E1803D All-In-One XY2-100 Scanner Controller

Users Manual

Table of Contents

1 Copyright

Portions of the E1803D firmware are based on lwIP 1.4.0 (or newer):

Portions of the E1803D firmware are based on FatFS R0.10a (or newer):

Portions of the E1803D firmware are based on StarterWare 2.0 (or newer):

Portions of the E1803D firmware are based on libzint-backend 2.0 (or newer):

PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The Motion Extension firmware bases on motion5 version 1.1 or newer*:

Copyright (c) 2018 Oxygenic, (c) 2012-2016 Sungeun K. Jeon for Gnea Research LLC, Copyright (c) 2009-2011 Simen Svale Skogsrud

motion5 is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
motion5 is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

*) GPL notice: the motion extension firmware is running separately and completely independent from the main controller firmware, they are neither linked nor compiled together with each other. The related motion5 sources, as used on motion extension, can be found at https://sourceforge.net/p/axis5/code/ci/master/tree/

2 History

Date	Changes in document
07/2019	Extended description of card state flags
07/2019	Added description about how to write stand-alone data via API
05/2019	Numbering of DIn and DOut of second digital port (Intelli-IO-Extension) corrected
02/2019	Added description of new API commands `E1803_digi_set_motf2()` and `E1803_digi_set_motf_powerctl()`
02/2019	Added "`pethd`" configuration parameter
01/2019	Added PID control loop API function description
12/2018	Added "`haltedloopbuffer`" configuration parameter
11/2018	Stepper motor pinout added for Intelli-IO extension
10/2018	Added commands `cscor` and `cgcor`
10/2018	Added "`haltedlooptimeout`" configuration parameter
09/2018	New tune-flag added
09/2018	Added description of Intelli-IO Extension Board and related API functions
05/2018	Added description of Motion Extension Board and motion API functions
04/2018	Description of new parameters "digiinit" and "digimask" added
04/2018	Description of new tune-flags added
04/2018	Added description of command E1803_digi_pulse()
03/2018	Added description of Multi-IO Extension Board
02/2018	New "d"-command 0x45 to update firmware
02/2018	Added description of flag `E1803_COMMAND_FLAG_ASYNC`
01/2018	Added description for u0bits and u0parity configuration parameters
12/2017	Added description for matrix-"d"-commands 0x40 and 0x41
10/2017	Added wiring description for IPG YLM lasers
08/2017	Added description for config parameters `wetout` and `mipout`
08/2017	Description for `E1803_set_sync()`/`E1803_get_sync()` added
07/2017	Description of USB license retrieval clarified
07/2017	TrueType support in stand-alone mode
04/2017	Added description of command `cscnc`
04/2017	Added wiring scheme for IPG YLR lasers
03/2017	Added description of supported G-Code commands
02/2017	Images updated
01/2017	Added wiring scheme for IPG type E lasers with APD index mode
01/2017	Added "`iohaltedloop`" stand-alone mode
12/2016	Description of `corrtable0` parameter corrected
12/2016	Initial version

3 Safety

The hardware component described within this document is designed to control a laser scanner system. Laser radiation may effect a person's health or may otherwise cause damage. Prior to installation and operation compliance with all relevant safety regulations including additional hardware-controlled safety measures has to be secured. The client shall solely be responsible to strictly comply with all applicable and relevant safety regulations regarding installation and operation of the system at any time.

Beside of that some laser equipment can be damaged in case it is controlled with wrong signals or signals outside a given specification. Thus it is highly recommended to check the output generated by this device using e.g. an oscilloscope to avoid problems caused by wrong configurations. This should be done prior to putting a system into operation for the first time, whenever some parameters have been changed or whenever any kind of software update was installed.

The hardware component described here is shipped without any cover and without prefabricated equipment for electric installation. It is intended to be integrated in machines or other equipment. It is not for use "as is". Prior to operation compliance with all relevant electric / electromagnetic safety regulations including additional hardware-controlled safety measures has to be secured. The client shall solely be responsible to strictly comply with all applicable and relevant regulations regarding installation and operation of the system at any time.

The hardware described here is an electrostatic sensitive device. This means it can be damaged by common static charges which build up on people, tools and other non-conductors or semiconductors. To avoid such a damage, it has to be handled with care and including all relevant procedures (like proper grounding of people handling the devices, shielding/covering to not to let a person touch the device unwanted, proper packaging in ESD-bags, ...). For more information please refer to related regulations and standards regarding handling of ESD devices.

This document describes the E1803D-hardware but may contain errors and/or may be changed without further notice.

4 Overview

This document describes the E1803D compact scanner controller board, its electrical characteristics and usage.

The E1803D scanner controller board is designed for controlling galvanometric scanner systems with two or three axes. It also supplies extensive signals for laser and external control. The communication between the host system and the controller boards is done via Ethernet or USB.

This is an all-in-one controller which provides all interfaces that are required to control common lasers and additional hardware. For a modular controller which can be configured according to some special requirements please refer to https://halaser.eu/E1803.php.

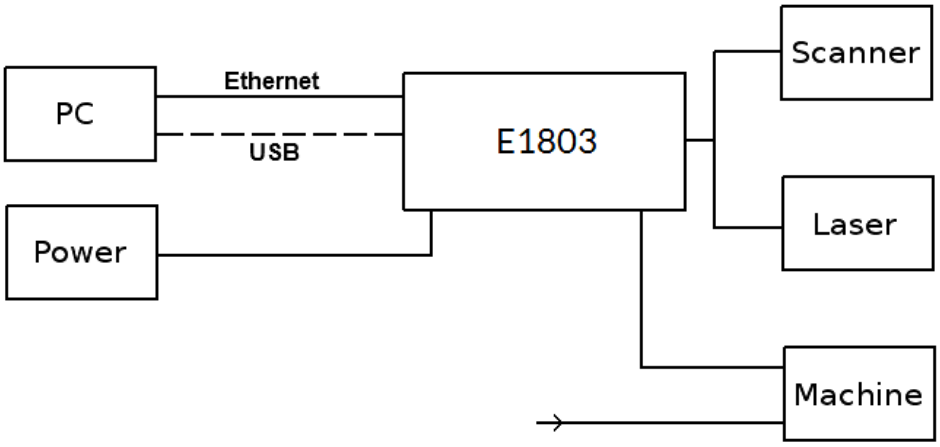
4.1 Features

The E1803D controller can be used to control 2D or 3D scanheads that come with a XY2-100 interface. It offers the following features:

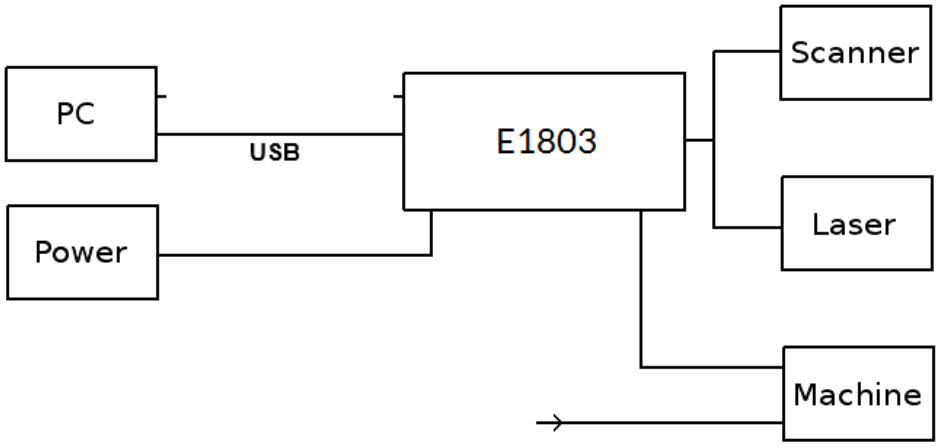
- XY2-100 and XY2-100-E interface to scanhead with X, Y and optional Z channel
- 100 Mbit Ethernet connection
- USB 2.0 connection
- online XYZ grid correction with support for several correction table file formats (like SCAPS™ .ucf, Scanlab™ .ctb and .ct5, Raylase™ .gcd, CTI™ .xml, Sunny™ .txt)
- high-definition online XYZ grid correction with BeamConstruct HD correction files (.bco)
- switching between up to 16 preloaded grid correction tables
- 10 microseconds vector cycle time and resolution (microstep period)
- command execution time down to 0,5 microseconds
- realtime processing of laser and scanner signals
- 26 bit internal resolution (for better accuracy also with 16 bit or 18 bit hardware output)
- 512 MByte DDR3 RAM
- 1 GHz CPU clock
- support for microSD and microSDHC cards
- optional matrix laser dot marking mode with up to 13 dots, up to two independent lines of text and up to 2 MHz dot frequency
- internal command and vector data list with more than 20 million entries
- continuous list concept, no need to swap between buffers
- BeamConstruct PRO license included
- open source compatibility library that emulates existing programming interface for fast and easy usage with existing software (contains e.g. Scanlab™ RTC4™, SCAPS™ USC™/SCI and other compatible interfaces)
- LP8 8 bit CMOS level parallel digital output e.g. for controlling laser power or laser waveform type
- LP8 latch CMOS level digital output for usage with IPG™ and compatible laser types
- Master Oscillator CMOS level digital output for usage with IPG™ and compatible laser types
- 12 bit 0..10V analogue output e.g. for controlling laser power (this output is a slave of LP8 outputs)
- two laser CMOS level digital outputs for usage with YAG, CO_2, IPG™, SPI™ and compatible laser types (outputs can provide PWM frequency, Q-Switch, FPK-pulse, CW/continuously running frequency, stand-by frequency) running with frequencies of up to 20 MHz
- 8 freely usable digital outputs providing either CMOS level or electrically insulated outputs via external power supply
- 8 freely usable digital inputs expecting either CMOS level or electrically insulated inputs via external power supply
- 4 digital inputs usable for quadrature encoder signals for 1D and 2D marking on-the-fly applications
- RS232/RS485 serial interface for communication with external devices
- scanhead power supply via controller card to save additional wiring

5 Position Within The System

The E1803D scanner controller system can be connected to the host via Ethernet or USB to receive laser marking data from BeamConstruct laser marking application, from ControlRoom process control software or from any other application which makes use of one of the provided programming possibilities (as described below). When using Ethernet connection, it optionally can be connected via USB too. In this case USB connection is used to retrieve BeamConstruct PRO license from the board:

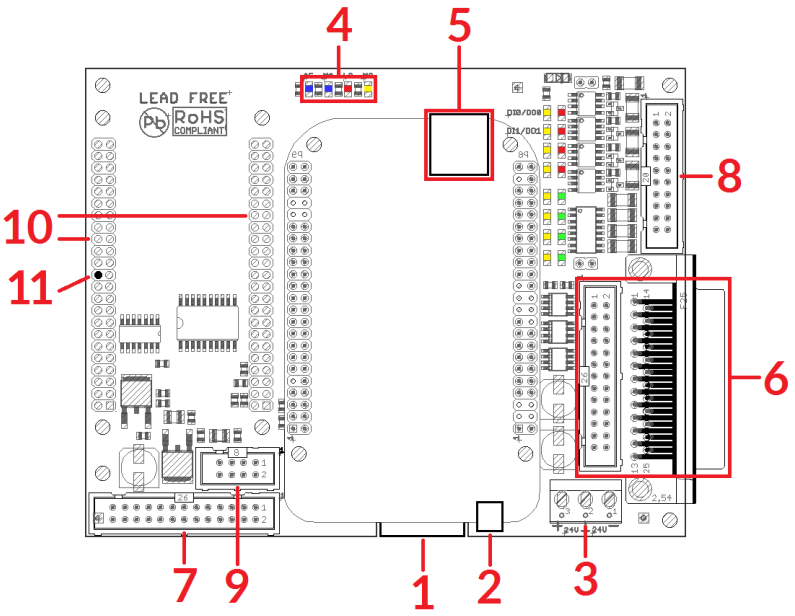


Since 100 Mbit Ethernet provides much faster data transfer than USB 2.0, this connection type is preferred. Especially in case complex marking data with many short lines that result in many separate jump and mark commands are used, Ethernet connection is more responsive.
When using USB connection with such data, time from sending data to the card until marking operation can be started may be longer (up to several seconds in worst case) caused by slower USB data transfer:



In both cases the board itself is connected with the scan head to submit 2D or 3D position information to it. Beside of that it is connected to a laser to submit motion-synchronous laser data. Additional communication channels between the E1803D scanner controller board and a connected machine can be done via separate IOs of the digital interface.

6 Board And Connectors



The E1803D Digital Laser Scanner Controller Card provides following connectors and interfaces:

1. Ethernet – for communication with the host system, marking information are submitted via this path
2. USB – via microUSB connector for providing BeamConstruct PRO license to host system and optionally for submitting marking data from host to E1803D card (in case Ethernet is not used)
3. Power – connect with power supply
4. User LEDs – show operational and error states of card
5. microSD-card – storage place for firmware and extended configuration file, can be used to upgrade firmware, to change the card's IP and other things more
6. Scanner signals – white 26 pin and D-SUB25 scanner output connector which provides XY2-100 scanner signals and power to scanhead
7. Laser signals – connector with different signal for controlling a laser and for starting/stopping mark operation
8. Digital interface – in- and output connector for control of external devices and for connecting marking on-the-fly encoder(s)
9. Serial interface connector for connections to RS232 or RS485 devices
10. Extension connectors
11. Code-pin for correct placement of extension boards

6.1 Ethernet

This is a standard RJ45 Ethernet plug for connection of the board with the host system. The controller board is accessed via this connection, all scanner and laser control data are sent via Ethernet. Thus it is recommended for security reasons to have a separate 1:1 connection from the host to the scanner controller card by using a separate Ethernet port. In case this is not possible, at least an own, physically separated sub-net for all scanner controller cards should be set up. This network of course should be separated from normal network completely. Ethernet connection is initialised during start-up only, thus Ethernet cable connecting E1803D board and host system needs to be plugged before the board is powered up.
By default the E1803D board is using IP 192.168.2.254, thus the Ethernet network the card is connected with needs to belong to subnet 192.168.2.0/24.
PLEASE NOTE: For security reasons it is highly recommended to not to mix a standard communication network with an E1803D network or to connect the scanner controller card with a standard network. Here it may be possible someone else in that network (accidentally) connects to that scanner controller and causes laser emission.
The IP of the scanner controller can be changed. This is necessary e.g. in case an other subnet has to be used or in case the E1803D board has to be operated in multi-head environments where more than one card will be

accessed at the same time. The IP can be configured using e1803.cfg configuration file that is placed on microSD-card. To change the IP, please perform the following steps:

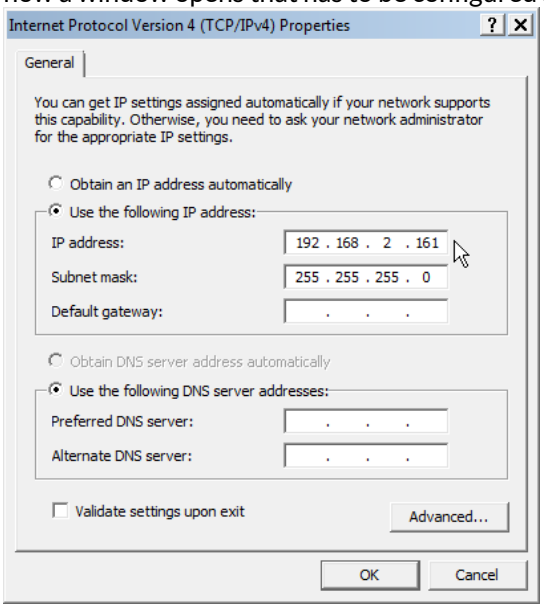
1. disconnect E1803D board from power and USB
2. remove microSD-card
3. put microSD-card into a desktop computer, this may require a microSD- to SD-card-adapter
4. open the drive that is assigned to the card
5. open file e1803.cfg using a text editor like Notepad or kwrite
6. add a line or edit an existing line "`ip0=`", here the desired IP has to be appended (as example: when you want to configure IP 192.168.2.13 the line has to be "`ip0=192.168.2.13`" – without any quotation signs
7. save the file
8. eject the drive the card is assigned to
9. place the microSD-card in E1803D board (place without the use of force, notice correct orientation with connectors of SD-card to top!)
10. power up card

When User LEDs do not light up as described below, please check if microSD-card is placed in board correctly.

6.1.1 Ethernet Configuration With Windows

When E1803D scanner controller is accessed via Ethernet, it is recommended to have a 1:1 connection to the host PC for security reasons. Since the controller is working with a static IP (default is 192.168.2.254) the Ethernet port on host PC has to be configured with an IP of same subnet in order to allow access to it. For Windows 7 (and similar) this configuration has to be done using following steps:

1. select “Start”-button and choose entry “Control Panel”
2. Select “Network and Sharing Center”
3. Select “Change adapter settings” in upper left corner
4. find the network interface E1803D has to be connected with and select it
5. find entry “Internet Protocol Version 4 (TCP/IP4)”
6. select it and press button “Properties”
7. now a window opens that has to be configured as follows:



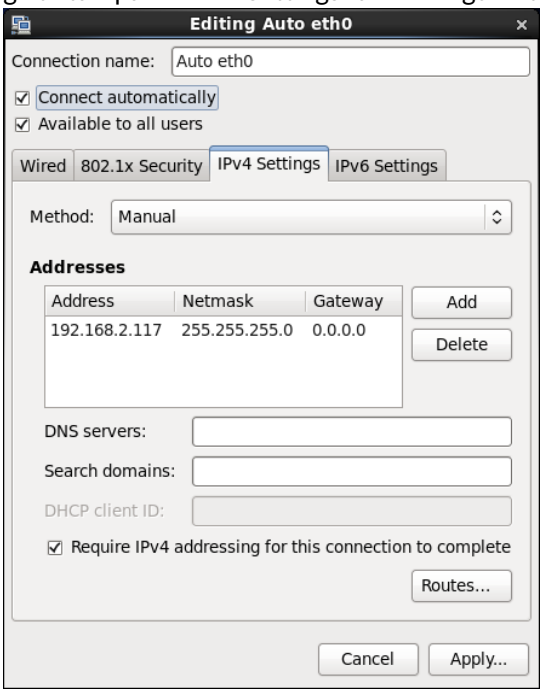
There you can specify an IP for your host PC. It has to belong to network 192.168.2.xxx and can be any number except than 192.168.2.254 (this is already the IP of the scanner card), 192.168.2.0 or 192.168.2.255.

6.1.2 Ethernet Configuration With Linux

When E1803D scanner controller is accessed via Ethernet, it is recommended to have a 1:1 connection to the host PC for security reasons. Since the controller is working with a static IP (default is 192.168.2.254) the

Ethernet port on host PC has to be configured with an IP of same subnet in order to allow access to it. For Linux (with NetworkManager) this configuration has to be done using following steps:

1. right-click the network-symbol in taskbar
2. click "Edit Connections..."
3. select the "Wired" network interface the scanner card is connected with and press button "Edit"
4. go to tab-pane "IPv4 Settings" and configure it as shown below:



There you can specify an IP for your host PC. It has to belong to network 192.168.2.xxx and can be any number except than 192.168.2.254 (this is already the IP of the scanner card), 192.168.2.0 or 192.168.2.255.

6.2 USB

This is a standard microUSB-connector for connection of the board with the host system. It is used to retrieve BeamConstruct PRO license and optionally to send marking data to the card. When USB is used for sending all scanner and laser data, Ethernet cable does not need to be connected.

PLEASE NOTE: USB 2.0 is much slower than a standard 100 Mbit Ethernet connection, so expect slower execution in case of complex marking data!

Required device driver is installed together with OpenAPC-setup (Windows) or comes with operating system by default (Linux). E1803D card appears as COM-interface on Windows using any free number for the port. With Linux it appears as /dev/ttyACMx where "x" is any number. These numbers are provided by the operating system automatically.

When no external power supply is connected, USB provides 5V power supply too. So whenever the card has to be stopped, both USB and power have to be disconnected in order to shut it down completely. It is not recommended to use USB as power supply, additional, external power should be connected in order to operate E1803D controller correctly. When E1803D is powered via USB only, not all functions are available. Here things like power supply of connected scanhead and 0..10V analogue output signals AOut0 and AOut1 will not work.

Depending on the capabilities of the used USB host, there also may be other failures and limitations caused by power-brownouts and drop-outs.

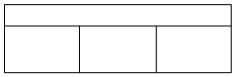
When the controller is connected via USB, a BeamConstruct PRO license is provided via this interface automatically. This is done without the need to configure anything, and as long as following conditions are true:

- physical USB connection from controller to host PC exists
- the COM-port (Windows) has a number smaller than COM20
- the controller is working and the Alive-LED in blinking

It is also possible to have the USB-connection for license retrieval only and to use the Ethernet-connection to transfer marking data to the controller, both can exist beside each other.

6.3 Power

Power supply for E1803D scanner controller board is done via 3 pin screw connector. Here a power in range +12..+24V or +-12..+-24V can be connected. This connector powers the board and optionally can be used to power the scanhead too (for details please refer to description of XY2-100 connector below).



Following possibilities to connect power exist:

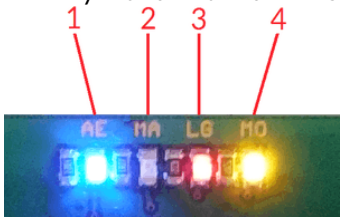
+V	GND	-V	Supported Features / Remarks
12..24 V	connected	unused	Power E1803D and analogue outputs AOut0 and AOut1
12..24V	connected	12..24V	Power E1803D, analogue outputs AOut0 and AOut1 and scanhead via XY2-100 connector, input voltage has to be the same voltage that is required to operate the scanhead
12..24V	unused	12..24V	**Not allowed!**
unused	connected	12..24V	**Not allowed!**
unused	unused	unused	Power supply needs to be done via USB, then only E1803D is powered
other	other	other	**Not allowed!**

When all three inputs are connected to a bipolar power supply providing +-12..+-24V to power both, the controller and a connected scanhead, the input voltage is feed to the scanhead directly via XY2-100 connector. Here the voltage has to be equal to the voltage required by the scanhead (typically +-15V or +-24V). Power supply needs to provide 1A plus current required for connected scanhead. For more details please refer to section about XY2-100 connector below.

ATTENTION: When connecting wires to the screw terminals of the power connector, do not transmit any force to the PCB where the green connector is soldered at! While screwing tight the wires, hold the connector by hand to catch the force but **do not hold the PCB**!

6.4 User LEDs

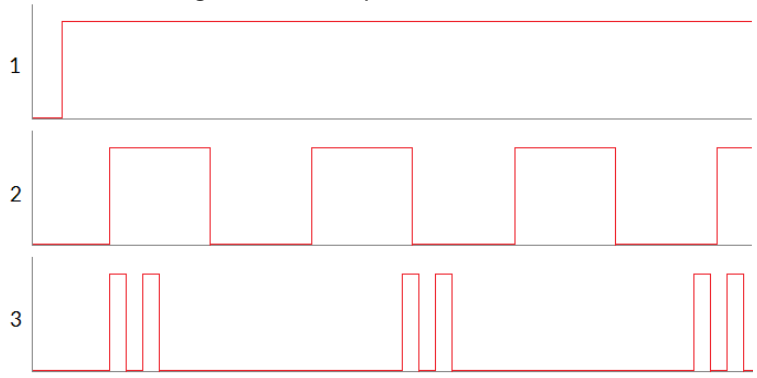
The real operational state of the card is shown by four additional LEDs described here:



1. AE (Alive/Error) – blue – this LED is turned on permanently (with full brightness) as soon as the card was powered up and the firmware boots properly. When it is not turned on with full brightness after some seconds, please check if the microSD-card is placed properly and if it contains a working firmware file (for details please refer below).
   After boot process has completed successfully, it starts blinking slowly and with same on and off times. This is an alive-notification, as long as it blinks, the board is working and ready for operation. During marking operations the blink frequency may vary.
   When this LED starts blinking with a changed on-time (LED is off for a long time and flashes twice for a short time only), a fatal error has occurred that normally should never happen. When this happens, in most cases the board can't continue with operation until the reason for error is removed and the board is restarted. In case this LED flashes signalling an error-state, please:
   - check if you are using valid E1803D extension boards only (and no other 3rd party hardware)
   - check if you are using latest firmware and host software
   - check all connections and cables

- undo your latest changes in hardware and configuration
If these steps do not help, please contact us for further assistance.

So this LED can signalise three operational states:

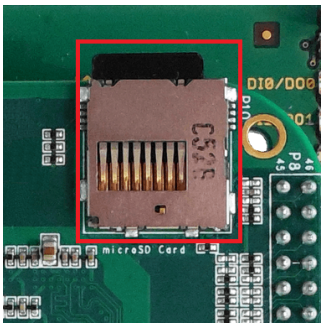


1 – Not ready / booting – turned on permanently with full brightness
2 – Operational – blinking, on and off times are the same
3 – Error – flashing twice, LED is off most of the time and is turned on for a very short time only

2. MA (Marking Active) – blue – this LED is turned on as long as a marking operation is running. This LED does not correspond to the laser gate signal, comparing to it it's also enabled during jumps when laser is turned off but marking operation itself is active.
3. LG (LaserGate) – red – this LED shows modulation state of the laser and signal the state of the laser gate output. It is turned on as long as the laser is turned on and the laser gate is high. This LED does NOT signal the same like the marking active LED described above since it will be turned off during jumps.
4. MO (Master Oscillator) – yellow – this LED is specific to the Master Oscillator output signal described below. As long as the signal is on (HIGH-signal at output pin), the LED is turned on.

6.5 microSD-Card

The microSD-card is storage place for firmware and configuration files. Here SD and SDHC cards with a capacity of up to 32 GB are supported. It is plugged with the contacts of the SD-card oriented to upper side.



To remove the microSD-card, first disconnect all power from the E1803D board completely (including USB). Next press microSD-card gently into the board until you can hear a click-noise. Then you can pull it out of the board. To place a microSD card, the same has to be done in reverse order: place it into the E1803D board's card slot and press it gently until a noise signals locking of the card. Now the board can be powered.
E1803D board is shipped with a card containing firmware and configuration files:

- e1803.fwi – firmware file that is used to operate the board, to be replaced when a firmware update is provided;
- e1803.cfg – configuration text file, can be edited using a text editor in order to modify cards configuration
- e1803.dat – additional data file that is used to operate the board, to be replaced when a firmware update is provided
- fonts/ – subdirectory containing fonts for dot marker mode

To use an other microSD card than the one shipped with the board, following conditions have to be met:

- maximum total size of 32 GB (SD or SDHC card)
- FAT32 formatted
- using only one partition
- BOOT-flag is set
- e1803.fwi and e1803.dat file available on card (e1803.cfg is optional)

An additional file e1803.cfg can be placed on the card too. It contains plain ASCII text, acts as configuration file and can contain several parameters and its values which are separated by an equal-sign. Every of the possible parameter/value pairs has to be located in an own line. Following configuration parameters are possible within this file:

Parameter	Description	Example
ip0	Configures IP of Ethernet port. Here only IPs in xxx.xxx.xxx.xxx notation are allowed but no host or domain names.	`ip0=192.168.2.100` specifies IP 192.168.2.100 to be used for Ethernet interface on next startup
corrtable0	Specifies a correction table file in .bco, .ctb, .ct5, .ucf, .gcd, .xml or .txt format to be loaded on start-up. When this parameter is set, the specified correction table is used exclusively and all correction data possibly sent from the host are ignored. The correction file itself has to be located on microSD-card too. When the Error-LED is turned on after a correction table file was configured, E1803D board was not able to load it for some reason.	`Corrtable0=0:/D2_200.ctb` use file D2_200.ctb as correction file and ignore all correction tables possibly sent from host application
corrtable<idx>	Specifies one of up to 16 correction table file in .ctb, .ct5, .ucf, .gcd or .bco format to be loaded on start-up. When this parameter is set, the specified correction table is used exclusively and all correction data possibly sent from the host are ignored. The correction file itself has to be located on microSD-card too. This method has also to be used when running the controller in stand-alone mode with .EPR files that require such a correction. When the Error-LED is turned on after a correction table file was configured, E1803 baseboard was not able to load it for some reason. `<idx>` can be any value in range 0..15 and specifies the storage location index of the correction file to be loaded. Later the related correction file can be used via command `cscor`. When `<idx>` has to be set to values greater than 0, a firmware version 6 or newer is needed.	`corrtable7=0:/200_200.bco` use file 200_200.bco as correction file at index position 7 and ignore all correction tables possibly sent from host application

Parameter	Description	Example
passwd	Specifies an access password that is checked when card is controlled via Ethernet connection. This password corresponds to password specified with function `E1803_set_password()`, please refer below for a detailed description. When a client computer connects to the card without sending the correct password, Ethernet connection to this host is closed immediately. PLEASE NOTE: this password does not replace any network security mechanisms and does not give the possibility to operate E1803D controller via insecure networks or Internet! It is transferred unencrypted and therefore can be "hacked" easily. Intention of this password is to avoid collisions between several E1803D cards that operate in same network and are accessed by several software instances. Maximum allowed length of the password is 48 characters. It is recommended to not to use any language-specific characters.	`passwd=myCardPwd` set a password "myCardPwd"
standalone	This command can be used to disable or enable a specific stand-alone operation mode. For a detailed description of possible parameters, operation modes and usage please refer related section below.	
haltedlooptimeout	This parameter is used in stand-alone modes “haltedloop” and “iohaltedloop” (please refer to section “7 Stand-Alone Operation” for detailed information). It defines a timeout for the laser in unit seconds. If the current operation is active for a longer time, the laser is turned off. It then can be turned on only by toggling the enable-input (ExtStart) again. This parameter requires firmware version 6 or newer.	`haltedlooptimeout=5` sets the laser timeout to 5 seconds
haltedloopbuffer	This parameter is used in stand-alone modes “haltedloop” and “iohaltedloop” (please refer to section “7 Stand-Alone Operation” for detailed information). It defines a maximum buffer size for the marking data. The buffer size should have a size of 20000000 at max. The minimum size depends on the specific application, in fact, when it is set to some too small values, drop-outs in marking operation may occur. Data which are already buffered in this marking mode can't be modified any longer. So any change on marking speed, laser power or similar (done e.g. by commands “`cjsor`”, “`cmsor`” or “`cpwor`”) will apply only to data which are not yet buffered. And as bigger as this buffer is, as longer it takes until the first new data after change of any of these parameters can be emitted. This parameter requires firmware version 5 or newer.	`haltedloopbuffer=100000` set the buffer to a maximum size of 100000 commands which is similar to data for about 1 second marking time

autofile	Loads a special .EPR stand-alone file or .CNC G-Code file from SD-card in some specific stand-alone modes. For a detailed description of possible parameters, operation modes and usage please refer related section below. For a description of supported G-Code commands, please check out related section “13 Supported CNC G-Code Commands“	`autofile=0:/markdata.epr`
		loads a file markdata.epr from disk; here 0:/ specifies the SD-card to be used. The .EPR-file itself can be generated within BeamConstruct out of a normal .BEAMP project file
		
