HEIDENHAIN

User's Manual
for Application Development

EIB 8791

External Interface Box
for Connecting
HEIDENHAIN Encoders

11/2018

Target group of this User's Manual

This Manual must be read and followed by every person who performs the following tasks:

- Qualified personnel for application development:
  The qualified personnel for application development have the required technical training, knowledge, and experience, as well as the knowledge of the applicable regulations, to perform the assigned tasks with regard to the given application and to identify and avoid potential risks on their own.

Documentation

The documentation for the EIB 8791 comprises the following documents:

- Operating instructions:
  - Documents required for commissioning, as well as technical specifications.
- User's Manual for Application Development:
  - Description of features on the EIB 8791.
  - Description of the installation and function calls of the driver software.

Remark:
Due to the modular internal structure of the EIB 8791, the term "EIB8" is often used in this User's Manual, particularly with respect to the driver functions and in connection with the basic component of the EIB 8791.

Firmware version

This document describes Firmware version:

- 816477-xx
- 1128592-xx

Change history

Changes from the previous versions are listed in the change history.
The document on change history is on the CD in the subdirectory EIB_8791/doc. Please read this document, particularly the notes on new, changed, or obsolete function calls.

Glossary

Term	Explanation
EIB	External Interface Box: interface electronics for precise position measurement, particularly for inspection stations and multipoint inspection apparatuses, as well as for mobile data acquisition, such as for machine inspection and calibration
SLOT	The EIB 8791 provides eight encoder inputs. The eight inputs are grouped into four slots with two encoder inputs each.
Axis	Designates a measuring axis of the application for which the EIB 8791 is used. For each axis, a connection for an encoder is provided.
Encoder	Position encoder from HEIDENHAIN for measuring the length or angle
Trigger	Asynchronous signal that initiates a position value output
Synchronization	Synchronization of the position value formation
PTM	Position Trigger Marker: signal that marks individual clock pulse edges of the SynClk signal
SynClk	Synchronization Clock: clock signal for high-precision temporal synchronization of the position value formation sampling time
PD	Position data
PDL	Position data link
Online compensation	Automatic, concurrent adjustment of offset, phase, and amplitude of the scanning signals of a position measuring system
Waveform compensation	Compensation-point-based, local compensation of the effect that the waveform distortion of the scanning signals of a position measuring system has on the position value
Trigger event	Edge of a configured trigger, or clock pulse edge of the SynClk signal marked with the PTM
FTP	File Transfer Protocol
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
GPIO	General Purpose Input Output

1 Description of functions

1.1 General information

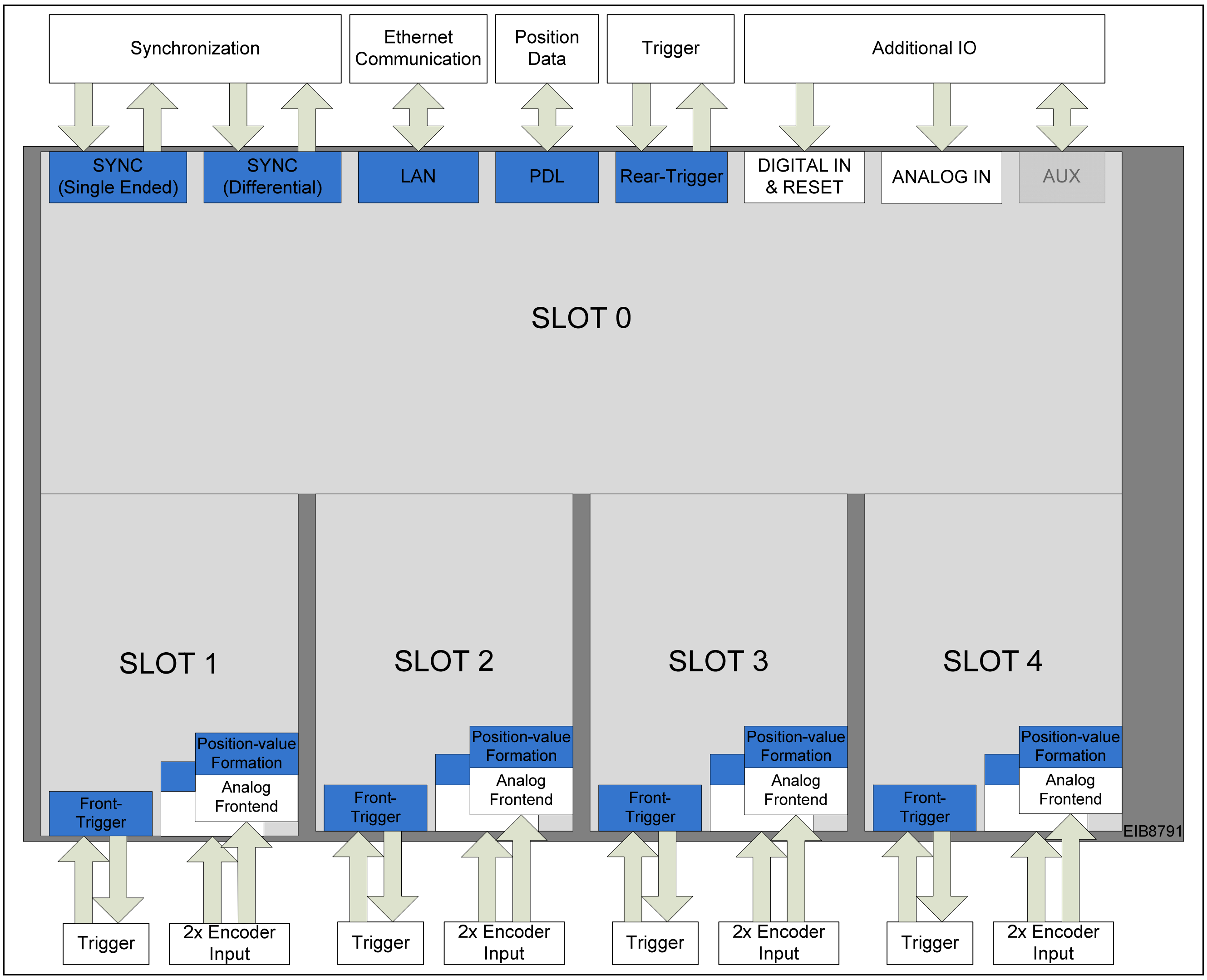
The EIB 8791 is an external interface box for precise position measurement. It is ideal for inspection stations and multipoint inspection apparatuses as well as for manufacturing systems and machines with numerous axes. The EIB 8791 is ideal for applications requiring high-resolution encoder signals and fast measured-value acquisition. Synchronous position value formation in the EIB 8791 has been optimized particularly for highly dynamic machines. Ethernet transmission also enables you to use switches or hubs for connecting more than one EIB 8791. Up to eight HEIDENHAIN encoders with sinusoidal incremental signals (1 V_{PP}) can be connected to the EIB 8791.

The EIB 8791 subdivides the periods of the incremental signals 65 536-fold (16 bits) for generating the measured value. The deviations within one signal period are automatically reduced by adjusting the sinusoidal incremental signals (online compensation[1]). In addition, further signal errors can be reduced via local compensation based on compensation points (waveform compensation[2]). The high internal sampling rate of 50 MHz also makes it possible to digitally filter the position values. In addition to position values, you can also output speed and/or acceleration values.

The integrated measured-value memory enables the EIB 8791 to store up to 125 000 measured values per axis while in "recording" mode. Internal or external triggers can be used for axis-specific storage of the measured values. A standard Ethernet interface (using TCP or UDP communication) or a high-speed synchronous position data link (PDL) is available for data output. This permits direct connection to a PC, laptop, or industrial PC as well as connection to dedicated control hardware.

The method of measured value transmission can be set via the operating mode. For the processing of measured values in a PC, driver software for Windows, Linux, and LabVIEW is included. The driver software enables customers to easily program their own applications. It also contains example programs demonstrating the performance range of the EIB 8791.

Basic circuit diagram:



[1] Automatic, concurrent adjustment of offset, phase, and amplitude of the scanning signals of a position measuring system

[2] Compensation-point-based local compensation of the waveform distortion of the position measuring system scanning signals, affecting the position value

1.2 Configuration of the encoder inputs

A maximum of eight HEIDENHAIN encoders with the following interfaces can be connected to the EIB 8791:

- 1 V_{PP} incremental signals

The power supply to the encoders is provided by the EIB 8791 and is protected by a resettable overcurrent cutoff.

For the specifications, see the *Operating Instructions*.

After power-up, the power supply of the encoders is deactivated. The parameters for operating the encoder input must be configured via initialization. During this process, type-dependent information about the encoder is transferred to an axis:

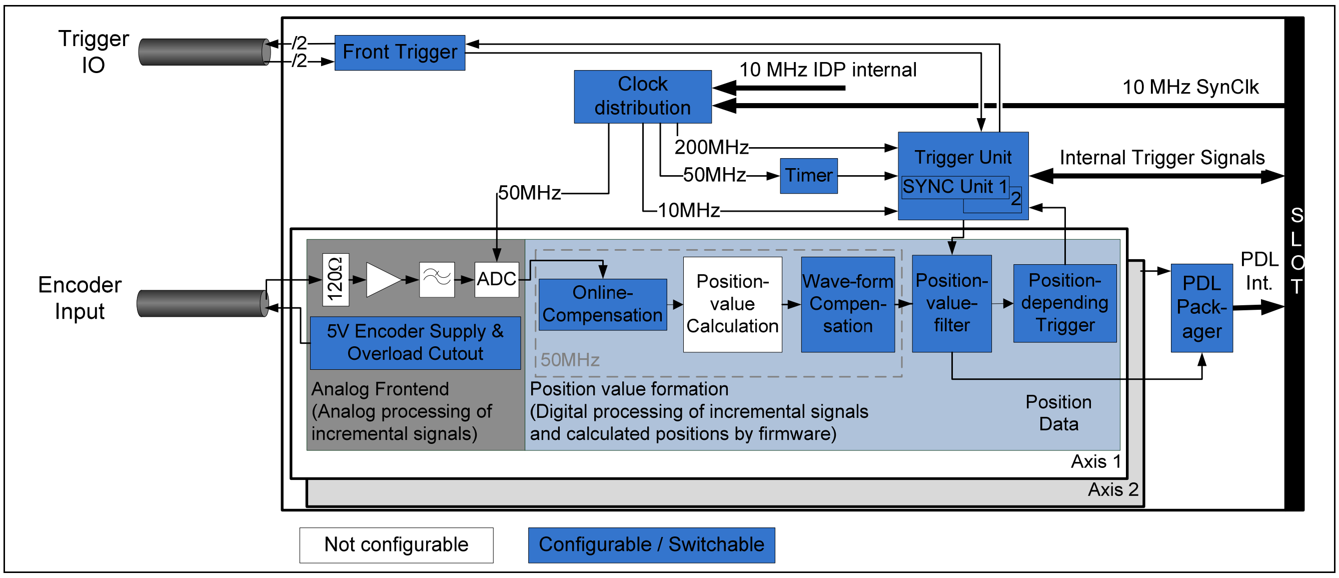
- Interface type
- Processing of the reference marks
- Processing of the homing/limit signals

The configuration of an axis can be changed only if its supply voltage is switched off.

Remarks:
When the voltage supply of the encoder is switched off,

- the referencing of the axis is lost,
- transferred position, speed, acceleration, and reference-mark packets are marked as invalid (these packet types all have the same status word), and
- a reset of the configuration is necessary.

Block diagram of the encoder connections and the signal processing path:



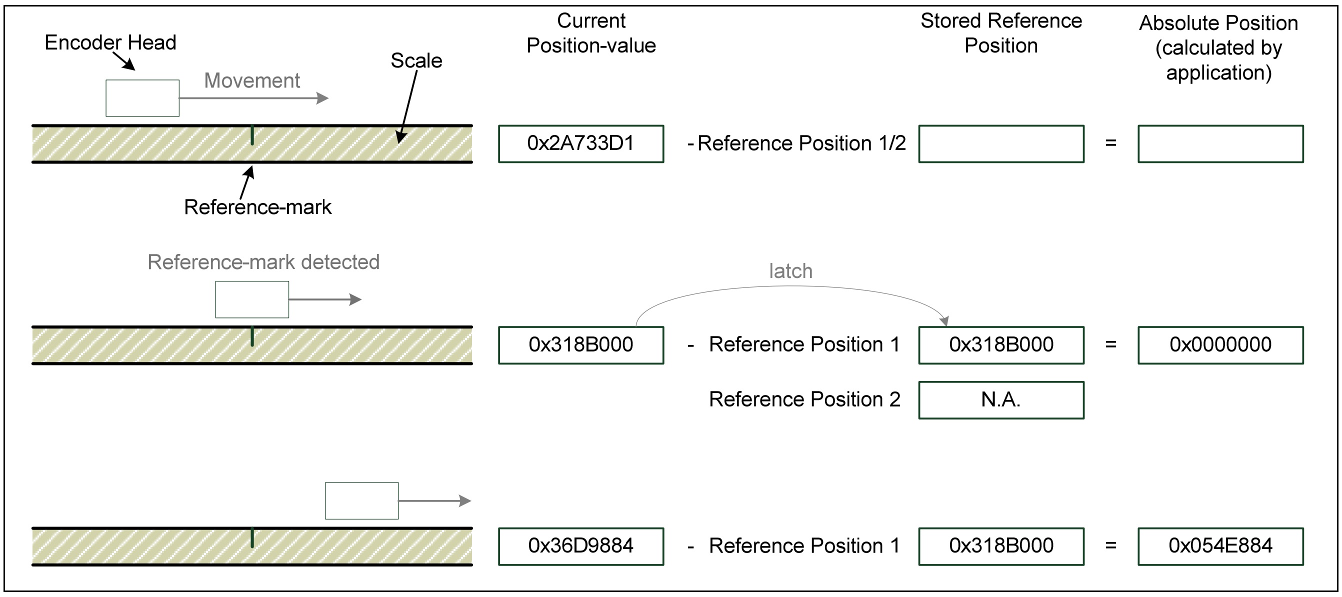
1.3 Treatment of reference marks

With incremental encoders, the reference mark or reference marks are used to establish an absolute reference between the axis position and displayed value. After an interruption of the supply voltage for the axis, this reference is lost.

1.3.1 Encoders with one reference mark

For encoders with a single reference mark, this mark has a unique reference to a specific signal period. This signal period can be used as a reference to generate absolute position values. Traversing the reference mark does not affect the period counter or the interpolation value. The period counter value at the time of the traversing of the reference mark is simply stored in a register for the reference position. This value can be used in the customer software application to calculate absolute position values.

The figure below shows the general process of determining a reference position. The displayed values are provided only as an example. To improve clarity, only a section of the position-value register is shown.



Automatic saving of the reference position must be activated via software. After this command, the axis waits for the next reference mark and then saves the reference position. You must activate this feature again if renewed saving is desired.

The reference position can either be queried by command or transferred as a position data packet.

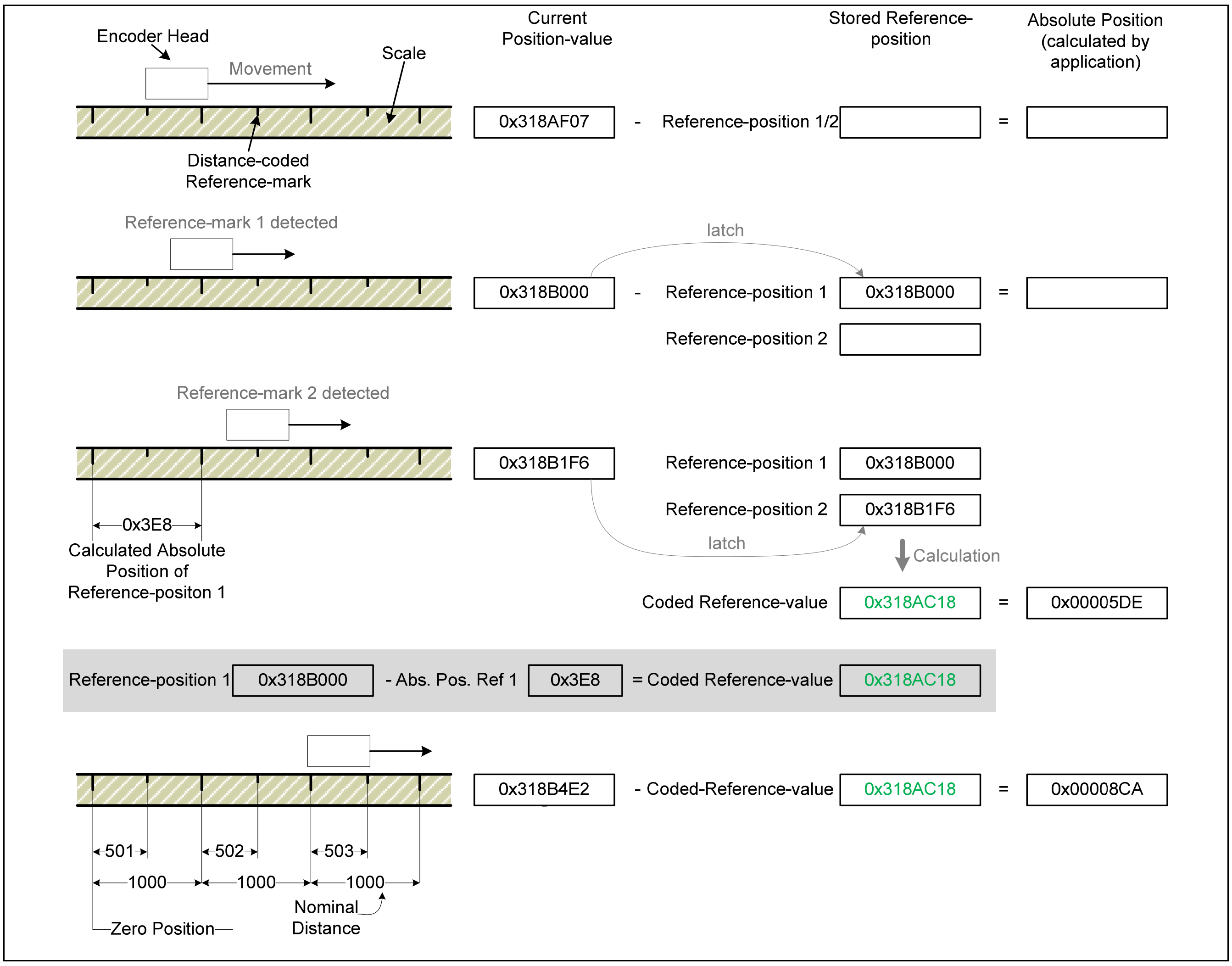
1.3.2 **Encoders with distance-coded reference marks**

In the case of encoders with distance-coded reference marks, reference marks are located at fixed distances (nominal increment in signal periods) along the entire measuring range. Between two of these reference marks, there is a third reference mark whose distance from the other two is varied such that each distance is a multiple of the grating period and occurs only once over the entire measuring range (see figure below). The reference for generating absolute position values is obtained from the counter values through the distance of two traversed (and adjacent) reference marks.

For this purpose, the period counter value is saved twice—once each time a reference mark is traversed. The coded reference value is formed from the distance of the (adjacent) reference marks, and the reference for the formation of absolute position values is thereby created.

When the absolute position value is calculated by the customer software application, this value is treated in exactly the same way as a saved reference position value in the case of encoders with a single reference mark. The coded reference value thus corresponds to the offset between the absolute position value and the generated (incremental) position value.

The figure below shows the general process of determining a reference position. The displayed values are provided only as an example. To improve clarity, only a section of the position-value register is shown.



Remarks:
Scale tapes with distance-coded reference marks are evaluated correctly as well if they are mounted on a circular mounting surface and therefore have an irregular reference interval at the butt joint.

1.3.3 Reference run process

Before a reference run can be started for an axis, the following requirements must be met:

(1) The axis is configured
(2) The supply voltage for the axis is activated
(3) The position must be valid

The reference run can now be activated via software command. The reference position is saved when the next reference pulse is traversed. If the axis is configured for distance-coded reference marks, one position value is saved with each of the two subsequent reference pulses. The saved values correspond to the count of the period counter at the respective reference mark.

The reference run can be stopped via software command. If reference pulses have already been traversed, the corresponding position values will be retained. If the Start Reference Run function is called again, then the previous reference position values become invalid. This is indicated by a bit in the status of the position data packet.

Remarks:
The referencing of the axis also becomes invalid (bit in the status word) if:

- the supply voltage for the axis was switched off
- the "eib8_set_position_value" function was called
- the position became invalid

There are various ways to query the status of the reference run:

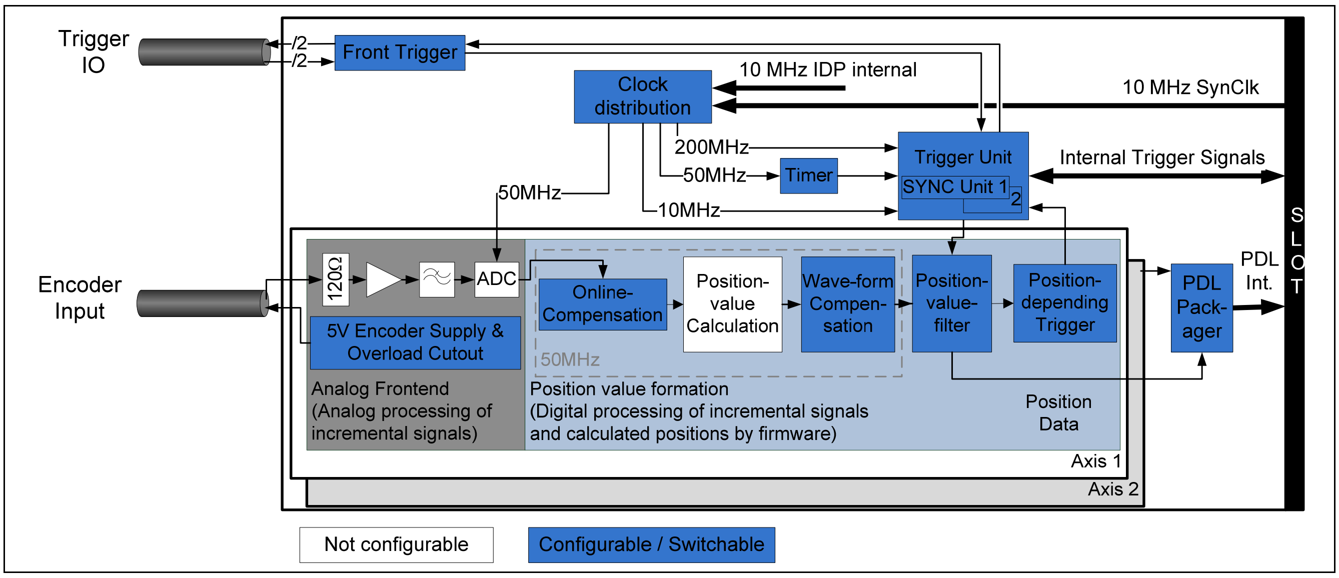
- By querying the parameter "ref_mark:valid" (see 2.6.3)
- In the status word of the position data output

However, this only allows it to be determine whether a valid reference mark was found. In addition, the position data output must be correctly configured as described in Section "2.7 PD position data." In the case of distance-coded encoders, the transferred "reference" position data packet always contains the coded reference value.

1.4 Position value formation

The term "position value formation" denotes the entire signal processing path that the scanning signals of a position encoder follow for the determination of the position value. It starts with the analog path and the subsequent A/D conversion, followed by the real-time position value calculation with all activated compensation and correction procedures, and ends with the determination of the position values in the position value filter.

Block diagram of the signal processing path



1.4.1 Online compensation

Online compensation is an automatic, concurrent adjustment of the offset, phase, and amplitude of the sampled incremental signals. It reduces the deviations within one signal period. Compensation can be activated or deactivated via software. The requirements for the validity of the online compensation, such as the signal amplitude and waveform, must be met.

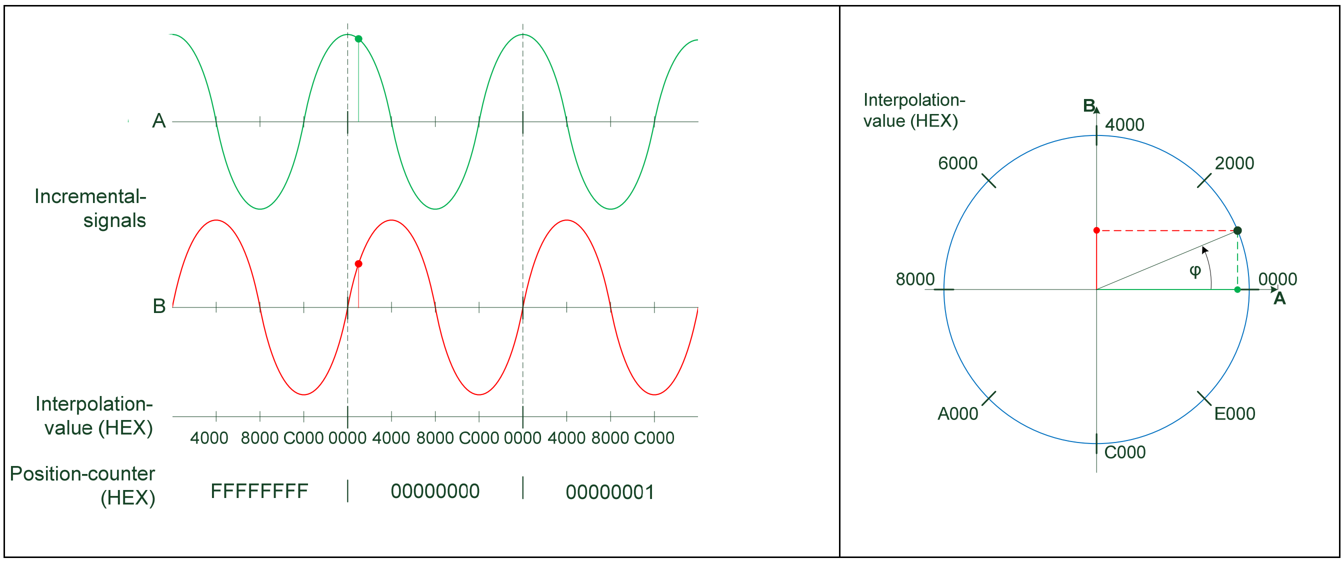
Remarks:
During initial activation of the axis supply voltage, online compensation is activated by default—unless it has been explicitly deactivated beforehand. An interruption of the current supply to the EIB likewise activates online compensation. If the axis supply voltage is deactivated and reactivated, then the most recent valid online compensation configuration applies (on or off).

1.4.2 Position value calculation

For generating the position value, the EIB 8791 subdivides the periods of the incremental signals 65 536-fold (16 bits). The period counter width is 32 bits. The count is incremented or decremented by the value "1" with each signal period of the connected encoder.

The 48-bit-wide position value at the time of the trigger event is formed from the interpolation value (16 bits) and the value of the period counter (32 bits). The position value is saved in a 64-bit-wide register (see *2.7.3.1 Position*). Depending on the encoder type (linear or rotary), the higher-level customer software application can use this value to calculate the corresponding length or angle. The overflow of the period counter does not affect the functionality of the period counter or the interpolator. The overflow, however, must be handled by the higher-level customer software application.

At the time of the trigger event, the incremental signals are sampled and used to calculate a 16-bit-wide interpolation value. The correlation between the interpolation value and incremental signals (A, B) is derived as follows:



1.4.3 **Waveform compensation**

Waveform compensation corrects the effects that the local waveform distortions of a position encoder's scanning signals have on the position value.

Any section of the position encoder's measuring range can be selected as the range in which waveform compensation is to be active during operation.

For waveform compensation, a "map" of the waveform distortions across the measuring range to be corrected is created and saved for the respective axis in a specified non-volatile memory area.

For this purpose, the waveform distortions are locally determined from the scanning signals, and their mean values are calculated in accordance with a predefined compensation point grid after the respective compensation run (1.4.4) used for determining the corresponding information about signal property changes has been completed.

While the compensation values are being acquired, the respective position encoder can be operated as a position value transmitter in the closed loop of the drive axis for performing the compensation run.

A combination of waveform compensation and online compensation (1.4.1) is permitted and is suitable for reducing position errors to the maximum possible extent.

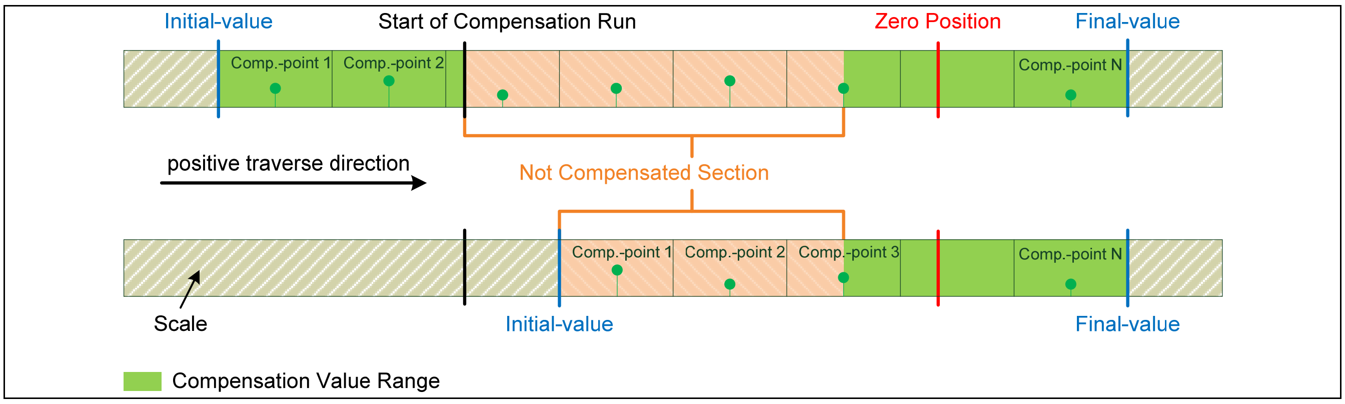
In this case, the compensation run must be performed with online compensation activated.
Note: The same online compensation setting (on or off) must be used for both the compensation run and the subsequent operation.

If a valid set of compensation values is stored in the memory for the respective axis, then they can be used for the automatic compensation of local signal waveform distortions during operation of the position encoder.

Before waveform compensation can be activated, the following requirements must be met:

(1) Axis is configured and supply voltage for the axis is activated
(2) Axis is referenced and position is valid
(3) Valid compensation values are available (for determining the compensation values, see Section *1.4.4 Waveform compensation run*). Note: Only valid if the compensation run was performed with the same configuration.

The function call (activation of waveform compensation) is permitted also while the axis is moving; the axis is permitted to be located outside or inside the compensation range during the function call. Before the position values can be corrected, the stored compensation values need to be loaded. For this purpose, a minimum of four compensation points must be traversed in the same direction of motion (only after waveform compensation has been activated).



There is a maximum traversing speed that depends on the number of compensation points. The resulting maximum signal frequency can be determined with the formula shown below. If this condition is not fulfilled, then an error message is generated. The position values are output without being corrected, but still remain valid. The maximum permissible traversing speed depends on the distance between the compensation points. When the axis is once again operated in the permitted speed range, the position value can be properly corrected again.

$$f_{Signal}^{max} = \frac{Distance\ in\ signal\ periods\ between\ the\ compensation\ points}{0.2}\ [kHz]$$

1.4.4 Waveform compensation run

Measured data acquisition for the automatic determination of compensation values for waveform compensation.

Waveform compensation corrects local errors (waveform distortions) of the encoder signals (scanning signals) that affect the position value. These errors (distortions) are identified in a compensation run. The compensation run can be performed with or without online compensation (activation or deactivation). A slot can perform the compensation run for only **one** axis at a time. When both channels of a slot are to be corrected, two compensation runs must be performed. However, it is possible to simultaneously start several axes on different slots for a compensation run. The compensation run can be performed in the positive or negative traverse direction.

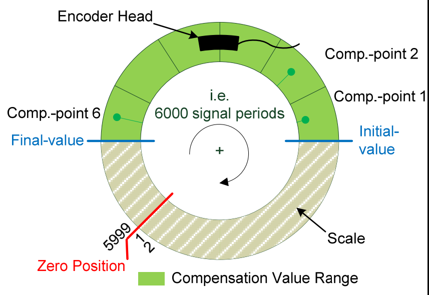
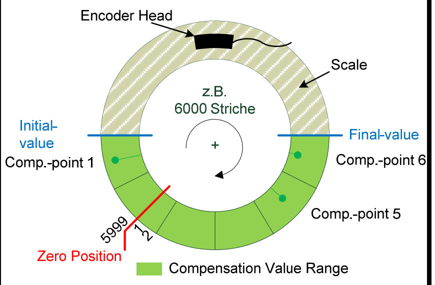
1.4.4.1 Configuration of the compensation run

The following parameters must be configured for each axis:

waveform_corr_run:start_value	= Initial value of the compensation range with respect to the zero position
waveform_corr_run:basic_frequency	= Fundamental signal frequency (phi_1 = one, phi_2 = two signal periods)
waveform_corr_run:direction	= Traverse direction (negative, positive)
waveform_corr_run:number_of_correction_points	= Number of compensation points
waveform_corr_run:distance_of_correction_points	= Distance (in signal periods) between the compensation points

At a fundamental signal frequency of phi_2 = two signal periods, the distance in signal periods between the compensation points must be a value that is divisible by two (i.e., an even number).
The parameters start_value, number_of_correction_points, and distance_of_correction_points define the compensation value range. The start_value parameter designates the initial value of the compensation range. It is the starting value of the compensation range if the traverse direction is positive, and the final value if the direction is negative. The examples below illustrate this.

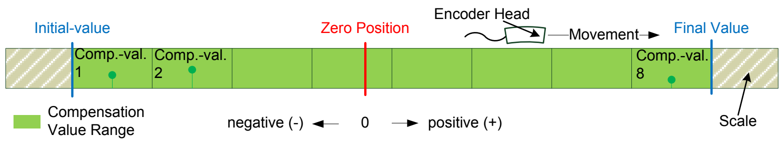
Examples: Rotary encoder

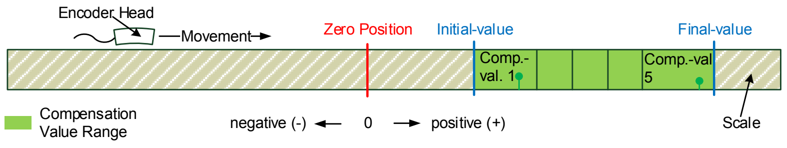
Axis configuration		
Encoder type	Rotary encoder	
Number of increments	6000	
Reference mark type	Single reference mark	
Configuration of the compensation run		
start_value	+2250	+5250
Direction	Positive	
number_of_correction_points	6	
distance_of_correction_points	500	
Compensation range of encoder		

Remarks:

- Positive direction of rotation for incremental signals as per the dimension drawing for the encoder
- With rotary axes, the initial value always refers to the positive traverse direction
- With rotary axes, the product of the distance and the number of compensation points must not be greater than the number of increments per revolution

Examples: Linear encoder

Axis configuration	
Encoder type	Linear encoder
Reference mark type	Single reference mark
Configuration of the compensation run	
start_value	–750
number_of_correction_points	8
distance_of_correction_points	200
Compensation range of encoder	 Initial-value Zero Position Encoder Head Movement Final Value Comp.-val. 1 Comp.-val. 2 Comp.-val. 8 Compensation Value Range negative (-) 0 positive (+) Scale

Axis configuration	
Encoder type	Linear encoder
Reference mark type	Single reference mark
Configuration of the compensation run	
start_value	+1500
Direction	Positive
number_of_correction_points	5
distance_of_correction_points	1000
Compensation range of encoder	 Encoder Head Movement Zero Position Initial-value Final-value Comp.-val. 1 Comp.-val 5 Compensation Value Range negative (-) 0 positive (+) Scale

Remarks:

- Positive direction of motion (traverse direction) for incremental signals as per the interface description (for a detailed description, please refer to the *Interfaces of HEIDENHAIN Encoders* brochure)
- With linear axes, the initial value must always be less than the final value

1.4.4.2 Compensation run process

Before the compensation run can be started for **one** axis, the following requirements must be met:

(1) Waveform compensation is deactivated on both axes of a slot
(2) Compensation run is deactivated on both axes of a slot

To start the compensation run for **one** axis, perform the following steps:

(1) Configure the axis
(2) Activate the supply voltage for the axis
(3) Activate online compensation, if required
(4) Configure the compensation run
(5) Perform a reference run for the axis
(6) Move the scanning head to at least two signal periods before the starting value or after the final value of the compensation range, depending on the defined traverse direction
(7) Start the compensation run with the *eib8_exec_waveform_corr_run* function

When the function is called (start compensation run), the axis can be at standstill, but it must be moving at the correct speed at the beginning of the compensation range.
In this context, the following wide range of frequencies is permitted for the scanning signals: 125 Hz $\leq f_{signal} \leq$ 31.25 kHz.
To ensure the highest possible quality of the compensation values, however, the signal frequency should not significantly exceed f_{signal} = 20 kHz. If the position encoder is operated as a position-value transmitter in the closed loop of the drive axis during the compensation run, the signal frequency of the encoder must always be greater (by a factor of about 5 to 10) than the cutoff frequency of the drive system with respect to its reference transfer behavior.
Although the procedure for determining the compensation values has been designed to ensure a high tolerance to speed fluctuations, high-frequency accelerations are particularly to be avoided during the compensation run.

If the speed of the axis within the compensation range is too high or too low, then the compensation run is aborted, and the corresponding error status is set. The compensation run is also aborted if the distance to the compensation range (initial value) was not at least two signal periods when the compensation run was activated. You can stop the axis movement when the entire compensation range has been traversed (waveform_corr_status:data_acquisition_done = 1). The slot then calculates the sampled values. The calculation of the compensation values may take up to 03:00 (minutes:seconds), depending on the configuration. During this time, waveform_corr_status:running is set. During the calculation, deviations in the compensation values can be detected; for example, deviations resulting from excessive speed fluctuations caused by disturbances due to vibrations.
If an excessive influence on the compensation values is detected during the compensation value calculation, the calculation is aborted, and waveform_corr_status:failed is set. If the compensation run and the calculation are completed without error, then the compensation values are stored in the internal memory. The procedure and the respective functions to be called are described in Part II of this User's Manual.

1.4.5 **Position value filter**

Configurable filter with which the position value at the effective measuring time can be determined from a selectable number of unprocessed position values.

Due to the high internal sampling rate of 50 MHz, it is recommended that the position values be digitally filtered prior to being output.

For this purpose, the dynamics of the position values are reduced in order to increase the resolution of the position information.

You can also preset the age of the position values (Data Age) relative to the time of the external position query (trigger) within a wide range.