		`autofile=0:/markdata.cnc`
		same as above but a G-Code file is provided which contains marking information

Parameter	Description	Example
iobuff	Pre-loads one or more .EPR files to the RAM of the controller to allow faster switching in “ioselect” stand-alone mode. This command can not be used to load file “0.EPR”	`iobuff=1` pre-load file 1.EPR on board start-up
mipout	Configure a Digi I/O output pin to be used as “mark in progress”-signal by default; here an output bit number in range 0..7 has to be configured which will be set to HIGH as long as a marking operation is in progress, the value given here can be overwritten by API-function `E1803_digi_set_mip_output();` this parameter requires firmware version 3 or newer	`mipout=1` use DOut1 for mark-in-progress signal
wetout	Configure a Digi I/O output pin to be used as “wait for external trigger”-signal by default; here an output bit number in range 0..7 has to be configured which will be set to HIGH as long as a marking operation is in progress and the controller is waiting for an external trigger signal to arrive at ExtStart input, the value given here can be overwritten by API-function `E1803_digi_set_wet_output();` this parameter requires firmware version 3 or newer	`wetout=0` use DOut0 for mark-in-progress signal
digiinit	Initialises the digital outputs on firmware start-up with the given defaults. This overrides the hardware defaults. The default digital values set here are NOT available on power up but a few seconds later after firmware has been loaded and started. This function requires firmware version 5 or newer.	`digiinit=2` set DOut1 to HIGH initially and all other outputs to LOW
digimask	Masks the digital inputs and specifies which inputs can be read. All input bits which are ignored by this command by setting the related value to 0, are no longer read. This may be useful for applications where encoder inputs are used together with a IOSelect stand-alone operation and where the random state of the encoder has to be masked out. This function requires firmware version 5 or newer.	`digimask=253` use only DIn2..DIn7 as input and ignore DIn0 and DIn1
digidebc	Sets a debouncing time / filter time for the digital inputs of the digital interface in order to not to let the inputs react on noise or bouncing of mechanical inputs. The debouncing value is given in time-units where every time-unit is equal to 31 usec. By default 7 time-units are set.	`digidebc=10` set the debounce-time to 310 usec
fastdebc	Sets a debouncing time / filter time for the ExtStart and ExtStop inputs in order to not to let the inputs react on noise or bouncing of mechanical inputs. The debouncing value is given in time-units where every time-unit is equal to 31 usec. By default 7 time-units are set.	`fastdebc=10` set the debounce-time to 310 usec
u0brate	Set the bitrate of UART0 RS485/RS232 serial interface on E1803D. By default this port is initialised with a speed of 115200 bps, this value can be changed with this parameter. Setting an u0brate of 0 disables the serial port completely	`u0brate=9600` set a new bitrate of 9600 bps for UART0 E1803D on-board serial port
u0bits	Set the number of data bits of UART0 RS485/RS232 serial interface on E1803D. By default this port is initialised with 8 data bits, this value can be changed to a word length of 5, 6, or 7 bits with this parameter. This parameter requires firmware version 4 or newer.	`u0bits=7` set a new word length of 7 bits for UART0 E1803D on-board serial port
u0parity	Set the parity of UART0 RS485/RS232 serial interface on E1803D. By default this port is initialised no parity (=0). For odd parity a value of 1 has to be set, for even parity a value of 2 has to be used. This parameter requires firmware version 4 or newer.	`u0parity=2` enable even parity for UART0 E1803D on-board serial port

Parameter	Description	Example
u1brate	Set the bitrate of UART1 RS485/RS232 serial interface on E1803D Multi-IO Extension Board. By default this port is disabled and has to be activated by setting a bitrate. This parameter requires firmware version 4 or newer and a Multi-IO Extension Board.	`u1brate=115200` set a new bitrate of 115200 bps for UART1 serial port
u1bits	Set the number of data bits of UART1 RS485/RS232 serial interface on E1803D Multi-IO Extension Board. By default this port is initialised with 8 data bits, this value can be changed to a word length of 5, 6, or 7 bits with this parameter. This parameter requires firmware version 4 or newer and a Multi-IO Extension board..	`u1bits=7` set a new word length of 7 bits for UART1 serial port
u1parity	Set the parity of UART1 RS485/RS232 serial interface on E1803D Multi-IO Extension Board. By default this port is initialised no parity (=0). For odd parity a value of 1 has to be set, for even parity a value of 2 has to be used. This parameter requires firmware version 4 or newer and a Multi-IO Extension Board.	`u1parity=1` enable odd parity for E1803D on-board serial port

Parameter	Description	Example
tune	Enables special functions and features that are not activated by default. As parameter a number can be handed over that specifies the functions to be enabled. Several of these functions can be combined by adding their related numbers: 1 – use DIn7 of digital interface connector as external trigger, this disables ExtStart input on laser signal connector 2 – use additional marking encoder inputs on DIn2 and DIn3 for 2D marking on-the-fly operations 4 – enable storage of serial number count values to microSD card; this option is useful in case of stand-alone operation mode when dynamic data with serial number counting is used. When it is set, the current count value of all used serial numbers is stored and reloaded on next power up. Thus their values are not get lost when power was turned off. The values are stored in a file with the same name like the "autofile" or the currently loaded .epr file but with extension ".ser".  ATTENTION: The file is saved on the FatFS formatted microSD card. FatFS is NOT fault-proof, means it can be corrupted when power is turned off during writing. So when this option is enabled, user has to ensure power is NOT turned of while the card writes to disk. Writing of serial number states is always done in case they have changed, then it is started when Alive/Error LED of E1803D board is switched off. Write operation is finished when this LED is turned back on the next time. So to ensure data are written successfully, it is recommended to let this LED blink two times after last mark operation has been finished or to wait for about 4 seconds.  ATTENTION: due to this limitation it is not recommended to work with this option but to save the state of the serial numbers by sending ASCII command "`cssta`" instead (please refer below for details)! 8 – invert LaserGate output to work as active HIGH signal; when this option is set, logic of LaserGate-LED changes too, it is on as long as laser is turned off and it is off as long as laser is on 16 – invert LaserA output to work as active HIGH signal 32 – invert LaserB output to work as active HIGH signal 4096 – operate in enhanced XY2-100 18 bit mode; when this value is added to the tune-parameter, the controller outputs more accurate 18 bit position data instead of the standard 16 bit values in normal operation mode; this mode needs to be supported by the connected scanhead, elsewhere the results are unpredictable. 32768 – invert the mark-in-progress signal (requires firmware version 5 or newer) 65536 – invert the wait-external-trigger signal (requires firmware version 5 or newer) 262144 – enable a more exact correction calculation (requires firmware version 5 or newer). With this option set, a	`tune=1` disables ExtTrig inputs and switches over external trigger function to DIn7 input

Parameter	Description	Example
sntp0	Allows to specify the IP of an SNTP time server. This option can be used in case of Ethernet usage to synchronise controller with an external time source. E1803D tries to connect to this server after initialisation of Ethernet interface and – if not successful – a few more times. These additional connection attempts are done whenever the Alive/Error-LED is switched on. ATTENTION: when this function has to be used, the network or host-computer the controller is connected with needs to be able to route this request. This is a potentially dangerous operation because a connection between encapsulated machine network and open and dangerous Internet has to be established. Since this is NOT RECOMMENDED in general, this option should be used ONLY when it is 100% sure there is no possibility for people from outside to intrude the machine network! Instead of that is is recommended to set system time manually using host-computer and ASCII command "cstime" (please refer below). Alternatively it is also possible to contact an own, network-internal NTP-server. When this option is used, the gateway and netmask have to be configured for the controllers Ethernet interface	`sntp0=83.170.1.42` – IP of time server at 3.de.pool.ntp.org is used for SNTP time retrieval (not recommended since this requires a connection to potentially dangerous Internet!)
sntp0offset	This value corresponds to sntp0 parameter above, it is used when system time is retrieved from an external time server to set an offset to the time returned from this server. The offset has to be specified in unit seconds.	`sntp0offset=-3600` – specifies an offset of minus one hour to the time returned from time-server. So when the time server would return a current time of 11:42:17, the system time of the controller would be set to 10:42:17 with this value
gw0	Specifies a gateway-address for the scanner controllers Ethernet interface. This option belongs to parameter "ip0" and has to be set in case "sntp0" is used.	`gw0=192.168.2.1` – use 192.168.2.1 as gateway
nm0	Specifies the netmask for the scanner controllers Ethernet interface. This option belongs to parameter "ip0" and has to be set in case "sntp0" is used.	`nm0=255.255.255.0` – use upper 24 bits of current IP for netmask
usb	When this parameter is set to 0, USB interface is disabled completely. This means it is no longer possible to connect to E1803D USB serial interface via terminal software or via BeamConstruct and it is also no longer possible to retrieve BeamConstruct PRO license via USB. This option can be used to suppress illegal access to USB and saves some power.	`usb=0` – turn off USB interface
eth	When this parameter is set to 0, Ethernet network interface is disabled completely. This means it is no longer possible to connect to E1803D via Telnet or via BeamConstruct. All SNTP-functionalities are disabled too. This option can be used to suppress illegal access to Ethernet, to save several seconds of startup-time and some power.	`eth=0` – turn off Ethernet interface
pethd	When Ethernet connection is used, it has to be established on power-up of the controller card as this connection is set-up and configured by the controller only once during boot. There may be situations where the other side of the Ethernet connection can not boot up as fast as E1803. In such cases this parameter can be used. It delays initialisation of Ethernet by the time given as parameter. The time is specified in unit “delayticks” where one “delaytick” is equal to about 0,5 seconds. This feature requires a firmware version 7 or newer.	`peth=20` – halt initialisation of the controller for about 10 seconds prior to initialisation of Ethernet interface

Parameter	Description	Example
dotfont0 dotfont1 dotfont1y dotdist dottime	These commands are related to matrix laser dot marking mode. For details please refer related section 8 below.	

6.5.1 Firmware Update

As described above, the firmware is located on microSD-Card and therefore can be updated easily:

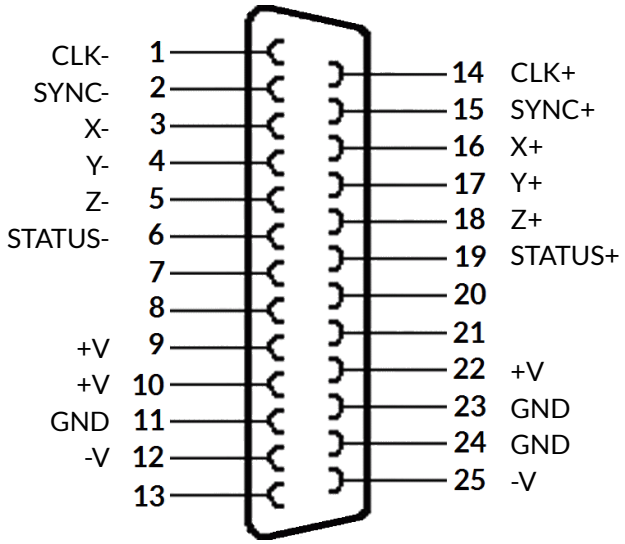
1. remove the microSD-Card as described above
2. download a new firmware from https://openapc.com/download/Firmware/E1803/ (the higher the number in the file name, the newer the firmware is)
3. copy the contents of this ZIP-file to microSD-Card (please take care about e1803.cfg in case it contains a changed configuration)
4. reinsert microSD-Card as described in previous section

6.6 Scanner Signals

The white 26 pin connector provides signals to be used to control up to three galvos of a scanhead and to power it up. The connector is a white one to avoid confusion with the 26 pin but black LP8 laser signal connector. The connector provides following signals:

Upper Row Of Pins	Signal	Voltage	Remarks	Lower Row Of Pins	Signal	Voltage	Remarks
1	CLK-		XY2-100-compatible signals	**2**	CLK+		XY2-100-compatible signals
3	SYNC-			**4**	SYNC+		
5	X-			**6**	X+		
7	Y-			**8**	Y+		
9	Z-			**10**	Z+		
11	STATUS-			**12**	STATUS+		
13				14			
15				16			
17	+V	+12..24V	Power supply to scanhead	**18**	+V	+12..24 V	Power supply to scanhead
19	+V	+12..24V		**20**	GND	GND	
21	GND	GND		**22**	GND	GND	
23	-V	-12..24V		**24**	-V	-12..24V	
25	-V	-12..24V		26			

The D-SUB25 connector provides the same signal as described above on a default XY2-100 connector:



-V

The connections -V, GND and +V can be used to power the scanhead with 12..24V and **max. 3A**. This requires a bipolar external power supply connected to the controllers three-pin power connector described above. Power from this power connector is routed to the -V, GND and +V pins directly, so the provided voltage should be stabilised according to the requirements of the scanhead.

PLEASE NOTE:

- do not connect scanheads that consume more than 3A (peak and continuously), this may damage the controller and voids warranty!
- do not feed more than 24V into the three-pin power connector of E1803!
- feed a stabilised voltage into E1803D controller according to requirements of connected scanhead!
- when E1803 card is powered via three-pin power connector but scanhead has not to be powered out of the card, the 9 lines for -V, GND, +V (9..13 and 22..25) need to be disconnected, means the used D-SUB25 cable needs to leave these pins open!
- **Violating one of these rules may damage the E1803D card irreversibly!**

6.7 Laser Signals

The black 26 pin connector provides several signals to be used to control a laser source. It can be used e.g. together with YAG, CO_2, IPG™, SPI™, fiber and compatible lasers since it provides additional signals and frequencies these laser types may require for proper operation.

The connector provides following signals:

Upper Row Of Pins	Signal	Voltage	Remarks	Lower Row Of Pins	Signal	Voltage	Remarks
1	LP8_0	CMOS, 0/5V, max 8 mA		**2**	GND	GND	
3	LP8_1	CMOS, 0/5V, max 8 mA		4			
5	LP8_2	CMOS, 0/5V, max 8 mA		**6**	5V	5V	
7	LP8_3	CMOS, 0/5V, max 8 mA		**8**	MO	CMOS, 0/5V, max 8 mA	Master Oscillator
9	LP8_4	CMOS, 0/5V, max 8 mA		10			
11	LP8_5	CMOS, 0/5V, max 8 mA		**12**	AOut0	0..10V, max 15 mA	Analogue output
13	LP8_6	CMOS, 0/5V, max 8 mA		**14**	AOut1	0..10V, max 15 mA	Analogue output
15	LP8_7	CMOS, 0/5V, max 8 mA		**16**	ExtStart	CMOS, 0/5V	Input control signal
17	LP8 Latch	CMOS, 0/5V, max 8 mA		**18**	5V	5V	
19	LaserB	CMOS, 0/5V, max 14 mA	FPK	**20**			Connected to pin 21
21			Connected to pin 20	**22**	LaserA	CMOS, 0/5V, max 14 mA	PWM, frequency or Q-Switch
23	GND	GND		**24**	ExtStop	CMOS, 0/5V	Input control signal
25	5V	5V		**26**	Laser Gate	CMOS, 0/5V, max 14 mA	

LP8_0...LP8_7 provide a parallel 8 bit output signal (e.g. for power control with IPG™/fiber lasers, waveform selection for SPI™ lasers and other).

LP8 Latch pin signals valid output at LP8_0..LP8_7 by submitting a latch pulse of software-controlled length.

MO can be used to enable master oscillator (e.g. for IPG™/fiber lasers or compatible), this signal is also visualised by the MO LED described above.

LaserA usage depends on software configuration and control, it is able to output a pulse-width modulated frequency (e.g. for controlling CO2 lasers), CW/continuously running frequency (e.g. for fiber lasers) or Q-Switch signal (e.g. for YAG lasers) in range 25 Hz..20 MHz.

LaserB can be used for emitting a FPK pulse (e.g. for YAG lasers).

AOut0 and AOut1 provide unipolar analogue output for controlling e.g. laser power or additional equipment or can be used for controlling power and simmer for SPI™ lasers.

PLEASE NOTE: output of 10V at AOut0 and AOut1 depends on the used power supply. So in case board is powered via USB, these outputs do not work, they require an external power supply via three-pin power connector described above.

ExtStart expects a CMOS-level input signal in respect to GND and can be used as external trigger signal to start operations when a HIGH-signal is detected at input pin.

ExtStop expects a CMOS-level input signal in respect to GND and can be used as external stop-signal in order to stop a running marking operation by using a HIGH-signal at input pin.

6.8 Digital Interface

This interface consist of different parts which belong together:

1. a 20 pin connector for connecting digital in- and output signals
2. two red jumpers to select opto-insulated or internal powered mode for the digital in- and outputs
3. 4 green and 4 red LEDs which signal the state of the digital outputs
4. 8 yellow LEDs which signal the state of the digital inputs

The 20 pin connector provides 8 lines for input and 8 lines for output of digital signals that can work on CMOS level (non-insulated mode) or via opto-couplers (electrically insulated mode with external power supply) optionally. The operation mode depends on jumper settings described below. The connector is used as follows:

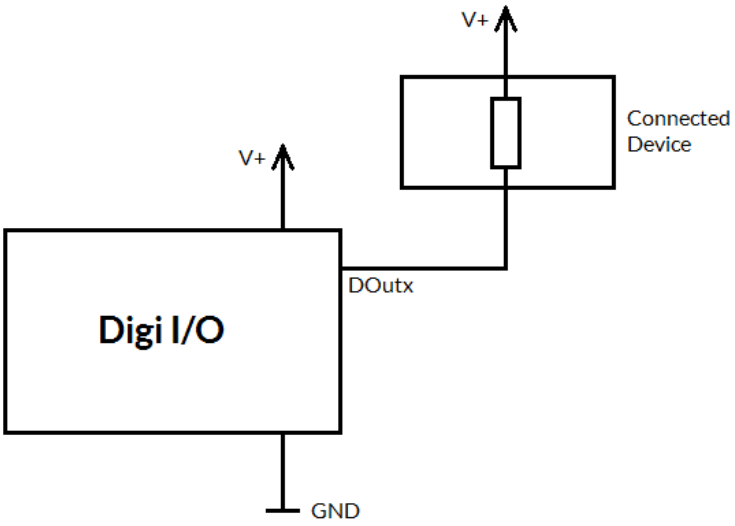
Upper Row Of Pins	Signal	Voltage	Remarks	Lower Row Of Pins	Signal	Voltage	Remarks
1	V_{ext}	5..24V	Input voltage to be used in opto-insulated mode only	**2**	GND_{ext}	GND	External ground
3	DOut0	CMOS, 0/5V or 0/V_{ext}	Default level: LOW	**4**	DIn0	CMOS, 0/5V or 0/V_{ext}	Encoder-input A1 for marking on-the-fly
5	DOut1	CMOS, 0/5V or 0/V_{ext}	Default level: LOW	**6**	DIn1	CMOS, 0/5V or 0/V_{ext}	Encoder-input B1 for marking on-the-fly
7	DOut2	CMOS, 0/5V or 0/V_{ext}	Default level: LOW	**8**	DIn2	CMOS, 0/5V or 0/V_{ext}	Encoder-input A2 for marking on-the-fly
9	DOut3	CMOS, 0/5V or 0/V_{ext}	Default level: LOW	**10**	DIn3	CMOS, 0/5V or 0/V_{ext}	Encoder-input B2 for marking on-the-fly
11	DOut4	CMOS, 0/5V or 0/V_{ext}	Default level: HIGH	**12**	DIn4	CMOS, 0/5V or 0/V_{ext}	
13	DOut5	CMOS, 0/5V or 0/V_{ext}	Default level: HIGH	**14**	DIn5	CMOS, 0/5V or 0/V_{ext}	
15	DOut6	CMOS, 0/5V or 0/V_{ext}	Default level: HIGH	**16**	DIn6	CMOS, 0/5V or 0/V_{ext}	
17	DOut7	CMOS, 0/5V or 0/V_{ext}	Default level: HIGH	**18**	DIn7	CMOS, 0/5V or 0/V_{ext}	
19	V	5V	Board voltage, to be used only when not operating in insulated mode	**20**	GND	GND	Board-internal ground

V_{ext} and GND_{ext} depend on opto-configuration as described below. In opto-insulated mode (opto-configuration jumpers not set) external power supply has to be connected to these inputs. Then DIn0..DIn7 and DOut0..DOut7 work in respect to this external power.

WARNING: When no opto-insulated mode is selected (opto-configuration jumpers are set), do NOT FEED ANY POWER into V_{ext}, this would cause damage to the E1803D board! In this case V_{ext} is equal to V (5V) of the board and GND_{ext} is connected to boards ground GND.

Maximum current for every output is 15 mA when internally powered (non-insulated mode), here it is recommended to use an external power supply.
Maximum current for outputs DOut0..DOut3 is 50 mA when externally powered (V_{ext} in insulated mode).

Signal output lines DOut0..DOut7 operate in open collector mode and have to be wired as follows:



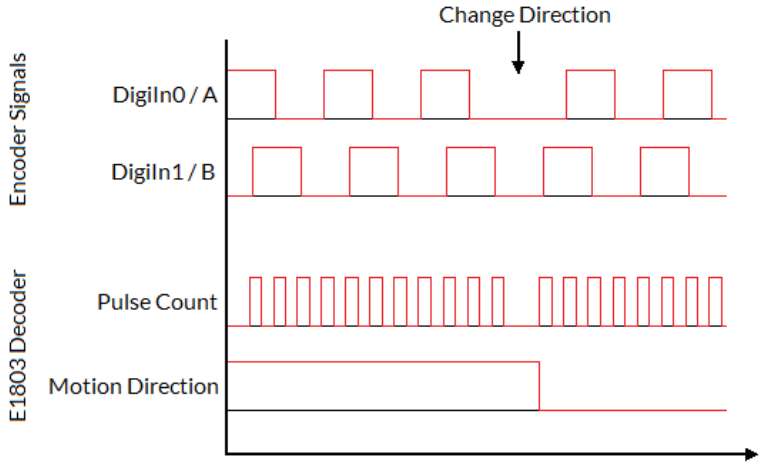
Here "DOutx" symbolises one of the digital outputs DOut..DOut7. V+ is either V (5V internal, non-insulated mode) or V_{ext} (up to 24V external, insulated mode). GND is either GND (non-insulated mode) or GND_{ext} (insulated mode). The internal resistor of the connected device is not allowed to have less than 490 Ohms in order to not exceed the given current limits.
DOut0..DOut3 provide LOW signal level by default, DOut4..DOut7 provide HIGH level by default. These levels are valid immediately on power-up of the card.

The DOut-lines make use of following logic:

Signal	LED	LED turned on on output level	Default output level on power-up
DOut0	red	HIGH	LOW
DOut1	red	HIGH	LOW
DOut2	red	HIGH	LOW
DOut3	red	HIGH	LOW
DOut4	green	LOW	HIGH
DOut5	green	LOW	HIGH
DOut6	green	LOW	HIGH
DOut7	green	LOW	HIGH

6.8.1 Marking On-The-Fly Signals

Digital inputs 0 and 1 (and optionally 2 and 3) can be used as position encoder signal inputs for marking on-the-fly applications. Here 90 degree phase-shifted input pulses are expected signalling motion direction and position change:



When these pulses are generated from a motion stage that moves the working piece, the resulting position information is used in marking on-the-fly mode to correct the marking positions accordingly. Resulting from that, marking will follow motion as far as available scanhead range and working area allows it.
The pulses generated out of the encoder signals have to be multiplied with a factor reflecting the resolution of the used encoder. To set up and adjust a marking on-the-fly-system properly, following steps have to be performed:

1. Connect encoder signals A and B to DigIn0 and DigiIn1 and configure E1803D controller for encoder usage (either from within ControlRoom/BeamConstruct or via programming interface as described below)
2. Mark a square without any encoder signals feed into the controller
3. When the square does not have exact size and/or is distorted, modify correction table and/or gain settings
4. When the square has correct size, mark it again but now with a slow motion (using encoder pulses)
5. When the square is damaged (means open on one side or compressed) the on-the-fly-factor has to be changed (set to a smaller or higher value)
6. Mark the same square again with a fast motion (using encoder pulses)
7. When the square is damaged (means open on one side or compressed) the on-the-fly-factor has to be changed (set to a smaller or higher value)

The on-the-fly-factor controls the strength of compensation and is the relation between speed of external device/encoder pulses and card-internal compensation calculation. When this factor is wrong, the marking results are distorted. For a square (as recommended to be used in calibration steps above) following results are imaginable:

The left drawing shows an over-compensated system, here the internal compensation is too strong, the factor is too big. The right drawing shows an under-compensated set-up, here the factor is too small causing a too weak compensation. Only when marking result is really a square, the on-the-fly-factor is correct.

When tune-flag 2 is set, a second encoder can be used for 2D marking on-the-fly applications. In this mode digital inputs 0 and 1 (encoder inputs A1 and B1) correspond to X axis and on-the-fly factor for X direction. Additionally digital inputs 2 and 3 (encoder inputs A2 and B2) correspond to Y axis and on-the-fly factor for Y direction. Operation principle is the same as for 1D on-the-fly described above: the incremental values received

from the encoders for X and Y are added to the current X and Y coordinates to be marked. Procedure for adjusting the encoder factor is also the same, here it is recommended to perform this operation for X and Y movements separately and finally try both motion directions together.

6.8.2 Opto-Configuration Jumpers

Using these jumpers the operation mode for digital I/Os 0..7 can be chosen. When they are set, the opto-couplers are powered internally. In this mode it is not working in opto-insulated mode and I/Os are using CMOS level signals.

When they are not set, external power and ground has to be provided at 20 pin connector (as described above) and these digital I/Os are working in electrically insulated, opto-coupled mode.

6.8.3 Output State LEDs

The green and red LEDs close to the connector signal the output state of the digital outputs. As shown in table above, four outputs have default state LOW (non inverted) and four have default state HIGH (inverted). The same is signalled by these green/red LEDs, they are on/off for output state HIGH/LOW. So please note: one of the output LEDs turned on does NOT necessarily mean the output is at HIGH level, this depends on the output and its corresponding default output level/output logic!

6.8.4 Input State LEDs

These 8 yellow LEDs show the state of corresponding 8 digital inputs. As long as a HIGH signal is detected on an input, the related LED is turned on.

6.9 Serial Interface

This is an 8 pin connector which provides access to UART0 RS232 and RS485 connection lines.
PLEASE NOTE: both, the RS232 and the RS485 interface are connected to the same serial interface internally! This means although there are two interface types available, only one logical serial line exists! Connecting two signal lines to RS232 and RS485 at the same time may damage the board irreversibly!

Upper Row Of Pins	Signal	Voltage	Remarks	Lower Row Of Pins	Signal	Voltage	Remarks
1	RX0	12V	UART0 RS232	**2**	TX0	12V	UART0 RS232
3	GND			4			
5	RX0+	+5V	UART0 RS485	**6**	RX0-	-5V	UART0 RS485
7	TX0+	+5V		**8**	TX0-	-5V	

6.10 Extension Connectors

These connectors can be used to plug additional boards which provide new functions and additional interfaces. For a detailed description of available boards please refer below.

The two extension connectors on left hand side of the board can be used to place extension boards with additional peripheral interfaces. The extension connectors are designed to place/remove boards from time to time but they are not intended for constant hardware changes. So changing extension boards repeatedly and often e.g. as permanent part of a production process is not recommended.

PLEASE NOTE: when placing a new extension board
1.check correct orientation and position of the code pin which is closed in connector
2.place the pins of the extension boards onto the extension connectors exactly
3.move down the extension board by pressing on its extension connectors gently; DO NOT PRESS THE BOARD ITSELF BUT ONLY THE CONNECTORS!

PLEASE NOTE: When removing an extension board DO NOT pull on the extension connectors but hold both boards on their long side directly at the PCBs edges!

Due to of the large number of pins, it is easy to plug in an extension but more difficult to pull it out. So when removing an extension board, it is recommended to be very slow and to carefully pull each side up just a little bit to avoid bending of the pins as they exit.

7 Stand-Alone Operation

E1803D scanner controller cards can be operated in stand-alone mode. In this mode all marking data are stored on microSD-card and the board can operate without direct control of a host-PC that sends the data to be marked. Such stand-alone marking data can be created e.g. in BeamConstruct marking software.

7.1 Create Stand-Alone Data with BeamConstruct

To use BeamConstruct for generation of stand-alone data for E1803D scanner controllers, the card has to be fully configured (including all scanner, laser and pen-parameters). Next the marking data to be stored on microSD-card have to be created. To generate stand-alone data, menu "Processing", submenu "Write Marking Data to File" or "Send Named Marking Data" has to be selected.

First one gives the possibility to write the data to microSD card when E1803D is switched off and the microSD card is plugged into host PC. Here it is recommended to use file extension ".EPR" for the file generated by BeamConstruct. Next it is also recommended to always let BeamConstruct write to microSD card directly because sometimes more than only one file is created. Direct write operation to BeamConstruct ensures all files are available on microSD and no data can be forgotten to be copied.

The second variant allows to download the stand-alone data to the controller while it is connected and running. Precondition for sending data to a running controller are:

- no mark operation is in progress (controller is idle)
- no stand-alone project is loaded (please refer to description of ioselect-mode and stand-alone control commands below).
- a valid name is given in style 0:/filename.epr

This operation creates the .EPR-file and all additional files on microSD card of the running controller automatically.

PLEASE NOTE: such an .EPR-stand-alone file can NOT be converted back to vector data that could be edited in BeamConstruct! Creating these files is a one-way-conversion of your projects. Thus it is recommended to save these projects twice – once as normal .BEAMP-File which can be loaded and modified later and once as .EPR-file which has to be used on SD-card. This also means such .EPR-files are protected so that it is possible to give away own designs to some end-users which shall not be able to modify them.

E1803D controller supports all static data in stand-alone mode (like all kinds of static geometries, output signals, waiting for input commands, waiting for trigger, all laser- and scanner parameters as well as elements which set outputs directly). But it does not store the vector data using a possibly configured correction table! To get a valid correction for stand-alone operations, the related correction file has to be saved on microSD card and needs to be activated using parameter "corrtable0" in e1803.cfg configuration file (please refer to description above).