In addition to position values, the filter can also determine and output speeds and/or acceleration values.

Bandwidth:	Defines the number of unprocessed position values that are taken into account in determining the position values.
Effective measuring time:	Age of the position value relative to the time at which the position was queried (trigger)
Characteristic:	Selection of a filter with linear characteristic (speed is taken into account) or parabolic characteristic (speed and acceleration are taken into account).

1.5 Position data output

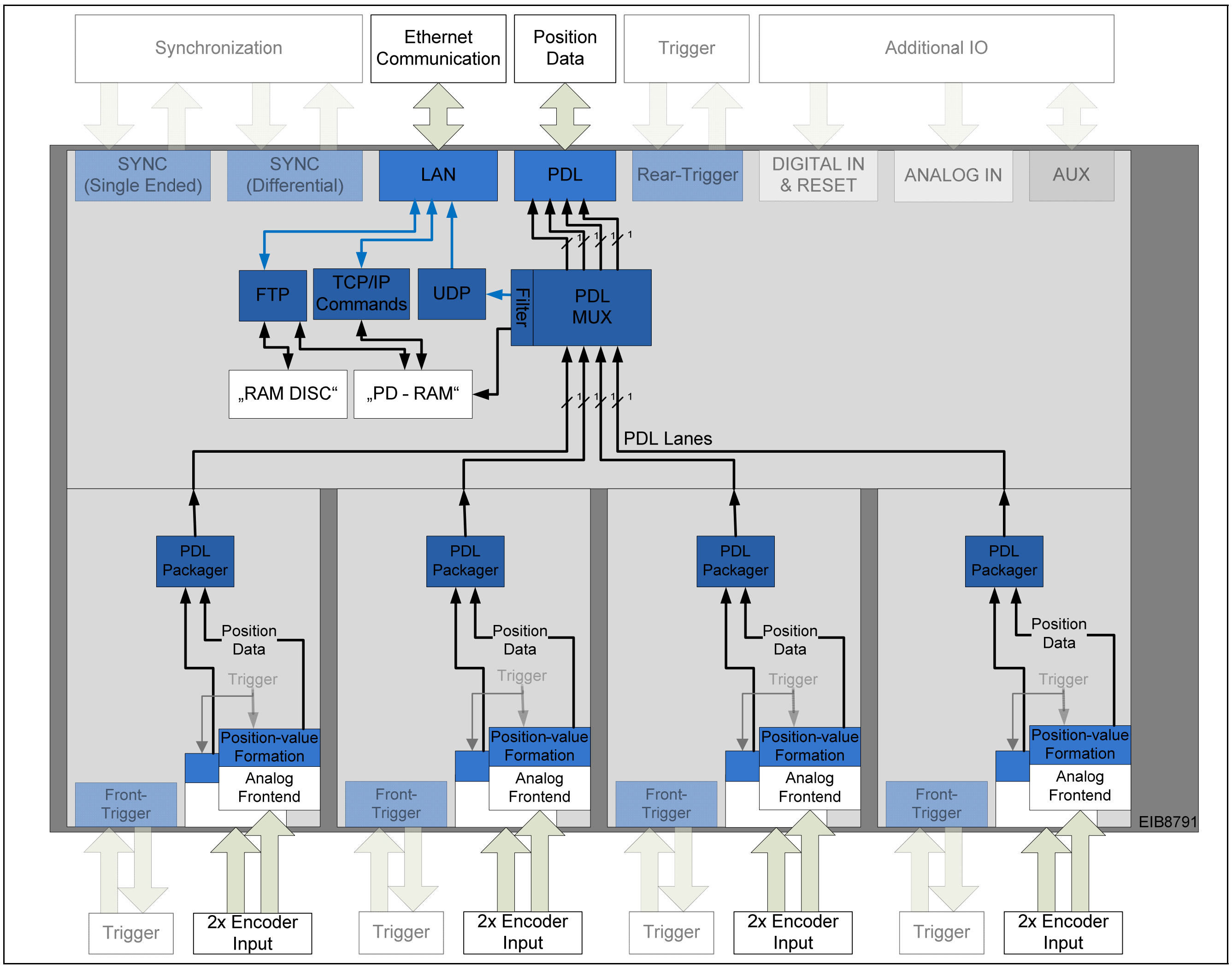
In the EIB 8791, the **P**osition **D**ata **(PD)** are generated as a result of trigger events in slots 1 to 4. These position data are processed internally via a **P**osition **D**ata **L**ink **(PDL)**.
The PDL is designed as a serial interface (full duplex) for real-time communication with minimum latency. It is also provided as an external interface.

Users can output these real-time position data in two different ways. The PDL and the LAN connection are available as interfaces for this purpose. The LAN interface can be used to transmit real-time data via UDP and to read position data recorded in internal memory (via TCP/IP or FTP).

Real-time position data can be processed or output via the following methods:

- ➔ LAN interface
    - o Polling via TCP/IP command
    - o UDP mode
        - ▪ Batched streaming (soft real-time)
        - ▪ Direct streaming (soft real-time)
        - ▪ RAM recording
            - • Single shot
                - o Read out via TCP/IP
                - o Read out via FTP
            - • Rolling mode
                - o Read out via TCP/IP
                - o Read out via FTP
- ➔ PDL interface
    - o PDL interface with proprietary PDL protocol (for highest transfer rates)

Basic circuit diagram:



1.5.1 **Position packets via TCP/IP command link**

With the EIB 8791, it is possible to read position data with via a command. This mode of operation is referred to as polling. The transfer is performed via TCP/IP and is thus not real-time capable.

Configuration of the polling mode → See Section *2.8.5 Polling of position data*

1.5.2 **Position data output via UDP**

The EIB 8791 can output internal position data via UDP over the LAN interface. In the process, specific packet numbers can be filtered. This means that not all packets transferred via PDL are necessarily also output via UDP. It is possible to transfer individual position values via UDP, whereby
the modes "direct_soft_realtime" and "batch_soft_realtime" are supported; for details, please refer to Section 2.8.6.1.

Position data transfer via UDP is generally not protected against the loss of data. Therefore, it must be checked on the host side whether the transfer was interrupted and whether packet data were lost. This can be done by analyzing the frame counter. This 8-bit counter must be incremented sequentially. To ensure the unambiguous assignment of the data to the respective axes, the packet numbers must be set such that they are unique.

The maximum possible (usable) UDP packet rate usually depends on the receiving remote station (host computer). With the EIB 8791, UDP packet rates of up to 400 kHz can be verified.

Configuration of the UDP mode → See *"2.8.6 Receiving position data via UDP"*

1.5.3 **Storing position packets in the EIB 8791 (recording)**

The recording mode can be seen as a subfunction of the UDP mode. With it, internal position data from the EIB 8791 can be stored in an internal memory. As with UDP output, specific packet numbers can be filtered. This means that not all packets transferred via PDL are necessarily also stored in memory.

Up to 10 million position values can be stored during recording. If all eight axes of the EIB 8791 are recorded, then the resulting maximum recording depth is 1.25 million packets per axis.

The maximum data rate for recording is 240 MB/s. A position packet is 12 bytes long. You can thus record up to two axes at a maximum rate of 10 MHz, or all eight axes at up to 2.5 MHz. If more than one
packet is generated per axis, then these packets can be recorded at an appropriately reduced rate.
The user is responsible for ensuring that the maximum data rate for recording is complied with.

The recording memory (PD RAM) is read out via TCP/IP. For this, data packets with a size of 1024 bytes are read out from the memory and transferred. This query must be repeated until the recording memory has been completely transferred.

Reading out via TCP/IP → See *"2.8.6.3 Reading position data from the position data buffer"*

Alternatively, reading out can be performed via FTP. However, this requires an intermediate step that limits the maximum memory depth. Therefore, the position data from the PD RAM must first be copied to the so-called RAM disk.
There, the data are stored as a file and can be queried via simple FTP access.

Saving to RAM disk → See *"2.8.6.4 Writing position data to a file"*
FTP file transfer → See *"1.12 Server of the EIB 8791 "*

1.5.4 **Position data via position data link**

The position data link has been developed for serial real-time data transfer. This serial interface with a proprietary protocol has been optimized for the transfer of small information units (position data) with low latency and minimum overhead. The communication topology requires a point-to-point connection.

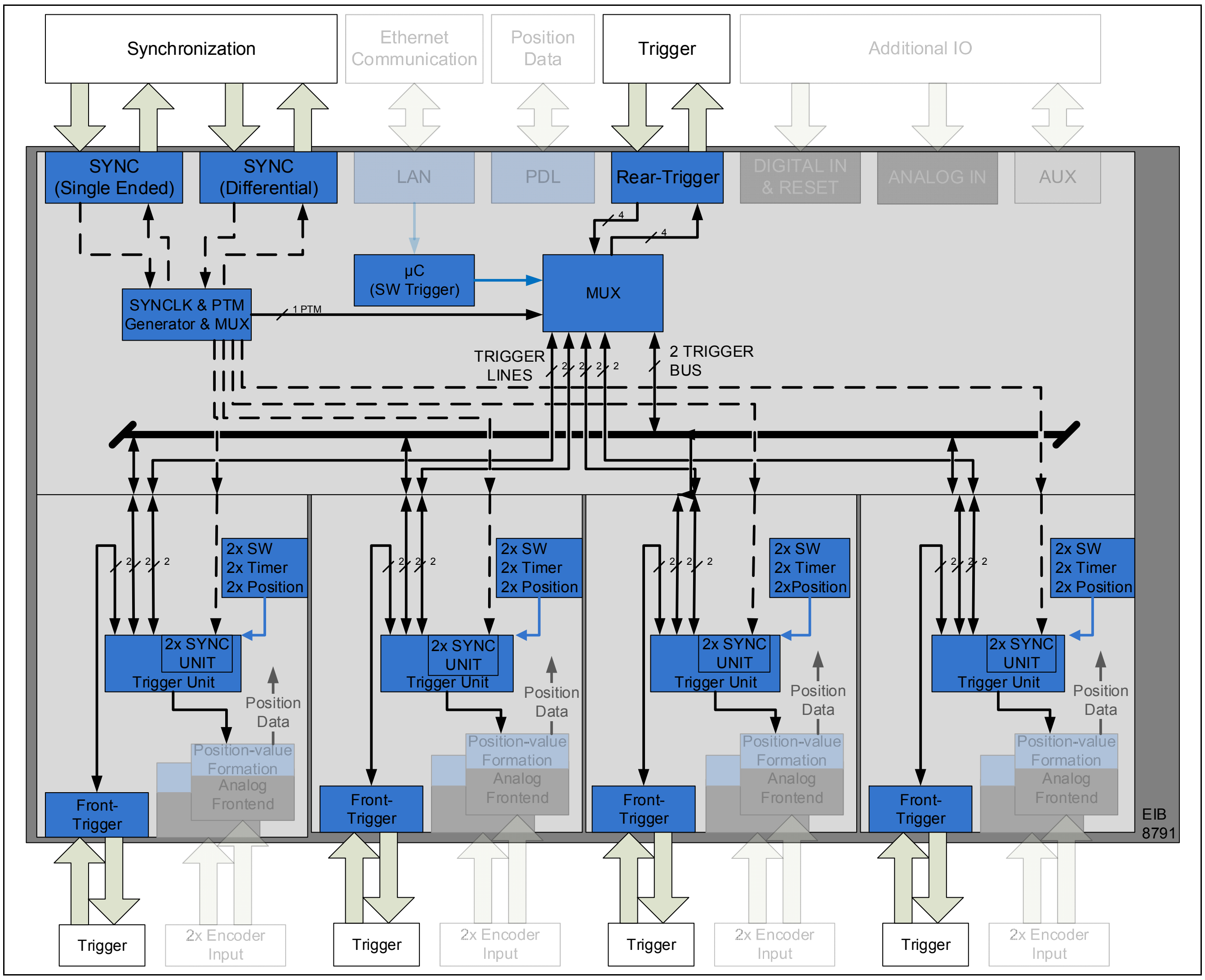
The position data link is a continuous serial data stream. The levels, the 8B/10B encoding, and the data rates are identical to those of serial data transfer via PCI Express (PCIe 1.1, Gen1).

For more information, please contact encoder support at HEIDENHAIN.

1.6 Synchronization and asynchronous triggering

The EIB 8791 provides the option of generating synchronous or asynchronous position data. The triggering of the position data can thus occur in a fixed time interval established by a precise synchronization clock signal (SynClk) and a position trigger marker (PTM). As an alternative, position data can also be generated via triggering at asynchronous events. The mixing of these two modes of operation should be avoided.

Basic circuit diagram of the synchronization and trigger resources:



1.6.1 **Asynchronous operation with triggers**

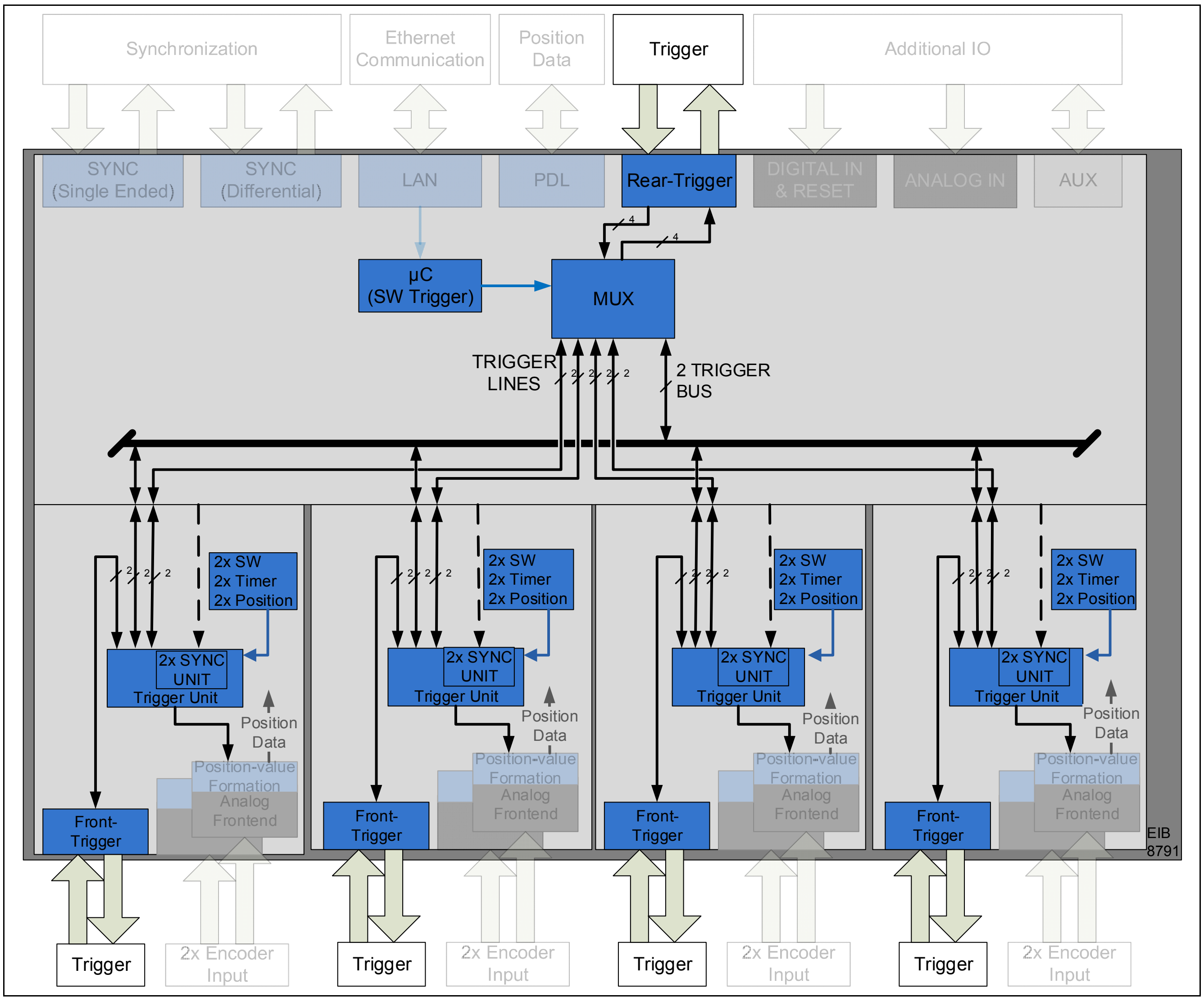
The EIB 8791 features a total of twelve asynchronous trigger inputs and twelve asynchronous trigger outputs. These outputs are divided into four trigger I/Os on the rear side and eight trigger I/Os on the front side.

In principle, any axis can be triggered by any trigger I/Os, with the only limitation being the utilization of internal trigger resources. If multiple inputs of a multiplexer are switched to the same output, then the input signals are OR-gated. Each slot (with two axes per slot) provides two trigger lines (Rx and Tx separately, therefore full-duplex) and two trigger buses (half-duplex). Trigger lines are point-to-point connections between the EIB 8791 basic component and the slots. Trigger buses are connections between the slots and the EIB 8791 basic component, in which there can be only one source node, but any number of receiving nodes.

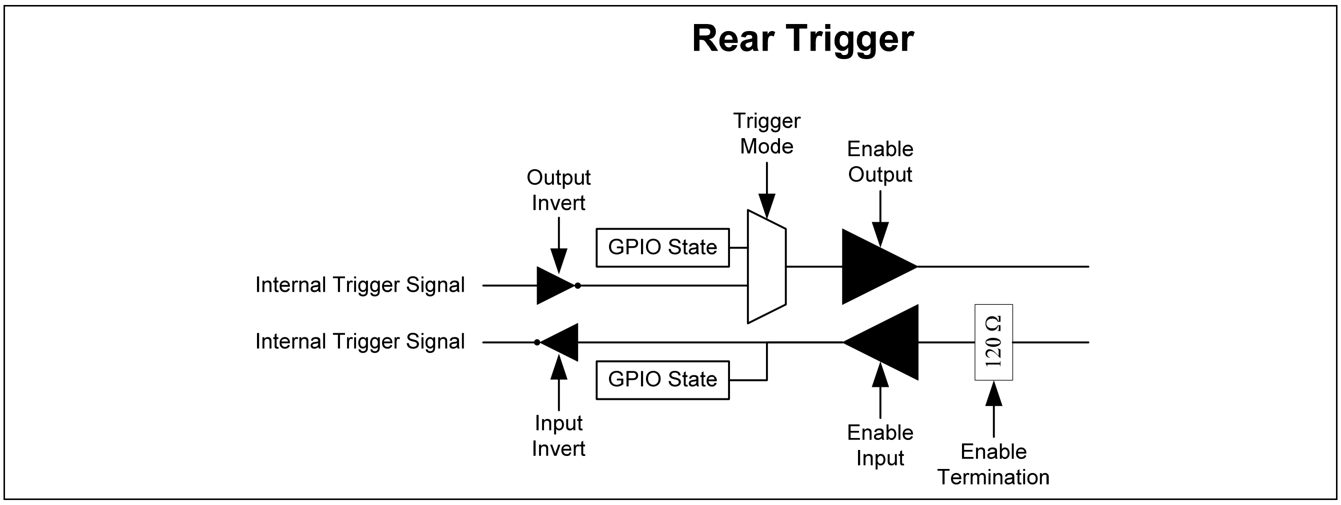
Each component (EIB 8791 basic component and slots 1 to 4) has its own trigger multiplexer or trigger unit that routes the inputs and outputs shown in the figures.

Configuration of the triggers -> See *"2.9.2 Configuration of the triggers"*

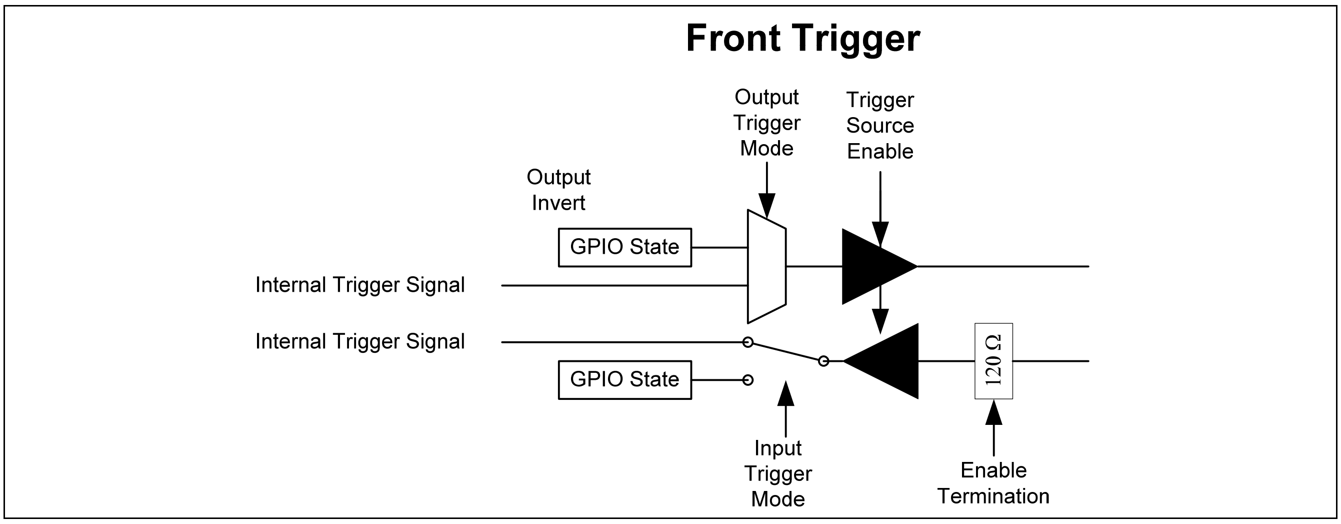
Basic circuit diagram of asynchronous trigger resources:



Block diagram of rear trigger I/Os (four I/Os available):



Block diagram of front trigger I/Os (eight I/Os available):



The trigger I/Os shown (for the front and rear) can also be used as GPIOs.

1.6.2 **Synchronous operation with SynClk and PTM**

During synchronous operation, the EIB 8791 is synchronized to an external clock signal. The PTM is a logic signal that marks the rising clock pulse edges of the SynClk signal at which a position value is to be determined and output.

1.6.2.1 **SynClk clock pulses (internal and external)**

The SynClk signal is always a 10 MHz clock that can either be generated by the EIB 8791 itself or provided via an interface of the EIB 8791. Two interfaces are available for this purpose—a single-ended LVTTL interface with SMA connector and a differential SYNC interface that provides the SynClk signal and the PTM in the form of LVDS signals.

This synchronization clock can also be output and used for synchronizing other devices.

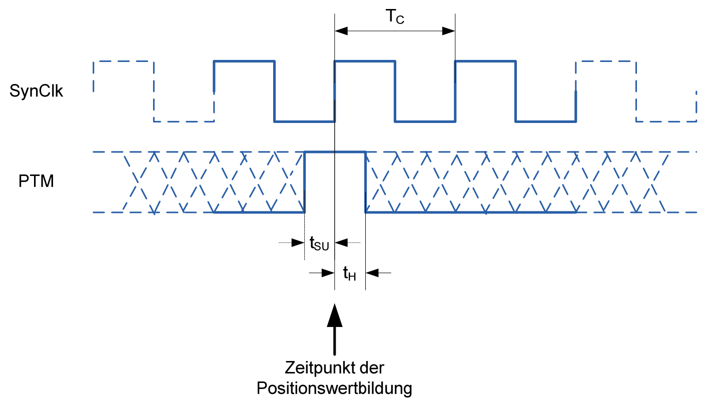
1.6.2.2 PTM – Position Trigger Marker (internal and external)

The PTM is a signal that marks a rising edge of the SynClk signal. This PTM can either be generated internally (which only makes sense if the SynClk signal is also generated internally) or provided via an interface of the EIB 8791. Two interfaces are available for this purpose—a single-ended LVTTL interface with SMA connector and a differential SYNC interface that provides the SynClk signal and the PTM in the form of LVDS signals. The PTM is usually a periodic pulse that defines the sampling rate of the position-value output. However, the exact time of sampling depends on the edge of the SynClk signal that has been marked by the PTM!

→ Jitter in the PTM signal does not affect sampling if the timing requirements are complied with!

The PTM can also be output and used for synchronizing other devices.

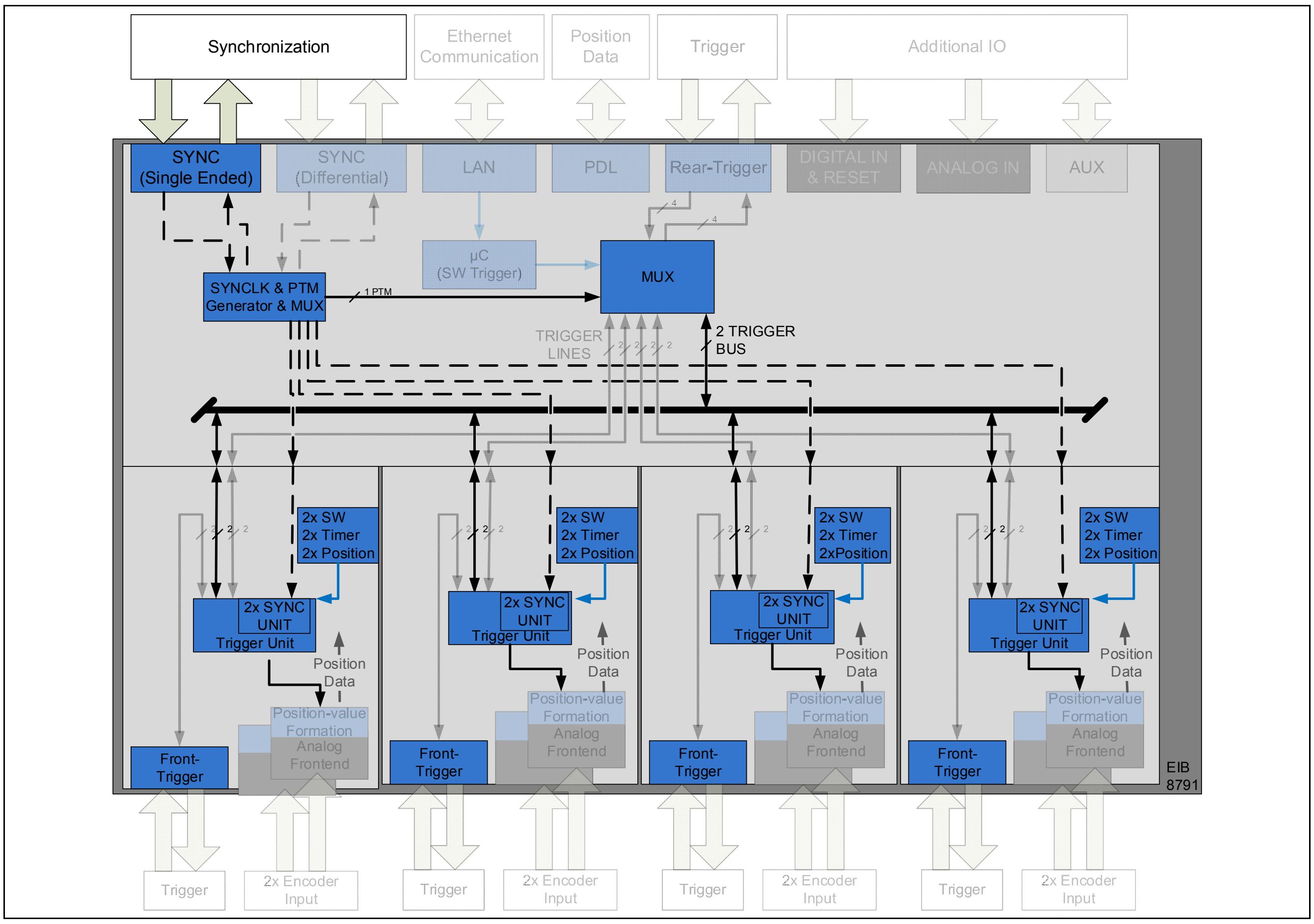
The PTM signal must meet the following timing requirements with respect to the SynClk signal:



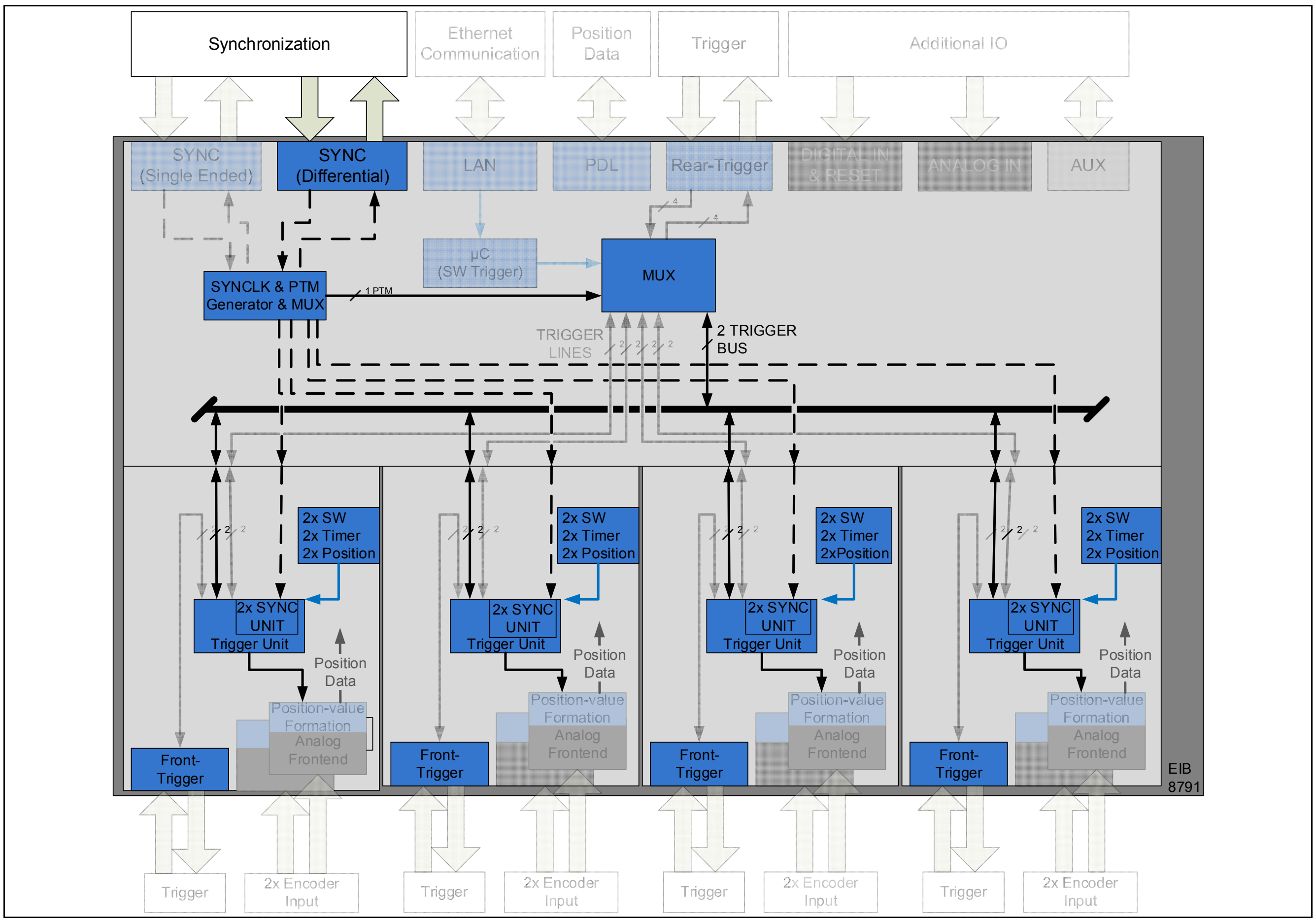
Symbol	**Meaning**	**Requirement**
T_C	SyncClk period	100 ns
t_{SU}	PTM setup time	≥ 20 ns
t_H	PTM hold time	≥ 20 ns

Internally, the PTM must be routed to the slots via the available trigger lines and trigger buses.

Basic circuit diagram of the synchronization resources (single-ended):

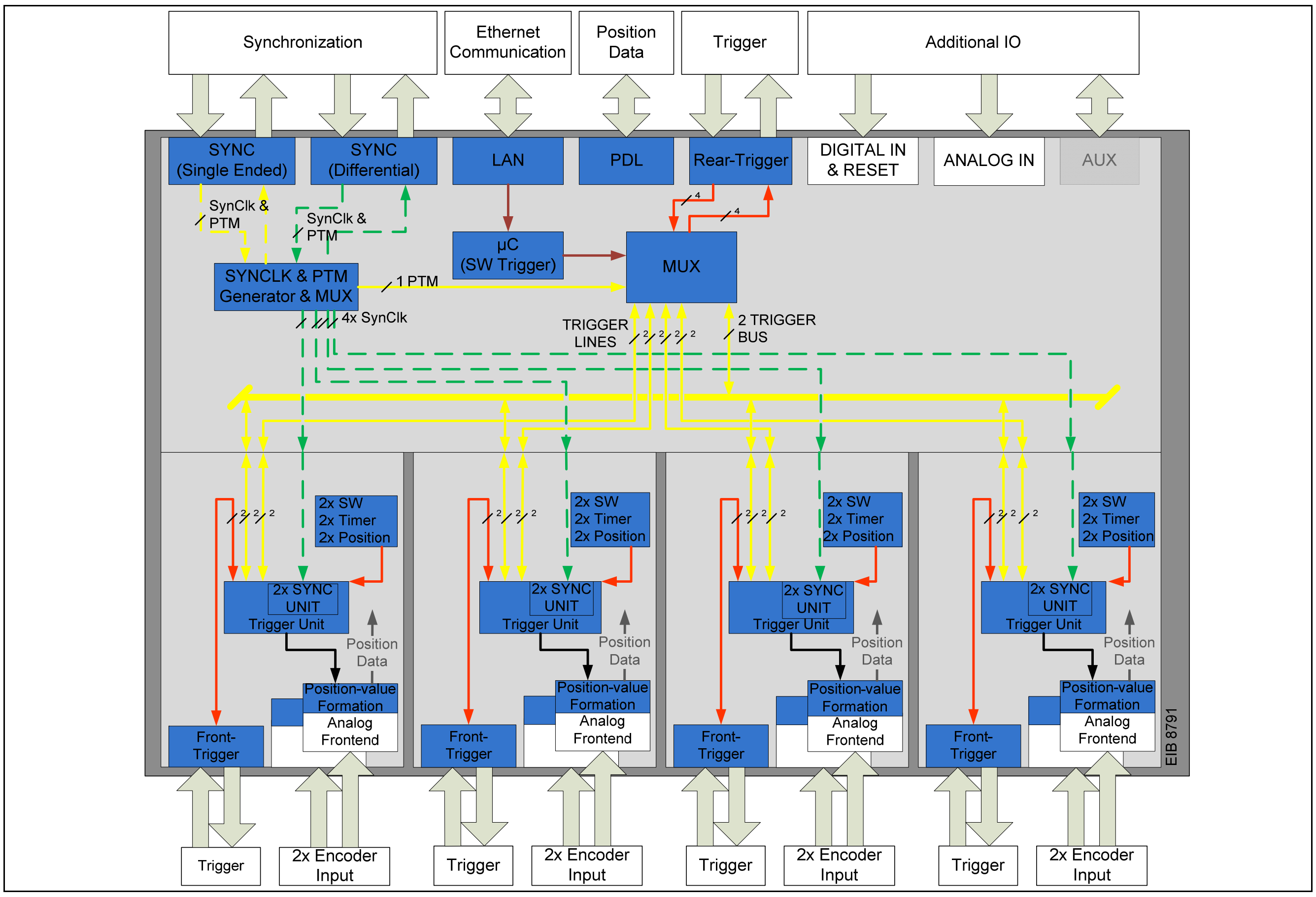


Basic circuit diagram of the synchronization resources (differential):



Keep in mind that the signal paths offer differing temporal performance:

Overview of the performance for synchronization and asynchronous triggering:



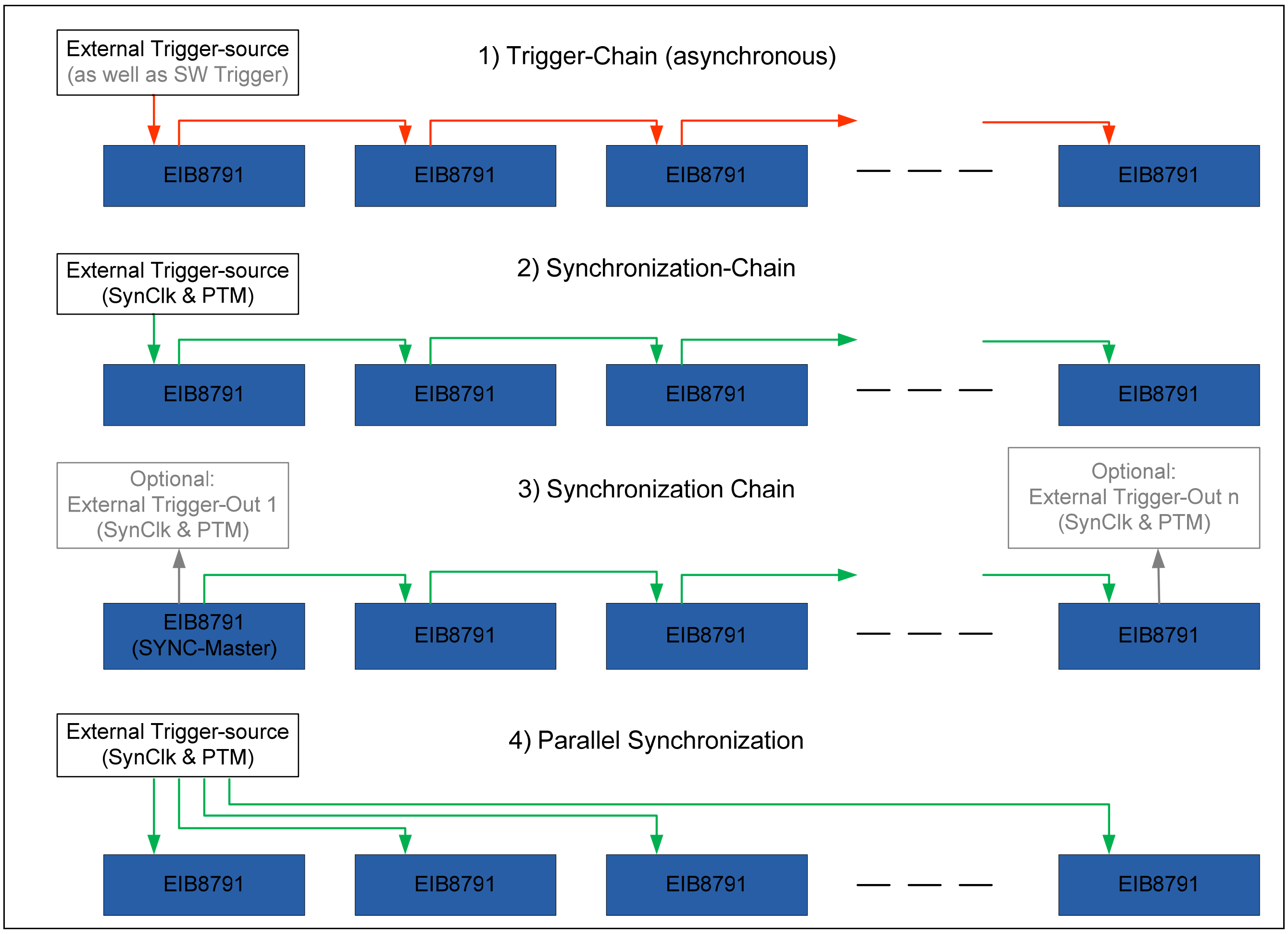
Brown:	**Lowest temporal accuracy performance**
Red:	**Accuracy performance (~1 ns)**
Yellow:	**Accuracy performance (<1 ns)**
Green:	**Best accuracy performance (<50 ps)**

1.6.3 "Synchronization" of multiple EIB 8791 interface boxes

The EIB 8791 can be synchronized or triggered in different ways.
For general-purpose applications, different "basic configurations" are recommended: *"2.9.2 Configuration of the triggers"*

When using more than one device, you can also create trigger chains and synchronization chains. Depending on the application, the resulting performance needs to be taken into account.