Next E1803D scanner card supports dynamic content when following conditions are met:

- a text element uses one of the laser vector font families "Roman", "Script" or "Times" and it makes use of an input element
- a text element makes use of a TrueType font and it makes use of an input element; here any available TrueType font can be used and several hatch-patterns can be applied but some limitations apply (only left to right orientation, no scaling/rotation/slant/mirroring is applied to the font and only the characters ' ', !, ", #, $, %, &, \, (,), *, +, ,, -, ., /, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, :, ;, <, =, >, ?, @, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, [, \,], ^, _, `, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, {, |, } and ~ can be used, firmware version 3 or newer is required, BeamConstruct version 4.8 or newer is required)
- a barcode element uses type "DataMatrix" or “QR” with option "Merge cells" disabled and it makes use of an input element
- any kind of hatch and combined hatches can be applied to a barcode element, not to a text element
- marking output is neither mirrored (along x- or y-axis) nor flipped

When these conditions are met, a text or barcode can be modified during stand-alone operation either via stand-alone control commands as described below or via a Serial Number input element that is applied to it in

BeamConstruct. Here all serial number, time, date and formatting functions of this input element are supported. To get a valid time in stand-alone mode, it needs to be set after boot-up via stand-alone control commands (as described below) or a SNTP time server has to be configured to retrieve current time from an external source (please refer to description e1803.cfg parameters above).

7.2 Stand-Alone Configuration Parameters

Within e1803.cfg configuration file of E1803D scanner controller one of the following stand-alone operation modes can be selected via the configuration parameter "`standalone`":

`standalone=off`

Stand-alone mode is fully disabled, the card acts as normal slave of a host-PC and all .epr-files on the SD-card are ignored. Digital outputs are not toggled since no stand-alone operational states have to be signalised here (please refer next section).

`standalone=dotmark`

This is a special stand-alone marking mode where E1803D controller card acts as dot matrix marker. For details please refer section “8 Matrix Laser Dot Marking Mode“ below.

`standalone=auto`

Stand-alone mode is enabled, a file specified by and additional parameter "autofile" is loaded and prepared for marking. Marking of this file is started only when an external trigger signal is detected. The file itself has to be specified via additional configuration parameter that gives the filename of the stand-alone file to be loaded. As an example a parameter: "`autofile=0:/myfile.epr`" would try to load the file "myfile.epr" from SD-card and prepare it for marking. In this mode the digital outputs are toggled as described in next section.

`standalone=loop`

This is the same like mode "auto" described above but using "loop" E1803D controller does NOT wait for an external trigger signal! So when no trigger points are set in stand-alone datafile itself, in this mode marking would be done in an infinite loop, repeating the given "`autofile`" again and again.

`standalone=haltedloop`

This is the same like mode "loop" described above, but marking does not start immediately. By default the controller is in state “halt” until the ExtStart input is set to HIGH level. Marking continues only as long this input stays at HIGH. When it goes back to LOW, marking is continued until the laser is turned off the next time and it is halted again. Next time ExtStart goes to HIGH, marking continues at the position where it was halted before. In this mode the timeout-parameter “`haltedlooptimeout`” can be used.

`standalone=iohaltedloop`

This mode is a combination out of "haltedloop" described above and “ioselect” described below (please refer there for usage details). In this mode a project can be selected via digital inputs but it is started immediately and marked in an endless loop as long as ExtStart input is HIGH (so the level at ExtStart is checked, not the rising edge of an applied signal). When a different project is selected by applying a different input pattern at DIn digital inputs, the current project is cancelled and the new one is started in a loop again.
This mode requires firmware version 2 or newer.
In this mode the timeout-parameter “`haltedlooptimeout`” can be used.

`standalone=ioselect`

This mode makes use of the digital interface (please refer above). Here it is possible to select one of 256 stand-alone marking jobs via the digital inputs. The number that results out of the input pattern of the Digi I/O input lines specifies the filename of the marking job that has to be loaded from SD card:

Selected input(s)	Stand-alone file loaded from SD-card
All inputs set to LOW (not recommended to be used)	0.epr
DIn0 set to HIGH	1.epr
DIn1 set to HIGH	2.epr
DIn0 and DIn1 set to HIGH	3.epr
DIn2 set to HIGH	4.epr
DIn0 and DIn2 set to HIGH	5.epr
DIn1 and DIn2 set to HIGH	6.epr
DIn0, DIn1 and DIn2 set to HIGH	7.epr
DIn3 set to HIGH	8.epr
...	...
DIn4 set to HIGH	16.epr
...	...
DIn5 set to HIGH	32.epr
...	...
DIn6 set to HIGH	64.epr
...	...
DIn7 set to HIGH	128.epr
...	...
All inputs set to high	255.epr

PLEASE NOTE: 0.epr (no inputs set to HIGH) can be used but it is not recommended to do that. This value should be reserved for "no job active" to set the card into an inactive mode also in stand-alone operational mode. This may be necessary e.g. when new project data are downloaded to the controller without removing the SD-card.

Marking of a IO-selected job is started by external trigger signal (ExtStart input). When the input pattern at DIn0..DIn7 changes during marking, the currently running operation is continued and the other stand-alone job is loaded after marking operation has finished. In this mode the digital outputs are toggled as described in next section.

In stand-alone mode “ioselect” .EPR-files are loaded from microSD card as soon as a new input pattern is detected at digital inputs. Depending on the size of the .EPR file and the speed of the microSD card, this may take a time that is too long for high-speed applications. Thus it is possible to operate such projects from controller's RAM completely:

- in e1803.cfg the numbers of the files to be loaded have to be specified with parameter “iobuff”, it can be used up to 20 times and expects the number of the file (so a line “iobuff=3” would be responsible for pre-loading file “0:/3.epr”). File “0.epr” can not be preloaded by this command.
- serial interface/Telnet command “crlbf” can be used to reload such a predefined file during operation, e.g. in case it has been changed from outside (for details please refer to section “12.2 Stand-Alone Control Commands“ below)

Now these files are loaded into RAM and switching from one to an other is done much faster since toggling between them is done controller-internal and no more disk-operations are necessary for that.

PLEASE NOTE: when too much too large .EPR files are selected for preloading, this may exceed the available memory on card. This is signalled by the Error LED turned on and an appropriate message is stored in log buffer.

7.3 Stand-Alone Control

The current stand-alone operational state is signalised via digital outputs:

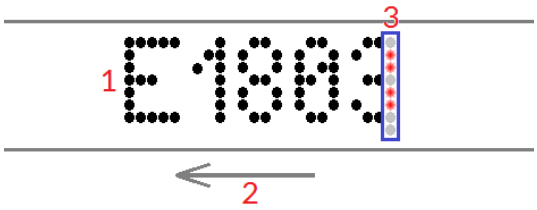
DOut0 – ready for marking – this output goes to HIGH as soon as a stand-alone job could be found on disk, was loaded successfully and is ready for marking. So external start signal should not be given until this output is HIGH. When a new stand-alone file is selected (e.g. via digital inputs in "`ioselect`"-mode) this output goes to LOW. It is switched back to HIGH only when the new file could be loaded successfully too.

DOut1 – marking active – as long as this output is HIGH, a marking operation is in progress. When a different stand-alone file is selected (e.g. via digital inputs) as long as this output is HIGH, marking is continued and the new file is NOT loaded. Once the current marking operation is completed, the output goes to LOW. After that the board continues with current marking data (when no new ones have been selected), or it tries to load new ones (when a new file was selected).
Please note: in firmware versions prior to v3 in stand-alone modes "`haltedloop`" and "`iohaltedloop`" this signal is not used since the user had full control over the marking process via the ExtStart input. So in case of these modes the logic is: when both, DOut0 and ExtStart are at HIGH, the controller is marking. Starting with firmware version 3 the DOut1 signal is used in same way as for all other stand-alone modes.

8 Matrix Laser Dot Marking Mode

Please note all functions described in this chapter are deprecated. They are available only for firmware version 5 of older. Before using these functions in new products, please contact HALaser Systems!

The E1803D controller card can be operated in a special mode where it does not make use of XY2-100 interface but acts as dot marker for matrix marking applications. Here a matrix laser, a dot peen or an other matrix printing device can be used. The dots are controlled via digital outputs turning the related dot on and off:



The controller works in stand-alone mode with no laser marking software connected to it. The objects to be marked have to be moved (2, from right to left in image above) and the movement information is feed into the controllers encoder inputs A1 and B1 of digital interface connector (for description please refer to section 6.8.1). Depending on the current position, the related dot-outputs are turned on and off marking one column on every distance step (3 symbolises the dot marking device in image above, red dots are lasers which are firing for the current column). This way the motion of the working piece forms up to two separate lines of text to be marked (1 in image).

8.1 Dot Mode Configuration Parameters

Within e1803.cfg configuration file of E1803D controller card the dot marking mode has to be configured by using different parameters:

Parameter	Description	Example
standalone	When set to "`dotmark`" this parameter enables the dot matrix marking mode. This is mandatory to use the following parameters, elsewhere they do not have any effect on the operation of the card.	`standalone=dotmark` Enables the dot matrix marker mode of E1803D
dotfont0	Specifies which font has to be used. The font size specifies how much of the max. 13 dots are used. All the available fonts are stored on microSD-card in sub-folder "fonts". Own fonts can be added to this, please contact HALaser Systems for details. This parameter is mandatory, for a single line of text it specifies the font for this line, when two lines of text have to be marked, it specifies the font for the upper line.	`dotfont0=0:/fonts/mono7.dfn` load the 7 pixels high, mono-spaced font "mono7.dfn" out of the "fonts"-folder of microSD-card
dotfont1	Specifies which font has to be used for the (optional) second line. This parameter is optional and has to be set only when two lines of text have to be marked below of each other. It specifies the font for the lower line. When the second line is used, parameter "dotfont1y" has to be set too.	`dotfont1=0:/fonts/straig6.dfn` load the 6 pixels high, mono-spaced font "mono6.dfn" out of the "fonts"-folder of microSD-card
dotfont1y	When two lines of text are marked this parameter specifies the y-offset of the second line (in unit dots). Using it vertical the position of the lower text line has to be set.	`dotfont1y=7` sets an Y-position of 7, means the upper border of the second (optional) text line will start at Dot7 output
dotdist	This parameter specifies the horizontal distance between two columns of dots. The value given here is in unit "increments" of connected encoder. When the value is positive, the encoder has to be	`dotdist=7500` starts a new column of dots whenever 7500 encoder increments have elapsed

	connected in a way where it counts to positive direction. In this case the text is marked from left to right, motion has to be done from right to left. When the value is negative, the encoder has to be connected in a way where it counts to negative direction. In this case the text is marked in reverse direction from right to left (mirrored in horizontal direction) and the motion of the working pieces need to have a suitable movement direction in order to have the expected results and no mirrored texts. The encoder counting direction has to fit to the configuration. When it works in wrong direction, only the first column of a text is marked, then nothing happens for a very long time. In this case it is necessary to exchange the two encoder lines connected to A1 and B1 in order to correct the counting direction.	
dottime	Specifies how long the dots have to be turned on at maximum. The value has to be given in unit nanoseconds and has a resolution of 500 nsec. When no dot-time is specified, turned on dots stay on until the motion has made enough progress to switch on the next row of dots. So in this case when the motion stops, the already turned on dots stay on endless.	`dottime=20000` turn off the dots latest after 20 usec

8.2 Dot Mode Hardware Interface

In dot matrix marking mode the laser connector is used to control the single dots. As soon as stand-alone mode “dotmark” is enabled, pinout of this connector is different:

Upper Row Of Pins	Signal	Voltage	Remarks	Lower Row Of Pins	Signal	Voltage	Remarks
1	Dot0	CMOS, 0/5V, max 8 mA		**2**	GND	GND	
3	Dot1	CMOS, 0/5V, max 8 mA		4			
5	Dot2	CMOS, 0/5V, max 8 mA		**6**	5V	5V	
7	Dot3	CMOS, 0/5V, max 8 mA		**8**	Dot10	CMOS, 0/5V, max 8 mA	
9	Dot4	CMOS, 0/5V, max 8 mA		10			
11	Dot5	CMOS, 0/5V, max 8 mA		**12**			Do not connect!
13	Dot6	CMOS, 0/5V, max 8 mA		**14**			Do not connect!
15	Dot7	CMOS, 0/5V, max 8 mA		**16**	ExtStart	CMOS, 0/5V	Input control signal
17	Dot8	CMOS, 0/5V, max 8 mA		**18**	5V	5V	
19	Dot9	CMOS, 0/5V, max 14 mA		**20**			Connected to pin 21
21			Connected to pin 20	**22**	Dot11	CMOS, 0/5V, max 14 mA	
23	GND	GND		**24**	ExtStop	CMOS, 0/5V	Input control signal
25	5V	5V		**26**	Dot12	CMOS, 0/5V, max 14 mA	

Dot0 is always the uppermost dot. Usage of all other dots depends on the height of the used font. So when a font with a size of 8 is chosen, the dots Dot0..Dot7 are used.
The dot-outputs are switched to HIGH signal whenever a dot has to be marked. This behaviour can be changed by setting the "`tune`"-parameter to value 8 (for details please refer section 6.5 above), then they are inverted and are set to HIGH when they are off.
The ExtStart input is used to start marking of one line of predefined text.
Dots can be toggled with a maximum frequency of 2 MHz.

8.3 Dot Mode Control

The dot matrix marking mode is a stand-alone operation mode which can be controlled from outside easily. After proper configuration of the related parameters in e1803.cfg, data to be marked can be sent to the card via commands "`cdtl0`" and optional "`cdtl1`" (for a more detailed description of this command interface please refer to section 12.2 below). This command can be used to enqueue several texts in advance. On every rising edge on ExtStart input output of the next text in queue is started according to the current encoder position.

During operation the current dot mode stand-alone operational state is signalised via digital outputs:

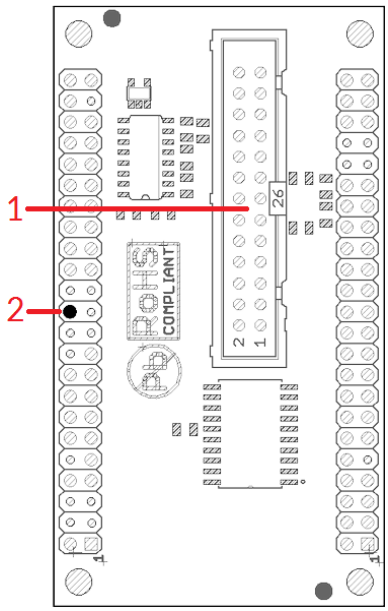
DOut0 – ready for marking – this output goes to HIGH as soon as some text was received which can be marked in dot mode. So external start signal to should not be given until this output is HIGH. When no more data are available to be marked or when marking is currently in progress, this output goes to LOW.
When dot mark mode is configured to use one line of text only (parameter `dotfont1` not set), the controller becomes ready for marking as soon as at least one text is available (to be set with command "`cdtl0`").
When dot mark mode works using two lines of text (parameter `dotfont1` set), the controller always requires pairs of data, means it becomes ready for marking only when text for two lines is available (to be set with both commands "`cdtl0`" and "`cdtl1`").

DOut1 – marking active – as long as this output is HIGH, a marking operation is in progress. During this time it is still possible to feed new marking data via commands "`cdtl0`" and "`cdtl1`", this does not influence the current operation.

9 Multi-IO Extension Board

The controller card can be extended by the Multi-IO board which utilitises the extension connector as described in section “6 Board And Connectors”.
To operate the controller with the Multi-IO Extension Board, firmware version 4 or newer is needed.

9.1 Board Connectors



The E1803D Multi-IO Extension Board provides following connectors and interfaces:

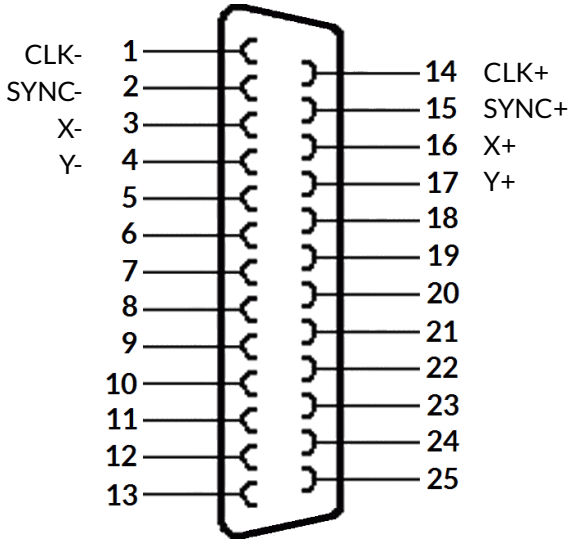
1. Multi-IO-connector offering different signals and interfaces including a secondary XY2-100 interface, analogue inputsm RS232/Rs485 serial ouput
2. Code pin for proper placement of extension board on E1803D controller card

9.1.1 Multi-IO Interface

The 26 pin connector provides signals and inputs to be used for different purposes:

Upper Row Of Pins	Signal	Voltage	Remarks	Lower Row Of Pins	Signal	Voltage	Remarks
1	CLK-		Secondary head, XY2-100-compatible output signals	**2**	CLK+		Secondary head, XY2-100-compatible output signals
3	SYNC-			**4**	SYNC+		
5	X-			**6**	X+		
7	Y-			**8**	Y+		
9			Do not connect!	10			Do not connect!
11	GND_{AIn}	GND	12 bit analogue inputs	**12**	AIn0	0..5V	12 bit analogue inputs
13	AIn1	0..5V		**14**	AIn2	0..5V	
15			Do not connect!	16			Do not connect!
17			Do not connect!	18			Do not connect!
19	RX1	12V	UART1 RS232	**20**	TX1	12V	UART1 RS232
21	GND			**22**	GND		
23	RX1+	5V	UART1 RS485	**24**	RX1-	5V	UART1 RS485
25	TX1+	5V		**26**	TX1-	5V	

The first 8 pins of the 26 pin connector provide signals to be used to control up to two galvos of a scanhead. These signals are fully parallel to the ones from XY2-100 interface of E1803D main board and can be used for secondary head applications where two scanheads work in parallel. With a flat wire belt that makes use of these first 8 pins only, a direct connection with a standard D-SUB25 connector can be made:



The pins 11 to 14 offer three analogue measurement inputs AIn0 .. AIn2 which can be operated in respect to GND_{AIn} and work with a voltage range of 0..5V. This range is converted to a 12 bit digital value in range 1..4095 and can be read out of a connected application or can be used to perform different automated tasks.

PLEASE NOTE: under no circumstances apply a voltage higher than 5V to any of these inputs, this may damage the complete scanner controller card irreversibly!

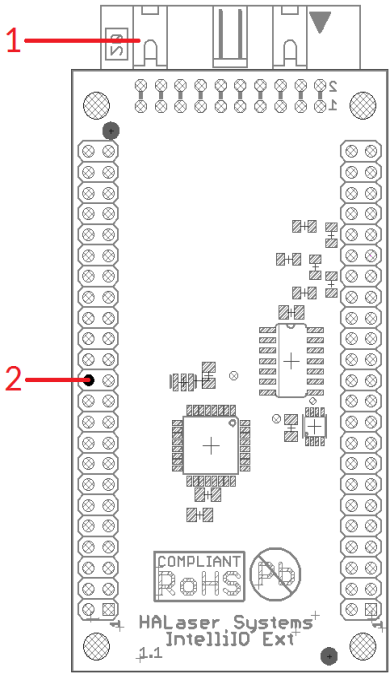
The last 8 pins provide access to UART1 RS232 and RS485 connection lines.

PLEASE NOTE: both, the RS232 and the RS485 interface are connected to the same serial interface internally! This means although there are two interface types available, only one logical serial line exists! Connecting two signal lines to RS232 and RS485 at the same time may damage the complete scanner controller card irreversibly!

10 Intelli-IO Extension Board

The controller card can be extended by the Intelli-IO Extension board which utilises the extension connector as described in section “6 Board And Connectors”.
To operate the controller with the Intelli-IO Extension Board, firmware version 5 or newer is needed.
Comparing to the Multi-IO Extension Board this one offers different types of IO and also provides the possibility for software customisation according to customers special needs. Due to flexibility it can be operated in different modes and therefore used for different purposes.

10.1 Board Connectors



The E1803D Intelli-IO Extension Board provides following connectors and interfaces:

1. IO-connector offering different signals and interfaces as described below including digital inputs, digital outputs and analogue inputs
2. Code pin for proper placement of extension board on E1803D controller card

10.2 Intelli-IO Interface in IO mode

When operated in IO-mode, the 20 pin connector provides following signals and inputs:

Upper Row Of Pins	Signal	Voltage	Remarks	Lower Row Of Pins	Signal	Voltage	Remarks
1	DOut0	5V	Digital outputs, second 8 bit port	**2**	DOut1	5V	Digital outputs, second 8 bit port
3	DOut2	5V		**4**	DOut3	5V	
5	DOut4	5V		**6**	DOut5	5V	
7	DOut6	5V		**8**	DOut7	5V	
9	DIn0	5V	Digital inputs, second 8 bit port (6 bits used)	**10**	DIn1	5V	Digital inputs, second 8 bit port (6 bits used)
11	DIn2	5V		**12**	DIn3	5V	
13	DIn4	5V		**14**	DIn5	5V	
15	GND_{AIn}	GND	12 bit analogue inputs	**16**	AIn0	0..5V	12 bit analogue inputs
17	AIn1	0..5V		**18**	AIn2	0..5V	
19	5V	5V		**20**	GND	GND	

The pins 1 to 8 provide 8 general purpose digital outputs. Maximum current to be pulled out of each output is 20 mA. It is recommended to supply power to E1803D controller via 3pin screw connector but not via USB when this extension board is used.
The pins 9 to 14 are digital general purpose input pins.

The pins 15 to 18 offer three analogue measurement inputs AIn0 .. AIn2 which can be operated in respect to GND_{AIn} and work with a voltage range of 0..5V. This range is converted to a 12 bit digital value in range 1..4095 and can be read out of a connected application or can be used to perform different automated tasks.
PLEASE NOTE: under no circumstances apply a voltage higher than 5V to any of these inputs, this may damage the complete scanner controller card irreversibly!

This extension board makes use of an own MCU (**Mic**rocontroller **U**nit) which handles all digital IOs and also has access to analogue inputs AIn0 and AIn1. This MCU can work fully parallel to the main scanner controller card and therefore is suitable for special, customised control applications. To get more information about such applications and to discuss your requirements, please contact HALaser Systems.

10.3 Intelli-IO Interface in motion mode

The hardware described within this section is designed to control motors. Motions caused by these motors may effect a person's health or may otherwise cause damage. Prior to installation and operation compliance with all relevant safety regulations including additional hardware-controlled safety measures has to be secured. The client shall solely be responsible to strictly comply with all applicable and relevant safety regulations regarding installation and operation of the system at any time.

When operated in motion-mode, the extension can be used to drive up to four stepper motor axes via step/direction signals. Here the 20 pin connector provides following signals and inputs:

Upper Row Of Pins	Signal	Voltage	Remarks	Lower Row Of Pins	Signal	Voltage	Remarks
1	Step0	5V	Stepper pulse output signals	**2**	Dir0	5V	Stepper motor direction output signals
3	Step1	5V		**4**	Dir1	5V	
5	Step2	5V		**6**	Dir2	5V	
7	Step3	5V		**8**	Dir3	5V	
9	Ref0	5V	Reference inputs	**10**	Ref1	5V	Reference inputs
11	Ref2	5V		**12**	Ref3	5V	
13	Do not connect!			14	Do not connect!		
15	GND_{AIn}	GND	12 bit analogue inputs	**16**	AIn0	0..5V	12 bit analogue inputs
17	AIn1	0..5V		**18**	AIn2	0..5V	
19	5V	5V		**20**	GND	GND	

The pins 1 to 8 provide the stepper motor control signals for axes 0..3 (step/direction signals to be used with a separate, external power driver). Maximum current to be pulled out of each output is 20 mA. It is recommended to supply power to E1803D controller via 3pin screw connector but not via USB when the Intelli-IO extension board is used.
The pins 9 to 12 are input pins for axes 0..3 to be used with the reference/homing position.

The pins 15 to 18 offer three analogue measurement inputs AIn0 .. AIn2 which can be operated in respect to GND_{AIn} and work with a voltage range of 0..5V. This range is converted to a 12 bit digital value in range 1..4095 and can be read out of a connected application or can be used to perform different automated tasks.
PLEASE NOTE: under no circumstances apply a voltage higher than 5V to any of these inputs, this may damage the complete scanner controller card irreversibly!

11 Quick Start into E1803D

Following a few steps are described that give users the possibility to quick start into usage of E1803D scanner controller. It makes use of BeamConstruct and USB connection. For this quick start manual it is assumed correct wiring of the controller is already done according to the description above. For more detailed information about BeamConstruct usage please also refer to quick start manual from https://openapc.com/download/manual_quickstart.pdf and to full user manual which is available at https://openapc.com/download/manual.pdf.

To start with E1803D controller:

1. **SECURITY CHECK:** The following steps describe how to set up E1803D scanner controller card and how to control laser equipment with them. Thus all laser safety rules and regulations need to be respected, all required technical security mechanisms need to be available and active prior to starting with it.
2. Install latest software version from https://openapc.com/download.php – for Windows this package contains all required drivers, for Linux no separate drivers are needed.
3. Connect E1803D controller via USB and apply +-12V .. +-24V power (depending on scanhead, as described above).
4. Now the Alive/Error-LED should light up and then start blinking after some time. When this does not happen, please turn power off, check if the microSD-card is placed correctly and then try again.
5. Evaluate the serial interface the controller is connected with – for Windows the Device Manager (can be found in Control Panel) will list a new COM-port (e.g. “COM3”); for Linux type “dmesg” in console to find out to which interface it was connected with (typically “/dev/ttyACM0”).
6. Start BeamConstruct laser marking software.
7. Go to menu “Project” → “Project Settings...”, then tab-pane “Scanner”.
8. Now you can select “E1803D” as scanner controller card. If is is not preselected, please choose the appropriate scanner controller in the related combo box.
9. Press the “Configure”-button to get into the settings dialogue for E1803D plug-in.
10. Enter the serial interface name in field “IP/Interface” (e.g. “COM3” or “/dev/ttyACM0”).
11. Leave everything with “OK”.
12. Draw some geometries as described in “BeamConstruct Quick Start Manual”.
13. **SECURITY CHECK:** Next the scanner controller card will be accessed for the first time. That means it is opened and initialised and all connected equipment may start working now. Thus it is very important to ensure all security regulations are met and nobody can be injured and no damage can be caused also in case laser output or other motion starts spontaneously and unexpectedly!
14. Press “F2” or go to menu “Process” → “Mark” to open the mark dialogue.
15. Start marking by pressing the yellow laser-button!

12 Command Interface

When E1803D scanner card is connected via USB and the USB-connection is NOT used for transmitting marking information, it can be used to send control commands to the card. Some of them are independent of the current operating mode and some of them can be used only in case the controller is operating in stand-alone mode.
Alternatively control commands can also be sent via Telnet using Ethernet connection. Here a Telnet-client has to connect to port 23 using the IP of the scanner controller. This Telnet client should work in passive mode.
So when E1803D scanner card is connected this way via Ethernet and the Ethernet-connection is NOT used for transmitting marking information, it can be used to send control commands to the card. Some of them are independent of the current operating mode and some of them can be used only in case the controller is operating in stand-alone mode.
Such a control command always consists of ASCII-text. An appropriate client has to connect to the serial port (COMx for Windows and /dev/ttyACMx for Linux where “x” is a number identifying the specific serial interface or TCP/IP port 23). As soon as the connection is established, commands can be sent to the card. All commands come with following structure:

```
cxxxx [parameter(s)]
```

The commands always start with character “c”. Next four characters identify the command itself. Depending on the command one or more optional or mandatory parameters may follow. The command always returns with an "OK" or with an error.

12.1 General Commands

The following commands can be used in all scenarios, they do not depend on a specific operation mode of the card. Nevertheless it is recommended to not to send a command during card is marking to not to influence marking operation.

`cvers`

"**vers**ion" – return version information of controller card. This command returns a version string specifying version of hard- and software.

`cecho <0/1>`

"**echo**" – when typing commands in a serial console communicating with the controller, all the typed characters are echoed, means they are sent back to the host so that a user can see what is typed. This is an unwanted behaviour when some kind of control software communicates with this interface. Using this command the serial echo mode can be turned off (parameter 0, only return values are sent back) or on (parameter 1, all data are sent back). When called with no parameters, the current echo mode value is returned.
Example: `cecho 0` – turn off echo mode

`cginp`

"**g**et **inp**ut" – get the current state of the digital inputs. The input state is returned as a decimal number representing the bitpattern at the inputs. So when e.g. a value "15" is returned, this means the lower four inputs of the digital interface are set to HIGH while the upper ones are at LOW level

`csout <value>`

"set output" – set the state of the digital outputs. The output to be set is specified as a decimal number representing the bitpattern. When no parameter is given, the behaviour is undefined.
Example: `csout 128` - set DOut7 at the digital interface to HIGH while all others stay at LOW

`cglog`

"**g**et **log**line" – returns a single logging line. This command has to be called repeatedly until an error is returned to get logging information from the controller. On each call of this function one logging line is

returned. When "cglog" isn't used for a longer time it may be possible the internal log-buffer has overrun. In this case "cglog" will not return all log information.

cgbsr

"**g**et **b**oard **s**er**i**al number" – returns the serial number of the card. This number is a unique, internal value that is used e.g. to identify a controller on host PC when more than one scanner card is used.

cjsor <factor>

"**j**ump **s**peed **o**verw**r**ite" - this command modifies the actual jump speed by using the given factor (in unit 1/10000). All operations make use of the changed jump speed until a factor of 10000 is set or until the controller is restarted. This is true for both, stand-alone applications where an .EPR-file is loaded from microSD-card and for host-controlled marking operations (via libE1803D or BeamConstruct).

cmsor <factor>

"**m**ark **s**peed **o**verw**r**ite" - this command modifies the actual mark speed by using the given factor (in unit 1/10000). All operations make use of the changed jump speed until a factor of 10000 is set or until the controller is restarted. This is true for both, stand-alone applications where an .EPR-file is loaded from microSD-card and for host-controlled marking operations (via libE1803D or BeamConstruct).

cpwor <factor>

"**pow**er **o**verw**r**ite" - this command modifies the actual power by using the given factor (in unit 1/10000). All operations make use of the changed jump speed until a factor of 10000 is set or until the controller is restarted. This is true for both, stand-alone applications where an .EPR-file is loaded from microSD-card and for host-controlled marking operations (via libE1803D or BeamConstruct).
This command influences following methods of setting laser power:

- pulse width, here user has to ensure the resulting pulse width is smaller than the period of the related frequency, elsewhere the output will be a continuous signal
- LP8 laser port
- AOut0 and AOut1 analogue outputs

cgana

get **ana**logue input values – this command returns the values which have been read at analogue inputs AIn0..AIn2 recently. Reading of the analogue inputs is done cyclically and automatically with a low frequency. This command does not read the analogue values but returns the values which have been read at last cycle. So when this command is repeated too fast, it may return the same values.
This command returns the current values of all inputs AIn0, AIn1 and AIn2 all together.
To use this command, firmware version 4 and the Multi-IO Extension Board are needed.