Basic circuit diagram of different trigger chains or synchronization chains:



1) Asynchronous triggering without clock signal
   a. Propagation delay per device: approx. 70 ns
   b. Drift of up to 4 ns per device
   c. High additive jitter
2) Synchronization chain to external clock source
   a. Propagation delay per device < 10 ns with single-ended or < 1 ns with differential
   b. Drift of < 1 ns (single-ended) or < 50 ps (differential) per device
   c. Low additive jitter
3) Synchronization chain to a "master EIB2
   a. Propagation delay per device < 10 ns with single-ended or < 1 ns with differential
   b. Drift of < 1 ns (single-ended) or < 50 ps (differential) per device
   c. Low additive jitter
4) Parallel synchronization
   a. No propagation delay
   b. Drift of < 1 ns (single-ended) or < 50 ps (differential) per device
   c. Minimum additive jitter

1.7 Digital and analog auxiliary inputs

Via additional auxiliary inputs, the EIB 8791 provides the option of inputting eight digital sensors and four analog sensors. The status of these inputs and their analog values can be read out only via TCP/IP command.

1.7.1 **DIGITAL IN**

Depending on the TCP/IP command interface, a polling rate of up to approx. 50 Hz can be achieved.

For the pin layout and electrical data, see the *Operating Instructions*

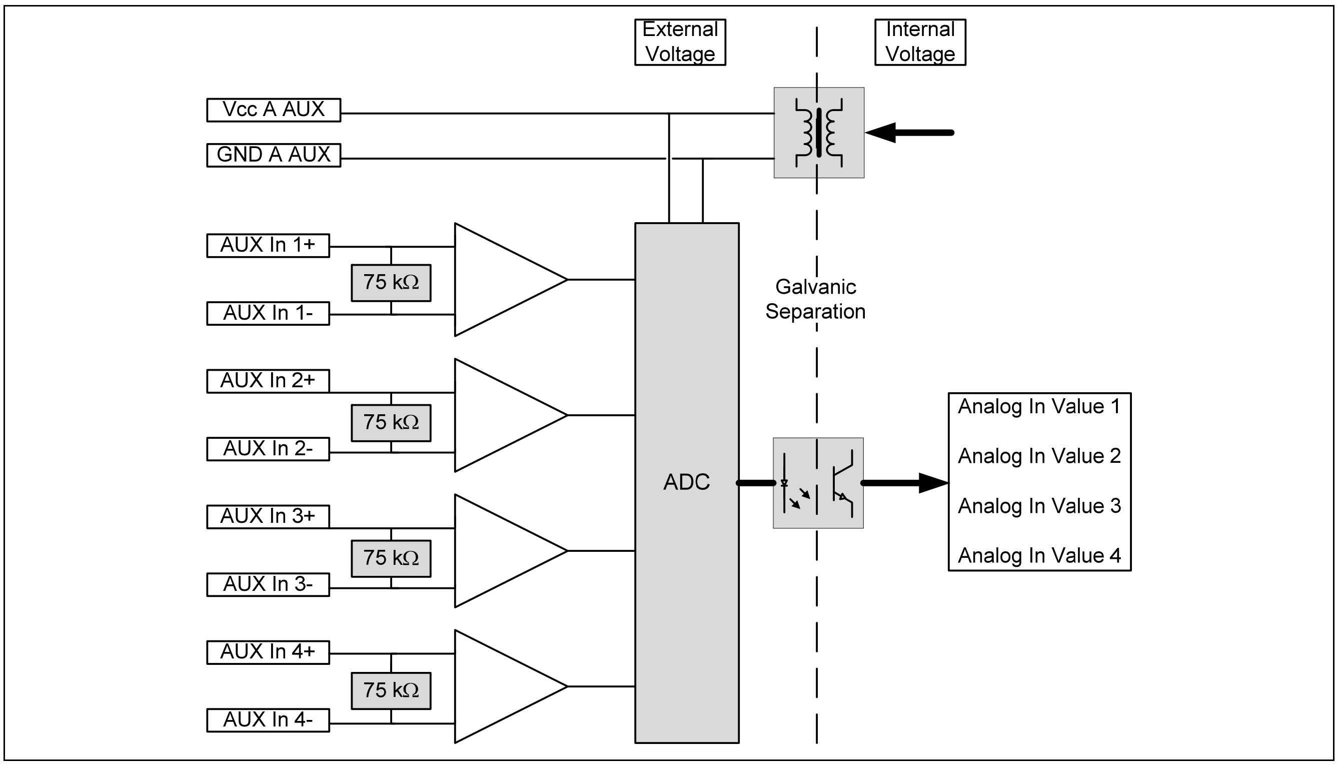
1.7.2 **ANALOG IN**

The ANALOG IN connection can be used to connect standard transducers that supply a voltage signal in a range between –10 V and +10 V. The digital resolution is 14 bits with an internal sampling rate of 2 samples/s. The ANALOG IN connection is galvanically isolated from the other connection. There is no internal separation between the four channels. The differential input impedance is typically 75 kΩ (at least 40 kΩ). The unprocessed ADC value (Analog In Value [n]) that is read out must be interpreted as a 32-bit unsigned integer.

Calculate the analog voltage value as follows:

$$\text{Analog Voltage [V]} = \frac{Analog\ In\ Value\ [n]\ -\ 2^{13}}{2^{13}} * 10.24\,V$$

Basic circuit diagram of the analog input:



For the pin layout and electrical data, see the *Operating Instructions*

1.7.3 **AUX I/O**

The AUX connection is not supported and must not be used.

1.8 Events, self-test, and diagnostics

The EIB 8791 provides different means of detecting errors. When the device is switched on, a self-test (BIST) is performed automatically. During operation, diagnostic tasks that monitor the functions and the operating status of the device are continuously performed in the background.

If an error or impermissible operating status is detected, then they are written to an event queue as a warning or error.
Each hardware module (EIB 8791 basic component, slots 1 to 4) has its own event queue.
The user must actively query whether events have occurred. It is recommended that a cyclical check for the occurrence of errors be performed once every second.
Only events that, in the slots, can lead directly to a faulty position data output over the real-time interface (PDL or UDP) are marked with a corresponding bit directly in the position data packet.

For details, see Section *2.8.2 Querying the event queues*

1.9 Reset

In the EIB 8791, a reset of the EIB 8791 can be triggered by TCP/IP command, by reset key, or by remote reset. The software reset provides the option of triggering a simple reset after the device has started in user mode. The different boot modes can be selected with the reset key and remote reset. See *Device Resets* in the operating instructions.
The remote reset line is part of the DIGITAL IN connection and has the same electrical parameters.

For the pin layout and electrical data, see the *Operating Instructions*

1.10 Shutdown

The EIB 8791 should not be simply switched off. To ensure that internal data (diagnostic data, etc.) are saved correctly, use the shutdown command. When the command is sent to the EIB 8791, the device saves all of the data and switches off the position output and all of the other outputs. You can switch off the EIB 8791 after the STATUS LED goes out.

1.11 Firmware update

The EIB 8791 firmware can be updated by the user with an FTP client. An FTP server is provided by the EIB 8791 for this purpose. For configuration (user name, password, FTP port), use the network commands of the EIB 8791 (see *2.10.6 Setting the network parameters*).

Only special update files from HEIDENHAIN may be used for the EIB 8791.

1.11.1 **Update via EIB 8791 Application**

You can update the firmware by using the EIB8 Application provided on the driver CD (see Section 3.3.8). You will need a Windows PC for this purpose.

1.11.2 **FTP update**

Since the FTP server of the EIB 8791 uses a FAT16 file system, the file names uploaded to the FTP server must be in 8.3 format. This means that the file supplied by HEIDENHAIN (e.g., 816477-06-00-A.bin) needs to be renamed (e.g., to "update.bin") before the upload.

The example below is based on a firmware update from a computer running a Windows operating system. The EIB 8791 must be connected to the computer via Ethernet. In this example, the file name for the update is "816477-06-00-A.bin".

1.11.2.1 **Update via command line**

First, save the firmware file to a defined directory. It is recommended that you immediately rename the file (e.g., 816477-06-00-A.bin) to the 8.3 format (update.bin); for example, "C:\temp\EIB\update.bin".

The Windows command line will not be started and switched to the "C:\temp\EIB" directory. The FTP client program is launched with the command "ftp -A 192.168.168.2".

With the -A option, you log on to the FTP server as "anonymous" (this is the default setting). If the FTP user name and/or the password has changed, leave out the "-A" option. The FTP server of the EIB 8791 will then prompt you for the user name and password.

The IP address is "192.168.168.2" (default setting), or use the customer-specific setting.

The command line of the FTP client program (ftp>) is now displayed. Enter the following commands:

```
ftp> binary
ftp> cd update
ftp> put update.bin
ftp> quit
```

The "binary" command switches from ASCII to binary file transfer. The "cd update" command switches to the "update" directory of the FTP server in the EIB 8791. The "put update.bin" command transfers the "update.bin" file from the host (in the current directory) to the FTP server of the EIB 8791 and saves it to the FTP server's current directory ("update"). With the "quit" command, you log off from the FTP server and exit the FTP program.

When the file has been successfully transferred, the FTP client displays a corresponding message in the command line. The status LED of the EIB 8791 flashes while the update is running. This process can take up to 7 minutes. While the status LED is flashing, the power supply must not be switched off, nor can commands be transmitted to the EIB 8791 via the Ethernet interface (EIB 8791 blocks commands as long as the update procedure is running).

When the status LED is active again, the relevant software command must be used to query whether the update has been successfully completed. The status of the update process can be polled until the EIB 8791 is booted again. If an error occurred during the update, the update procedure should be repeated.

The EIB 8791 boots the new version of the firmware after the next reset.

If an error occurred during the firmware update (power interruption, CRC error during transfer, invalid firmware file, etc.), the EIB 8791 is automatically started in factory mode because no valid user image exists. In factory mode, the update of the user image must now be repeated.

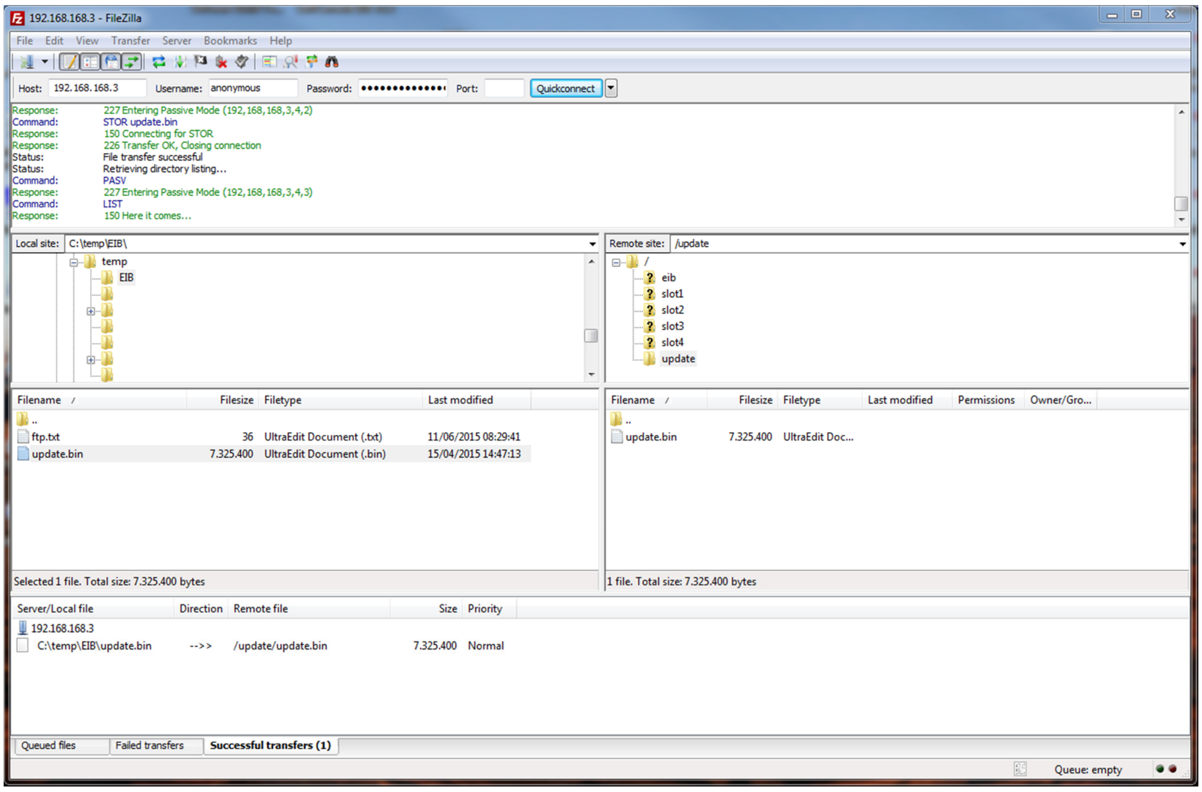
Note on automating the update procedure:
In the "C:\temp\EIB" directory, save the file ftp.txt with the following content:

```
binary
cd update
put update.bin
quit
```

The update with "ftp -s:ftp.txt -A 192.168.168.2" can now be started; the FTP client program automatically executes the commands contained in the ftp.txt file.

1.11.2.2 Update via FileZilla

First, save the firmware file to a defined directory. It is recommended that you immediately rename the file (e.g., 816477-06-00-A.bin) to the 8.3 format (update.bin); for example, "C:\temp\EIB\update.bin".
Then open the FileZilla FTP program.



192.168.168.3 - FileZilla

Host: 192.168.168.3 Username: anonymous Password: •••••••••••••• Port:

```
Response:    227 Entering Passive Mode (192,168,168,3,4,2)
Command:     STOR update.bin
Response:    150 Connecting for STOR
Response:    226 Transfer OK, Closing connection
Status:      File transfer successful
Status:      Retrieving directory listing...
Command:     PASV
Response:    227 Entering Passive Mode (192,168,168,3,4,3)
Command:     LIST
Response:    150 Here it comes...
```

Local site: C:\temp\EIB\

Filename	Filesize	Filetype	Last modified
..			
ftp.txt	36	UltraEdit Document (.txt)	11/06/2015 08:29:41
update.bin	7.325.400	UltraEdit Document (.bin)	15/04/2015 14:47:13

Selected 1 file. Total size: 7.325.400 bytes

Remote site: /update

Filename	Filesize	Filetype	Last modified	Permissions	Owner/Gro...
..					
update.bin	7.325.400	UltraEdit Doc...			

1 file. Total size: 7.325.400 bytes

Server/Local file	Direction	Remote file	Size	Priority
192.168.168.3				
C:\temp\EIB\update.bin	-->>	/update/update.bin	7.325.400	Normal

Enter the FTP data of the EIB 8791 into the Host, Username, and Password fields (in this case: 192.168.168.3, anonymous, "*"), and press Quickconnect. The FTP program connects to the EIB 8791 and displays the directory structure of the host (on the left) and of the EIB 8791 (on the right).

Select Transfer->Transfer type in the menu and choose "Binary".

Then switch to the "update" directory (on the right) and drag the "update.bin" file (on the left) to this directory with the mouse. This starts the data transfer from the host to the EIB 8791.

On completion of the transfer, the EIB 8791 starts the update procedure described above.

1.12 Server of the EIB 8791

The EIB 8791 includes an FTP server. With the aid of an FTP client program, you can transfer update files and MAP data to the EIB 8791 and receive PDL data, memory dump data, and MAP data from the EIB 8791.

To be able to establish an FTP connection between the host and the EIB 8791, the control channel (port 21) must be open if a firewall exists. For an active FTP connection, the data channel (port 20) must be opened as well. This is not the case for a passive FTP connection, because the FTP server of the EIB 8791 does not establish a connection to the host by itself. The FTP port for the control can be defined by the customer (default 21).

1.12.1 **Configuration of FTP parameters**

With the network commands of the EIB 8791, you can modify the configuration of the FTP server and query its current status. The following FTP parameters can be edited (see Section *"2.10.6 Setting the network parameters"*).

- FTP port (default: 21)
- FTP user name (default: "anonymous")
- FTP password (default: "*")

1.12.2 **Directory structure of the FTP server**

The following directories are always provided on the FTP server:

- eib—PDL data and the memory dump of the EIB 8791 basic component are stored in this directory.
- slot1—the memory dump data of slot1 are stored in this directory.
- slot2—the memory dump data of slot2 are stored in this directory.
- slot3—the memory dump data of slot3 are stored in this directory.
- slot4—the memory dump data of slot4 are stored in this directory.
- update—the firmware file for the EIB 8791 update must be stored in this directory (see previous section).

Each slot directory (slot1, slot2, slot3, slot4) can contain directories for the MAP data (sid_0000, sid_0001, etc.). In these directories, MAP files can be transmitted to the respective slots or retrieved from the slots.

1.12.3 **Storing the PDL data**

With the "EIB8_SaveDataToRD" command, you can transfer recorded PDL data to the "eib" directory of the FTP server. A maximum of 2 000 000 PDL packets can be stored in the FTP server.

The PDL data must be in the RAM of the EIB 8791 basic component (see *"2.10.1 Recording of position data"*). The "EIB8_SaveDataToRD" command with the file name, the slot parameter 0x00000000, and the data ID 0x00000002 is now transmitted to the EIB 8791. Upon completion of the command, the PDL data (max. 2 000 000 PDL packets) are located in the "eib" directory of the FTP server and can be retrieved by an FTP client.

1.12.4 **Storing the memory dump data**

The memory dump data contain additional diagnostic data of the EIB 8791 system and can be used by HEIDENHAIN for support purposes.

With the "EIB8_SaveDataToRD" command, you can create memory dump data from the EIB 8791 basic component, slot1, slot2, slot3, and slot4. The memory dump data are stored in the "eib" FTP directory for the EIB slot and in "slotX" for the SlotX (whereby X = 1 ... 4).

The "EIB8_SaveDataToRD" command with the file name, the respective slot parameter (0x00000000 for EIB slot, etc.), and the data ID 0x00000001 is transmitted to the EIB 8791. Upon completion of the command, the memory dump data are located in the respective slot directory of the FTP server and can be retrieved by an FTP client.

1.12.5 **Transfer of MAP data to a slot**

Use an FTP client to transmit MAP data to the slots. The file name of the MAP data must be in the 8.3 format because the FTP server has a FAT16 file system.

For example, the data are loaded into the FTP server directory "slot1/sid_0000". The EIB 8791 then transfers the MAP data to StorageID0 of slot1.
The status LED flashes while the EIB 8791 is storing the data. During the phase in which the LED is flashing, the power supply must not be switched off, nor can commands be sent to the EIB 8791 via the Ethernet interface (the EIB 8791 blocks commands as long as the update procedure is running).

1.13 DHCP

The EIB 8791 can work with static IP addresses or with dynamic IP addresses that are obtained from a DHCP server. By default, DHCP is deactivated and the EIB 8791 uses static IP addresses. This address can be set by the user in order to conform to the requirements of a specific network (the factory default address is 192.168.168.2).

If DHCP is activated, then after it has booted, the EIB 8791 tries to obtain an IP address from a DHCP server. This address is used until the duration of validity shown under “Lease” expires. If necessary, the EIB 8791 can renew the lease by itself. If, during booting, no DHCP server can be found to make an address available before the timeout expires, the EIB 8791 uses the static IP address set by the user. If DHCP is selected, but no DHCP server is available, then booting takes longer. The DHCP client requests an IP address, the subnet mask, and the default gateway.

If an address was obtained via DHCP at the beginning (during booting), then an address must again be obtained via DHCP when the lease time expires!
Possible remedy -> infinite lease time.

In addition, the host name of the EIB 8791 is transmitted to the DHCP server. If the DHCP server is connected to a DNS server, then the host name can be used instead of the IP address for communication with the EIB 8791. The default host name is distinct for each EIB 8791 and contains the device name along with a unique serial number.

Example of a host name: EIB 8791   123456789
The device name is "EIB 8791" and the serial number is "123456789". The serial number is printed on the ID label on the rear of the EIB 8791. The host name can be changed by a software command.

Note:
To change the network settings, you can use, for example, the "EIB8 Application" program provided on the CD (see Section 3.3).

2 Configuring and using the EIB 8791 with the driver software

2.1 General information

Functions are provided for accessing the EIB 8791 from a software application. This group of functions is supplied as DLL for Windows systems and as SO library for Linux. The following operating systems are supported:

- Windows 7
- Linux/Unix with kernel 2.6 – 64-bit (i386 systems)

For Windows, the library is available in a 32-bit and a 64-bit version. Use the 32-bit library for a 32-bit software application, and the 64-bit library for a 64-bit software application. For Linux, only the 64-bit version is provided.

In addition to the libraries, a header file that enables the functions to be integrated into C/C++ programs is also supplied. To create a program, you must incorporate the library into the project.

For initial tests with the EIB 8791, you can use the supplied command-line tools EIB8AppXX.exe for Windows and EIB8App for Linux. These tools can be run without installation (see Section 3.2).

Virtual instruments (VIs) that are based on the Windows DLL are provided for LabVIEW™ (Version 2012, Service Pack 1, 32-bit). The designation, functionality, and input/output parameters of the VIs are oriented to the respective function calls, which are documented below. Particularly for complex data types, a VI may need an adaptation specific to LabVIEW™, along with the DLL call. The corresponding adaptations are apparent when the VI is opened.
In addition, the EIB 8791 Application (EXE and VI) implemented in LabVIEW™ is also supplied and can be used for operating the EIB 8791 (see Section 3.3).

2.2 Contents of the driver CD

The driver CD contains the following directories and files:

Directory	Files	Description
\bin\32Bit	EIB8Driver32.dll	EIB 8791 driver DLL for Windows 32-bit
	EIB8Driver32.lib	EIB 8791 driver LIB for Windows 32-bit
	EIB8LogServer32.exe	Logging server for capturing logging messages of the driver
	EIB8TestApp32.exe	Test application for special applications
	EIB8App32.exe	Command-line tool that uses the driver
\bin\64Bit	See \bin\64Bit	Files for Windows 64-bit
\include	eib8_api.h	Include file that is integrated by the software application.
	eib8.h	Declaration of all driver functions and data types for standard applications
	*.h	Additional header files for special applications (must always be available together with eib8_api.h)
\doc	html.zip, Operating Instructions	Driver interface documentation: contains all of the information and safety precautions needed for the proper operation of the product in accordance with its intended use.
	Release Notes	Change history of the driver
	User's Manual	Driver software user's manual: contains all of the information regarding the range of functions, installation, and function calls of the driver software.
	EIB8_Parameter.xlsx	Table of all parameters of the EIB 8791
\example	*.c	Example programs for using the driver in a C/C++ application (see Section 3)
	*.txt	Example configurations that can be transmitted to the EIB 8791 (see Section 3)
\labview\32Bit\EIB8Driver\	EIB8ApiVI\*.*	LabVIEW™ Vis, with which the driver's standard functions can be called.
	ExampleVI \*.*	Example VIs for operating an EIB 8791
	HelperSubVI \*.*	Helper VIs for the examples and the LabVIEW™ application
	ApplicationVI \*.*	VI of the LabVIEW™ EIB 8791 Application
\labview\32Bit\EIB8Application	\EIB8 Application EXE\*.*	LabVIEW™ EIB 8791 Application as an executable file (exe). If LabVIEW™ is available on the PC, EIB8Application.exe can be started directly.
	\EIB8 Application Setup\*.*	Setup of the LabVIEW™ EIB 8791 Application, with installation of a free runtime version of LabVIEW™
\linux\64Bit\bin	EIB8App	Command-line tool that uses the driver
	EIB8LogServer	Logging server for capturing logging messages of the driver
	libEIB8Driver.so	EIB 8791 driver library for Linux 64-bit
\linux\64Bit\include	See \include for Windows	

2.3 Installation instructions for the driver DLL

The stated directories and files refer to the driver CD for the EIB 8791.

2.3.1 Windows

Before an application can load the DLL, the EIB8Driver32.dll or EIB8Driver64.dll file must be copied to the Windows system directory (e.g., "C:\Windows\system32").

For compatibility with 32-bit applications, the file EIB8DriverXX.dll should additionally be copied to the "SysWOW64" system directory in the Windows folder (e.g., "C:\Windows"). As an alternative, the path for the DLL can be specified in the system, or the DLL can be copied to the directory of the application.

The interface of the DLL is defined by the two files "EIB8DriverXX.lib" and "eib8_api.h". For C/C++ projects, the header file must be integrated as follows:

#include "<path>eib8_api.h"

In this way, eib8_api.h includes additional header files that must be available under the same path (all files that are located in the "\include" directory of the driver CD). The actual interface of the DLL is declared in "eib8.h".

In addition, the "EIB8DriverXX.lib" library must be made known to the development environment. An additional library can usually be specified in the linker area (e.g., under "linker" – "input" – "additional dependencies").

2.3.2 **Linux**

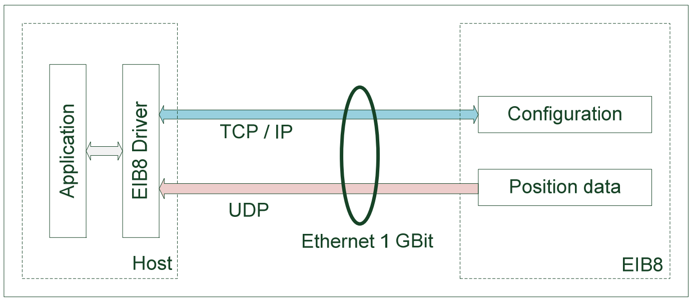
Before an application can load the SO library, the "libEIB8Driver.so" file on the CD must be copied to the "usr/lib" directory. The library interface is defined by the "eib8_api.h" file. This file should be copied to "usr/local/include" together with the other header files from the driver CD ("\include") and must be integrated into the software project in the development environment. The stated directories are based on the "Filesystem Hierarchy Standard" for Linux operating systems. The "libEIB8Driver.so" library was compiled for i386 systems under kernel 2.6, 64-bit.

The EIB8App command-line tool was also compiled for i386 systems under kernel 2.6, 64-bit. It can be started directly from the command line if "libEIB8Driver.so" is located in the directory given above.

2.4 Establishing an Ethernet connection to the EIB 8791

The EIB 8791 is connected to the host via Ethernet. Commands from the host to the EIB 8791 are transmitted via TCP/IP. In addition, UDP can be used to transfer position data from the EIB 8791 to a host.

Note:
The position data are usually transferred to the host that configures the EIB 8791. However, it is also possible to transfer the position data to a second host. For this purpose, the respective hosts need to be connected to the EIB 8791 via a switch.



The factory default IP address of the EIB 8791 is 192.168.168.2. To enable the host to connect to the EIB 8791, the host's IP address must be in the same subnet with the subnet mask 255.255.255.0. The host can be set to 192.168.168.100, for example.

To test the connection, you can use the EIB8AppXX.exe or EIB8App command-line tool provided on the CD (see Section 3.2). The command *c* (connect) tries to establish a connection with the default IP address. The command *i*, for example, can then be used to request some information from the EIB 8791. The *ion* command sets the LAN LED of the EIB 8791 to blinking mode, and *ioff* disables it again.

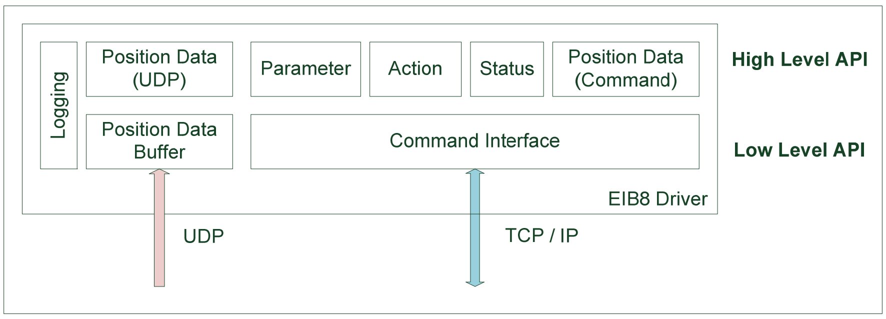
You can then use the *ipeib* command to assign the desired IP address to the EIB 8791. This change does not become effective until you restart the EIB 8791 (*reset* command). Before you can then reconnect with the command-line tool, you need to set the new IP address in the command-line tool (*iphost* command).

Note:
If the IP address of the EIB 8791 is unknown, it is recommended that you start the EIB 8791 in factory mode with the factory settings (hold down the reset switch for at least 10 seconds). This resets the IP address to 192.168.168.2. You can then assign the desired IP address for the user mode as described above.

Another easy way to perform tests with the EIB 8791 is to use the LabVIEW™ EIB 8791 Application (see Appendix 3.3).

2.5 Overview of the EIB 8791 driver DLL

2.5.1 **Driver structure**



The driver uses a wide range of commands to communicate with the EIB 8791. The individual commands are available via the driver's low-level API (declarations in device_cmd.h, device.h, err_int.h), but they are not recommended for standard applications. It is generally preferable to use the abstracted high-level API, which is defined in eib8.h. The description below refers only to the high-level API.

The EIB 8791 transfers position data via the UDP interface. The data are not buffered in the EIB 8791 but is rather transmitted as quickly as possible. The driver is responsible for buffering the position data. The application can use the applicable functions to configure the buffer and access the position data.

Position data can also be queried through the command interface, but only at a rate of approx. 10 Hz.

The logging functionality is especially used to support troubleshooting during operation of the driver. Depending on the logging level, error messages or the entire data traffic with the EIB 8791 can be logged. The logging messages can optionally be output to a file, to the command line, or on UDP. Logging messages that are output on UDP can be displayed with the following logging servers supplied with the CD: "EIB8LogServerXX.exe" for Windows or "EIB8LogServer" for Linux.

Remark:
The driver does not support the reception of position data via the PDL interface, for which special hardware is required.

2.5.2 **Function calls**

The driver's function calls are usually blocking. The return code of the *eib8_err_t* type indicates whether the function call was successful (*EIB8_ERROR_OK*, corresponds to the value 0) or whether an error occurred.

The return code can be converted into a readable string with the
eib8_get_last_error()
function.

2.5.3 **Strings**

For convenient use of the functions, the driver frequently uses strings. To ensure that the memory area is not exceeded, return strings (e.g., for the *eib8_get_driver_version*() function), in particular, are always declared in the format of the *eib8_std_str_t* data type, which has a fixed maximum length.

2.5.4 **Basic functions**

This section explains the basic functionality of the driver. For details on the described functions and data types, please refer to the interface definition (eib8.h) or the supplied HTML documentation.
The "example_identify" example on the driver CD shows how to use the commands described below.

2.5.4.1 **Opening and closing the driver**

Before you call a function of the driver, the driver must be initialized with the following function:
eib8_driver_open()

Before you exit the application, you should close the driver with the
eib8_driver_close()
function to ensure that the driver frees the allocated memory. When the function is called, all EIB 8791 connections are disconnected and all handles are released.

If a driver has already been opened in an application and you want to reopen it with *eib8_driver_open()*, you need to close it first with *eib8_driver_close()*.

2.5.4.3 Version of the driver

The driver version can be queried with the following function:

eib8_get_driver_version()

Example:

2.1.13, Date: Sep 30 2014 12:35:52 [release32], JH-ID: 1065235-02-A-01

2.5.4.4 Logging function of the driver

The logging function of the driver is disabled after the driver is opened. Logging messages can be written to the command line (stdout)

eib8_activate_logging_stdout()

or to a file:

eib8_activate_logging_file()

Logging messages can also be transmitted as UDP packets (recommended method):

eib8_activate_logging_udp()

Example:

eib8_activate_logging_udp(EIB8_DRIVER_LOG_LEVEL_WARN, "127.0.0.1", 3000u)

In this example, the messages are transmitted to the "internal IP" 127.0.0.1 and port 3000. To make the logging messages visible, launch the EIB8LogServerXX.exe log server on the computer to which the logging messages are transmitted (with IP 127.0.0.1, it is the computer on which the application is running):

EIB8LogServerXX.exe 3000

Logging level:
For all three methods, the logging level (*eib8_driver_log_level_t*) must be transferred in order to specify, for example, whether only errors should be output, or also warnings as well.

Recommended logging levels:

Logging level	Meaning
EIB8_DRIVER_LOG_LEVEL_NON	Logging is deactivated.
EIB8_DRIVER_LOG_LEVEL_WARN	All warnings and errors are output.
EIB8_DRIVER_LOG_LEVEL_INFO2	All warnings and errors are output. The entire communication with the EIB 8791 is logged.

2.5.4.5 EIB 8791 handles

The driver can communicate with up to 256 EIB 8791 devices (*EIB8_MAX_HANDLES*) simultaneously. For this purpose, use the function

eib8_get_handle()

to request a handle for each EIB 8791 from the driver. The handle representing an EIB 8791 is subsequently transferred with each function call that communicates with an EIB 8791.

The function

eib8_release_handle()

can release a handle again. The TCP connection to an EIB 8791 is then automatically disconnected.
An invalid handle has the value *EIB8_INVALID_HANDLE_ID* (0).

Note:
The driver has been implemented as thread-safe. This means that multiple threads of the application can access one or more handles simultaneously.
If a handle is accessed a second time while it is still executing a function, the second access is acknowledged with the error *EIB8_ERROR_EIB8_CMD_BUSY*, and the function is not executed. The first access is not affected by this.

2.5.4.6 Connecting and disconnecting the EIB 8791

To communicate with the EIB 8791, use the
eib8_connect_tcp()
function to establish a TCP/IP connection.

Example using the factory default settings of the EIB 8791:
eib8_connect_tcp(&myEIB8handle, "192.168.168.2", 1050, &CheckResult)

Upon successful execution of the function, the handle is connected to the respective EIB 8791.
During connecting, the driver checks the basic system states of the EIB 8791 and provides them to the application in the
eib8_connect_check_result_t
structure. The elements have the following meanings:

Entry	**Meaning**	**Action**
u32BusyUpdate	The EIB 8791 is running a software update	Wait and try again later (up to 10 min)
u32ErrorUpdate	An error occurred during the most recent update	Repeat the update. If the error recurs, please contact HEIDENHAIN's encoder support.
u32FactoryMode	The EIB 8791 is running in factory mode	The range of functions of the EIB 8791 is not completely available. If the user did not select this mode, then the most recent software update was interrupted, or an error occurred in flash memory. Repeat the software update. If the error recurs, please contact HEIDENHAIN's encoder support.
u32ErrorHardware	The EIB 8791 detected a hardware error	Restart the EIB 8791. If the error recurs, please contact HEIDENHAIN's encoder support.
u32ErrorBistDiag	The EIB 8791 detected an error during self-test	Read out the event queues (see Section 2.8.2).
u32PendingEvents	There are entries in the event queues of the EIB 8791	Read out the event queues (see Section 2.8.2).
u32IncompDriver	The driver is incompatible with the EIB 8791	Install a new driver version (driver version must be greater than or equal to the EIB 8791 software version).

If an EIB 8791 is already connected, the basic system states can be updated in the application at any time with the
eib8_connect_result_update()
function (the system states can change due to the integrated diagnostic functionality of the EIB 8791).

To disconnect the TCP connection of an EIB 8791, use the
eib8_disconnect_tcp()
function (also performed automatically when the handle is released or the driver is closed).

2.5.4.7 "Identify" to the EIB 8791

To check the connection to the desired EIB 8791, you can use the
eib8_identify()
function to set the LAN LED on the front panel of the EIB 8791 to blinking mode.

Example (see also: example_identify.c in the example programs):
Activate blinking mode:
eib8_identify(&myEIB8handle, "EIB8", EIB8_ENABLE)

Deactivate blinking mode:
eib8_identify(&myEIB8handle, "EIB8", EIB8_DISABLE)

2.6 Parameters

2.6.1 Nomenclature

The EIB 8791 driver provides an interface for exchanging parameters between the EIB 8791 and the host in a standardized way.

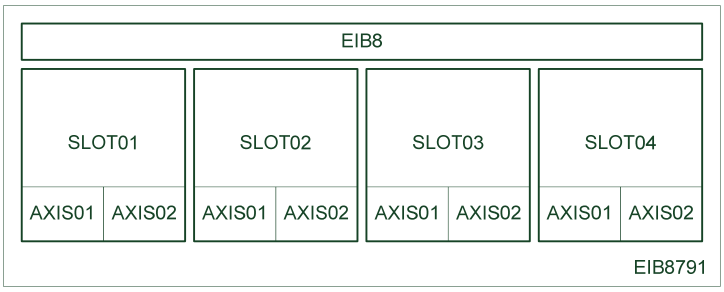
The following nomenclature applies:

Name	Meaning
Configuration	Parameters that are transmitted to the EIB 8791 and can be read from the EIB 8791 (R/W parameters)
Status	Parameters that can be read only from the EIB 8791 (R parameters)
Parameter	Configuration and status

A parameter is defined as follows (regardless of whether it is a configuration parameter or a status parameter):

Designation	Meaning	Example
Node	Node	"EIB8"
Param	Parameter	"network_user:ip_address"
Value	Value	"192.168.168.2"

Node:
The node designates the component in the EIB 8791 to which the parameter applies. The following nodes are available in the EIB 8791:



Examples of nodes:

Node	Meaning
"EIB8"	EIB8 basic component
"SLOT01"	Slot 1
"SLOT00"	All slots
"SLOT02_AXIS01"	Axis 1 in slot 2
"SLOT00:AXIS00"	All axes in all slots

A parameter can be transmitted to multiple nodes at the same time via multi-addressing with "SLOT00" or "AXIS00".