12.2 Stand-Alone Control Commands

Following commands are useful in case scanner controller is operating in stand-alone mode where marking data are loaded from microSD-card using special EPR-fileformat.

cstop

"**stop**" – stop marking as fast as possible. A running marking operation is stopped and LaserGate is turned off.

chalt <0/1>

"**halt**" – halts or continues the processing and output of marking data. When given parameter is equal to 1, marking is stopped next time the laser is off but no vector data are flushed. On continue (parameter equal 0) controller continues processing at the point where halt occurred. When marking is stopped with cstop the halt-condition is cleared too, means on next transmission of new marking data they are processed without the need to explicitly continue operation.

`cstrt`

"**st**a**rt**" – start marking operation. This command can be called only when no marking operation is running and when a valid project (.epr) file was loaded. In this case the currently loaded project is marked once.

`ctrig`

"**trig**ger" – send an external trigger signal by software. When scanner card is in state "marking" but waiting for an external trigger, this command releases this trigger. So behaviour is the same like a rising edge on the ExtStart input of the controller card.

`cstat`

"**stat**e" – return the current state of the card. This command returns one of the following texts identifying the operational state:

- `marking` – card is processing some marking data currently, means either actively outputting them or waiting for an external trigger to start marking
- `stand-alone` – controller is in stand-alone mode
- `idle` - card is waiting and not marking
- `waiting` - a project file was loaded, is ready for execution and waits for a trigger signal (either via ExtStart input or via command "`ctrig`")

`cscnc`

"**s**et **CNC** data" – switch to a mode where G-Code process data are accepted via Telnet/serial interface. When this command is set, the G-Code mode stays active until a command "M2" (end of G-Code program) is detected. Only with this command the controller returns to normal operation mode and again accepts native "c"- and "d"-commands.
Using of command "cscnc" requires stand-alone mode "auto" in order to store the received G-Code data in memory for further processing and to control execution of the G-Code data. After the G-Code data have been transferred and transmission has been ended with "M2", marking of these data can be started by applying an external trigger at ExtStart or by sending a trigger-event via command (e.g. "`ctrig`").
To successfully send G-Code data, some preconditions have to be met. For data transmission via Telnet:

- a G-Code line is limited to 255 characters maximum and always has to end with carriage return and/or line feed
- when transferring more than one line at the same time, the maximum packet size is 1460 bytes, at the end of such a packed a G-Code line has to end too and it is recommended to flush the full output buffer in order to invoke a data transmission over TCP/IP; this procedure is recommended in order to have an as fast as possible data transfer independent from the implementation a TCP/IP stack really uses

For data transmission via USB serial interface:

- a G-Code line is limited to 255 characters maximum and always has to end with carriage return and/or line feed

The structure and supported G-Code commands are described in section " 13 Supported CNC G-Code Commands" below.
This function requires firmware version 3 or newer.

`cgtin`

"**g**et **t**rigger **in**puts" – get the state of the external input signals. This command is not related to digital inputs of digital interface but provides information regarding signal state of external start and stop. It returns a value that specifies which of these input signals are currently HIGH:
0 – ExtStart and ExtStop are both LOW
2 – ExtStart is HIGH
4 – ExtStop is HIGH
6 – ExtStart and ExtStop are both HIGH

`cscor <idx>`

“**s**et **cor**rection” - specifies a new index for a previously loaded correction file (see description of configuration parameter `corrtable` in section “6.5 microSD-Card” above). The parameter `idx` can be a value in range 0..15 and needs to correspond to a previously loaded correction table. The newly set correction table applies to all vector data which are processed after this call. Thus it is recommended to use it only when marking operation was stopped – elsewhere it is not predictable how many vector data already have been pre-calculated with the previous correction table and starting with which vector data the new correction file is used.
When a `idx`-value is set which corresponds to no correction file data, no more correction is performed on vector data.
This command requires firmware version 6 or newer.

`cgcor`

“**g**et **cor**rection” - this command is the counterpart of `cscor` and displays the index number of the currently used correction file
This command requires firmware version 6 or newer.

`clepr <path>`

"**l**oad **epr**" – loads an EPR stand-alone file or CNC G-Code file from microSD card for outputting it on next marking operation. This command can be executed in stand-alone mode only and expects the path to the file to be loaded as parameter. Since this is the only parameter, no quotes are allowed for the pathname. The pathname itself has to be in format
`0:/filename.epr`
or
`0:/filename.cnc`
where `0:/` specifies the microSD-card, `.epr` is the standard extension of E1803D stand alone marking data files (this name is a short-cut for "**E1803D Pr**ocessing Data") and `.cnc` is the extension which has to be used when an ASCII-G-code file is provided. EPR-files can be created out of BeamConstruct, CNC-files are text files containing valid G-Code commands as described in section “13 Supported CNC G-Code Commands”
During loading the ready-for-marking output signal is turned off and it is turned on only in case the file could be loaded successful (please refer to related section above).
Examples: `clepr 0:/test.epr` – loads a stand-alone file "test.epr" from microSD card

`crlbf <number>`

“**rel**oad **buf**fered file” - this command can be used in stand-alone mode “ioselect” and for all files that are configured for preloading using configuration parameter “iobuff”. When called without any parameter it reloads all files that are configured for pre-buffering, when called with a number as parameter it reloads the file with the given number. This command can be useful in cases where a pre-buffered .EPR file was changed on disk (e.g. by downloading it via controller to microSD card) and has to be loaded into the RAM without rebooting the card.

`cgepr`

“**g**et **epr**” – returns the name of the currently loaded stand-alone file or an error “no file specified” when no file is loaded.

`cstxt <"elementname"> <"text">`

"**s**et **t**e**xt**" – set a new text value to an element in currently loaded project. The parameters for this command both have to be given with quotes ("). Setting a text is possible only for dynamic elements like DataMatrix or QR barcodes or texts. Here "elementname" is the name of the element that has to be modified (this is the same name like shown in Elementtree of BeamConstruct) and the new text to be set. The "text" itself can be a format string as used within BeamConstruct when a serial number input element is involved
Example: `cstxt "Barcode 1" "Hello :-)"` - sets a new text "Hello :-)" for the element with name "Barcode 1"

`cgtxt <"elementname">`

"**g**et **t**e**xt**" – gets the currently used text value of an element in loaded project. The parameter for this command has to be given with quotes ("). Getting a text is possible only for dynamic elements like DataMatrix or QR barcodes or texts.

Example: `cgtxt "Barcode 1"` – gets the text from the element with name "Barcode 1"

`csser <"elementname"> <cnt>`

"**s**et **ser**ial number" – sets a new serial count value to an element in currently loaded project. The element name for this command has to be given with quotes ("). Setting a new count is possible only for dynamic elements like DataMatrix or QR barcodes or texts that have a serial number input element assigned. Setting the value has to be handled with care, here every value can be specified independent if it fits to possibly exiting beat count values.
Example: `csser "Text 2" 42` – set a new serial number count value 42 for element with the name "Text 2"

`cgser <"elementname">`

"**g**et **ser**ial number" – gets the current serial count value from an element in loaded project. The element name for this command has to be given with quotes ("). Getting the count is possible only for dynamic elements like DataMatrix or QR barcodes or texts that have a serial number input element assigned.

`ciser <"elementname">`

"**i**ncrement **ser**ial number" – increments the current serial count value of an element according to its serial number parameters. The element name for this command has to be given with quotes ("). Incrementing the count is possible only for dynamic elements like DataMatrix or QR barcodes or texts that have a serial number input element assigned. This function is more secure than forced setting of a new count value with "csser" since it can't violate the counting rules.

`cdser <"elementname">`

"**d**ecrement **ser**ial number" – decrements the current serial count value of an element according to its serial number parameters. The element name for this command has to be given with quotes ("). Decrementing the count is possible only for dynamic elements like DataMatrix or QR barcodes or texts that have a serial number input element assigned. This function is more secure than forced setting of a new count value with "csser" since it can't violate the counting rules.

`crser <"elementname">`

"**r**eset **ser**ial number" – resets the current serial count value of an element to its start-value (according to its serial number parameters). The element name for this command has to be given with quotes ("). Resetting the count is possible only for dynamic elements like DataMatrix or QR barcodes or texts that have a serial number input element assigned. This function is more secure than forced setting of a value with "csser" since it can't violate the predefined serial number parameters and uses the correct reset value.

`cstim <seconds>`

"**s**et **tim**e" – this command sets the system time to the value specified with the parameter. Here the number of seconds have to be specified that have elapsed since 01.01.1970 at 00:00:00. After sending this command the controller card operates at the given time. The time value is lost after next power cycle and has to be set again.
Example: `cstim 1420113600` – set the internal time of E1803D controller to 01.01.2015 12:00:00, here 1420113600 represents the number of seconds that have been elapsed between 01.01.1970 00:00:00 and 01.01.2015 12:00:00

`crtim`

"**r**etrieve **tim**e" – this command schedules time retrieval from an SNTP server asynchronously. It always returns with "OK" since the command is scheduled for execution during next working cycles. To use this command, controller has to be configured with IP, netmask, gateway and SNTP server IP correctly and needs to

be able to access this SNTP server from its position in network. For details please refer to description of configuration parameters in section about microSD card above.

`cgtim`

"**g**et **tim**e" – returns the current time of the board in number of seconds that have elapsed since 01.01.1970 at 00:00:00. After powering up the board and before a valid time has been set, this value is undefined.

`cftim`

"**g**et **f**ormatted **tim**e" – returns the current time of the board as formatted string in style DD.MM.YYYY hh:mm:ss. After powering up the board and before a valid time has been set, this value is undefined.

`cstyr <year>`

"**s**et **t**ime **y**ea**r**" – sets the year of the current system time to the value given as parameter. This value has to be in range 1900..2038

`cstmo <month>`

"**s**et **t**ime **mo**nth" – sets the month of the current system time to the value given as parameter. This value has to be in range 1..12 according to the number of the month.

`cstdy <day>`

"**s**et **t**ime **d**a**y**" - sets the day of the current system time to the value given as parameter. This value has to be in range 1..28, 1..30 or 1..31 according to the length of the current month.

`csthr <hour>`

"**s**et **t**ime **h**our" – sets the hour of the current system time to the value given as parameter. This value has to be in range 0..23.

`cstmi <minute>`

"set time minute" – sets the minute of the current system time to the value given as parameter. This value has to be in range 0..59.

`cstsc <second>`

"**s**et **t**ime **se**cond" – sets the second of the current system time to the value given as parameter. This value has to be in range 0..59.

`cgsta`

"**g**et **s**erial s**ta**te" – this command applies only when working in stand-alone mode with dynamic serial number data that change on every mark operation. It returns information if the state of serial numbers has changed and is not yet saved (in this case "pending" is returned) or if they have been saved and therefore do not get lost when power is turned off now ("saved" is returned in this case).

`cssta`

"**s**ave **s**erial s**ta**te" - this command applies only when working in stand-alone mode with dynamic serial number data that change on every mark operation. When it is called, a command to save the current state of serial numbers is enqueued and will be processed as soon as controller is able to store these data. So when this command returns with "OK" that doesn't necessarily means the serial number states are saved now. The current save state still has to be checked by calling "cgsta" after "cssta" has been issued.

`cdtl0 <text>`

“**d**ot **t**ext **l**ine **0**” - this command is used in dot mode matrix marking mode only. It can be used to set a new line of text for the upper line to be marked in dot matrix mode. It can be called several times in advance to set some more texts. They are marked in the order they have been set on every marking cycle started by an ExtStart input signal. Stopping a marking operation by calling “cstop” or by applying a signal to ExtStop input deletes all previously set texts and empties the whole text buffer.

`cdtl1 <text>`

“**d**ot **t**ext **l**ine **1**” - this command is used in dot mode matrix marking mode only. It can be used to set a new line of text for the lower, optional line to be marked in dot matrix mode. It can be called several times in advance to set some more texts for the second line. When the controller card is configured to work with two lines of texts, it always expects pairs of texts to be set, means “cdtl0” and “cdtl1” needs to be used always together. They are marked in the order they have been set on every marking cycle started by an ExtStart input signal. Stopping a marking operation by calling “cstop” or by applying a signal to ExtStop input deletes all previously set texts and empties the whole text buffer.

`crrrr`

"**r**eboot" – perform a warm reboot of the hardware and restart the firmware. Reboot is done immediately, means this command does not return anything but connection to the board will be interrupted as soon as it has been sent.

12.3 Mark Control Commands

The following section describes commands that can be used to send marking data (including vector data and laser/scanner parameters) to the controller. These commands can be mixed with the commands described above but have a different structure:

- they always start with a character "d"
- the total length of one frame (means one command) is always 14 bytes
- they mustn't be terminated with CR/LF, the end of a frame is determined by its size of 14 bytes
- they contain binary, means not human-readable data and therefore can't be sent manually

PLEASE NOTE: when using Network/Telnet connection and when switching from a Mark Control Command ("d"-command) to a general command ("c"-command as described above) it is recommended to flush all output before sending a command of other type.

These commands always have the following structure:
`dCAAAABBBBEEEE`
`d` – marks starting point of a frame and identifies a mark control command with a fixed length of 14 bytes (including this character)
`C` – 8 bit value that specifies what command has to be executed
`AAAA` – 32 bit little-endian value, it's meaning and usage depends on "C"
`BBBB` – 32 bit little-endian value, it's meaning and usage depends on "C"
`EEEE` – 32 bit little-endian value, it's meaning and usage depends on "C"

It is recommended to collect commands before they are sent to the controller, especially in case Ethernet connection is used. In case of TCP/IP the used payload length of a TCP-frame is 1460 bytes which should be filled as much as possible in order to avoid additional data transfers. So when sending larger amounts of data to the controller, up to 104 command frames should be collected and then sent all together (104 * 14 = 1456 bytes which is close to 1460).

From time to time the controller sends back an answer to give back some state information. In case of Ethernet/ Telnet connection this answer is not sent periodically but as response to a complete block of data sent to the card. Since the size of such a block is not specified and depends on the underlying TCP/IP implementation (in case of Ethernet connections), no predictions can be made after what amount of data a response frame is sent. Thus it is recommended to try to receive such a response frame every time some data have been transmitted until at least one frame was received. When host software is idle, it can try to read response frames

permanently. To trigger transmission of a new response frame, "ping" control command `0x0A` can be used (for details please refer below).
In case of USB/serial connection this response is sent automatically after every 14 byte frame submitted, so it is necessary to always read them in order to avoid overrun of receive buffers.
Such a response frame gives back information about the current operational state of the card and comes in following structure:
`dRLLLLSSSSIIII`
`d` – marks starting point of a response frame with a fixed length of 14 bytes (including this character), this character can be used to re-synchronise
`R` – 8 bit value, currently always 0xFF; this value has to be checked for future compatibility, in case it is not 0xFF the frame has to be ignored!
`LLLL` – 32 bit little-endian value, here the amount of free command buffer space is returned; sending application has to ensure this buffer never overruns, so it is recommended to always leave a space of at least 200 commands (recommended: 1000), new commands should be sent only when there is more space than this left in this buffer
`SSSS` – 32 bit little-endian value, signalling operational state; this value can consist of following or-concatenated flags:

- `0x00000001` – card is currently marking
- `0x00000002` – the external start input is currently HIGH
- `0x00000004` – the external stop input is currently HIGH
- `0x00000008` – the external start input was set to HIGH after last response frame, this value is set only once for every rising edge on this input
- `0x00000010` – the external stop input was set to HIGH after last response frame, this value is set only once for every rising edge on this input
- `0x00000080` – the controller has received some data which may result in a marking operation; these data are currently processed but marking has not yet started
- `0x00000400` – card is active but currently waiting for an external trigger to continue operation

`IIII` - 32 bit little-endian value, lower 8 bit show the actual state of digital inputs

Currently following mark control commands (identified by the 8 bit hexadecimal value for position "`C`" in a frame) can be sent to the controller:

Jump to Position
Move to a given coordinate position using the current jump speed and with laser turned off
C = 0x00
AAAA = x-position to move to in range 0..67108863
BBBB = y-position to move to in range 0..67108863
EEEE = z-position to move to in range 0..67108863

Mark to Position
Move to a given coordinate position using the current mark speed and with laser turned on
C = 0x01
AAAA = x-position to move to in range 0..67108863
BBBB = y-position to move to in range 0..67108863
EEEE = z-position to move to in range 0..67108863

Start output
This command has to be called at the end of every marking sequence to ensure marking output really starts. This is important in case only a few vectors are sent to ensure marking is started but it is recommended to always use this command.
C = 0x02
AAAA = unused, set to 0
BBBB = unused, set to 0
EEEE = unused, set to 0

Wait for external trigger

Set a trigger point to current position of stream; emission of output data will stop until an external trigger signal is detected

C = 0x03
AAAA = unused, set to 0
BBBB = unused, set to 0
EEEE = unused, set to 0

Set speed values

Specify the speeds to be used during jump or mark movements (invoked by commands 0x00 and 0x01)

C = 0x04
AAAA = jumpspeed in unit bits per microsecond
BBBB = markspeed in unit bits per microsecond
EEEE = unused, set to 0

Set laser delays

Specify the delays to be used when laser is turned on or off

C = 0x05
AAAA = laser on delay in unit microseconds and in range -10000000..10000000
BBBB = laser off delay in unit microseconds and in range 0..10000000
EEEE = unused, set to 0

Set scanner delays

Specify the delays to be used before and after mark and within a polygon

C = 0x06
AAAA = jumpdelay in unit microseconds
BBBB = markdelay in unit microseconds
EEEE = in-polygondelay in unit microseconds

Stop marking

Tries to halt, continue or stop current output depending on the chosen option

C = 0x07
AAAA = stop option:

0 - tries to stop operation as fast as possible and rejects all data that still may be enqueued for execution

1 - marking is stopped next time the laser is off but no vector data are flushed, card is still active

2 - controller continues processing at the point where halt occurred (requires a previously called command 0x07 with stop option 1)

BBBB = unused, set to 0
EEEE = unused, set to 0

Set wobble parameters

Specify the wobble settings to be used for next marking operations

C = 0x08
AAAA = wobble amplitude in X-direction using unit bits and with maximum range of 0..10000000 bits
BBBB = wobble amplitude in Y-direction using unit bits and with maximum range of 0..10000000 bits
EEEE = wobble frequency in unit Hz*100 and in range 1..2500000

Set LP8 outputs

Set LP0..LP7 output pins on laser signal connector

C = 0x09
AAAA - bitpattern to be set on LP0..LP7 output pins, here only lower 8 bits are used.
BBBB = unused, set to 0
EEEE = unused, set to 0

Ping

This command can be used to let the controller send back a state-information. So it can be used to check if the card is still operating or not.

ATTENTION: this command should not be sent repeatedly and without any delay! This could cause E1803D scanner controller to stall because the massive data transfer has to be handled. So it is recommended to have a delay of at least 150 msec between every ping.

C = 0x0A
AAAA - unused, set to 0
BBBB = unused, set to 0
EEEE = unused, set to 0

Set digital outputs

Set DOut0..DOut7 output pins on digital interface connector

C = 0x0B
AAAA - bitpattern to be set on DOut0..DOut7 output pins, here only lower 8 bits are used.
BBBB = bitmask specifying which of the bits in AAAA have to be set or cleared, all these bits in AAAA are left unchanged, where the corresponding bit in BBBB is 0
EEEE = unused, set to 0

Set lasermode

Specify the laser mode the card has to operate with

C = 0x0C
AAAA - flags specifying the laser mode, here following values have to be or-concatenated to specify the behaviour of a laser:

- `0x40000000` - laser frequency on LaserA output is turned on immediately and together with laser gate signal, this flag can't be used together with `0x20000000`
- `0x20000000` - laser frequency on LaserA output is turned on after FPK time, this flag can't be used together with `0x40000000`
- `0x10000000` - laser supports FPK on LaserB output
- `0x08000000` - laser frequency has to be turned off and switched to standby-frequency
- `0x04000000` – a frequency can be emitted at LaserB permanently, the related frequency can be specified with command 0x15

Using these flags following laser types can be configured:

- CO_2-laser:
  `0x40000000 + 0x08000000`
- YAG-laser with FPK:
  `0x40000000 + 0x08000000 + 0x10000000` or
  `0x20000000 + 0x08000000 + 0x10000000`
- laser with continuously running frequency: `0x40000000`

BBBB = unused, set to 0
EEEE = unused, set to 0

Set marking on-the-fly parameters

Specify the parameters used for marking on-the-fly applications

C = 0x0D
AAAA = marking on-the-fly resolution in X-direction in unit bits per encoder increment
BBBB = marking on-the-fly resolution in Y-direction in unit bits per encoder increment
EEEE = unused, set to 0

Set laser frequency

Specify the frequency the laser has to be operated with during marks, usage of these parameters depends on the lasermode specified with command 0x0C

C = 0x0E

AAAA = frequency in unit Hz and in range 25..20000000 Hz
BBBB = pulse-width in unit microseconds and in range 1..65530 usec
EEEE = unused, set to 0

Set laser standby frequency

Specify the frequency the laser has to be operated with during jumps, usage of these parameters depends on the lasermode specified with command 0x0C
C = 0x0F
AAAA = frequency in unit Hz and in range 25..20000000 Hz
BBBB = pulse-width in unit microseconds and in range 1..65530 usec
EEEE = unused, set to 0

Set first pulse killer

Specify the pulse width of the FPK signal when laser is turned on, usage of these parameters depends on the lasermode specified with command 0x0C
C = 0x11
AAAA = FPK pulse width in unit microseconds*100
BBBB = the time the laser frequency has to be started after beginning of FPK using unit microseconds*2, this value is used only when lasermode flag 0x20000000 is set
EEEE = unused, set to 0

Switch MO-output

Turns the MO-output of laser interface connector on or off
C = 0x12
AAAA = turn MO output on when equal 1, turn it off when 0
BBBB = unused, set to 0
EEEE = unused, set to 0

Release external trigger

When card is waiting for an external trigger this command can be sent to release this external trigger by software and to continue execution without the need to receive a real external signal
C = 0x13
AAAA = unused, set to 0
BBBB = unused, set to 0
EEEE = unused, set to 0

Wait for external input signal

Stop execution until a defined input bitpattern is detected at configurable input pins DIn0..DIn7 of digital interface connector
C = 0x14
AAAA = a bitpattern specifying which signals LOW or HIGH have to be detected at digital input pins
BBBB = a bitpattern specifying which of the digital input pins have to be watched for a signal, these bits that are set to 0 are ignored while these bits, that are set to 1 have to get the state specified in previous parameter in order to let operation of card continue
EEEE = unused, set to 0

Set LaserB frequency

Specify the frequency LaserB output has to emit; this function can only be used when operating using a laser mode with flag `0x04000000` set (see command 0x0C above).
C = 0x15
AAAA = frequency in unit Hz and in range 25..20000000 Hz
BBBB = pulse-width in unit microseconds and in range 1..65530 usec
EEEE = unused, set to 0

Wait until on-the-fly-increments have been elapsed

This command adds some special kind of delay to the application. It can be used only when marking on-the-fly is enabled (by setting the on-the-fly factors), and halts laser marking not for a given time but for a given distance. Marking is continued only when the given number of increments has elapsed. When no or not enough increments are counted by the controller, operation only can be stopped.
This command requires firmware version 27 or newer.
C = 0x16
AAAA = positive or negative number of increments to wait for until operation continues; here it depends on the used counting direction of the encoder if the given distance-value has to be positive or negative, when sign of the number and counting direction of the encoder do not fit to each other, the controller will halt at this position for a very long time
BBBB = unused, set to 0
EEEE = unused, set to 0

Specify output for MIP-signal

Specify a single output pin of digital interface connector to be used for "Mark in progress"-signal, this output pin will be HIGH as long as a marking operation is in progress.
C = 0x2A
AAAA = the number (not a bitpattern!) of the digital output pin to be used for MIP-signal (in range 0..7)
BBBB = unused, set to 0
EEEE = unused, set to 0

Halt/continue current marking operation

Stops the current marking operation on very next appearance of a jump or continue a previously halted operation.
C = 0x2F
AAAA = 1 to halt marking and 0 to continue a halted operation
BBBB = unused, set to 0
EEEE = unused, set to 0

Specify output for WET-signal

Specify a single output pin of digital interface connector to be used for "Wait External Trigger"-signal, this output pin will be HIGH as controller is waiting for an external trigger.
C = 0x33
AAAA = the number (not a bitpattern!) of the digital output pin to be used for WET-signal (in range 0..7)
BBBB = unused, set to 0
EEEE = unused, set to 0

Set first row of 2x2 output matrix

Specify the elements m11 and m12 of a 2x2 output matrix which is applied to all coordinate values as soon as the second half is applied with command 0x41. This matrix can be used to scale, slant, rotate and mirror the input coordinates in respect to the output positions. For details please check out description of command 0x41 below
This command requires firmware version 3 or newer.
C=0x40
AAAA = the m11 part of the 2x2 matrix multiplied with 1000000
BBBB = the m12 part of the 2x2 matrix multiplied with 1000000
EEEE = unused, set to 0

Set second row of 2x2 output matrix

Specify the elements m21 and m22 of a 2x2 output matrix which is applied to all coordinate values together with the first row of matrix data which has to be set using command 0x40 in a preceding call. This matrix can be used to scale, slant, rotate and mirror the input coordinates in respect to the output positions. Assumed a matrix bases on a 4-elements array, it has following structure:

```
{m11, m12, m21, m22}
```

then these matrix values can be used and even combined with each other by multiplying them:

- rotation: `{cos(angle), -sin(angle), sin(angle), cos(angle)}`
- scaling: `{factorX, 0.0, 0.0, factorY}`
- slant X: `{1.0, 1.0/tan(angle), 0.0, 1.0}`
- slant Y: `{1.0, 0.0, 1.0/tan(angle), 1.0}`
- mirror X: `{-1.0, 0.0, 0.0, 1.0}`
- mirror Y: `{1.0, 0.0, 0.0, -1.0}`

This command requires firmware version 3 or newer.
C=0x41
AAAA = the m21 part of the 2x2 matrix multiplied with 1000000
BBBB = the m22 part of the 2x2 matrix multiplied with 1000000
EEEE = unused, set to 0

Download new firmware

Download a new firmware file to the controller and write it to the microSD-card so that it can be used after next reboot. The binary data of the new firmware have to be appended directly to this command. This command has to be used in a specific sequence in order to ensure the current firmware file is updated correctly:

- ensure the card is idle (state-flag `SSSS` is 0)
- send command 0x45 with length of firmware data and checksum
- send binary firmware data directly after this command
- wait until card state returns “active” (by repeatedly sending ping-commands), now in state flag `SSSS` bit 0x4000 (`E1803_CSTATE_WRITING_DATA`) is set
- wait until card state returns “idle” again (by repeatedly sending ping-commands), the flag 0x4000 no longer should be set in state flag `SSSS`
- check if an error occurred: when flags 0x8000 (`E1803_CSTATE_WRITING_DATA_ERROR`) is set in state flag `SSSS`, downloading or writing or checksum calculation failed and the original file was not replaced; for debugging in such a case the command “cglog” can be called repeatedly until the related error text was found; when this error flag is set it can be reset only by using command 0x45 again
- reboot the controller
- check if the version of the firmware has changed

This command requires firmware 4 or newer.
C = 0x45
AAAA = the length of the firmware file in bytes
BBBB = checksum for verification of the downloaded data, only when this checksum is correct, the old firmware file will be replaced; the checksum can be calculated using following function (C example code):

```
unsigned int crc32b(const char *buf,size_t len)
{
   int          k;
   unsigned int crc=0xFFFFFFFF;

   while (len--)
   {
      crc^=*buf++;
      for (k=0; k<8; k++)
       crc=crc&1 ? (crc>>1) ^ 0x82f63b78 : crc>>1;
   }
   return ~crc;
}
```

EEEE = specifies the file which has to be overwritten by the current data download:

- 0 – overwrite file 0:/version.txt when downloading of data was successful
- 1 – overwrite file 0:/e1803.fwi when downloading of data was successful
- 2 – overwrite file 0:/e1803.dat when downloading of data was successful
- 3 – overwrite file 0:/e1803.cfg when downloading of data was successful

Reset the board

This function performs a warm reboot of the hardware and restarts the firmware. Reboot is done immediately, means this command does not return anything but connection to the board will be interrupted as soon as it has been sent.

C = 0xFF
AAAA = unused, set to 0
BBBB = unused, set to 0
EEEE = unused, set to 0

13 Supported CNC G-Code Commands

Starting with firmware version 2 E1803D supports G-Codes stored in a file on microSD card. A related CNC file has to be placed at microSD card. The file extension decides how it is loaded and interpreted, all G-Code ASCII files need to end with ".CNC".

Starting with firmware version 3 E1803D supports G-codes sent to the controller via USB serial interface or Telnet. Such a data transfer has to be started with command "cscnc" (for details please refer to description above).

To allow fast and efficient processing of a CNC file within E1803D, some points have to be noticed. So in order to improve loading performance it is recommended to:

- not to have lots of leading or trailing spaces
- not to make use of large comments
- have exactly one space between code and related parameter

Beside of that it is mandatory to

- have a space or CR/LF between two different codes (so e.g. "`G21 G90`" is valid but "`G21G90`" will result in an error)
- have no space within a code or within a parameter of a code (so e.g. "`G0 X-0.5 Y.75 Z10`" is valid but "`G 0 X-0 .5 Y. 75 Z 10`" is not and will result in an error)
- use a dot as separator in floating point variables (so e.g. "`T1 F6000.0`" is valid but "`T1 F6000,0`" is not and will result in an error)

Following the G-Code commands are described which are supported.