Parameter:
A parameter is defined by its parameter group and its parameter name. The parameter's group and name are separated by a colon ":".

Example:
"network_user:ip_address"
"network_user" is the parameter group and "ip_address" is the parameter name.

Value:
The value of the parameter is the result when a parameter is read, or it is the nominal value when a parameter is written.

Example of an IP address:
"192.168.168.2"

2.6.2 **Parameters of the EIB 8791**

All parameters available in the EIB 8791 are listed with a brief description in the "EIB8_Parameter.xlsx" table in the "doc" directory on the driver CD.

An up-to-date list can also be created with the driver (see Section 2.6.3). With the command-line tool and a connected EIB 8791 (see Section 2.4), you can use the *pg* ("parameter get") command to read out the entire parameter list from the EIB 8791. Likewise, the entire configuration (*cg*) or the complete status (*sg*) can be read from the EIB 8791.

Note:
The EIB 8791 provides a wide range of parameters that are relevant only to special applications. This section deals primarily with the parameters that are used in general.

2.6.3 **Reading and writing of parameters**

The driver provides various functions for reading and writing parameters. The following basic distinction is made:

- Reading or writing of a single parameter
- Reading of all parameters and output as a file or list
- Writing of a configuration list: Input via file or list

For files, you can choose between a simple text format or the XML format. The text format is preferable because it is clearer and easier to read.

The following table shows an overview of the functions for reading and writing parameters:

Function	**Meaning**

eib8_set_param_string()
eib8_get_param_string()
	Read or write a parameter; the value is a string (always possible, including in the case of numerical values).

eib8_set_param_bool()
eib8_get_param_bool()
	Read or write a parameter; the value is Boolean (0 or 1).

eib8_set_param_hex32()
eib8_get_param_hex32()
	Read or write a parameter; the value is an unsigned 32-bit integer.

eib8_set_param_int32()
eib8_get_param_int32()
	Read or write a parameter; the value is a signed 32-bit integer.

eib8_get_config_all_file()
eib8_get_config_all_list()
	Read the entire current configuration of the EIB 8791 and write it to a file or list.

eib8_get_status_all_file()
eib8_get_status_all_list()
	Read the entire current status of the EIB 8791 and write it to a file or list.

eib8_get_parameter_all_file()	Read all parameters from the EIB 8791 and write them to a file.

eib8_set_config_file()
eib8_set_param_list_string()
	Write a configuration to the EIB 8791: Input via file or list.

Examples:
Writing a configuration file to the EIB 8791:

eib8_set_config_file(&myEIB8Handle, "config_std_EIB 8791.txt", EIB8_FILE_TXT)

where *"config_std_EIB 8791.txt"* is the file name, and "*EIB8_FILE_TXT"* specifies that the file is in text format.

Writing the IP address:

eib8_set_param_string(&myEIB8Handle,
"EIB8", " network_user:ip_address ", "192.168.168.2")

Writing the internal PTM frequency (10 kHz):

eib8_set_param_int32(&myEIB8Handle, "EIB8", "trig_ptm_in:freq_config", 10000)

A detailed description of how to use the functions is provided in the interface definition ("eib8.h"). Examples are also available in the "example" directory of the driver CD.

2.6.4 **Configuration files**

An example configuration file is provided on the driver CD: config_std_EIB 8791.txt.

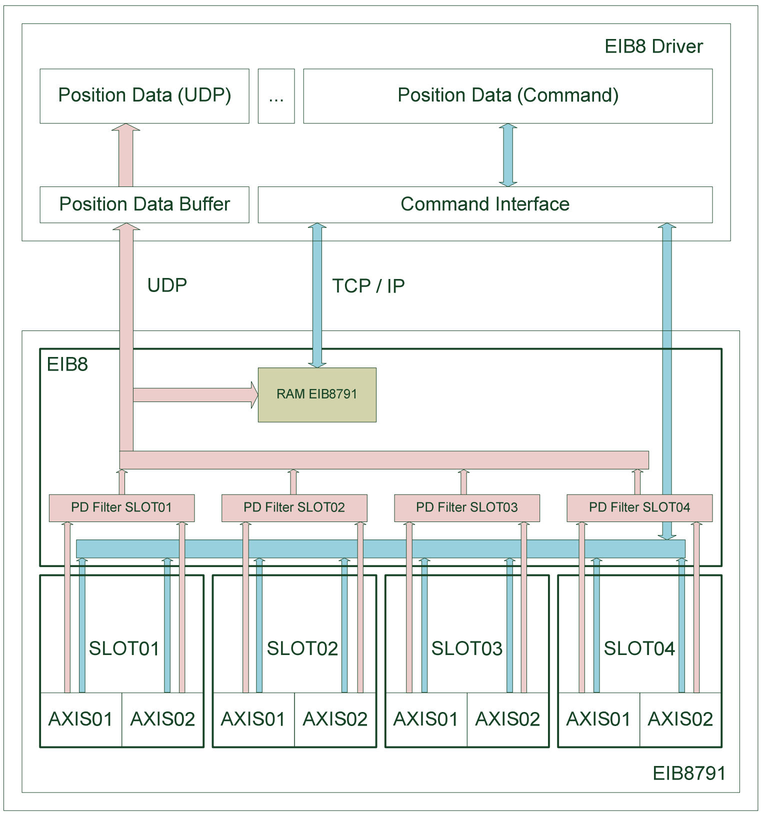
Please note the following information regarding the format of the configuration files:

- Each configuration entry is given in a separate line in the format:
  node ; parameter ; value ;
  The node, parameter, and value must be separated by a semicolon ";". Additional spaces between them are allowed.
- Every line that is not a configuration entry (comments, blank lines) must be preceded by a "#" character.
- The configuration entries can be made in any order. The driver sorts them in the order in which they are output with *eib8_get_config_all_file()*.

2.7 PD position data

2.7.1 **Data path**

Position data are generated individually for each axis when the respective trigger event occurs at the axis.



The EIB 8791 can output PD position data in different ways:

- Polling mode (blue data path):
  The position data are retrieved from the EIB 8791 by commands. Update rates of approx. 10 Hz can be achieved, depending on the application.
- UDP mode (red data path):
  The position data are transferred to a host via UDP protocol. The destination host is determined from its MAC address and IP address as well as from the port to which the UDP packets are transmitted. The destination of the UDP transfer does not necessarily have to be the host that transmits commands to the EIB 8791. It is possible, for example, to connect the EIB 8791 to two PCs via a switch, and to have one PC transmit the commands to the EIB 8791 while the other PC receives position data via UDP.
  The EIB 8791 is able to transfer position data at a trigger rate of up to 400 kHz. The host must be checked to ensure that it can process the selected data rate.
- RAM recording mode
  Position data can also be recorded in the RAM of the EIB 8791 and retrieved via the command interface upon completion of recording. Up to 10 million single positions (position data packets) can be stored.

Notes:

- A maximum of 32 different types of position data packets can be configured for transfer via UDP or for storage in the EIB 8791's RAM.
- Position data packets can either be transferred via UDP or stored in RAM, but not both at the same time. Position data, in contrast, can be queried via the command interface at any time (even while position data are being transferred via UDP or written to RAM).

2.7.2 **Position data format**

The position data are transferred or stored in a 12-byte position data packet, regardless of their type:

Byte no.	Meaning	Value range	Note
0	Packet number	1 – 255	The packet number for identifying the position packet (*u32PacketNumber*)
1	Frame counter	0 – 255	Counter that is incremented by 1 with every trigger event (*u32FrameTransCnt*). The frame counter is always 0 in polling mode.
2	Payload 0	$0 – (2^{64} - 1)$	64-bit value (payload) that contains the actual position value or the user data, depending on the position data type. The value is transferred in little-endian format (i.e., the least significant byte first). (*u64Value*)
3	Payload 1		
4	Payload 2		
5	Payload 3		
6	Payload 4		
7	Payload 5		
8	Payload 6		
9	Payload 7		
10	Filler byte		Filler byte always with the value 0xBC
11	Checksum		8-bit CRC over bytes 0 to 9. Calculation in accordance with the formula ITU $x^8 + x^2 + x + 1$

All driver functions that receive or retrieve position data packets use the following data structure (eib8.h):

```
typedef struct
{
        uint64_t   u64Value64;
        uint32_t   u32PacketNumber;
        uint32_t   u32FrameTransCnt;
        uint32_t   u32ChecksumError;  /* 0: checksum OK, 1: checksum fail */
        uint32_t   u32Dummy;          /* Due to 64-bit alignment on various platforms */
} eib8_pdl_packet_t;
```

With this, the checksum is already evaluated by the driver.

2.7.3 **Position data types**

The position of the encoder is usually transferred with the abovementioned position data packet. In addition, the packet can be configured in such a way that the 64-bit value takes on a different meaning. In this way, several position data packets can be created and transferred for an axis. If a trigger event is detected at an axis, all configured position data packets of this axis are transmitted.

2.7.3.1 **Position**

The *position* data type has the following structure in the position data packet:

Byte no.	Meaning	
Payload 0	16-bit status	Bits 0 – 3: Reserved Bit 4: Limit signal (L2) active Bit 5: Homing signal (L1) active Bit 6: Reference mark valid Bits 7 – 13: Reserved Bit 14: Event in event queue Bit 15: Position error
Payload 1		
Payload 2	16-bit phase	48-bit position
Payload 3		
Payload 4	32-bit period counter	
Payload 5		
Payload 6		
Payload 7		

The position is given as a 48-bit value in the upper six bytes of the payload. The upper four bytes indicate the value of the period counter, while the lower two bytes indicate the phase of the incremental signal (interpolation value within the signal period).

In the case of position packets, it must particularly be checked whether the position is valid (bit 15 in status: Position error). If the position is invalid—for example, because no encoder was initially connected—then the position can be set to valid again only via the

eib8_clear_position_error()

command.

The status also provides the following information:

- Event in event queue
  Events have occurred in the slot. The event queue should be queried (see Section 2.8.2).
- Reference mark is valid:
  The bit is set if a reference mark search was successful and the reference position is valid.
- Homing signal:
  Status of the homing signal
- Limit signal:
  Status of the limit signal

For converting a general position data packet of the *eib8_pdl_packet_t* type, the driver provides a function that extracts the position and status:

eib8_pdl_convert_pos_IDP8791()

The position value is given as a 64-bit value whose lower 16 bits are set to zero.

2.7.3.2 **Speed**

The "speed" data type has the following structure in the position data packet:

Byte no.	**Meaning**	
Payload 0	16-bit status	Bits 0 – 3: Reserved Bit 4: Limit signal (L2) active Bit 5: Homing signal (L1) active Bit 6: Reference mark valid Bits 7 – 13: Reserved Bit 14: Event in event queue Bit 15: Position error
Payload 1		
Payload 2	48-bit speed	Signed 48-bit value Unit $\left[\frac{\text{Signalperioden}}{2^{22} \cdot \mu\text{s}}\right]$
Payload 3		
Payload 4		
Payload 5		
Payload 6		
Payload 7		

The status corresponds to Section 2.7.3.1 (position). Accordingly, the packet can be converted with *eib8_pdl_convert_pos_IDP8791()*. The unit of speed can be converted with *eib8_pdl_convert_velocity_IDP8791* from $\left[\frac{\text{Signal Periods}}{2^{22} \cdot \mu\text{s}}\right]$ into signal periods per second.

2.7.3.3 Reference mark

The "reference" data type has the following structure in the position data packet:

Byte no.	Meaning	
Payload 0	16-bit status	Bits 0 – 3: Reserved Bit 4: Limit signal (L2) active Bit 5: Homing signal (L1) active Bit 6: Reference mark valid Bits 7 – 13: Reserved Bit 14: Event in event queue Bit 15: Position error
Payload 1		
Payload 2	16-bit phase	Always 0x00
Payload 3		
Payload 4	32-bit period counter	Reference position for encoders with a single reference mark Coded reference position for encoders with distance-coded reference marks
Payload 5		
Payload 6		
Payload 7		

The status corresponds to Section 2.7.3.1 (position). Accordingly, the packet can be converted with *eib8_pdl_convert_pos_IDP8791()*.

2.7.3.4 Other position data packets

Type	Meaning	Application
Constant value (*constant*)	A constant 64-bit value that is transferred as a position data packet can be configured individually for an axis.	Testing of the data transfer
Incremental value (*incremental*)	A 64-bit value that is incremented individually for each axis every time a trigger event occurs.	Testing of the data transfer / triggers
Time stamp (*timestamp*)	A 64-bit value that returns the time stamp of the trigger event on which the data packet was triggered. The time stamp is reset to 0 when the EIB 8791 is started, and incremented in a 10 MHz grid.	Time stamp of the trigger event, temporal assignment of the position data packet
Analog values (*adc*)	The 64-bit value contains four analog values: Payload 0/1: Channel A Payload 2/3: Channel B Payload 4/5: Channel C (if available) Payload 6/7: Reference pulse (if available)	Examination of the analog input signals before the filter chain

2.7.3.5 Packet configuration

The packet number of a position data packet is defined during the configuration of the EIB 8791. In the process, a packet number is assigned to a position data type at a specific axis.

Example:

Packet number	Axis	Type	Configuration
0x11	Slot 1, axis 1	Position	*SLOT01;pdl_packets:axis_1;0x11:position*
0x12 0x13	Slot 1, axis 2	Position and constant value	*SLOT01;pdl_packets:axis_2;0x12:position, 0x13:adc*

Three position data packets are created in this example: 0x11, 0x12, 0x13. In the application, position data packets are received and need to be interpreted and processed in accordance with the packet number.

To ensure that the packet numbers are unique, each packet number should be assigned only once in a device.

2.8 Standard use cases

This section illustrates the driver's function calls and parameters by describing use cases. As explained in Section 2.5.4, you first open the driver and connect the EIB 8791. You can then configure the EIB 8791 in order to perform measurements.

Note:
The examples provided on the driver CD illustrate how to use the following functions.

2.8.1 **Reset – Shutdown of the EIB 8791**

One way to initiate a hardware reset in the EIB 8791 is to press the reset switch on the front panel. The preferred way, however, is to use a command to initiate the hardware reset because this enables the EIB 8791 to complete internal processes. For the same reason, the EIB 8791 should be shut down with the shutdown command before it is switched off. The driver offers the following functions for this purpose:

eib8_reset()
eib8_shutdown()

2.8.2 **Querying the event queues**

In the EIB 8791, the EIB8 and SLOTXX nodes have event queues. There, error messages or warnings that occur due to internal diagnostics or self-test functions are logged (if an error occurs during the execution of a command, the error is reported with the acknowledgment of the command and handled in the driver via the return parameter of the function).

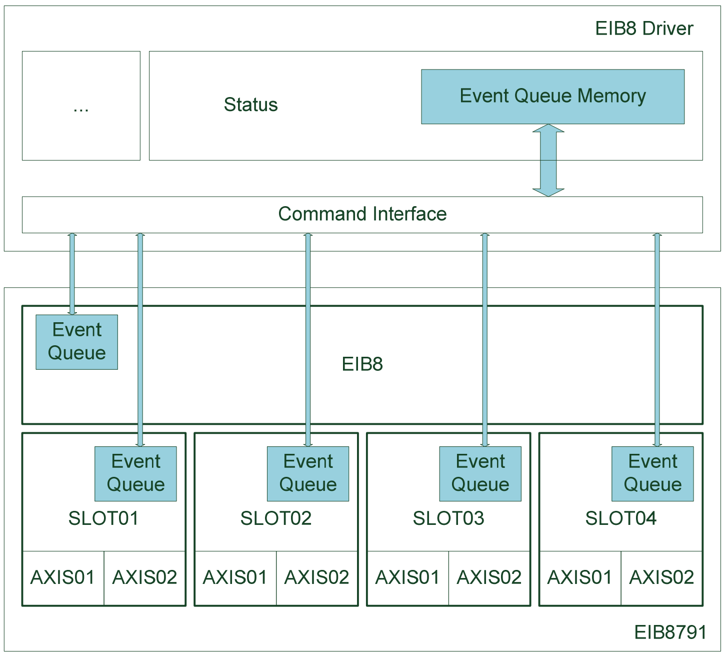
If an error occurs multiple times, it is written to the event queue only once. If the queue was emptied (i.e., queried), however, pending errors are added to the event queue again.

Event queue entries are distinguished between warnings and errors. In the case of errors, the application must respond immediately. Warnings do not have to be handled immediately, but they can subsequently lead to errors (e.g., temperature exceeds warning threshold, temperature exceeds error threshold).

It is important that the application query the event queues cyclically (e.g., once per second) in order to ensure the early detection of faults in the device.

Notes:

- When the driver connects to the EIB 8791, it already determines whether there are events in the event queues (u32PendingEvents). The events are not queried yet at that point, however.
- The status in the position data packets contains the information "Event in event queue" (see Section 2.7.3.1)



The driver provides functions that facilitate the querying of event queues. The function

eib8_check_event_queues()

queries all event queues in the EIB 8791 and stores them in the driver's event queue memory (this clears the event queues of the nodes). The function returns the number of all warnings and errors determined. The next time that *eib8_check_event_queues()* is called, the driver's event queue memory is cleared, and the event queues of the nodes are queried again.

If the event queue memory contains events, then they can be retrieved individually by the application with the

eib8_get_event_queue_entry()

function (when the events are read, they are deleted in the event queue memory). During this process, the driver converts the error codes into readable strings.

As an alternative, the entire event queue memory of the driver can be written to a text file:

eib8_event_queue_to_txt()

The event queue memory of the driver is not cleared in this case.

2.8.3 Loading a default configuration

The driver offers the possibility of transmitting a default configuration to the EIB 8791, allowing measured values to be immediately generated and queried.

eib8_set_default_conf_EIB 8791()

With the function call, the MAC address and IP address, as well as the port for the output of position data, must already be specified via UDP.