13.1 General G-Code Characters

Following codes and identifiers are supported by E1803D G-Code interpreter:

Code	Description	Example
%	Marks the begin of a G-Code file, this code is optional and does not have any effect	
;	Begin of a comment, the remaining line is ignored; in order to improve loading speed of a G-Code file comments and spaces at the end of a line should be removed	`G21 ; set unit to mm`
( )	Encapsulate a comment, all data within the brackets are ignored; in order to improve loading speed of a G-Code file comments and spaces at the end of a line should be removed	`G21 (set unit to mm) G90 (use absolute positioning)`
G	G-commands, please refer below for a description	`G1 X25.75 Y31 Z0.25`
M	M-commands, please refer below for a description	`M3`
T	T-commands, please refer below for a description	`T1 F3000`

13.2 Supported "G"-codes

Following "G" codes and identifiers are supported by E1803D G-Code interpreter:

Code	Description	Example
G0	Jump to a specified position using predefined unit mm or inch and with maximal speed or – when set with command M704 – with the jump speed that was defined before. The position to jump to is specified by two or three parameters X, Y and Z. This movement is done with the laser turned off and by taking laser and scanner delays into account.	`G0 X0 Y0 Z0`
G1	Move to a specified position using predefined unit mm or inch and with a default or – when set with command M704 – with the mark speed that was defined before. The position to move to is specified by two or three parameters X, Y and Z. This movement is done with the laser turned on and by taking laser and scanner delays into account.	`G1 X10 Y10.5 Z11.75`
G4	When followed by a parameter „P“ execution is delayed by the given time (in unit seconds)	`G4 P0.002`
G17	Select the XY plane for marking operations. This means, given jump or movement coordinates in X and Y directions are applied to X and Y plane, optional Z coordinate is applied to remaining Z direction.	`G17`
G18	Select the ZX plane for marking operations. This means, given jump or movement coordinates in X and Y directions are applied to Z and X plane, optional Z coordinate is applied to remaining Y direction.	`G18`
G19	Select the YZ plane for marking operations. This means, given jump or movement coordinates in X and Y directions are applied to Y and Z plane, optional Z coordinate is applied to remaining X direction.	`G19`
G20	Set measurement unit to inch, means all positions handed over e.g. with G0 or G1 will be followed by coordinates in inch. In reality this has no effect for E1803D since calculation of marking positions is done based on the given working area which has to be defined with command M709	`G20`
G21	Set measurement unit to mm, means all positions handed over e.g. with G0 or G1 will be followed by coordinates in mm. In reality this has no effect for E1803D since calculation of marking positions is done based on the given working area which has to be defined with command M709	`G21`
G70	Set measurement unit to inch, means all positions handed over e.g. with G0 or G1 will be followed by coordinates in inch. In reality this has no effect for E1803D since calculation of marking positions is done based on the given working area which has to be defined with command M709	`G70`
G71	Set measurement unit to mm, means all positions handed over e.g. with G0 or G1 will be followed by coordinates in mm. In reality this has no effect for E1803D since calculation of marking positions is done based on the given working area which has to be defined with command M709	`G71`
G90	Enable absolute positioning, means all positions handed over e.g. with G0 or G1 will be followed by absolute coordinates according to the used coordinate system.	`G90`
G91	Enable relative positioning, means all positions handed over e.g. with G0 or G1 will be followed by coordinates that are relative to the previously used position in used coordinate system.	`G90`

13.3 Supported “M”-codes

Following “M” codes and identifiers are supported by E1803D G-Code interpreter, here all codes in range 700..799 are specific to the E1803D and contain all laser-related parameters and values:

Code	Description	Example
M2	End of program. When this code is found, parsing of the CNC file is stopped and all following codes are ignored	`M2`
M3	Set laser on. This command does NOT turn on the laser but sets the internal state to „on“. This can be used to set e.g. the mark speed by a following command T1.	`M3`
M5	Set laser off. This command does NOT turn on or off the laser but sets its internal state to „off“. This can be used to set e.g. the jump speed by a following command T1.	`M5`
M700	Set the used laser type. This command is mandatory and has to be called prior to every laser-related command. As parameter it expects a decimal number which corresponds to the lasermode-types `E1803_LASERMODE_xxx` as described in section „14.1 E1803D Easy Interface Functions“	`M700 1073741831`
M701	Set the frequency for the laser in unit Hz. This command is a place holder and has no effect when used with E1803D. Instead of that, the frequency has to be set lasertype-dependent via pulse length parameter of commands M715 (stand-by frequency) and M718 (mark or continuously running frequency)	`M701 25000`
M702	Set the laser power in unit 1/1000%. This command has to be used only in case of specific lasermodes. Resulting from the mode the effect is different: E1803_LASERMODE_YAG – used in case of an SPI-laser, the power-value is set at analogue output AOut0 `E1803_LASERMODE_IPG` – used in case of an IPG-laser, the power-value is latched out at LP8 laserport	`M702 75555`
M703	Set laser-on-delay (at parameter A) and laser-off-delay (at parameter B) in unit usec.	`M703 A100 B200`
M704	Set jump-speed (at parameter A) and mark-speed (at parameter B) in unit mm/min.	`M704 A8936.592 B3000.00`
M705	Set jump-delay (at parameter A), mark-delay (at parameter B) and in-polygon-delay (at parameter C) in unit usec.	`M705 A500 B200 C2500`
M707	Set the working area left position (at parameter X), top position (at parameter Y) and depth position (at optional parameter Z) in unit specified with commands G70 or G71. Together with M709 this command specifies the valid working field, all positioning parameters handed over with e.g. G0 and G1 need to be located within this area.	`M707 X-50 Y50`
M708	Set jump-delay (at parameter A), mark-delay (at parameter B) and variable polygon-delay (at parameter C) in unit usec. Different to command M705 here the third parameter specifies a variable delay which applies only to polygons at maximum angle of 180°.	`M705 A500 B200 C2500`

Code	Description	Example
M709	Set the working area width (at parameter X), height (at parameter Y) and depth (at optional parameter Z) in unit specified with commands G70 or G71. Together with M707 this command specifies the valid working field, all positioning parameters handed over with e.g. G0 and G1 need to be located within this area.	`M709 X-50 Y50`
M710	IPG pulse length value in unit nsec. This command is for future use and currently does not have any effect.	`M710 10000`
M711	Set wobble distance in X-direction (at parameter X), Y-direction (at parameter Y) in unit specified by commands G70 or G71 and wobble frequency (at parameter C) in unit Hz. When all values are set to 0, wobble marking is disabled completely.	`M711 X2.5 Y2.5 C10000`
M713	Set first pulse killer value (FPK) for YAG laser types using unit usec	`M713 10000`
M714	Set simmer value in unit 1/1000% for SPI laser types, the corresponding analogue voltage will be set at output AOut1	`M714 55000`
M715	Set standby-frequency (at parameter A) in unit Hz and standby-pulsewidth (at parameter B) in unit nsec.	`M715 A50000 B1000`
M717	Set waveform number for SPI laser types latched out at LP8 laser port.	`M717 17`
M718	Set laser frequency (at parameter A) in unit Hz and pulsewidth (at parameter B) in unit usec.	`M718 A50000 B10000`
M719	Switch master oscillator on (1) or off (0). This command can be used together with SPI or IPG laser types prior to starting a mark operation to turn MO on or afterwards to turn it off. Required delays to turn on the MO are handled by E1803D internally.	`M719 1`

13.4 Supported “T”-codes

Following “T” codes and identifiers are supported by E1803D G-Code interpreter:

Code	Description	Example
T1	Set jump or mark speed to be used with commands G0 and G1 in unit mm/min. Here it depends on a previous M-command if this value specifies the mark-speed (M3) or the jump speed (M5).	`T1 F6000.0`

14 Programming Interfaces

The e1803.dll/libe1803.so shared library provides an own programming interface that gives the possibility to access and control the E1803D scanner controller card.

Beside of that e1803inter.dll / libe1803inter.so come with some compatibility interfaces for different other scanner cards like Scanlab™ RTC4™ and SCAPS™ USC1/2. They can be used to access E1803D scanner card with existing software easily. To do that, the "e1803inter" shared library just has to be renamed to the library name of the original vendor (like RTC4DLL.DLL, libslrtc4.so or sc_optic.dll). In order to operate E1803D card with different than the default connection settings, recompilation of code to be reused is necessary. Here at the very beginning and before initialisation a call to E1803_set_connection() has to be added in order to specify the connection to the card. Apart from this single case, none of the different programming interfaces should be mixed.

Sources of e1803inter.dll/libe1803inter.so are open and available for free, they can be found in OpenAPC SDK (available for download at https://openapc.com/download.php#sdk) or in GIT (available at https://sourceforge.net/p/oapc/code/ci/master/tree/).

14.1 E1803D Easy Interface Functions

The following functions belong to the native programming interface of E1803D scanner card and should be used preferential in order to get access to all features and full performance of the scanner card. Functions of E1803D Easy Interface are either stream commands that are executed in the order they are called, or functions that are executed immediately.

The E1803D does NOT use the concept of two or more lists that have to be managed and switched by the calling application. Here all stream commands simply are sent to the card without the need to provide some additional management information. Output of data is started only when `E1803_execute()` is called or when a card-internal threshold is exceeded. This card-internal triggered output of data can be held back by calling function `E1803_set_trigger_point()` as very first so that marking starts only after an external trigger signal was detected by the card or when the trigger point was released out of software by calling `E1803_release_trigger_point()`.

E1803D Easy Interface uses unit “bits” as base for all units and parameters. Since E1803D card internally uses 26 bits resolution for a better accuracy and to minimize round-off errors, all calculations are done with these 26 bits. So the working area always has a size of 26 x 26 bits equal to 67108864 x 67108864. Independent from real resolution and output of hardware all calculations have to be done within this 26 bit range.

E1803D Easy Interface provides following functions:

14.1.1 General functions

This section describes all general functions related to accessing the scanner controller card, starting and stopping operation of, checking the current operational state and other things more which are necessary to control flow of data and commands,

`unsigned char E1803_set_connection(const char *address)`

This function has to be called as very first. It is used to specify the IP address where the card is accessible at (in case of Ethernet connection) or the serial interface (in case of USB connection, “COMx” for Windows and “/dev/ttyACMx” for Linux where “x” is the number of its interface). By default IP 192.168.2.254 is used. This is the only function that has to be called in case of both, when compatibility functions and when the E1803D easy function interface is used.
It returns a card index number that has to be used with all following functions (this is true for Easy Interface and RTC-compatible functions).
PLEASE NOTE: calling this function does not open the connection to scanner controller card! This is done on first call to `E1803_load_correction()`!

Parameters:
`address` – a char-array containing the IP in xxx.yyy.zzz.aaa notation or the name of the serial interface (COMx or /dev/ttyACMx) to be used

Return: the board instance number or 0 in case of an error

`void E1803_set_password(const char n,const char *ethPwd)`

Sets a password that is used for Ethernet connection of E1803D card. The same password should be configured on E1803D configuration file e1803.cfg with parameter "`passwd`" to add an additional level of security to an Ethernet controlled card.

PLEASE NOTE: usage of this password does NOT provide enough security to control the card via networks that are accessible by a larger audience, publicly or via Internet! Also when this password is set, the card always should operate in secured, separated networks only!
Every card and every connection should use an own, unique password that can consist of up to 48 characters containing numbers, lower- and upper-case letters and punctuation marks. Due to compatibility reasons no language-specific special characters should be used.
When connected via USB serial interface, this password is ignored. In this case no authentication is done.

Parameters:
`ethPwd` – the password to be used to authorise at an E1803D card. To reset a local password for connecting to a card that doesn't has an Ethernet password configured, hand over an empty string "" here

`void E1803_close(unsigned char n)`

Closes the connection to a card and releases all related resources. After this function was called, no more commands can be sent to the card until `E1803_set_connection()` and `E1803_load_correction()`/`n_load_correction_file()`/`load_correction_file()`/ `ScSCIInitInterface()` is called again.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`

`int E1803_set_debug_logfile(const unsigned char n,const char *path,const unsigned char flags)`

This function can be used during development to check an own application regarding called commands and their parameters. It lets libe1803 write all function calls into a logfile so that it is possible to evaluate the real order of commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`path` – full path to the file which has to be used as debug log file
`flags` – a bunch of OR-concatenated flags which specify what function calls have to be written into or filtered from the log output; when 0x00 is specified here, the log file is kept quite small. When 0x01 is set, all motion-related function calls are added too, when 0x02 is set, all calls which check the state of the card are added to the log file.

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_sync(const unsigned char n,const unsigned int flags,const unsigned int value)`

This function sends a synchronisation `value` to the controller. As soon as marking reaches the related position in stream, the value returned by function `E1803_get_sync()` changes to the value given here.
This command delays execution of the data by 0,5 usec, so it should not be used excessively. A value of 0xFFFFFFFF disables this function.
This function requires firmware version 3 or newer.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`flags` – currently unused, set to 0 for future compatibility
`value` – the value to be used as sync-identifier, here on every call a different value should be handed over in order to differentiate what is returned by `E1803_get_sync()`.

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`unsigned int E1803_get_sync(const unsigned char n)`

Returns a sync-identifier as set by `E1803_set_sync()` as soon as the related position in stream was reached.
This function requires firmware version 3 or newer.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`

Return: the last sync-identifier which was identified and processed in stream of commands or 0xFFFFFFFF when function is not used/turned off

`int E1803_execute(unsigned char n)`

Starts execution of all previously sent commands in case card is not already outputting these data. The E1803D Easy Programming Interface does not use the concept of two or more lists that have to be handled and switched by the calling application. Nevertheless the user has to ensure the card can start marking by calling this function after all vector data have been sent to the card. Here it does not matter if the card is already executing or not, subsequent calls to `E1803_execute()` do not influence marking behaviour. More than this: in case very much data are sent to the card, it starts marking automatically after a defined fill level was reached. Due to this automated, fill level dependent start it would not be necessary to call `E1803_execute()`. But in situations where only very few data are sent to the card it is necessary to call this function always in order to start marking also in these cases where the internal fill threshold is not reached and where the card would not start marking immediately. Thus it is recommended to always call this function after all marking data have been sent.
Marking is finished only when STOP is invoked or when the internal buffer is empty. When internal buffer runs empty because subsequent data are not sent fast enough, an additional call to `E1803_execute()` is necessary in order to output the remaining data.
This is not a stream command since it controls the already sent stream of commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_stop_execution(unsigned char n)`

Stops the currently running execution as fast as possible and drops all data and commands that still may be queued. Calling this function also would drop all laser and scanner parameters that are already sent to the controller but not yet processed. Thus after calling this function it may be necessary to set scanner and laser parameters again in order to ensure they are used for following operations.
This is not a stream command since it controls the current stream of commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_halt_execution(unsigned char n,unsigned char halt)`

Halts or continues the processing and output of marking data. On `halt=1` marking is stopped next time the laser is turned off. Different to a full stop, no vector data are dropped. On continue (`halt=0`) controller continues processing at the point where halt occurred. When marking is stopped with `E1803_stop_execution()` the halt-condition is cleared too, means on next transmission of new marking data they are processed without the need to explicitly continue last operation.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`halt` – 1 to halt operation next time the laser is off, 0 to continue a previously halted operation

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_trigger_point(unsigned char n)`

Specifies a point in data stream where execution has to stop until an external trigger signal (mark start) or a manual release of this trigger point is detected. This expects a rising edge on ExtStart input or calling of function `E1803_release_trigger_point()`.
This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_release_trigger_point(unsigned char n)`

This function should be called only when a previous call to `E1803_set_trigger_point()` was done. It acts like an external trigger signal, releases the waiting condition and lets the controller start processing. So this function provides some kind of software-simulated external start-signal.

ATTENTION: this command will not arrive at the controller when there is no more space left on it, means when all controller-internal buffers are filled. So after a call to `E1803_set_trigger_point()` and during sending of commands and data to the controller, application has to ensure there is some space left in controller's buffers. This can be done by calling `E1803_get_free_space()` with flag `E1803_FREE_SPACE_PRIMARY` for checking the available space in primary buffer. It is recommended to leave space for at least 10000 elements in primary buffer in order to let a call to `E1803_release_trigger_point()` work properly.
When the buffers already have been filled completely, this function will no longer work and marking can be started only by applying the ExtStart hardware signal.
This is not a stream-command, it is applied to controller immediately.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`unsigned int E1803_get_startstop_state(unsigned char n)`

This function returns a bit pattern that informs about state of the start and stop input pins.
This is not a stream command since it returns the current state immediately. Here “current state” means the last known state. When the state changes during this call, it may be possible the previous, no longer actual state is given back since transmission of data from controller to host is done asynchronously and independent from a call to this function.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`

Return: a bit pattern specifying the current state:

- bit 0 and 1 (0x00000003) specify if the start input was set after last call of this function, when these bits are set, a rising edge has been detected at this input; calling this function resets the internal state

of these bits, means when it is called again and when no new rising edge has been detected meanwhile, these bits will be 0
- bit 2 and 3 (0x0000000C) specify if the stop input was set after last call of this function, when they are set, a rising edge has been detected at this input; calling this function resets the internal state of these bits, means when it is called again and when no new rising edge has been detected at top input meanwhile, these bits will be low
- bit 12 (0x00001000) this bit signals the start input is low, as long as this bit is set no start input signal is detected

`int E1803_get_card_state2(const unsigned char n,unsigned int *state)`

This function returns a bit pattern that informs about cards current operational state. Here "current state" means the last known state. When the state changes during this call, it may be possible the previous, no longer actual state is given back since transmission of data from controller to host is done asynchronously and independent from a call to this function.
The card-states are enqueued internally in order to not to lose a "busy"-state which may be available for a very short time only in case of very small and fast marking cycles. So every state change caused by the calling application results in on state change returned by this function. This means for every marking cycle the application has to wait for two state changes: first wait until this function signals "busy" (`E1803_CSTATE_MARKING|E1803_CSTATE_PROCESSING`), next wait until this function signals "ready" (0).
During transfer of vector data and scanner/laser parameters this function should be called as rarely as possible: every call of `E1803_get_card_state()` performs a fully cycle of transmission and receiving of data to and from the controller. Dependent on the current transmission state this may result in submission of a small block of data which does not uses the full available bandwidth. On excessive use of this function this can slow down the whole transfer of data.
This is not a stream command, it returns the current state immediately.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`state` – pointer to a variable where the card state has to be written to: a bit pattern of or-concatenated constants specifying the current state:
- `E1803_CSTATE_MARKING` – card is currently marking
- `E1803_CSTATE_PROCESSING` – card has received some data that are enqueued for marking
- `E1803_CSTATE_WAS_START_PRESSED` – the ExtStart input was triggered, this flag is cleared after it has bean read and is set again only when ExtStart was triggered again
- `E1803_CSTATE_WAS_STOP_PRESSED` – the ExtStop input was triggered, this flag is cleared after it has bean read and is set again only when ExtStop was triggered again
- `E1803_CSTATE_FILE_WRITE_ERROR` – this flag is returned only in case stand-alone data are written to the microSD card and in case an file error occurs during this procedure. As writing an EPR file is done as asynchronous stream, errors during this procedure are not announced by the functions which are called but only by this error state. For more information about writing of stand alone data please refer to section "14.1.9 Writing of stand-alone data"
- `E1803_CSTATE_WAIT_EXTTRIGGER` – the controller is in state "marking" but is not yet processing any data as it is waiting for an external trigger
- `E1803_CSTATE_HALTED` – the controller is in state "marking" but is not yet processing any data as it is currently halted by function `E1803_halt_execution()`
- `E1803_CSTATE_WAIT_INPUT` – the controller is in state "marking" but is not yet processing any data as it is waiting for a specific input pattern at the digital inputs

When the function returns an error code instead of `E1803_OK`, this value is undefined and can't be used.

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_delay(unsigned char n, double delay)`

Pause marking for the given time/wait for execution of the next command in stream for the given time. This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`delay` – time to wait until marking continues in unit usec, smallest possible value is 0,500 usecs

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_get_free_space(unsigned char n,int buffer)`

This function returns the space (in unit "commands") that is free in one of the buffers of E1803. Here parameter `buffer` decides which buffer has to be checked.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`buffer` – expects a constant which decides what buffer has to be checked, it has to be set to one of the following values:

- `E1803_FREE_SPACE_PRIMARY` – return size of the primary buffer; it can be used to avoid memory on host system is filled which may happen when vector data are sent to the controller while it's internal buffers are already full. In this case these data would have been stored on host side consuming some memory there. Using this function this problem can be avoided by sending commands only in case this function returns a value that is (much) larger than 0.
  The primary buffer that can be checked by using this value is one of two available buffers on E1803D controller. The primary one has a size of 1 million and is used to feed the secondary buffer (with a size of 20 million). So when this function returns 1000000, this does not mean the buffer is empty and no vector data currently processed – they still may be stored in secondary buffer. So to check the operational state of the controller, only function `E1803_get_card_state2()` can be used.
  This buffer has also to be checked when function `E1803_release_trigger_point()` is used in order to ensure the command can arrive at the controller. For a detailed description please refer to explanation of `E1803_release_trigger_point()` above.
- `E1803_FREE_SPACE_SECONDARY` – return size of the secondary buffer; this one is filled by data from primary buffer and contains raw commands (like single micro vectors that concatenate to a full vector during output).

Return: -1 in case the function failed or the amount of free space in primary buffer.

`void E1803_get_version(unsigned char n, unsigned short *hwVersion, unsigned short *fwVersion)`

Get the hardware and software version of the used board. It is recommended to call this function after successful connect always and to check if used hardware and firmware version is at least a version that is known to work with own software.
This is not a stream command, it is executed immediately and independent from all other commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`hwVersion` – pointer to a variable where the hardware revision/version number is written into
`fwVersion` – pointer to a variable where the revision/version number of the firmware running on the board is written into

`const int E1803_get_library_version()`

Returns an integer value which is an identifier specifying the version of this shared library. In decimal notation this identifier uses format "Mmmrrr" where "M" is the major version, "m" the minor version number and "r" the release count. The bigger the whole returned number is, the newer the library is.

`int E1803_ana_read(const unsigned char n,const unsigned int flags,unsigned short *a)`

Read a value from one of the analogue inputs. Here the parameter `flags` decides which output has to be accessed in which way:

- E1803_COMMAND_FLAG_ANA_AIN0 – read data from analogue input AIn0
- E1803_COMMAND_FLAG_ANA_AIN1 – read data from analogue input AIn1
- E1803_COMMAND_FLAG_ANA_AIN2 – read data from analogue input AIn2

The value returned in a is always a 16 bit value in range 0..65535 independent from the real resolution of the hardware.

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – flags specifying when the command has to be executed and which analogue output has to be set
a – value read from the analogue input, independent on the real resolution of the hardware, here always a 16 bit value is returned, means a value of 65535 would correspond to full input voltage of 5V
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

This function requires firmware version 4 or newer and the Multi-IO expansion board or the Intelli-IO expansion board.

14.1.2 Laser and scanner related functions

This section describes all functions which are related to the scanhead and laser control and therefore have influence on the signals at the laser interface and the XY2-100 connector. The related interfaces are described in section „6.7 Laser Signals“ and section „6.6 Scanner Signals“.

int E1803_load_correction(unsigned char n, const char* filename, unsigned char tableNum)

Opens connection to the card and loads a correction file to be used during vector data output. In case a previously loaded correction table has to be flushed and no other correction has to be used, parameter "filename" needs to be empty.
This function has to be called for first time on initialisation and before any vector data are sent to the board. It is mandatory to call this function at least once since it establishes connection to E1803D card. So when no correction file has to be used this function still has to be called but with an empty filename “”.
This function supports different correction table file formats directly and without previous conversion:

- BeamConstruct .bco high resolution files
- Scanlab .ctb and .ct5 files
- SCAPS .ucf files
- Raylase .gcd files
- CTI/GSI .xml files
- Sunny .txt 5x5 point correction files

This is not a stream-command, means its data may be applied immediately and independent from current stream state.

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
filename – the full path to the correction file to be loaded from file system, when "" is specified here, a previously used correction file is flushed and no/neutral correction is used as long as no other correction table is given
tableNum – the 0-based correction table number these data have to be loaded for; it is possible to download up to 16 different correction tables and to switch between them during operation using function E1803_switch_correction()

Return: E1803_OK or an E1803_ERROR_- or RTC-compatible return code in case of an error

int E1803_switch_correction(unsigned char n, unsigned char tableNum)

Switches between up to 16 correction tables on the fly. When a table-number is given where no file was downloaded before using function E1803_load_correction(), no correction is performed on all following vector data.
This is a stream-command, means the new correction is applied to vector data sent to the card after this

command but NOT to already sent but not yet processed data. Thus on-the-fly switching between correction tables is possible.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`tableNum` – the 0-based table number of the correction that has to be used for all following vector data

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_xy_correction(const unsigned char n, const double gainX, const double gainY,const double rot,const int offsetX,const int offsetY,const double slantX,const double slantY)`

Sets size correction factor and offset for X and Y direction of working area as well as a rotation. This function will overwrite all corrections specified with `E1803_set_matrix()`.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`gainX` – scale factor in x-direction, 1.0 means no scaling
`gainY` – scale factor in y-direction, 1.0 means no scaling
`rot` – rotation of whole working area in unit degrees
`offsetX` – offset in x-direction in unit bits, 0 means no offset
`offsetY` – offset in y-direction in unit bits, 0 means no offset
`slantX` – trapezoidal correction along X-axis in range -45..45°
`slantY` – trapezoidal correction along Y-axis in range -45..45°

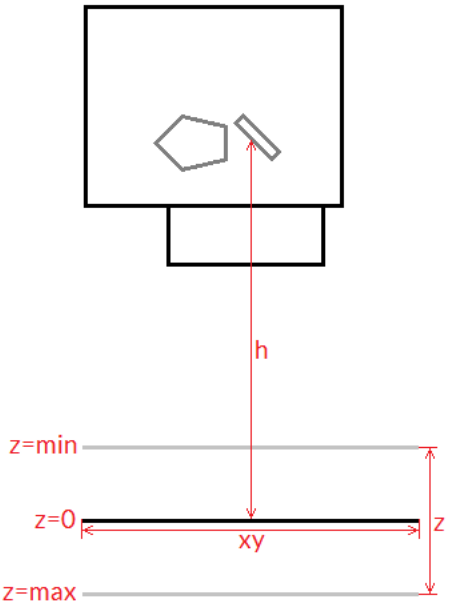
Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_z_correction(const unsigned char n,const unsigned int h,const double xy_to_z_ratio,const int res1)`

Set additional Z correction parameter. This function may be used in cases where third axis is used with a large Z working range. Here additional deviation occurs when no F-Theta lens is used caused by the fact that the beam is always sent from the centre of the scanhead – which causes some kind of projection resulting in larger or smaller X and Y positions depending on the real Z height.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands and changed values apply only to these vector data and coordinates, which are sent after calling this function.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`h` – the vertical height from last mirror of the scanhead to the working area (Z-position 0 of working area) in unit bits
`xy_to_z_ratio` – factor specifying the ratio between maximal horizontal working area size and maximal vertical movement size. As an example: when the working area has a size of 100 x 100 mm and the Z-axis has a maximum movement range of -20 mm .. 20 mm, the ratio to be set is 2,5 (100 mm horizontal divided by 40 mm vertical)
`res1` – reserved for future use, set to 0

For more details please refer to the image below:



Here “h” is the height from the position where the beam hits the last mirror to the position of the working area at z=0 position (in unit bits). “xy” is the width of the working area to be used together with the “z” range from “z=min” to “z=max” to calculate the `xy_to_z_ratio`. All working area parameters like its width “xy” and the “z”-range are expected to be the theoretical maximum of the full range, not the – possibly smaller – range used in a specific setup.

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_speeds(unsigned char n, double jumpspeed,double markspeed)`

Set scanner speed values to be used for all following vector data and until not replaced by other speed values.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands. So values set here apply only to these vector data that are sent after this command.

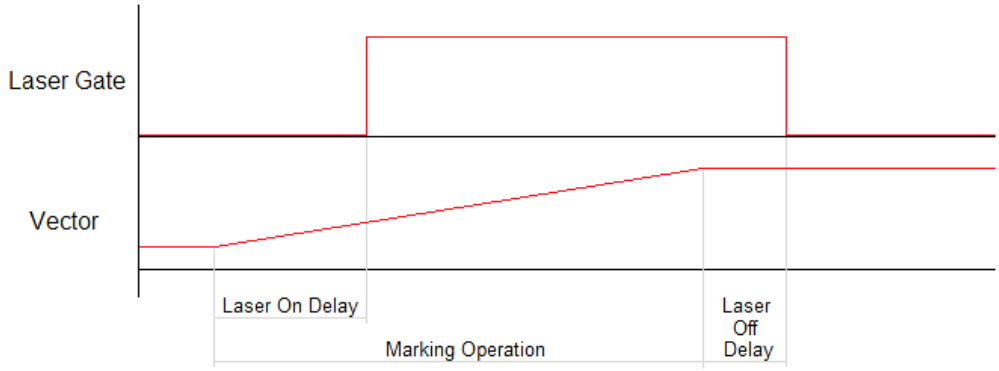
Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`jumpspeed` – scanner movement speed during jumps (movements when laser is off) in unit bits/msec and range 1..4294960000
`markspeed` – scanner speed during mark (movements when laser is on) in unit bits/msec and range 1..4294960000

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_laser_delays(unsigned char n,double ondelay,double offdelay)`

Set laser delay values to be used for all following vector data and until not replaced by other delay values.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands. So values set here apply only to these vector data that are sent after this command.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`ondelay` – laser on delay in unit microseconds, can be a negative or a positive value
`offdelay` – laser off delay in unit microseconds, must be a positive value



Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_scanner_delays(const unsigned char n,const unsigned int flags,const double jumpdelay,const double markdelay,const double polydelay)`

Set scanner delays in unit microseconds. Smallest possible value and resolution is 0.5 microseconds. This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands. So values set here apply only to these vector data that are sent after this command.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`flags` – here some flags can be set which add some further functional specifications and features to this function. At the moment following flags are supported and can be OR-concatenated with each other:

- `E1803_COMMAND_FLAG_SCANNER_VAR_POLYDELAY` – when this flag is set, the value set via `polydelay` is not applied statically to every point within a polygon, but it is set dynamically depending on the angle between two lines; no angle (a straight line) results in no delay while an 180 degree angle results in a full delay as set by value `polydelay`; this flag requires firmware version 2 or newer

`jumpdelay` – the jump delay value in unit microseconds
`markdelay` – the mark delay value in unit microseconds
`polydelay` – the in-polygon delay value in unit microseconds

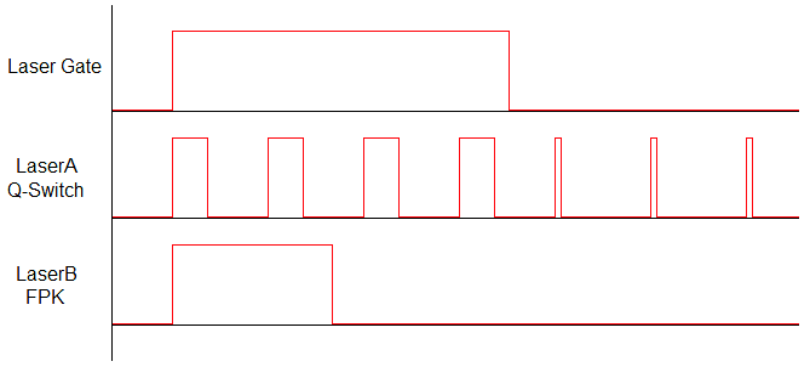
Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_laser_mode(unsigned char n, unsigned int mode)`

Sets the laser mode to be used for all following operations, this value influences the signals emitted at the connectors of the card. This function has to be called prior to setting any other laser parameters (like frequency, standby-frequency, power).
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

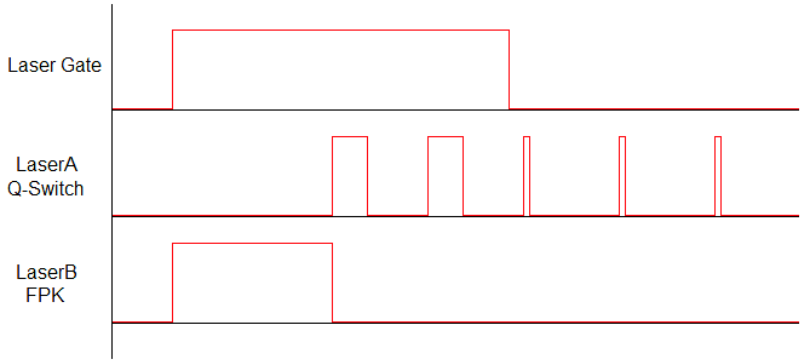
Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`mode` - the laser mode, here one of the following values is possible:

- `E1803_LASERMODE_CO2` – for controlling CO_2 lasers, this mode supports stand-by frequency at LaserA output (to be set with function `E1803_set_standby()`) and PWM-modulated frequencies during marking and for power control (to be set with function `E1803_set_laser_timing()`)
- `E1803_LASERMODE_YAG1` – for controlling YAG lasers, this mode supports stand-by and Q-Switch frequency at LaserA output (to be set with function `E1803_set_standby()`) and a first pulse killer signal at output LaserB that is issued on beginning of a mark together with the Q-Switch frequency (to be set with function `E1803_set_fpk()`):



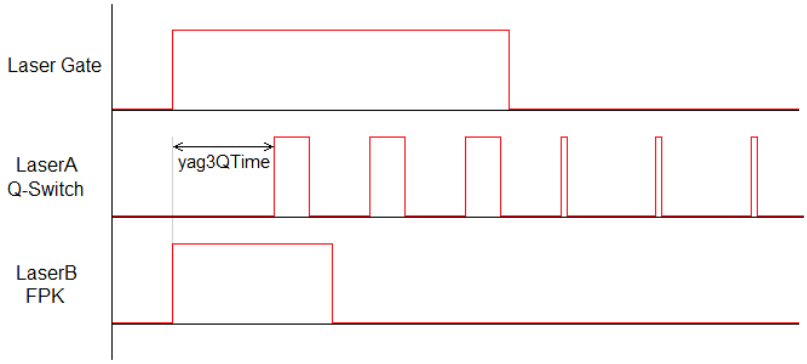
Here Q-Switch signal is started together with laser gate and FPK pulse. At end of mark when laser gate is turned off stand-by frequency is emitted at LaserA.

- `E1803_LASERMODE_YAG2` - for controlling YAG lasers, this mode supports stand-by and Q-Switch frequency at LaserA output (to be set with function `E1803_set_standby()`) and a first pulse killer signal at output LaserB that is issued on beginning followed by Q-Switch frequency that starts when FPK pulse has finished:



Here FPK and laser gate are started together. Q-Switch signal is started at end of FPK pulse. At end of mark when laser gate is turned off, stand-by frequency and pulse-width is emitted at LaserA instead of Q-Switch frequency.

- `E1803_LASERMODE_YAG3`– for controlling YAG lasers, this mode supports stand-by and Q-Switch frequency at LaserA output (to be set with function `E1803_set_standby()`) and a first pulse killer signal at output LaserB that is issued on beginning followed by Q-Switch frequency that starts after a freely configurable time period “yag3QTime”:



Here FPK and laser gate are started together. Q-Switch signal is started after yag3QTime has elapsed according to the beginning of FPK pulse. This time value can be set using function E1803_set_fpk(). At end of mark when laser gate is turned off, stand-by frequency and pulse-width is emitted at LaserA instead of Q-Switch frequency.

- `E1803_LASERMODE_CRF` – for controlling lasers that require a continuously running frequency (like fiber-lasers), this frequency is emitted at LaserA output and can be set and changed by calling function `E1803_set_standby()`.
- `E1803_LASERMODE_DUAL` – for controlling special lasers that require two frequencies, the second, continuously running frequency is emitted at LaserB output and can be set with function `E1803_set_laserb()`

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_laser(const unsigned char n,const unsigned int flags,const char on)`

Switches the laser on or off independent fro many mark or jump commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`flags` – handling flags specifying the behaviour of this command, `E1803_COMMAND_FLAG_STREAM` to use it as stream command, `E1803_COMMAND_FLAG_DIRECT` to execute it immediately and independent on current stream and execution state; in case `E1803_COMMAND_FLAG_STREAM` is used, please ensure this function call is followed by other stream commands, elsewhere the laser is turned off for security reasons as soon as no more data are available to process in order to not to let the laser fire while the card is waiting
`on` – set to 1 to turn the laser on or to 0 to turn it off

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_wobble(unsigned char n,unsigned int x,unsigned int y,double freq)`

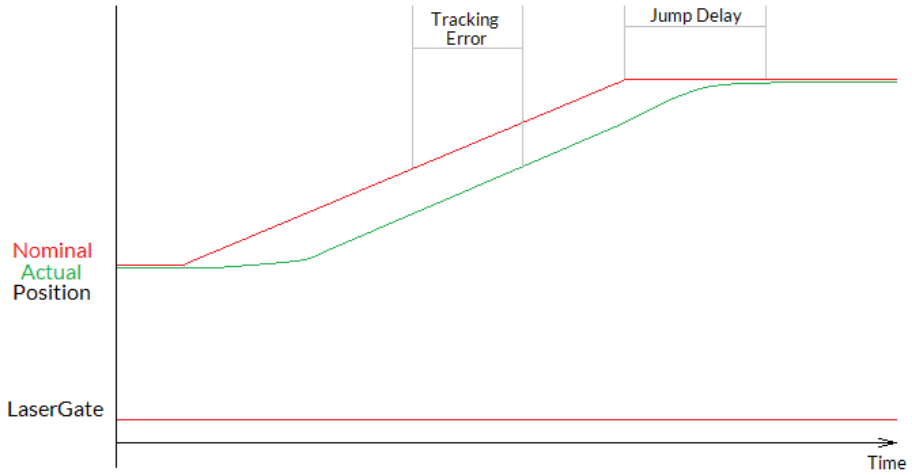
This function gives the possibility to not to let the laser beam follow the given path directly but to rotate around the specified path and lasers current position. Depending on chosen wobble-parameters and marking speed this results either in a thick or a sinusoidal line. This call sets wobble parameters to be used for all following vector data and until not replaced by other wobble values or by 0 which disables wobble mode. This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`x` – wobble amplitude in x direction in units bits and range 1..10000000
`y` – wobble amplitude in y direction in units bits and range 1..10000000
freq – wobble frequency in Hz in range 1..25000

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_jump_abs(unsigned char n,int x,int y,int z)`

Perform a jump (movement with laser turned off) to the given position. This causes a galvo movement from current position to the one specified by this functions parameters using the jump speed and taking the jump delay into account:



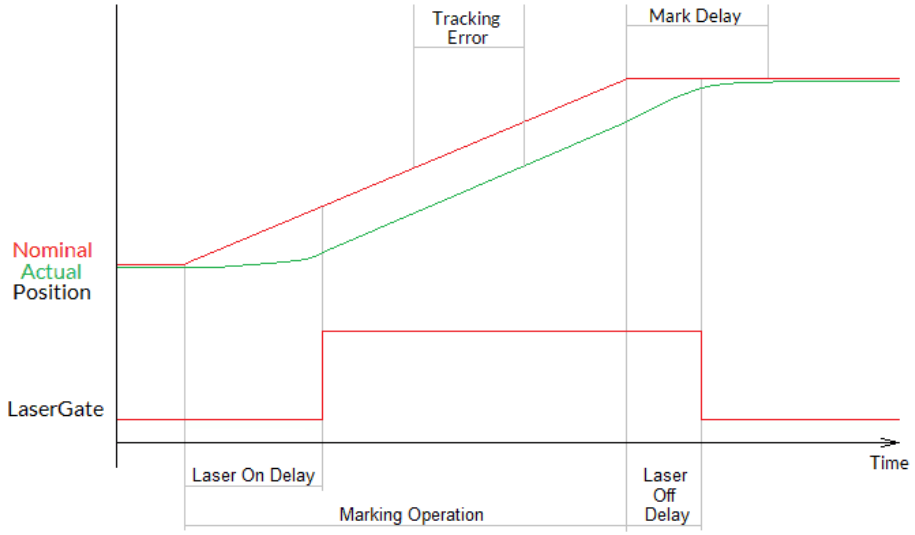
When laser was turned on before this function is called, laser is turned off at the beginning with a delay specified by laser off delay (please refer to description of `E1803_mark_abs()` for a diagram showing laser off delay too).
This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands. So values set here apply only to these vector data that are sent after this command.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`x` – the x-coordinate in unit bits the scanner has to jump to (in range -33554431..33554432)
`y` – the y-coordinate in unit bits the scanner has to jump to (in range -33554431..33554432)
`z` – the z-coordinate in unit bits the scanner has to jump to (in range -33554431..33554432, requires a hardware that is equipped with Z- channel)

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_mark_abs(unsigned char n,int x,int y,int z)`

Perform a mark (movement with laser turned on) to the given position. This causes a galvo movement from current position to the one specified by this functions parameters using the mark speed and taking the mark delay into account. When laser was turned off before this function is called, laser is turned on at the beginning with a delay specified by laser on delay:



This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`x` – the x-coordinate in unit bits the scanner has to move to (in range -33554431..33554432)
`y` – the y-coordinate in unit bits the scanner has to move to (in range -33554431..33554432)
`z` – the z-coordinate in unit bits the scanner has to move to (in range -33554431..33554432, requires a hardware that is equipped with Z- channel)

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_pos(unsigned char n,int x,int y,int z,unsigned char laserOn)`

Perform a raw, immediate movement to the given position.

HANDLE WITH CARE! This function causes galvo movement to the given position immediately, without respect to any mark or jump speed values, without micro-vectorisation or intermediate steps! This means it can result in a very heavy movement for the galvos and in worst case it may cause some damage! Since the resulting movement speed may be way too high for the used galvos, they may overshoot and need some time until the

desired position is reached. So this function is mainly intended to be used for very small position changes in respect to the galvos current position.
This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`x` – the x-coordinate in unit bits the scanner has to jump to (in range -33554431..33554432)
`y` – the y-coordinate in unit bits the scanner has to jump to (in range -33554431..33554432)
`z` – the z-coordinate in unit bits the scanner has to jump to (in range -33554431..33554432, requires a hardware that is equipped with Z- channel)
`laserOn` – specifies if the movement has to be done with laser turned on (1) or off (0)

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_set_pixelmode(const unsigned char n,const unsigned int mode,const double powerThres,const unsigned int res)`

Set the operational mode for `E1803_mark_pixelline()`. This function influences the behaviour when marking a pixel line. This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`mode` – pixel marking mode, this parameter can be set to:

- 0 – default mode, while marking a pixel line the controller tries to perform jumps when power is below of the given threshold `powerThres` to save marking time
- `E1803_PIXELMODE_NO_JUMPS` – no jumps are performed, the given power threshold is ignored and the full pixel line is done with marking speed; this mode is slower but can result in more accurate and more exact images
- `E1803_PIXELMODE_JUMP_N_SHOOT` – marking of the line is no longer done with a continuous movement but with a sequence "jump to position → shoot → jump to next position → shoot → jump to next position → shoot..."; here the shoot-time is equal to the laser-off-delay minus laser-on-delay as set with function `E1803_set_laser_delays()`

`powerThres` – this value is used only in default mode, when the marking power for some pixels is below of the given value (in unit percent), a jump is performed to save marking time, during this jump the laser is off and no marking is done
`res` – reserved, set always to 0

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_mark_pixelline(const unsigned char n,int x,int y,int z,const int pixWidth,const int pixHeight,const int pixDepth,unsigned int pixNum,const double *pixels,E1803_power_callback power_callback,void *userData)`

This function can be used to mark a single line of a bitmap image. Here horizontal, vertical and even 3D bitmap lines (going into depth) can be marked. Direction and orientation of the line to be marked can be chosen freely. A full image can be created by concatenating several lines. Power control during marking of such a bitmap line is not limited to some specific power outputs, it can be fully customised via a callback function.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`x, y, z` – the starting coordinates of the line in unit bits
`pixWidth` – the width of a single pixel (in unit bits), when this is set to a value greater or smaller than 0 while all the others are equal 0, a horizontal line is drawn; the sign of the value specifies the marking direction
`pixHeight` – the height of a single pixel (in unit bits), when this is set to a value greater or smaller than 0 while all the others are equal 0, a vertical line is drawn; the sign of the value specifies the marking direction

pixDepth – the depth of a single pixel (in unit bits, requires a 3D-capable scanhead), when this is set to a value greater or smaller than 0 while all the others are equal 0, line goes into depth; the sign of the value specifies the marking direction
pixNum – the number of pixel data contained in the array of intensity values handed over with the following parameter
pixels – an array of double-values with a length equal the number of pixels specified with pixNum and with an allowed range of 0.0..100.0 specifying the intensity; every entry of this array is equal to one pixel of the bitmap, so a greyscale-pixelline with brightness values in range 0..255 has to be converted to values in range 0.0..100.0
power_callback – this is a callback function of type

```
int (*E1803_power_callback)(unsigned char n, double power, void *userData)
```

which is used to set the power for every pixel. There these E1803_-functions have to be called that belong to the used laser type and set the power values according to it's hardware capabilities. Within the power callback function only stream commands are allowed to be called. It is not possible to use external devices that are not synchronous to E1803D command stream. The power callback has to return with E1803_OK when setting of power was successful. In case of an error the appropriate error code has to be returned, the pixel marking function will be cancelled in such a case too and does not finish marking of the line. Parameter n is the 1-based board instance number specifying the board the power has to be changed for, power is the power to be set in unit percent and userData are some free to use, custom data that can be handed over on call to E1803_mark_pixelline().

userData – here some custom data can be handed over which are forwarded on and handed over at every call of the power-callback

Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_set_matrix(unsigned char n, double m11, double m12, double m21, double m22)

Specify a 2x2 matrix that contains scaling and rotation corrections for the output. When a given matrix element parameter has a value smaller or equal -10000000 it is ignored and the previous/default value is kept at this position in matrix.
This function will overwrite all corrections specified with E1803_set_xy_correction().
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
m11 – first matrix element in first row
m12 – second matrix element in first row
m21 – first matrix element in second row
m22 – second matrix element in second row

Return: E1803_OK or an E1803_ERROR_-return code in case of an error

unsigned int E1803_get_head_state(const unsigned char n,const unsigned int flags)

Returns head status information in case the connected scanhead provides such data via STATUS signal of XY2-100 interface. When the head does not provide such information or returns invalid data or a proprietary data format, the function returns 0xFFFFFFFF. Otherwise the returned value can be AND-concatenated with HEAD_STATE_MASK to find out what kind of head is connected: a resulting value of HEAD_STATE_2D_HEAD identifies a 2D scanhead, HEAD_STATE_3D_HEAD a 3D scanhead. Depending on this, the returned value contains the following state information:

Bit	HEAD_STATE_2D_HEAD	2D Head Remarks	HEAD_STATE_3D_HEAD	3D Head Remarks
19 / C2	0	Identification bit	0	Identification bit
18 / C1	1	Identification bit	0	Identification bit
17 / C0	1	Identification bit	1	Identification bit
16 / S15	Power state		X servo ready	
15 / S14	Temperature state		X temperature state	
14 / S13	In-field		X tracking error	
13 / S12	X-position ACK		0	
12 / S11	Y-position ACK		Y servo ready	
11 / S10	1		Y temperature state	
10 / S9	0		Y tracking error	
9 / S8	1		0	
8 / S7	Power state		Z servo ready	
7 / S6	Temperature state		Z temperature state	
6 / S5	In-field		Z tracking error	
5 / S4	X-position ACK		0	
4 / S3	Y-position ACK		X channel parity error	
3 / S2	1		Y channel parity error	
2 / S1	0		Z channel parity error	
1 / S0	1		CLK channel error	
0 / Par	x	Always 0	Parity bit (even)	

The exact usage of these fields depends on the used head, so for further details please refer to the related scanhead manual.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`flags` – when flag `E1803_COMMAND_FLAG_HEAD_STATE_RAW` is set here, the state-information from the head are returned as they are received. When this flag is not used, the returned data are checked and filtered – only in case they fit to the bit patterns shown above, the received data are returned, elsewhere the error information 0xFFFFFFFF is given back.

Return: the received (filtered or raw) XY2-100 state data as received from the head or 0xFFFFFFFF in case of an error

`int E1803_set_laser_timing(unsigned char n, double frequency, double pulse)`
Set the frequency and pulse-width to be used during marking at LaserA output of laser connector. This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`frequency` – emitted frequency in unit Hz and in range 25..20000000 Hz
`pulse` – pulse width in usec, this value has to be smaller than period length that results out of frequency

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

int E1803_set_standby(const unsigned char n,const double frequency,const double pulse)

Set the frequency and pulse-width to be used during jumps, as stand-by frequency or as continuously running frequency at LaserA output of laser connector.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
`frequency` – emitted frequency in unit Hz and in range 25..20000000 Hz. When a value of 0 is given, the frequency at LaserA output is turned off at end of mark.
`pulse` – pulse width in usec, this value has to be smaller than period length that results out of `frequency`
`n` – the 1-based board instance number as returned by `E1803_set_connection()`

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

int E1803_set_laserb(const unsigned char n,const double frequency,const double pulse)

Set the frequency and pulse-width to be used at LaserB output of laser connector. To use LaserB as second frequency output, a laser mode with flag `E1803_LASERMODE_DUAL` has to be configured.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`frequency` – emitted frequency in unit Hz and in range 25..20000000 Hz
`pulse` – pulse width in usec, this value has to be smaller than period length that results out of frequency

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

int E1803_set_fpk(unsigned char n, double fpk, double yag3QTime)

Set the parameters for first pulse killer signal that is emitted via laser connector whenever the laser is turned on; this applies to YAG-modes only and is emitted as one single pulse at LaserB output.
This is a stream-command, means its parameters are applied at a point in stream that is relative to the other stream commands.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`fpk` – the length of the first pulse killer signal in usec
`yag3QTime` – the length of the first pulse killer signal in usec, this value is used only when laser mode `E1803_LASERMODE_YAG3` is set, elsewhere it is ignored

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

int E1803_lp8_write(const unsigned char n,const unsigned int flags,const unsigned char value)

Sets the LP8_0..LP8_7 outputs of 8 bit laser port of laser interface connector without touching the related latch output. Total execution time of this command is 1 usec.
Depending on the value of parameter flags this is either a stream-command (means it is executed at a point in stream that is relative to the other stream commands) or a direct command (means it is executed immediately on calling).

Parameters:
`n` – the 1-based board instance number as returned by E1803_set_connection()
`flags` – handling flags specifying the behaviour of this command, `E1803_COMMAND_FLAG_STREAM` to use it as stream command, `E1803_COMMAND_FLAG_DIRECT` to execute it immediately and independent on current stream and execution state

value – the 8 bit value to be set at LP8 port
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_lp8_write_latch(unsigned char n, unsigned char on, double delay1,unsigned char value, double delay2,double delay3)

Sets the LP8 8 bit laser port of laser interface connector with freely definable delays and toggles the related latch output automatically; calling this function causes the following sequence of commands:

- turn latch bit on/off
- wait for delay1 usecs
- set LP8
- wait for delay2 usecs
- turn latch bit off/on
- wait for delay3 usecs

The whole execution time of this sequence is 1.5 usecs for setting LP8 outputs and toggling latch plus delay1 plus delay2 plus delay3.
This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
on – specifies if the latch bit has to be set to HIGH (on=1) or LOW (on=0) on first step, on second step it will toggle to value !=on
delay1 – delay to be issued after setting/clearing the latch bit for the first time
value – the 8 bit value to be set at LP8 port
delay2 – delay to be issued after setting LP8 output and before clearing/setting the latch bit
delay3 – delay to be issued after clearing/setting the latch bit for the second time

Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_lp8_write_mo(unsigned char n, unsigned char on)

Sets the master oscillator output MO of laser interface connector to be used with e.g. fiber lasers.
This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
on – the state the MO output has to be switched to

Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_ana_write(const unsigned char n,const unsigned int flags,const unsigned short a)

Write a value to one of the analogue outputs. Here the parameter flags decides which output has to be accessed in which way:

- E1803_COMMAND_FLAG_ANA_AOUT0 – send data to analogue output AOut0
- E1803_COMMAND_FLAG_ANA_AOUT1 – send data to analogue output AOut1

One of these flags can be combined with E1803_COMMAND_FLAG_STREAM to use it as stream command or with E1803_COMMAND_FLAG_DIRECT to execute it immediately and independent on current stream and execution state. Parameter a needs to be always a 16 bit value independent from the real resolution of the hardware.

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – flags specifying when the command has to be executed and which analogue output has to be set

a – value to be set at the analogue output, independent on the real resolution of the hardware, here always a 16 bit value has to be given, means a value of 65535 would correspond to full output voltage of 10V
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

14.1.3 Digital interface functions

The following section describes all functions which can be used to describe data at the digital interface as described in section „6.8 Digital Interface“ and section „6.8.1 Marking On-The-Fly Signals“.

int E1803_digi_write(const unsigned char n,const unsigned int flags,const unsigned int value,const unsigned int mask)

Sets the 8 bit digital output port.
Depending on the value of parameter flags this is either a stream-command (means it is executed at a point in stream that is relative to the other stream commands) or a direct command (means it is executed immediately on calling).
Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – handling flags specifying the behaviour of this command, E1803_COMMAND_FLAG_STREAM to use it as stream command, E1803_COMMAND_FLAG_DIRECT to execute it immediately and independent on current stream and execution state
mask – specifies which of the bits in "value" have to be used for setting and clearing output data, only these bits that are set to 1 in mask are changed according to the given value
value – the 8 bit value to be set at digital out port

Return: E1803_OK or an E1803_ERROR_ return code in case of an error

int E1803_digi_pulse(const unsigned char n, const unsigned int flags, const unsigned int in_value, const unsigned int mask, const unsigned int pulses, const double delayOn, const double delayOff)

Send a sequence of pulses to the 8 bit digital interface. When the controller works with a firmware version 5 or later, this operation causes nearly no data transmission load.
This command is available as stream-command only (means it is executed at a point in stream that is relative to the other stream commands).
Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – currently only E1803_COMMAND_FLAG_STREAM is supported here
mask – specifies which of the bits in "value" have to be used for setting and clearing output data, only these bits that are set to 1 in mask are changed according to the given value
value – the 8 bit value to be set at digital out port
pulses – specifies how often the output has to be set/cleared
delayOn – the delay (in unit usec) which has to be issued every time after setting the output, the minimal resolution of this value is 0,5 usec
delayOff – the delay (in unit usec) which has to be issued every time after clearing the output, the minimal resolution of this value is 0,5 usec

Return: E1803_OK or an E1803_ERROR_ return code in case of an error

int E1803_digi_read2(const unsigned char n,unsigned int *value)

Reads the 8 bit digital input port.
This is not a stream-command, means it is executed immediately and returns current state of the digital inputs. When marking on the fly is enabled using function E1803_digi_set_motf2(), digital inputs 0 and 1 (and optionally also digital inputs 2 and 3 in case of 2D marking on-the-fly) are used for MOTF-encoder and therefore not available as standard inputs. In such a case state of these bits is undefined and does not reflect the current input state caused by the external encoder.

Parameters:
n - the 1-based board instance number as returned by E1803_set_connection()
value – pointer to a variable where the current digital input state has to be written into.
When the function returns an error code instead of E1803_OK, this value is undefined and can't be used.

Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_digi_wait(unsigned char n,unsigned long value,unsigned long mask)

Stop execution and output of data until the given bitpattern was detected at digital inputs of digital interface connector. Here parameter mask specifies which of the bits at the input have to be checked, they have to be set to 1. These bits within mask that need to be ignored have to be set to 0. Parameter value itself defines the states of the bits that has to be detected at the input to continue processing of data. All bits of value that correspond to bits of mask, that are 0, are ignored.

Parameters:
n - the 1-based board instance number as returned by E1803_set_connection()
value – the expected bitpattern at digital input
mask – specifies which of the input bits and value bits have to be used for comparison

Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_digi_set_motf2(const unsigned char n,const unsigned int flags,const double motfX,const double motfY)

Disables or enables marking on-the-fly functionality and specifies factors for X- and Y-direction. When this function is called with values for motfX or motfY greater than 0, marking on-the-fly is enabled and digital inputs 0 and 1 of the digital interface are no longer available as general purpose inputs. Now they are used as decoder inputs for a 90 degree phase shifted encoder signal for marking on-the-fly functions. When both parameters motfX and motfY are set to 0, marking on-the-fly is disabled and inputs 0 and 1 no longer work as encoder inputs.
When tune flag "2" is set (for 2D marking on-the-fly, please refer to description of e1803.cfg parameters above), the two factors for X and Y are assigned to separate encoder inputs. Here factor for X applies to values received on digital inputs 0 and 1 and factor for Y applies to values received on digital inputs 2 and 3.
Depending on value of parameter flags this is or is not a stream-command, means it switches states of digital inputs 0 and 1 (plus optionally 2 and 3) and marking on-the-fly functionality at the related position in stream or immediately.
Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – when flag E1803_COMMAND_FLAG_DIRECT is set, the new MOTF-factors are applied immediatley, when flag E1803_COMMAND_FLAG_STREAM is used instead, the command acts as stream-command and sets the new MOTF-factors as soon as this command is due in current stream of command.
motfX – marking on-the-fly factor for X-direction in unit bits per encoder increment
motfY – marking on-the-fly factor for Y-direction in unit bits per encoder increment
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_digi_set_motf_sim(unsigned char n, double motfX, double motfY)

Disables or enables simulated marking on-the-fly functionality and specifies factors for X- and Y-direction. When this function is called with values for motfX or motfY greater than 0, simulated marking on-the-fly is enabled and internal 100 kHz signal generator is used to create static marking on-the-fly pulses in positive direction. A possibly enabled on-the-fly operation using external signals on digital inputs 0 and 1 of digital interface connector is disabled. When both parameters motfX and motfY are set to 0, marking on-the-fly is disabled completely.
This is not a stream-command, means it enables simulated marking on-the-fly functionality immediately.
Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()

motfX – marking on-the-fly factor for X-direction in unit bits suitable for to be simulated movement-speed on 100 kHz encoder counting frequency
motfY – marking on-the-fly factor for Y-direction in unit bits suitable for to be simulated movement speed on 100 kHz encoder counting frequency
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_digi_wait_motf(const unsigned char n, const unsigned int flags, const double dist)

Halts the current marking operation for a given distance of the on-the-fly encoder. Different to E1803_delay() this function does not use a time to wait until marking is continued but a distance specified by parameter dist and measured by the connected encoder. To use this function marking on-the-fly has to be enabled by calling E1803_digi_set_motf() or E1803_digi_set_motf_sim() before.
This is a stream-command, means it is executed at a point in stream that is relative to the other stream commands.
Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – specifies how the distance value is handed over, with E1803_COMMAND_FLAG_MOTF_WAIT_INCS a value in unit "encoder increments" is expected, with E1803_COMMAND_FLAG_MOTF_WAIT_BITS a distance in unit "bits" is expected. In second case the X-on-the-fly factor of a preceding call to E1803_digi_set_motf() or E1803_digi_set_motf_sim() is used.
dist – the distance to wait for until marking has to be completed, the unit of this value is specified with preceding parameter flags
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_digi_set_motf_powerctl(const unsigned char n,const unsigned int flags,const double motfSpeed,const double lowValue, const double highValue)

Perform a marking on-the-fly speed-dependent power adjustment. This function uses the currently set power as well as the handed over nominal speed to calculate the real power based on the real marking on the fly speed. So to use this functionalities, following sequence of commands is necessary:
- set the on-the-fly factors/enable marking on the fly by calling E1803_digi_set_motf()
- set the current power, here dependent on the used laser type the appropriate function has to be called
- call E1803_digi_set_motf_powerctl() to define automatic power adjustment parameters
This function requires firmware version 7 or newer.
Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – specifies the method of power control/the power control output which has to be used. Here exactly one of the following flags can be used:
- E1803_COMMAND_FLAG_ANA_AOUT0 – to use analogue output A0 for power control
- E1803_COMMAND_FLAG_ANA_AOUT1 – to use analogue output A01for power control
- E1803_COMMAND_FLAG_FREQ_LASERA – this is not really a power control method but enables the option to change the frequency of LaserA output while the pulse-pause-ratio of the related waveform is kept constant
motfSpeed – nominal speed (in unit bits/sec) which corresponds to the current nominal power set in previous call, when the real, measured marking on-the-fly speed rises above this value, the related power output is adjusted to also ensure a higher power output, when the actual, measured MOTF-speed becomes lower than the nominal motfSpeed, power is adjusted to lower values
lowValue – lower clipping value, the automatically adjusted power will never become smaller than this
highValue – upper clipping value, the automatically adjusted power will never become bigger than this
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_digi_set_mip_output(unsigned char n,unsigned int value,unsigned int flags)

This function can be used to specify which of the digital outputs has to be used for signalling "marking in progress". When value is set to 0xFFFFFFFF, this function is disabled and scanner controller card does not provide this signal automatically. When the number of the digital output (in range 0..7) is given as value, the related digital output pin is used for "mark in progress" signal.

PLEASE NOTE: here the number (means the count) of one specific output pin has to be given, not a bitpattern specifying one or more pins!
During operation the selected "mark in progress" pin is HIGH as long as the scanner is moving and/or the laser is on and/or a delay is processed and when marking parameter are processed between these operations. It becomes LOW as soon as no more marking data are available and current operation is stopped or when scanner is waiting for an external trigger signal (ExtStart).
This is not a stream-command, when it is called it is applied to current configuration immediately.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`value` – the number of the digital output to be used for this signal
`flags` - currently unused, set always to 0 for compatibility
Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_digi_set_wet_output(const unsigned char n,const unsigned int value,const unsigned int flags)`

This function can be used to specify which of the digital outputs has to be used for signalling "waiting for external trigger". When `value` is set to 0xFFFFFFFF, this function is disabled and scanner controller card does not provide this signal automatically. When the number of the digital output (in range 0..7) is given as value, the related digital output pin is used for "waiting for external trigger" signal.

PLEASE NOTE: here the number (means the count) of one specific output pin has to be given, not a bitpattern specifying one or more pins!
During operation the selected "waiting for external trigger" pin is HIGH as long as the controller is waiting for an external trigger to be applied at ExtStart input. It becomes LOW as soon as this signal has been detected or when current operation is stopped.
This is not a stream-command, when it is called, it is applied to current configuration immediately.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`value` – the number of the digital output to be used for this signal
`flags` – currently unused, set always to 0 for compatibility

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

14.1.4 Serial interface functions

Following functions are described which can be used to access the serial interface as described in section “6.9 Serial Interface”

`int E1803_uart_write(const unsigned char n,const unsigned int flags,const char *sendData,const unsigned int in_length,unsigned int *sentLength)`

Send data to RS232/RS485 serial interface using the serial interface parameters which are configured in e1803.cfg configuration file. This command is executed immediately, means the data are sent to the serial interface independent from the context the function has to be called within.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`flags` – here pairs of flags a) and b) have to be OR-concatenated with each other to specify how to transmit the data exactly:

a) `E1803_COMMAND_FLAG_UART1` – this flag specifies which UART interface has to be used for transmitting data, at the moment only one UART interface exists, so only this flag can be set
b) `E1803_COMMAND_FLAG_DIRECT` – when this flag is combined with the flag from a), data are transmitted immediately (means as fast as possible and not within the regular stream of data)
b) `E1803_COMMAND_FLAG_ASYNC` – when this flag is combined with the flag from a), data are transmitted asynchronously (means as soon as the next package of data is sent to the controller which either is done on E1803_execute() or when enough other data are handed over, on arrival at the

controller these data are sent to the UART interface immediately and not within the regular stream of data)
sendData – pointer to byte-array which contains the data which have to be sent
in_length – length of the data in sendData
sentLength – pointer to a variable where the amount of data is returned which really has been sent by this function; when a different return code than E1803_OK is given back, this value is undefined.
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_uart_read(const unsigned char n,const unsigned int flags,char *recvData,const unsigned int maxLength,unsigned int *receivedLength)

Receive data from RS232/RS485 serial interface using the serial interface parameters which are configured in e1803.cfg configuration file. This command is executed immediately, means it checks for data arrived at serial interface independent from the context the function has to be called within.

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – here pairs of flags a) and b) have to be OR-concatenated with each other to specify how to receive the data exactly:
a) E1803_COMMAND_FLAG_UART1 – this flag specifies which UART interface has to be used for receiving of data, at the moment only one UART interface exists, so only this flag can be set
b) E1803_COMMAND_FLAG_DIRECT – when this flag is combined with the flag from a), data are tried to be read immediately, this means the function does not return until some data could be read from the serial interface or until a timeout occured
b) E1803_COMMAND_FLAG_ASYNC – when this flag is combined with the flag from a), data are received asynchronously, this means when no data are available on call of the function, it returns immediately with E1803_OK and setting receivedLength to 0. In such a case the function has to be called later again in order to receive some data. As soon as some data have arrived, receivedLength specifies the size of these data on return of the function
recvData – pointer to byte-array where received data have to be stored into, this buffer should have a size of at least 513 bytes
maxLength – maximum number of bytes the buffer specified by recvData is able to store
receivedLength – pointer to a variable where the amount of data is returned which really has been received by this function; when a different return code than E1803_OK is given back, this value is undefined.
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

14.1.5 Intelli-IO extension functions (IO-mode)

Following functions require the Intelli-IO Extension Board as described in section "10.2 Intelli-IO Interface in IO mode". They can be used to access the digital IOs of this board. The analogue inputs of the Intelli-IO board are not directly subject to this extension and therefore can be read by general function E1803_ana_read() as described above. All functions described here require firmware version 5 or newer.

int E1803_ext_digi_write(const unsigned char n,const unsigned int flags,const unsigned int in_value,const unsigned int mask)

Sets the second 8 bit digital output port which is located on the Intelli-IO extension.
Depending on the value of parameter flags this is either a stream-command (means it is executed at a point in stream that is relative to the other stream commands) or a direct command (means it is executed immediately on calling).
Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – handling flags specifying the behaviour of this command, E1803_COMMAND_FLAG_STREAM to use it as stream command, E1803_COMMAND_FLAG_DIRECT to execute it immediately and independent on current stream and execution state
mask – specifies which of the bits in "value" have to be used for setting and clearing output data, only these bits that are set to 1 in mask are changed according to the given value
value – the 8 bit value to be set at digital out port

Return: E1803_OK or an E1803_ERROR_ return code in case of an error

int E1803_ext_digi_read(const unsigned char n,unsigned int *value)

Reads the second 8 bit digital input port which is located on the IntelliIO extension. Since this extension board provides only six input bits, the upper two bits always will be 0.
This is not a stream-command, means it is executed immediately and returns current state of the digital inputs.

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
value – pointer to a variable where the current digital input state has to be written into.
When the function returns an error code instead of E1803_OK, this value is undefined and can't be used.

Return: E1803_OK or an E1803_ERROR_-return code in case of an error

14.1.6 Intelli-IO extension functions (motion mode)

Following functions require a Intelli-IO Extension Board as described in section "10.3 Intelli-IO Interface in motion mode" and can be used to control motion operations. Here command options that set a value or start a motion are always available in two options: as stream (flag E1803_COMMAND_FLAG_STREAM is set) or as direct command (flag E1803_COMMAND_FLAG_DIRECT is set). Stream-commands are always executed in the order they are sent to the controller and the controller always waits until one motion has been finished before any other command in this stream is executed. Direct commands are executed immediately, here it is up to the user to wait until a motion operation has ended. This wait-operation is the same as for every other scanner operation: first one has to wait until operation started, next one has to wait until operation has ended. It is mandatory to always wait for both state changes.
All the functions described here require firmware version 6 or newer.

int E1803_motion_set_steps(const unsigned char n, const unsigned int flags,const double steps)

Set the factor which defines the relation between steps (increments) of the used stepper motor and the distance that it travels. This value needs to be specified prior to all other operations in order to allow correct calculation of all distances and speeds as expected by the other functions as described below. For the E1803 motion extension no default value exists, so if no factor is set, motion operations are done with an undefined, random value which may lead to unexpected results.
The E1803 motion extension programming interface always makes use of real distances (mostly in unit mm) and does not expect the calling application to do the conversion from increments to mm.
Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – command flags specifying the type of function call (E1803_COMMAND_FLAG_STREAM or E1803_COMMAND_FLAG_DIRECT) and for which axes the given values have to be applied (E1803_COMMAND_FLAG_AXIS_0, E1803_COMMAND_FLAG_AXIS_1, E1803_COMMAND_FLAG_AXIS_2, E1803_COMMAND_FLAG_AXIS_3)
steps – factor which defines relation between stepper motor steps and travel distance (in unit increments/mm for longitudinal movements or increments/degree for rotational movements)
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_motion_set_limits(const unsigned char n,const unsigned int flags,const double llimit,const double hlimit,const unsigned double in_slimit)

Set motion limits for axis operations. When any follow-up command tries to set values beyond these limits, these values are clipped to the allowed range set with this function.

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()

flags – command flags specifying the type of function call (E1803_COMMAND_FLAG_STREAM or E1803_COMMAND_FLAG_DIRECT) and for which axes the given values have to be applied (E1803_COMMAND_FLAG_AXIS_0, E1803_COMMAND_FLAG_AXIS_1, E1803_COMMAND_FLAG_AXIS_2, E1803_COMMAND_FLAG_AXIS_3)
llimit – lower motion limit (in unit mm or degrees)
hlimit – upper motion limit (in unit mm or degrees)
in_slimit – speed limit (in unit mm/sec or degrees/sec)
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_motion_set_accel(const unsigned char n,const unsigned int flags,const double accel)

Set the acceleration to be used for start and stop for all motion operations and for the specified axes.
Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – command flags specifying the type of function call (E1803_COMMAND_FLAG_STREAM or E1803_COMMAND_FLAG_DIRECT) and for which axes the given values have to be applied (E1803_COMMAND_FLAG_AXIS_0, E1803_COMMAND_FLAG_AXIS_1, E1803_COMMAND_FLAG_AXIS_2, E1803_COMMAND_FLAG_AXIS_3)
accel – acceleration (in unit mm/sec^2 or degrees/sec^2)
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_motion_set_speed(const unsigned char n,const unsigned int flags,double speed)

Set the speed for the next motion operations and for the specified axes. Since all motions are combined movements where all axes start and stop at the same time, the speed value given here is some kind of recommendation which may not be used at the next motion operation. Here following rules apply:

- a speed value given here is never exceeded
- when only one axis is moved by a motion operation E1803_motion_move_abs() or E1803_motion_move_rel() at the same time, the speed value given here is used for this movement
- when more than one axis is moved by a motion operation E1803_motion_move_abs() or E1803_motion_move_rel() at the same time, the controller calculates speeds for all axes which ensure they all start and stop their movements at the same time

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – command flags specifying the type of function call (E1803_COMMAND_FLAG_STREAM or E1803_COMMAND_FLAG_DIRECT) and for which axes the given values have to be applied (E1803_COMMAND_FLAG_AXIS_0, E1803_COMMAND_FLAG_AXIS_1, E1803_COMMAND_FLAG_AXIS_2, E1803_COMMAND_FLAG_AXIS_3)
speed – motion speed (in unit mm/sec or degrees/sec)
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_motion_move_abs(const unsigned char n,const unsigned int flags,const double pos0,const double pos1,const double pos2,const double pos3)

Start a motion operation to the given absolute positions using at maximum the speeds specified with E1803_motion_set_speed().

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – command flags specifying the type of function call (E1803_COMMAND_FLAG_STREAM or E1803_COMMAND_FLAG_DIRECT) and for which axes the given values have to be applied (E1803_COMMAND_FLAG_AXIS_0, E1803_COMMAND_FLAG_AXIS_1, E1803_COMMAND_FLAG_AXIS_2, E1803_COMMAND_FLAG_AXIS_3); the additional flag E1803_COMMAND_FLAG_DONOTWAIT can be set to perform a movement parallel to other operations, for details please refer to description of function E1803_motion_stream_wait() below

pos0 – absolute motion position for axis 0 (in unit mm or dregrees); this value is used only when flag E1803_COMMAND_FLAG_AXIS_0 is set
pos1 – absolute motion position for axis 1 (in unit mm or degrees); this value is used only when flag E1803_COMMAND_FLAG_AXIS_1 is set
pos2 – absolute motion position for axis 2 (in unit mm or degrees); this value is used only when flag E1803_COMMAND_FLAG_AXIS_2 is set
pos3 – absolute motion position for axis 3 (in unit mm or degrees); this value is used only when flag E1803_COMMAND_FLAG_AXIS_3 is set
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_motion_move_rel(const unsigned char n,const unsigned int flags,const double pos0,const double pos1,const double pos2,const double pos3)

Start a motion operation which changes the current axis position by the value specified here and by using at maximum the speeds specified with E1803_motion_set_speed().
Please note: in case of a direct operation it is mandatory to wait for the end of all previous motion operations before this function is called. Elsewhere the real current axis position is not known and a relative movement is not possible.

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
flags – command flags specifying the type of function call (E1803_COMMAND_FLAG_STREAM or E1803_COMMAND_FLAG_DIRECT) and for which axes the given values have to be applied (E1803_COMMAND_FLAG_AXIS_0, E1803_COMMAND_FLAG_AXIS_1, E1803_COMMAND_FLAG_AXIS_2, E1803_COMMAND_FLAG_AXIS_3); the additional flag E1803_COMMAND_FLAG_DONOTWAIT can be set to perform a movement parallel to other operations, for details please refer to description of function E1803_motion_stream_wait() below
pos0 – change the motion position for axis 0 (in unit mm or degrees) by the value given here; this value is used only when flag E1803_COMMAND_FLAG_AXIS_0 is set
pos1 – change the motion position for axis 1 (in unit mm or degrees) by the value given here; this value is used only when flag E1803_COMMAND_FLAG_AXIS_1 is set
pos2 – change the motion position for axis 2 (in unit mm or degrees) by the value given here; this value is used only when flag E1803_COMMAND_FLAG_AXIS_2 is set
pos3 – change the motion position for axis 3 (in unit mm or degrees) by the value given here; this value is used only when flag E1803_COMMAND_FLAG_AXIS_3 is set
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_motion_stream_wait(const unsigned char n)

The two motion functions E1803_motion_move_abs() and E1803_motion_move_rel() described above can be called with the command flag E1803_COMMAND_FLAG_STREAM set to ensure sequential operation within the normal stream of commands. In this mode processing of further commands is halted until the related motion operation has finished. Since the motion extension board uses an own micro-controller, here also parallel operations are possible: when for these motion functions the flag E1803_COMMAND_FLAG_DONOTWAIT is set together with E1803_COMMAND_FLAG_STREAM, operation does not wait but continues to process other data in stream. In such a case motion is performed in parallel to these operations done on the main laser controller.
This is true only for non-motion operations, before the next motion operation is called, the application has to wait until the previous one has been finished. Within a stream this has to be done by calling function E1803_motion_stream_wait(), it re-synchronises the stream with the parallel motion operation. So the rule is: as long as a motion operation was started with the combined flags E1803_COMMAND_FLAG_STREAM|E1803_COMMAND_FLAG_DONOTWAIT, a following call to E1803_motion_stream_wait() is mandatory before any other motion-function is used.

Parameters:
n – the 1-based board instance number as returned by E1803_set_connection()
Return: E1803_OK or an E1803_ERROR_-return code in case of an error

int E1803_motion_stop(const unsigned char n)

Stop a currently running motion operation immediately. Since axes always perform a combined movement where all axes start and stop at the same time, the stop function always affects all axes which are moving. When E1803_stop_execution() is called instead, not only axis movements but also all other operations are stopped.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

int E1803_motion_get_pos(const unsigned char n,const unsigned char axisNum,double *pos)

Retrieves the current position of an axis. This command is always a direct command retrieving the current axis position. Thus it does not make use of command-flags.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`axisNum` – the number of the axis (but not the axis-flags!) in range 0..3 where the current position has to be retrieved for
`pos` – pointer to a variable where the current position (in unit mm) of the axis with the number specified in `axisNum` has to be stored into; when the function does not return E1803_OK, this value is undefined and can't be used
Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

int E1803_motion_reference(const unsigned char n,const unsigned int flags,const unsigned int mode,const double leaveDist,double speedStep0,double speedStep1)

Starts a referencing operation (=homing sequence) to have a defined position for the axis. The referencing sequence consists of following steps:

- move to reference switch (connected to reference-input) with first referencing speed `speedStep0`
- leave the reference switch by the given distance `leaveDist`
- move to reference switch (connected to reference-input) with second referencing speed `speedStep1`
- set the position of the referenced axis to -1 – this value can be used to check if referencing was successful or not, when `E1803_motion_get_pos()` returns a different value than -1 for the referenced axis, something went wrong and referencing failed

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`flags` – command flags specifying the type of function call (`E1803_COMMAND_FLAG_STREAM` or `E1803_COMMAND_FLAG_DIRECT`) and for which axes the given values have to be applied (`E1803_COMMAND_FLAG_AXIS_0`, `E1803_COMMAND_FLAG_AXIS_1`, `E1803_COMMAND_FLAG_AXIS_2`, `E1803_COMMAND_FLAG_AXIS_3`)
`mode` – specifies how referencing has to be done exactly, here a bunch of OR-concatenated flags can be handed over: one of the flags `E1803_MOTION_REFSTEP_N` (to search for the reference input in negative direction) or `E1803_MOTION_REFSTEP_P` (to search for the reference input in positive direction) which optionally can be combined with flag `E1803_MOTION_REFSTEP_INV_SWITCH` to have inverted logic on the reference input
`leaveDist` – distance (in unit mm or degrees) to move off the reference switch after the switch was found for the first time
`speedStep0` – referencing speed (in unit mm/sec or degrees/sec) to find the reference switch for the first time (this value can be larger than `speedStep1` but should be small enough to not to overrun the switch)
`speedStep1` – referencing speed (in unit mm/sec or degrees/sec) to find the reference switch for the second time (this value should be smaller than `speedStep0` and is responsible for the accuracy of the referenced position)
Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

int E1803_motion_set_pos(const unsigned char n,const unsigned int flags,const double pos)

This function does not cause any movement but resets the current axis position(s) to a new value. It can be used e.g. after successful referencing to set the initial positions to some own values. All following movement operations then are done in respect to the position values given here.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`flags` – command flags specifying the type of function call (`E1803_COMMAND_FLAG_STREAM` or `E1803_COMMAND_FLAG_DIRECT`) and for which axes the given values have to be applied (`E1803_COMMAND_FLAG_AXIS_0`, `E1803_COMMAND_FLAG_AXIS_1`, `E1803_COMMAND_FLAG_AXIS_2`, `E1803_COMMAND_FLAG_AXIS_3`)
`pos` – the new position value to be set for the specified axis/axes (in unit mm or degrees)
Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

14.1.7 PID control loop functions

The E1803D provides three integrated PID control loops which can be used for different control tasks and run fully parallel and autonomously to every marking function. Every of these three PID control loops makes use of an input parameter of one of the analogue inputs AIn0, AIn1 and AIn2 which e.g. are provided by the Intelli-IO Extension Board. These control loops can be initialised and used by the following functions:

int E1803_pid_init(const unsigned char n,const unsigned int flags,const unsigned char ctrl,const double p,const double i,const double d,const double s,const unsigned short setpoint)

Initialises and starts a control loop or halts and stops an already running control loop (dependent on value of parameter `flags`). When a control loop is running, the current analogue input value from related input AIn0 (used by PID0), AIn1 (used by PID1) or AIn2 (used by PID2) is read every 01 seconds, the related PID control loop calculations are done and depending on the result the related output is changed.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`ctrl` – a value in range 0..2 specifying the PID control loop 0..2 (which directly correspond to analogue inputs AIn0..AIn2) where the new setpoint has to be specified for
`flags` – specifies the operational mode and output port to be used for the selected PID control loop. When `flags` is set to 0, the related control loop (specified by parameter ctrl) is turned off completely. To enable a control loop, following values can be set for this parameter:

- - one of the flags `E1803_COMMAND_FLAG_PID_OUT_AOUT0`, `E1803_COMMAND_FLAG_PID_OUT_AOUT1`, `E1803_COMMAND_FLAG_PID_OUT_DOUT0`, `E1803_COMMAND_FLAG_PID_OUT_DOUT1`, `E1803_COMMAND_FLAG_PID_OUT_DOUT2`, `E1803_COMMAND_FLAG_PID_OUT_DOUT3`, `E1803_COMMAND_FLAG_PID_OUT_DOUT4`, `E1803_COMMAND_FLAG_PID_OUT_DOUT5`, `E1803_COMMAND_FLAG_PID_OUT_DOUT6`, `E1803_COMMAND_FLAG_PID_OUT_DOUT7` which specify which of the analogue outputs AOut0 or AOut1 or which of the digital outputs DOut0..DOut7 has to be used as the PID control output (please note: digital outputs should be used only with simple, slow environments, elsewhere analogue outputs are to be preferred)
- optionally combined (=OR-concatenated) with `E1803_COMMAND_FLAG_PID_OUT_INVERT` when the behaviour of the related output has to be inverted
- in case of `E1803_COMMAND_FLAG_PID_OUT_AOUT0` or `E1803_COMMAND_FLAG_PID_OUT_AOUT1` optionally combined (=OR-concatenated) with `E1803_COMMAND_FLAG_PID_OUT_POSITIVE`, when this flag is not set, the neutral middle point of the control output is at 5V, when the input value is smaller than the setpoint, the output voltage is regulated below of 5V, when the input value is bigger than the setpoint, the output voltage is regulated ablove these 5V; comparing to this the middle point

of the control output is at 0V when flag `E1803_COMMAND_FLAG_PID_OUT_POSITIVE` is set and the output voltage is regulated to values >0V only when measured input value is bigger than the setpoint
`ctrl` – a value in range 0..2 specifying the PID control loop 0..2 (which directly correspond to analogue inputs AIn0..AIn2) where the new setpoint has to be specified for
`p` – proportional part of the PID control loop, has to be adjusted according to the specific characteristics of the environment to be controlled
`i` – integral part of the PID control loop, has to be adjusted according to the specific characteristics of the environment to be controlled
`d` – differential part of the PID control loop, has to be adjusted according to the specific characteristics of the environment to be controlled
`s` – scale value for the PID control loop, this parameter normally can be set to 1.0 and is required only for a few very specific cases
`setpoint` – the target value in range 0..65535
Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_pid_set(const unsigned char n,const unsigned char ctrl,const unsigned short setpoint)`

Specify a new or modified setpoint to let the PID control loop work with. The setpoint is the target value which has to be reached and kept. After initialisation this function can be used several times in order to specify a new setpoint.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`ctrl` – a value in range 0..2 specifying the PID control loop 0..2 (which directly correspond to analogue inputs AIn0..AIn2) where the new setpoint has to be specified for
`setpoint` – the new target value in range 0..65535
Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_pid_halt_range(const unsigned char n,const unsigned char ctrl,const unsigned short lowlimit,const unsigned short highlimit)`

This function can be used to halt the current marking operation when the measured input values are out of a valid range. In this case marking is stopped the next time the laser is turned of and it is automatically continued as soon as the measured input values are back within the range specified with this function. Marking is halted when the input value is smaller than `lowlimit` or bigger than `highlimit`. When `lowlimit` is set to 0 and `highlimit` is set to 65535, this function is disabled and the PID control loop does not influence the marking operation.

Parameters:
`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`ctrl` – a value in range 0..2 specifying the PID control loop 0..2 (which directly correspond to analogue inputs AIn0..AIn2) where the new setpoint has to be specified for
`lowlimit` – lower limit to halt the marking operation when input value is below of it
`highlimit` – upper limit to halt the marking operation when input value is above of it
Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

14.1.8 Miscellaneous functions

`int E1803_write(unsigned char n,unsigned int flags,unsigned int value)`

Writes some specific data to outputs at E1803D controller. Here `flags` decides which output to use and `value` specifies what has to be written to this output. Additionally `flags` decides weather this is a stream-command (means it is executed at a point in stream that is relative to the other stream commands) or a direct command (means it is executed immediately on calling).

Parameters:

`n` – the 1-based board instance number as returned by `E1803_set_connection()`
`flags` – handling flags specifying the behaviour of this command, when `E1803_COMMAND_FLAG_STREAM` is set, it is used as stream command, `E1803_COMMAND_FLAG_DIRECT` specifies to execute it immediately and independent on current stream and execution state. Here exactly one of these flags can be used, it is not allowed to OR-concatenate them. Additionally exactly one of the following flags has to be set to specify which output need to be used to send the `value` to, this flag has to be OR-concatenated with one of the previously described ones:
`E1803_COMMAND_FLAG_WRITE_LP8MO` – set or unset MO-output of laser connector to a value of 1 or 0
`E1803_COMMAND_FLAG_WRITE_LP8LATCH` – set or unset latch-output of laser connector to a value of 1 or 0
`E1803_COMMAND_FLAG_WRITE_LASERGATE` – set or unset LaserGate-output to a value of 1 or 0, this functions should be used with jump or mark operations only since every switch from jump to mark (or vice versa) still sets the LaserGate output automatically and therefore would overwrite own values set with this function
`value` – the value to be sent to the output specified by flags
Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

14.1.9 Writing of stand-alone data

Using E1803D Easy Programming Interface it is also possible to write stand-alone data which are not marked immediately but are stored either locally or on scanner controller's micro-SD-card. In this mode sending of vector data, scanner- and laser parameters looks exactly the same as for direct operation mode where data are marked immediately. The difference can be found in initialisation (which tells the software to not to mark these data but to store them for later use) and when dynamic data are created.

E1803D supports two types of writing of stand-alone data:

- sending them to the controller via Ethernet or USB connection where they are written to micro-SD-card and
- writing one or more files to the local file system which later have to be copied to the micro-SD-card of the controller manually.

The general procedure for sending stand-alone data to the controller's micro-SD-card has to look as follows:

1. The controller needs to be in idle-state, means it should not mark and should not have loaded an already existing .EPR file. This can be ensured by calling stand-alone command `clepr` with a filename for a file that does not exists on micro-SD-card. For more details please refer to “12.2 Stand-Alone Control Commands“
2. Configure the connection to E1803D controller by calling `E1803_set_connection()`, the returned board instance number has to be used for all following function calls.
3. Enable stand-alone write mode and specify the filename of the .EPR file to be created on micro-SD-card by calling `E1803_set_filepath()` with mode `E1803_FILEMODE_SEND`.
4. Send all laser- and scanner-parameters as well as vector data as usual.
5. Optionally: send information about dynamic contents of the .EPR file to be created by calling `E1803_dynamic_data()` optionally followed by some vector data followed by an other call to function `E1803_dynamic_data()` which ends this section of dynamic data (please refer function description below for details).
6. Wait until `E1803_get_card_state2()` returns “busy”
7. Wait until `E1803_get_card_state2()` returns “idle” or an error
8. End data transmission and finish created file by calling `E1803_close()`.

The general procedure for writing stand-alone data to the local filesystem has to look as follows:

1. Since writing of local data does not require a working connection to the controller card, it does not need to be configured and the special board instance number 0 has to be used for all following function calls.
2. Enable stand-alone write mode and specify the filename of the .EPR file to be created by calling `E1803_set_filepath()` with mode `E1803_FILEMODE_LOCAL`.
3. Send all laser- and scanner-parameters as well as vector data as usual.

4. Optionally: send information about dynamic contents of the .EPR file to be created by calling `E1803_dynamic_data()` optionally followed by some vector data followed by an other call to function `E1803_dynamic_data()` which ends this section of dynamic data (please refer function description below for details).
5. End data transmission and finish created file by calling `E1803_close()`.

The functions which are specific to writing of stand-alone data have to be used as follows:

`int E1803_set_filepath(unsigned char n,const char *fname,unsigned int mode)`

This function enables operation mode where all following data are not marked immediately but written into an .EPR stand-alone file. This mode stays active until next call of `E1803_close()`. It has to be called prior to `E1803_load_correction()`. Valid parameters and their meaning depends on the usage scenario:

- when sending stand-alone data to a connected controller which writes the .EPR file to the micro-SD-card directly:
  `n` – the 1-based board instance number as returned by `E1803_set_connection()`
  `fname` – name of the file as it has to appear on micro-SD-card of the controller in style "0:/filename.epr" where "0:/" is a fixed prefix specifying the micro-SD-card, "filename" is a free to choose name with recommended 8 characters at max and ".epr" is a fixed, mandatory file extension specifying an E1803 stand-alone file
  `mode` – set to `E1803_FILEMODE_SEND` to specify the data have to be sent to the controller
- when writing stand-alone data to the local filesystem (no controller card directly involved):
  `n` – board instance number, has to be set to 0 (as well as for all other function calls in this mode)
  `fname` – name of the file to be written, this has to be a valid path to a location on a local filesystem which is writable and needs to have file extension ".epr"
  `mode` – set to `E1803_FILEMODE_LOCAL` to specify the data have to be written locally

Return: `E1803_OK` or an `E1803_ERROR_`-return code in case of an error

`int E1803_dynamic_data(unsigned char n,struct oapc_bin_struct_dyn_data *dynData)`

This function can be used to write dynamic data such as texts, serial numbers, barcodes which later can be changed during operation in stand-alone mode.
This function always has to be called in fixed sequences:

1. jump to the start position of the dynamic element by calling `E1803_jump_abs()`
2. first call of `E1803_dynamic_data(n,dynData)` with a valid `dynData` parameter describing the dynamic content and its capabilities
3. optionally and dependent on type of dynamic data that have to be created: some vector data which belong to the dynamic content and are required to build it up
4. second call of `E1803_dynamic_data(n,NULL)` with NULL handed over for parameter `dynData` to finish this element

A stand-alone file can contain up to ten dynamic data elements. So this function can be called up to ten times to create a new element on each call.

When this function is called, beside the .EPR-file an additional .DAT file is created which contains some specific data. During operation in stand-alone mode an other file with the same name and with extension .SER may be created which contains counting information of an included serial number. All these files belong together and deleting one or more of them may lead to unexpected results. When writing the data to local filesystem it also has to be ensured both, the .EPR and the .DAT fiel are copied to the controller later.

The structure `oapc_bin_struct_dyn_data` is defined in file "oapc_libio.h" which is part of the OpenSDK. The general usage is described in OpenSDK manual, both are available for download at http://openapc.com/download.php.

For E1803 scanner controller card following specific parameters and features of this structure can to be used:

Independent from what kind of dynamic element has to be created, following members of structure `oapc_bin_struct_dyn_data` **always have to be filled** with data:

UID – and unique identifier which can be created out of a plain, human readable text which should be unique too and later can be used to access this specific element via stand-alone control commands; this identifier has to created out of the 8 bit ASCII character using following CRC-function:

```
#define POLY 0x82f63b78

/* CRC-32 (Ethernet, ZIP, etc.) polynomial in reversed bit order. */
unsigned int crc32b(const char *buf)
{
   int          k;
   unsigned int crc=0xFFFFFFFF;
   size_t       len=strlen(buf);

   while (len--)
   {
      crc^=*buf++;
      for (k=0; k<8; k++)
       crc=crc&1 ? (crc>>1)^POLY : crc>>1;
   }
   return ~crc;
}
```

`uScaleX` – scaling factor in X-direction in unit 1/1000000
`uScaleY` – scaling factor in Y-direction in unit 1/1000000
`res1a`, `res1b`, `res2`, `res3`, `res4`, `res5`, `res6`, `res7` – these members are reserved for later use and all have to be set to 0

Every dynamic element can be a **serial number**. In such a case the serial number part of structure `oapc_bin_struct_dyn_data` has to be filled with data:

`fmtString` – an ASCII text with a maximum length of `DYN_DATA_MAX_STRING_LENGTH` describing the format of the serial number/date/time in the dynamic element, here the same notation has to be used as it is known from the serial number input element of BeamConstruct (please refer to the related manual)
`snBeatCount` – specifies how much numbers of mark operations have to elapse before the serial number has to be incremented, here a value of 1 has to be given to increment on every operation
`snBeatOffset` – specifies a counting offset for the beat count parameter
`snIncrement` – specifies the step width by which a serial number has to be incremented
`snNumericBase` – the numeric base of the serial numbers to be displayed, default is 10 for decimal numbers
`snResetAt` – the value at which the serial number has to be reset to it's initial value; set to a timestamp (in unit day of week/date/seconds) when it has to be reset at a given time
`snFlags` – a set of OR-concatenated flags which further specifies handling of the serial number:
0x0002 – reset the serial number at a specific counting value
0x0004 – reset the serial number at a specific day of the week
0x0008 – reset the serial number at a specific date
0x0010 – reset the serial number at a specific time of the day
`snStartValue` – the initial- and reset-to-value of the serial number
`snMinDigits` – the minimum number of digits the serial number has to consist of
`timeOffset` – a static offset (in unit seconds) to be added to the time-part of the current element

Dynamic **text elements** additionally need to fill following parts of the same structure `oapc_bin_struct_dyn_data`:

`fmtString` – an ASCII text with a maximum length of `DYN_DATA_MAX_STRING_LENGTH` which contains the text to be shown and which can be changed by appropriate stand-alone commands later; when used in combination with serial number data, here a format-string has to be given as described above
`type` – a number which specifies the font to be used for creating the dynamic texts, here one of following values can be used:
`0x01000000` – use “Rect Single” laser font

0x02000000 – use “Rect Double” laser font
0x03000000 – use “Roman Simple” laser font
0x04000000 – use “Roman Double” laser font
0x05000000 – use “Script Simple” laser font
0x06000000 – use “Script Double” laser font
0x07000000 – use “Script Complex” laser font
0x08000000 – use “Times Simple” laser font
0x09000000 – use “Times Bold” laser font
0x0A000000 – use “Times Italic” laser font
0x0B000000 – use “Times Italic Bold” laser font

flags – some OR-concatenated flags which specify orientation, alignment and style of the text to be generated, here no two flags of same type are allowed to be combined which would conflict with each other:
0x00000000 – orient text left to right
0x00010000 – orient text right to left
0x00020000 – orient text top to bottom
0x00030000 – orient text bottom to top

0x00000000 – horizontally align to the left
0x00000100 – centre-align horizontally
0x00000200 – horizontally align to the right

0x00000001 – style fixed char-size – all characters are forced to have same distance

param1 – kerning value in unit 1/1000%
param2 – reserved for future use, set to 0
param3 – spacing in unit 1/1000%

Dynamic **DataMatrix barcode elements** require vector data to be sent between two calls of function E1803_dynamic_data(), these vector data describe the pattern which has to be marked to create one single element (means square) of the DataMatrix barcode. Such an element needs to incorporate all that is needed including laser- and scannerdata as well as vector data for outline and possible hatches. During stand-alone operation the barcode itself is created by combining these single elements at these positions, where a bit (=square) has to be set).
Additionally following data of the structure oapc_bin_struct_dyn_data need to be filled for this type of element:

fmtString – an ASCII text with a maximum length of DYN_DATA_MAX_STRING_LENGTH which contains the text to be encoded as DataMatrix barcode and which can be changed by appropriate stand-alone commands later; when used in combination with serial number data, here a format-string has to be given as described above
type – set to 71 for DataMatrix barcode
flags – some OR-concatenated flags which further specify how the barcode has to be created, currently only one flag is supported:
0x0001 – create a square-shaped DataMatrix barcode instead of a rectangular one
param1 – set to 0
param2 – set to -1
param3 – specifies the size to be generated (in range 2..30) and implicitly the error correction level
quietZone – zone the barcode has to be surrounded with, the value given here is the multiple of the width of a single token multiplied with 1000

14.1.9.1 Example

Following a (simplified) example in some pseudo-code is given which demonstrates the correct usage of the programming interface to write stand-alone data. The laser- and scanner-parameters are dropped in this example since they are not specific to this operation mode and always have to be set.

Example: A serial number in format “000/hh/mm” where “000” is a continuously increased number, “hh” is the current hour and “mm” is the current minute has to be encoded into a DataMatrix barcode which has a size of 25x25 mm and is positioned at -30x30 mm within a 100x100 mm working area that itself is aligned to coordinates -50,50

1. not shown here: initialisation of libE1803 (with evaluation of parameter `boardIdx`), sending of default scanner and laser data as usual
2. `E1803_jump_abs(boardIdx, -20132659, 20132659, 0)` // jump to the starting position of the DataMatrix barcode to be created
3. `E1803_dynamic_data(boardIdx, dynData)` // initiate the dynamic data sequence, here the members of dynData are set to following values:
   ```
   UID           = 2340633892 – CRC-value of element name “Barcode 1”
   fmtString     = "$S/%I/%M" – display serial number, hour and minute
   type          = 71 – DataMatrix barcode
   flags         = 1 – barcode forced to square
   param2        = 4294967295
   param3        = 2
   uScaleX       = 1029654
   uScaleY       = 1029654
   snIncrement   = 1
   snNumericBase = 10
   snMinDigits   = 3
   ```
   all other values are set to 0
4. ```
   E1803_jump_abs(boardIdx, 0, 0, 0)
   E1803_mark_abs(boardIdx, 1197222, 0, 0)
   E1803_mark_abs(boardIdx, 1197222, -1197222, 0)
   E1803_mark_abs(boardIdx, 0, -1197222, 0)
   E1803_mark_abs(boardIdx, 0, 0, 0)
   ```
   // draw a single rectangle which describes one DataMatrix cell (in this example only the outline without any hatching is done, hatches would have to be added here too
5. `E1803_dynamic_data(boardIdx, NULL)` // end the sequence of dynamic data
6. `E1803_execute(boardIdx)`
7. Not shown here: waiting for card being busy, waiting for card being idle (which means writing of the Epr file to the microSd card has been finished), closing the connection to the controller

14.1.10 Error Codes

Most of the functions described above can return an error code in case an operation could not be completed successfully for any reason. So when it does not return with `E1803_OK` the error code informs about the reason for failure:

- `E1803_ERROR_INVALID_CARD` – a wrong or illegal card number was specified with function parameter `n`
- `E1803_ERROR_NO_CONNECTION` – a connection to card could not be established
- `E1803_ERROR_NO_MEMORY` – there is not enough memory available on the host
- `E1803_ERROR_UNKNOWN_FW` – card is running an unknown and/or incompatible firmware version
- `E1803_ERROR_TRANSMISSION` – data transmission to card failed
- `E1803_ERROR_FILEOPEN` – opening of a file failed
- `E1803_ERROR_FILEWRITE` – writing of data into a file failed
- `E1803_ERROR_INVALID_DATA` – data or parameters handed over to a function are invalid, out of range or illegal in current context
- `E1803_ERROR_UNKNOWN_BOARD` – trying to access a controller board that is not a motion controller
- `E1803_ERROR_FILENAME` – a file name handed over to a function was illegal, it is either too long, has an illegal or too long file extension, comes with too much sub-directories or contains illegal characters
- `E1803_ERROR_NOT_SUPPORTED` – the requested feature or function is not supported by the current firmware version
- `E1803_ERROR` – an other, unspecified error occurred
   ```

14.2 RTC4 Compatibility Functions

Beside the easy programming interface described above, a bunch of additional functions is provided that are compatible to the ones known from RTC4 scanner controller card. So to use E1803D scanner card with existing code that supports the RTC4 scanner controller, following few steps have to be done:

- insert a call to `E1803_set_connection()` into existing code as very first in order to specify the communication connection for E1803D card (this is the only exception where an E1803D Easy Interface Function should be used together with an RTC4 Compatibility Function)
- recompile the existing RTC4-application so that it uses e1803inter.DLL/libe1803inter.so instead of RTC4DLL.dll/libslrtc4.so

In case E1803D card has to be operated with default connection settings, no recompilation is necessary, here e1803inter.dll/libe1803inter.so just has to be renamed to RTC4DLL.dll/libslrtc4.so.

Since most relevant RTC4 functions are already provided, majority of existing applications should work now without any further modifications. Due to the completely different concept of E1803D scanner controller, there are some differences to the original RTC4 programming interface which should be checked in case of some errors:

- all RTC4 functions that exist as list- and non-list-commands are treated like a list command
- list switch commands are ignored since E1803D does not make use of separated lists internally
- output of already sent marking data is started on calls to `n_execute_list()`,`execute_list()`, `n_set_end_of_list()` or `set_end_of_list()`
- some functions are not implemented or always return a default value (please refer below for a list of not implemented RTC4 functions)

Following functions are specific to RTC4 hardware or do not make sense when E1803D scanner card is used and therefore aren't supported:

```
n_load_z_table()
load_z_table()
n_set_defocus_list()
set_defocus_list()
n_set_offset_list()
set_offset_list()
n_laser_on_list()
laser_on_list()
n_set_list_jump()
set_list_jump()
n_set_input_pointer()
set_input_pointer()
n_list_call()
list_call()
n_list_return()
list_return()
n_z_out_list()
z_out_list()
n_timed_jump_abs()
timed_jump_abs()
n_timed_mark_abs()
timed_mark_abs()
n_timed_jump_rel()
timed_jump_rel()
n_timed_mark_rel()
timed_mark_rel()
n_set_fly_rot()
set_fly_rot()
n_fly_return()
fly_return()
n_calculate_fly()
calculate_fly()
n_select_cor_table_list()
select_cor_table_list()
n_set_wait()
```

```
set_wait()
n_simulate_ext_start()
simulate_ext_start()
n_set_pixel_line()
set_pixel_line()
n_set_pixel()
set_pixel()
n_set_extstartpos_list()
set_extstartpos_list()
n_laser_signal_on_list()
laser_signal_on_list()
n_laser_signal_off_list()
laser_signal_off_list()
n_set_io_cond_list()
set_io_cond_list()
n_clear_io_cond_list()
clear_io_cond_list()
n_list_jump_cond()
list_jump_cond()
n_list_call_cond()
list_call_cond()
n_save_and_restart_timer()
save_and_restart_timer()
n_set_ext_start_delay_list()
set_ext_start_delay_list()
n_set_trigger()
set_trigger()
n_arc_rel()
arc_rel()
n_arc_abs()
arc_abs()
drilling()
regulation()
flyline()
n_get_input_pointer()
get_input_pointer()
n_get_marking_info()
get_marking_info()
n_auto_change_pos()
auto_change_pos()
aut_change()
n_start_loop()
start_loop()
n_quit_loop()
quit_loop()
n_write_da_2()
write_da_2()
n_set_max_counts()
set_max_counts()
n_set_offset()
set_offset()
n_disable_laser()
disable_laser()
n_enable_laser()
enable_laser()
n_stop_list()
stop_list()
n_restart_list()
restart_list()
n_get_xyz_pos()
get_xyz_pos()
n_get_xy_pos()
get_xy_pos()
```

```
n_select_list()
select_list()
n_z_out()
z_out()
n_laser_signal_on()
laser_signal_on()
n_laser_signal_off()
laser_signal_off()
n_set_delay_mode()
set_delay_mode()
n_set_piso_control()
set_piso_control()
n_select_status()
select_status()
n_get_encoder()
get_encoder()
n_select_cor_table()
select_cor_table()
n_execute_at_pointer()
execute_at_pointer()
n_get_head_status()
get_head_status()
n_simulate_encoder()
simulate_encoder()
n_set_hi()
set_hi()
n_release_wait()
release_wait()
n_get_wait_status()
get_wait_status()
n_set_ext_start_delay()
set_ext_start_delay()
n_home_position()
home_position()
n_set_rot_center()
set_rot_center()
n_read_ad_x()
read_ad_x()
n_read_pixel_ad()
read_pixel_ad()
n_get_z_distance()
get_z_distance()
n_get_time()
get_time()
n_set_defocus()
set_defocus()
n_set_softstart_mode()
set_softstart_mode()
n_set_softstart_level()
set_softstart_level()
n_control_command()
control_command()
load_cor()
load_pro()
n_get_serial_number()
get_serial_number()
n_get_serial_number_32()
get_serial_number_32()
get_hi_data()
n_auto_cal()
auto_cal()
n_get_list_space()
get_list_space()
```

```
teachin()
n_get_value()
get_value()
set_duty_cycle_table()
n_move_to()
move_to()
getmemory()
n_get_waveform()
get_waveform()
n_measurement_status()
measurement_status()
n_load_varpolydelay()
load_varpolydelay()
n_write_da_2_list()
write_da_2_list()
```

14.3 USC1/2 Compatibility Functions (SCI interface)

Beside the easy programming interface described above a bunch of additional functions is provided that are compatible to the ones known from SCI programming interface used for USC1/2 scanner controller card. So to use E1803D scanner card with existing code that supports the USC1 or USC2 scanner controller, following steps have to be done:

- insert a call to `E1803_set_connection()` into existing code as very first in order to specify the communication connection for E1803D card (this is the only exception where an E1803D Easy Interface Function should be used together with an USC1/2 Compatibility Function)
- recompile the existing SCI-application so that it uses e1803inter.DLL instead of sc_optic.dll

In case E1803D card has to be operated with default connection settings, no recompilation is necessary, here e1803inter.dll just has to be renamed to sc_optic.dll.

Since most relevant SCI functions are already provided, majority of existing applications should work now without any further modifications. Due to the different concept of E1803D scanner controller, there are some differences to the original SCI programming interface which should be checked in case of troubles:

- all USC1/2 functions/types that exist as stream- and non-stream-variants are treated like a stream command
- output of already sent marking data is started on call to `ScSCIFlush()` latest
- some functions are not implemented or always return a default value (please refer below of a list of not implemented SCI functions)

Following functions are specific to USC1/2 hardware or do not make sense in relation to E1803D scanner card and therefore aren't supported:

```
long ScSCISetContinuousMode()
long ScSCIGetContinuousMode()
long ScSCIDevicePixelLine()
long ScSCIRasterPixelLine()
long ScSCIRasterStart()
long ScSCIRasterEnd()
long ScSCIGetDeviceName()
long ScSCIGetDeviceCaps()
long ScSCIGetDeviceData()
long ScSCISetDeviceData()
long ScSCIGetExternalTrigger()
long ScSCISetExternalTriggerCount()
long ScSCIGetExternalTriggerCount()
long ScSCISetEnableHead()
long ScSCIGetEnableHead()
long ScSCISetZField()
long ScSCIGetZField()
long ScSCIGetZGain()
long ScSCISetZGain()
long ScSCIGetHomePosition()
```

```
long ScSCIGetZHomePosition()
long ScSCISetHomePosition()
long ScSCISetZHomePosition()
long ScSCIGetZOffset()
long ScSCISetZOffset()
long ScSCISetZWorkingArea()
long ScSCIGetZWorkingArea()
long ScSCIGetHomeJump()
long ScSCISetHomeJump()
long ScSCIMaxExternalTriggerCount()
long ScSCIResetExternalTriggerCount()
long ScSCISetDeviceEnableFlags()
long ScSCIGetDeviceEnableFlags()
long ScSCIGetDevicePath()
long ScSCISetDeviceMiscValueD()
long ScSCIGetDeviceMiscValueD()
long ScSCISetHeadCount()
long ScSCIStreamInfo()
long ScSCIGetSpeed()
long ScSCISetSpeed()
long ScSCIGetStyleSet()
long ScSCISetStyleSet()
long ScSCISetLoopMode()
long ScSCIGetLoopMode()
long ScSCISetLoop()
long ScSCIGetLoop()
long ScSCISetMessageWindow()
long ScSCISetAxisState()
long ScSCIGetAxisState()
long ScSCISaveSettings()
long ScSCILoadSettings()
long ScSCIEditSettings()
long ScSCIUpdateDeviceStyle()
long ScSCIGetInterfaceVersion()
long ScSCIGetDebugMode()
long ScSCISetDebugMode()
long ScSCIGetIdentString()
long ScSCIGetDeviceMapLaserPort()
long ScSCISetDeviceMapLaserPort()
long ScSCIGetUSCInfoLong()
```

APPENDIX A – Wiring between E1803D and IPG YLP Series Type B, B1 and B2 fiber laser

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Signal Name	E1803D Connector	E1803D Connector / Pin	IPG Pin
LP0	Laser signal connector	Pin 1	Pin 1
LP1		Pin 3	Pin 2
LP2		Pin 5	Pin 3
LP3		Pin 7	Pin 4
LP4		Pin 9	Pin 5
LP5		Pin 11	Pin 6
LP6		Pin 13	Pin 7
LP7		Pin 15	Pin 8
MO / Master Oscillator		Pin 8	Pin 18
LP8 Latch		Pin 17	Pin 9
LaserA / Frequency		Pin 22	Pin 20
Laser Gate / Modulation		26 pin connector, pin 26	Pin 19
Alarm, one of DIn0…DIn7	Digital interface connector	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 16
Alarm, one of DIn0…DIn7		Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 21
Pilot Laser, one of DOut0…DOut7		Pin 3, 5, 7, 9, 11, 13, 15 or 17	Pin 22 *)

*) may require additional power driver since some laser variants consume a current at this input which is higher than the maximum output allowed

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.

APPENDIX B – Wiring between E1803D and IPG YLP Series Type E fiber laser

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Variant using E1803D with support for APD index setting via DB-25 serial data interface

Signal Name	E1803D Connector	Connector / Pin	IPG Pin
LP0	Laser signal connector	Pin 1	Pin 1
LP1		Pin 3	Pin 2
LP2		Pin 5	Pin 3
LP3		Pin 7	Pin 4
LP4		Pin 9	Pin 5
LP5		Pin 11	Pin 6
LP6		Pin 13	Pin 7
LP7		Pin 15	Pin 8
MO / Master Oscillator		Pin 8	Pin 18
LP8 Latch		Pin 17	Pin 9
LaserA / Frequency		Pin 22	Pin 20
Laser Gate / Modulation		26 pin connector, pin 26	Pin 19
Alarm, one of DIn0..DIn7	Digital interface connector	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 16
Alarm, one of DIn0..DIn7		Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 21
Pilot Laser, one of DOut0, DOut1, DOut5..DOut7		Pin 3, 5, 13, 15 or 17	Pin 22 [1])
Serial Enable		Pin 7	Pin 24 [2])
Serial Clock		Pin 9	Pin 13 [2])
Serial Data		Pin 11	Pin 10 [2])

[1]) may require additional power driver since some laser variants consume a current at this input which is higher than the maximum output allowed
[2]) serial data transmission requires firmware version 2 or newer

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.

APPENDIX C – Wiring between E1803 and IPG YLR Series laser

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Signal Name	Board	Connector / Pin	IPG Pin
AOut0 or AOut1	Laser signal connector	Pin 12 or pin 14	Pin 12
MO / Master Oscillator		Pin 8	Pin 18
Laser Gate / Modulation		Pin 26	Pin 15
Pilot Laser, one of DOut0..DOut7	Digital interface connector	Pin 3, 5, 7, 9, 11, 13, 15 or 17	Pin 17

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.

APPENDIX D – Wiring between E1803 and IPG YLM Series laser

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Signal Name	Board	Connector / Pin	IPG Pin
AOut0 or AOut2	Laser signal connector	Pin 12 or pin 14	Pin 8 [1])
MO / Master Oscillator		Pin 8	Pin 22
Laser Gate / Modulation		Pin 26	Pin 17
Pilot Laser, one of DOut0..DOut7	Digital interface connector	Pin 3, 5, 7, 9, 11, 13, 15 or 17	Pin 21
Laser ready, one of DIn0..DIn7		Pin 4, 6, 8 ,10, 12, 14, 16 or 18	Pin 19
RS232 serial interface	Serial interface connector	Pins 1, 2 and 3	RS232 interface

[1]) The E1803D analogue output provides signals in range 0..10 V while the IPG YLM laser input expects signals in range 0..4 V. To avoid hardware damage the signal level has to be limited by additional hardware measures, e.g. by a voltage divider

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.

APPENDIX E – Wiring between E1803D and JPT YDFLP series fiber laser (“MOPA”) or IPG YLP Series Type D fiber laser

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Signal Name	E1803D Connector	E1803D Connector / Pin	JPT D-SUB25
LP0	Laser signal connector	Pin 1	Pin 1
LP1 / serial data		Pin 3	Pin 2
LP2 / serial clock		Pin 5	Pin 3
LP3		Pin 7	Pin 4
LP4		Pin 9	Pin 5
LP5		Pin 11	Pin 6
LP6		Pin 13	Pin 7
LP7		Pin 15	Pin 8
MO / Master Oscillator		Pin 8	Pin 18
LaserA / Frequency		Pin 22	Pin 20
Laser Gate / Modulation		26 pin connector, pin 26	Pin 19
LaserB / serial enable		Pin 19	Pin 22 *)
Alarm, one of DIn0…DIn7	Digital interface connector	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 16
Alarm, one of DIn0…DIn7		Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 21
Pilot Laser, one of DOut0…DOut7		Pin 3, 5, 7, 9, 11, 13, 15 or 17	Pin 22 *)

*) for details regarding double-usage of this pin, please refer to the manual of the laser

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.

APPENDIX F – Wiring between E1803D and SPI G4 Pulsed Fibre Laser series

PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Signal Name	E1803D Connector	E1803D Connector / Pin	SPI Pin
LP0	Laser signal connector	Pin 1	Pin 17
LP1		Pin 3	Pin 18
LP2		Pin 5	Pin 19
LP3		Pin 7	Pin 20
LP4		Pin 9	Pin 51
LP5		Pin 11	Pin 52
LP6		Pin 13	Pin 53
LP7		Pin 15	Pin 54
MO / Laser Enable		Pin 8	Pin 7
LP8 Latch		Pin 17	Pin 23
LaserA / Pulse Trigger		Pin 22	Pin 47
AOut0 / Power		Pin 12	Pin 65
AOut1 / Simmer		Pin 14	Pin 64
LaserGate / Modulation		Pin 26	Pin 5
Alarm, one of DIn0…DIn7	Digital interface connector	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 9
Pilot Laser, one of DOut0…DOut7		Pin 3, 5, 7, 9, 11, 13, 15 or 17	Pin 6

In these wiring-schemes no GND-connections are listed, they have to be added in order to get valid and working connections.

APPENDIX G – Wiring between E1803D and Raycus fiber laser

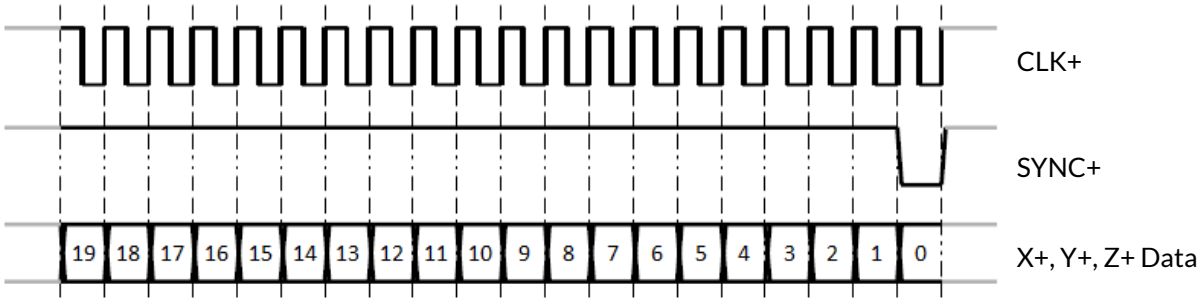
PLEASE NOTE: this wiring scheme is a non-binding policy, it may be incorrect due to changes in connected hardware. So in every case please check this table against specification and wiring documentation of the used laser!

Signal Name	E1803D Connector	E1803D Connector / Pin	Raycus Pin
LP0	Laser signal connector	Pin 1	Pin 1
LP1		Pin 3	Pin 2
LP2		Pin 5	Pin 3
LP3		Pin 7	Pin 4
LP4		Pin 9	Pin 5
LP5		Pin 11	Pin 6
LP6		Pin 13	Pin 7
LP7		Pin 15	Pin 8
MO / Master Oscillator		Pin 8	Pin 18
LaserA / Frequency		Pin 22	Pin 20
Laser Gate / Modulation		26 pin connector, pin 26	Pin 19
Alarm, one of DIn0…DIn7	Digital interface connector	Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 16
Alarm, one of DIn0…DIn7		Pin 4, 6, 8, 10, 12, 14, 16 or 18	Pin 21

In this wiring-scheme no GND-connections are listed, they have to be added in order to get valid and working connections.

APPENDIX H – XY2-100 protocol description

The data submitted at 26 pin or D-SUB25 connector of E1803D are conform to XY2-100 specification:



In standard 16 bit operating mode first three bits are set to 001, then 16 bit position data followed by a parity bit (even parity) are transmitted:

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	D15..D0 position data																Pe

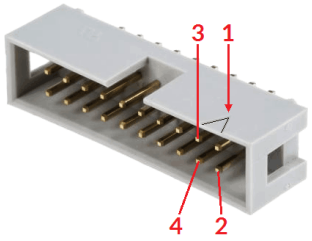
In enhanced XY2-100-E 18 bit operating mode first bit is set to 1, then 18 bit position data followed by a parity bit (odd parity) are transmitted:

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	D17..D0 position data																		Po

To use this mode, the related tune-value has to be set in configuration file (please refer to section “6.5 microSD-Card”)

APPENDIX I – IDC connector pin numbering

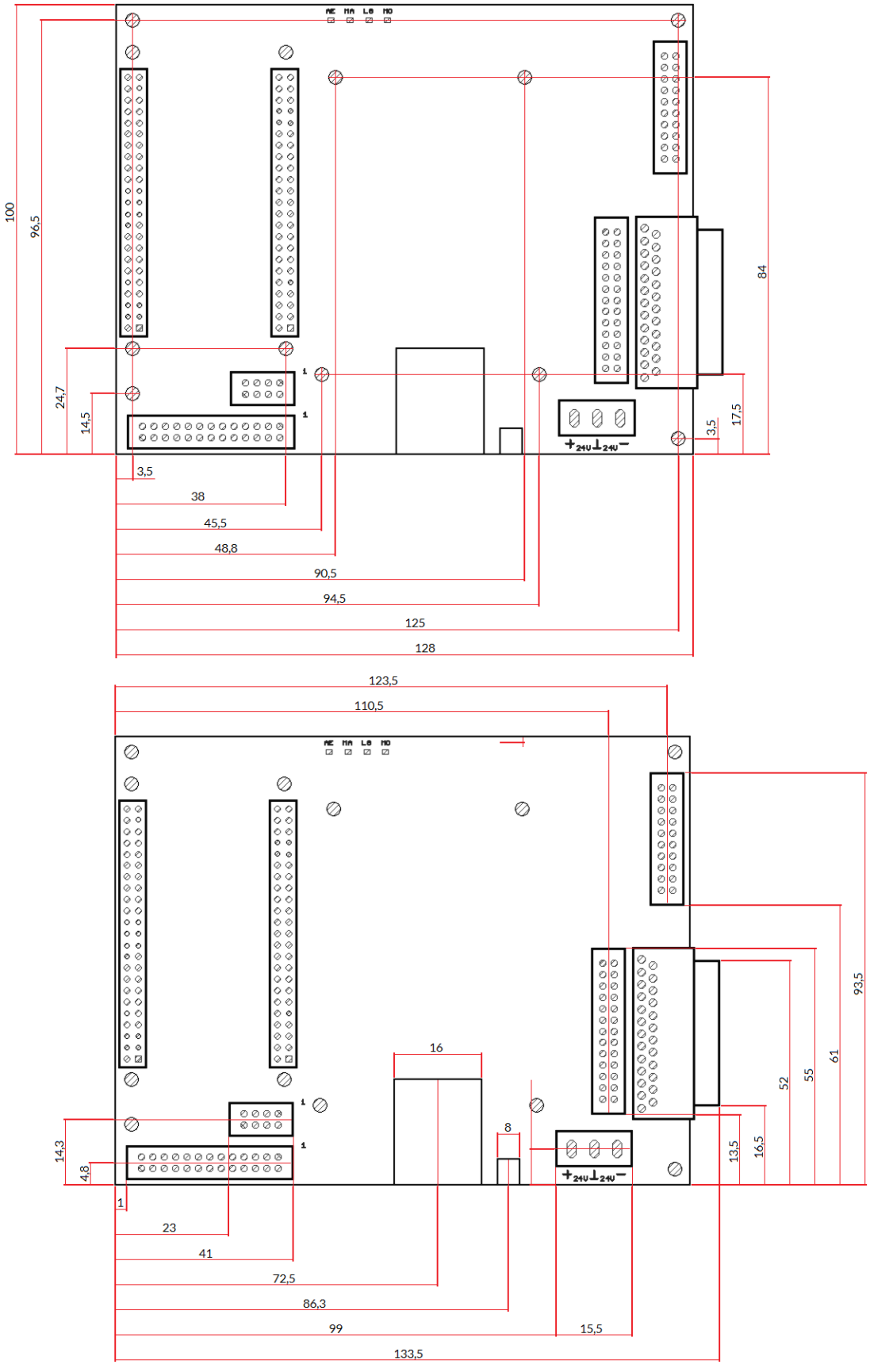
Pin numbering of the IDC connectors (according to pinout-tables shown in hardware description sections above) can be seen in below image:



The first pin is marked by a small arrow in connector. Second pin is below of it, counting continues column-wise.

APPENDIX J – Mechanical Dimensions

Mechanical dimensions, positions of connectors and holes, all units are given in mm:



Index

D

E

F

G

H

I

J

L

M

N

O

P

Q

V

W

X

Y

Z

.