Example:

eib8_set_default_conf_EIB 8791
(&myEIB8Handle, "90.e2.ba.03.9c.eb", "192.168.168.100", 3051)

If no UDP transfer is to be configured, then the parameters can also be set as follows:

eib8_set_default_conf_EIB 8791(&myEIB8Handle, "", "", 0)

The default configuration provides the following settings:

No.	Configuration	Explanation
1	*EIB8;trig_ptm_in:source;internal EIB8;trig_ptm_in:freq_config;1000*	Use of the internal PTM trigger at a frequency of 1000 Hz
2	*EIB8;trig_bus1_out:inputs;trig_ptm_in EIB8;trig_bus1_out:enable;1 EIB8;trig_bus1_out:transmitter;1*	Internal trigger routing in the EIB8 node (PTM as trigger)
3	*EIB8;pdl_tx_from_slots_to_lane_1:slot_1;1 EIB8;pdl_tx_from_slots_to_lane_1:slot_2;1 EIB8;pdl_tx_from_slots_to_lane_1:slot_3;1 EIB8;pdl_tx_from_slots_to_lane_1:slot_4;1*	Internal routing of position data in the EIB 8791 (should not be changed in standard applications)
4	*EIB8;pdl_forwarding_ram_udp:slot_1;0x11,0x12 EIB8;pdl_forwarding_ram_udp:slot_2;0x21,0x22 EIB8;pdl_forwarding_ram_udp:slot_3;0x31,0x32 EIB8;pdl_forwarding_ram_udp:slot_4;0x41,0x42*	Position data filter for packet numbers: Slot 1: 0x11, 0x12 Slot 2: 0x21, 0x22 Slot 3: 0x31, 0x32 Slot 4: 0x41, 0x42
5	*EIB8;udp_transfer:udp_dest_mac;90.e2.ba.03.9c.eb EIB8;udp_transfer:udp_dest_ip;192.168.168.100 EIB8;udp_transfer:udp_dest_port;3051*	MAC address, IP address, port for UDP transfer
6	*EIB8;pdl_forwarding_ram_udp:mode; batch_soft_realtime*	Position data path for UDP transfer
7	*SLOT00;trig_sync1_in:input;trig_bus1_in SLOT00;trig_bus1_in:enable;1 SLOT00;trig_sync1_in:enable;1 SLOT00:AXIS00;trigger:inputs;trig_sync1_in*	Internal trigger routing in the slots and axes, all axes identical (PTM as trigger)
8	*SLOT01;pdl_packets:axis_1;0x11:position SLOT01;pdl_packets:axis_2;0x12:position SLOT02;pdl_packets:axis_1;0x21:position SLOT02;pdl_packets:axis_2;0x22:position SLOT03;pdl_packets:axis_1;0x31:position SLOT03;pdl_packets:axis_2;0x32:position SLOT04;pdl_packets:axis_1;0x41:position SLOT04;pdl_packets:axis_2;0x42:position*	Configuration of the position data packets: Each axis supplies a packet of type "position". (The packets defined here must be entered in the position data filter)
9	*SLOT00:AXIS00;encoder_config:type;linear SLOT00:AXIS00;encoder_config:interface;1V_pp_0_90 SLOT00:AXIS00;encoder_config:reference_type;single SLOT00:AXIS00;encoder_config:line_cnt;0 SLOT00:AXIS00;encoder_config:ref_increment;0 SLOT00:AXIS00;encoder_config:limit_signal_1_present;0 SLOT00:AXIS00;encoder_config:limit_signal_2_present;0 SLOT00:AXIS00;encoder_config:homing_signal_present;0 SLOT00:AXIS00;encoder_config:limit_signal_2_present;0*	Same encoder configuration for all axes (see Section 2.9.3)
10	*SLOT00:AXIS00;encoder:supply_enable;1*	Activate encoder supply for all axes
11	*SLOT00:AXIS00;encoder_processing:online_comp_enable;1*	Activate online compensation for all axes

No.	Configuration	Explanation
12	*SLOT00:AXIS00;pos_value_filter:active;1* *SLOT00:AXIS00; pos_value_filter:characteristic;linear* *SLOT00:AXIS00; pos_value_filter:bandwidth;high* *SLOT00:AXIS00; pos_value_filter:measure_time;0*	Activate position value filter
13	*SLOT00:AXIS00;pdl:output_active;1*	Activate the creation of position data packets for all axes

Notes:

- When the eib8_set_default_conf_EIB 8791() function is used for setting the default configuration, the driver ensures that the frame counters for all axes start at 0.
- In the programming examples, the configuration is set using a configuration list (eib8_set_param_list_string()). In this case, the respective MAC and IP addresses as well as the respective port for the output of position data via UDP must be specified.
- The default configuration can also be set by loading the "config_std_EIB 8791.txt" example configuration from the driver CD with the eib8_set_config_file() function. The only thing to keep in mind is to specifically activate the PTM trigger after loading the configuration file.
- The default configuration is set for the output of positions for all eight axes. If no encoder is connected to an encoder input, an event is created because the signal amplitudes are too small.
  To prevent the output of events from unconnected encoder inputs, you need to either completely remove the respective axes from the overall configuration (e.g., configuration file) or at least set the following parameters:
  – *SLOT0X:AXIS0X;trigger:inputs;*
  – *SLOT0X:AXIS0X;encoder:supply_enable;0*

2.8.4 Resetting the configuration

After the EIB 8791 is switched on, the configuration is in the default state (i.e., no position data, triggers, etc. are generated or output). Once configured, the EIB 8791 can be reset to its default state with the

eib8_reset_idle()

function without the need to perform a hardware reset.
Executing the *eib8_reset_idle()* function is recommended, in particular, when you want to reconfigure the device.

Note:
eib8_set_default_conf_EIB 8791() automatically executes *eib8_reset_idle()*. As you can also see in the programming examples, *eib8_reset_idle()* is always called before a new configuration is loaded.

2.8.5 Polling of position data

Position data can be queried in polling mode regardless of whether data are being transmitted via UDP or recorded in RAM in the EIB 8791.

The following is required for this:

- Trigger configuration: Triggering of position data packets
- Configuration of position data packets
- Configuration of encoders and activation of encoder supply
- Creation of position data packets activated

The function

eib8_poll_pdl_packets()

supplies the driver with a list of all position data (for the format, see 2.7.2) that are configured in the device (the *pNode* input parameter must be set to *SLOT00*).

Notes:

- The default configuration (see above) meets all of the requirements for querying position data in polling mode.
- If no trigger event occurs, the position (including the other position data types) of the last trigger event is output.
- The "example_poll.c" example on the driver CD shows how to use the function eib8_poll_pdl_packets() and how to further process the position data.
- Depending on the host, an update rate of approx. 10 Hz can be achieved for the measured values in the polling mode. The polling mode is thus suitable, for example, for a measured value display unit.

2.8.6 Receiving position data via UDP

As described in Section 2.7.1, position data can be output via UDP at high clock rates.

2.8.6.1 **Configuration of the EIB 8791**

The following configuration parameters are relevant to the position data transfer via UDP:

Configuration (example)	Explanation
EIB8;pdl_tx_from_slots_to_lane_1:slot_1;1 *EIB8;pdl_tx_from_slots_to_lane_1:slot_2;1* *EIB8;pdl_tx_from_slots_to_lane_1:slot_3;1* *EIB8;pdl_tx_from_slots_to_lane_1:slot_4;1*	Internal routing of position data in the EIB 8791 (should not be changed in standard applications)
EIB8;pdl_forwarding_ram_udp:slot_1;0x11,0x12 *EIB8;pdl_forwarding_ram_udp:slot_2;0x21,0x22 EIB8;pdl_forwarding_ram_udp:slot_3;0x31,0x32 EIB8;pdl_forwarding_ram_udp:slot_4;0x41,0x42*	Position data filter for packet numbers: Slot 1: 0x11, 0x12 Slot 2: 0x21, 0x22 Slot 3: 0x31, 0x32 Slot 4: 0x41, 0x42
EIB8;udp_transfer:udp_dest_mac;90.e2.ba.03.9c.eb *EIB8;udp_transfer:udp_dest_ip;192.168.168.100* *EIB8;udp_transfer:udp_dest_port;3051*	MAC address, IP address, port for UDP transfer (address of the host to which the data are to be transferred)
EIB8;pdl_forwarding_ram_udp:mode;udp_batch	Position data path for UDP transfer

Position data filter:
For each slot, enter the packet numbers that are to be transmitted via UDP. Only packet numbers that are created at an axis should be entered. The number of configurable packet numbers of all slots taken together is 32. The allocation to the slots is arbitrary.

UDP destination:
The EIB 8791 must be informed about the destination of the UDP transmission by means of the MAC address, IP address, and the port. For determining the MAC address of a known IP address, the driver provides the

eib8_discover_MAC_from_IP()

function.

UDP mode:
The

EIB8;pdl_forwarding_ram_udp:mode;"value"

parameter defines the behavior of the position data output:

Value	Explanation
stop	No position data output (default)
batch_soft_realtime	Multiple position data packets are combined into a UDP packet. The data transfer requires little overhead. There are fewer UDP packets to be processed by the host. Recommended for synchronous operation with multiple axes and data packets.
direct_soft_realtime	Each position data packet is transmitted in a separate UDP packet. However, for technical reason, 24 bytes are sent rather than 12. Bytes 13 – 24 are always 0 and do not have any meaning. In particular when using different asynchronous trigger sources (if there is no fixed or equidistant time grid for different axes), the shortest possible latency is achieved in this way.
recording	Position data are written to RAM in the EIB 8791 (see Section 2.10.1)

Notes on udp_batch_soft_realtime:
The number of position data packets that are combined always depends on how many packet numbers have been configured for the position data filters (maximum of 32). For example, if five packet numbers have been configured in the filters, the EIB 8791 will wait until five position data packets are available and then transmit them. Whether the packets are associated with the same trigger event is irrelevant here.
To make sure that all position data packets in a UDP packet are associated with a single trigger event, ensure the following:

- All of the packet numbers set in the filter are configured for the axes and created with the same trigger event (synchronously)
- All of the axes already deliver valid position data packets when the first trigger event occurs.
  This is not be the case, for example, if the trigger is already pending and the axes are configured one after the other. The axes would then deliver their initial position data packets on different trigger events (for ensuring a synchronous start of the position data output, see Section 2.6.3).

Additional notes:

- Before *udp_batch_soft_realtime* can be activated, at least one position data packet must be configured in the position data filter.
- The position data filter can be configured only if *EIB8;pdl_forwarding_ram_udp:mode* was set to *stop*. This is usually already done automatically in the driver.

2.8.6.2 **Activating and deactivating UDP reception in the driver**

The driver provides functions for receiving position data via UDP. First, the data are buffered with a high priority task in an internal position data buffer of the drive (see Section 2.7.1). The application then retrieves the data from the buffer in accordance with the FIFO principle by means of the corresponding functions.

When the buffer is full, the most recent data in the buffer are discarded. This means that new data will not be stored in the buffer until memory is available again.

Data reception is activated with:

eib8_start_udp_pdl_sampling()

This function can be called either with the handle of a connected EIB 8791 or with a new handle. For UDP reception, the handle does not have to be connected to an EIB 8791. The function also expects the following input parameters:

UDP port:
Specify the port to which the EIB 8791 will be transmitting. The port was configured in the EIB 8791 with, for example,

EIB8;udp_transfer:udp_dest_port;3051

UDP reception can be performed with different handles at several different ports simultaneously (if there is more than one EIB 8791 in the network).

Size of the position data buffer:
Enter the size of the position data buffer in bytes. Each position data packet requires 12 bytes. For example, if eight position data packets are created at a rate of 10 kHz, then approx. 1 MB of data is generated per second.

The appropriate buffer size depends on the application: the larger the buffer, the less likely it is that data will be lost if an interruption occurs in the application during the retrieval of data from the buffer. On the downside, a large buffer might contain obsolete data.

UDP reception is stopped with

eib8_stop_udp_pdl_sampling()

This clears the position data buffer.

2.8.6.3 **Reading position data from the position data buffer**

The driver provides several ways of reading position data from the position data buffer (detailed description in eib8.h). Information is usually returned regarding whether an overflow of the position data buffer occurred.

eib8_get_udp_pdl_sampling():
Retrieves a position data packet from the position data buffer.

eib8_get_udp_multiple_pdl():
Retrieves multiple position data packets from the position data buffer. The application specifies the maximum number of packets that it can receive. The driver then returns the maximum number of packets to the application. If the buffer contains fewer packets than the application can receive, then all of the packets from the buffer are immediately returned (the function is not blocked until the maximum number of packets is available).

eib8_get_udp_pdl_latest():
The driver reads the entire position data buffer and determines the most recent position data packet of each packet number. The determined packets are entered into a list and returned to the application. The position data buffer is emptied completely. To prevent uncontrolled blocking of the function at high data rates, a timeout can be set at which the driver cancels the reading of the position data buffer. Even in this case, however, a valid list is returned.
This function is suitable, for example, for displays for which the most recent value is significant.

eib8_get_udp_synchronous_pdl():
The application specifies how many position data packets it expects with the same frame counter. The driver reads the position data memory until the expected packets have arrived and returns them to the application (with timeout). In the next call, the driver checks whether the frame counter was incremented by 1 and then waits for the desired number of packets again. An error is generated:

- If not all packets arrive within a timeout period
- If not all packets have the same frame counter
- If the frame counter is not incremented by exactly 1 during the next call

This function has been designed for synchronous systems in which a trigger source triggers all position data for all axes. During a call, the driver returns packets that are associated with a trigger event and simultaneous checks whether data have been lost. This requires a synchronous start of the position data output (see Section 2.6.3).

2.8.6.4 **Writing position data to a file**

The driver offers the possibility of writing position data in a background task to a file. A position data buffer is initialized, and the desired port for UDP reception is correspondingly initialized. The calls are identical to those described in Section 2.8.6.2, except that the respective file path and file format must be specified:

eib8_start_udp_pdl_dumping()
eib8_stop_udp_pdl_dumping()

The process can be monitored with the following function:

eib8_monitor_udp_pdl_dumping()

The output indicates, for example, how many packets have been written so far and whether a position data memory overflow occurred.

2.8.6.5 **Data loss during UDP reception**

Since UDP is an unsecured, connectionless protocol, there is no guarantee that the data will arrive at the host. The EIB 8791 cannot detect whether the host exists or whether the host is losing packets due to overload.

In particular, high data rates (the EIB 8791 can transmit packets at a rate of up to 400 kHz) lead to data loss in the host. There are several causes:

- Data are already lost in the network card
- The operating system is unable to process the network card's requirements quickly enough
- The EIB 8791 driver is unable to serve the operating system's sockets quickly enough
- The position data buffer overflows because the application is not retrieving the data quickly enough at the driver API.

Thus, the risk of data loss heavily depends on the hardware used (PC and network card) as well as on the operating system and the application. Tests conducted on different computers showed that trigger rates between 10 kHz and 50 kHz can be achieved for 16 position data with the mode *udp_soft_realtime*. Nevertheless, problems can still occur, for example, if a virus scanner is active.

Data loss, despite being very sporadic, must always be taken into account.

Therefore, an error detection feature should be integrated into the application. The frame counter of the position data packets can be analyzed for this purpose. For a packet number, the frame counter must be incremented by 1 with every new position data packet (value range 0 – 255). During synchronous operation (same trigger for all axes), this check is already performed by the *eib8_get_udp_synchronous_pdl()* function.

2.9 Creating a customized configuration

2.9.1 **Configuration of system clock**

As described in Section 1.6, it is possible to select the system clock for the EIB 8791 and to switch the internal system clock to external:

Configuration			**Explanation**
Node	**Parameter**	**Values**	
EIB8	*sync_clock:source*	*internal* *ext_single* *ext_diff*	Sets the system clock to: – Internal (default) – External at single-ended input – External at differential input
EIB8	*sync_clock:output*	*0* *1*	Activates the system clock at the single-ended output: – Deactivated (default) – Activated (The differential output of the system clock is always active)

2.9.2 **Configuration of the triggers**

The EIB 8791 provides a wide range of trigger options. This section describes three typical applications.

2.9.2.1 **Synchronous operation with PTM**

All axes together trigger on a PTM that is created internally or provided at an external input. The PTM signal refers to the system clock of the EIB 8791, as described in Section 1.6. Therefore, the internal PTM may be used only together with the internal system clock, and the external PTM may be used only together with the external system clock (for the configuration of the system clock, see Section 2.9.1). The following configuration parameters must be transmitted to the EIB 8791:

Configuration			**Explanation**
Node	**Parameter**	**Values**	
EIB8	*trig_ptm_in:source*	*internal* *ext_single* *ext_diff*	Sets the PTM to: – Internal (default) – External at single-ended input – External at differential input
EIB8	*trig_ptm_in:freq_config*	*<Hz>*	Frequency of the internal PTM in Hz (only relevant when using the internal PTM), default 0
EIB8	*trig_ptm_in:output*	*0* *1*	Activates the internal PTM at the single-ended output: – Off (default) – On (The differential output of the PTM is always active)
EIB8	*trig_bus1_out:inputs* *trig_bus1_out:enable* *trig_bus1_out:transmitter*	*trig_ptm_in* *1* *1*	Internal trigger routing in the EIB 8791 node for PTM as trigger
SLOT00	*trig_sync1_in:input* *trig_bus1_in:enable* *trig_sync1_in:enable*	*trig_bus1_in* *1* *1*	Internal trigger routing in the SLOT nodes for PTM as trigger (identical for all slots)
SLOT00:AXIS00	*trigger:inputs*	*trig_sync1_in*	Internal trigger routing in the AXIS nodes for PTM as trigger (identical for all axes)

Notes on the PTM:

- The frequency of the internal PTM is set in Hz with the *trig_ptm_in:freq_config* parameter. It is not possible, however, to set any desired frequency here, because the EIB 8791 operates with a 10 MHz system clock, and the PTM frequency is derived with a 16-bit prescaler. When the frequency is entered, the driver sets the nearest possible frequency. To query the currently set frequency, the application can read trig_ptm_in:freq_config. However, the value is rounded and is not the exact frequency. To obtain the actual value, you can query the system clock and the current prescaler:

  EIB8; sync_clock:freq_nominal and
  EIB8; trig_ptm_in:prescaler_config

  The frequency results from freq_nominal / prescaler_config.

  The following frequencies can be precisely set, for example:
  200 Hz, 250 Hz, 500 Hz, 1 kHz, 2 kHz, 5 kHz, 10 kHz, 20 kHz, 40 kHz, 50 kHz, 80 kHz,
  100 kHz, 200 kHz, 400 kHz, 500 kHz, 625 kHz, 1 MHz

  The lowest frequency that can be set is 152.59 Hz.
- If the value for the internal PTM is set to 0 in trig_ptm_in:freq_config, no PTM is created. In this manner, the internal PTM can be deactivated.
- If the last active frequency is set when the internal PTM is switched on, then the PTM is activated in-phase with the last PTM.
- The internal PTM (single-ended) can be provided at the output of the EIB 8791 with EIB8;trig_ptm_in:output;1. Multiple EIB 8791 devices can thereby share a common PTM, for example.
- The following procedure is recommended for defined activation of the external PTM (triggers are forwarded to the axes):
  – Activate the internal PTM with a frequency of 0 (EIB8;trig_ptm_in:source;internal and
  EIB8;trig_ptm_*in:freq_config;0*). In this way, no triggers are created.
  – Perform the configuration
  – Switch from the internal PTM to the external PTM
  (e.g., *EIB8;trig_ptm_in:source; ext_single)*

2.9.2.2 Single external trigger for all axes

If all axes are to trigger on the same external trigger input (see Section 1.6.1), then the following configuration parameters must be transmitted (table for trigger input 1: **ext1**; **ext2**, **ext3** or **ext4** can be used in the same way):

Configuration			Explanation
Node	**Parameter**	**Values**	
EIB8	*trig_ext1_in:enable*	*1*	Activates the external trigger input (The external trigger inputs are deactivated when the EIB 8791 is switched on)
EIB8	*trig_ext1_in:terminate*	*0* *1*	Termination of the trigger input: – Off (default) – On
EIB8	*trig_ext1_in:inverted*	*0* *1*	Inverting of the trigger input: – Off (default) – On
EIB8	*trig_bus1_out:inputs* *trig_bus1_out:enable* *trig_bus1_out:transmitter*	*trig_ext1_in* *1* *1*	Internal trigger routing in the EIB 8791 node for the external trigger
SLOT00	*trig_bus1_in:enable*	*1*	Internal trigger routing in the SLOT nodes for the external trigger (identical for all slots)
SLOT00:AXIS00	*trigger:inputs*	*trig_bus1_in*	Internal trigger routing in the AXIS nodes for the external trigger (identical for all axes)

Notes:

- An inverting of the external trigger signal is required if the axis is to be triggered on the falling edge. By default, axes are triggered on the rising edge.
- The following procedure is recommended for defined activation of the external trigger:
  – First, deactivate the external trigger (*EIB8;trig_**ext1**_in:enable;0)*
  – Perform the configuration
  – Activate the external trigger (*EIB8;trig_**ext1**_in:enable;1)*

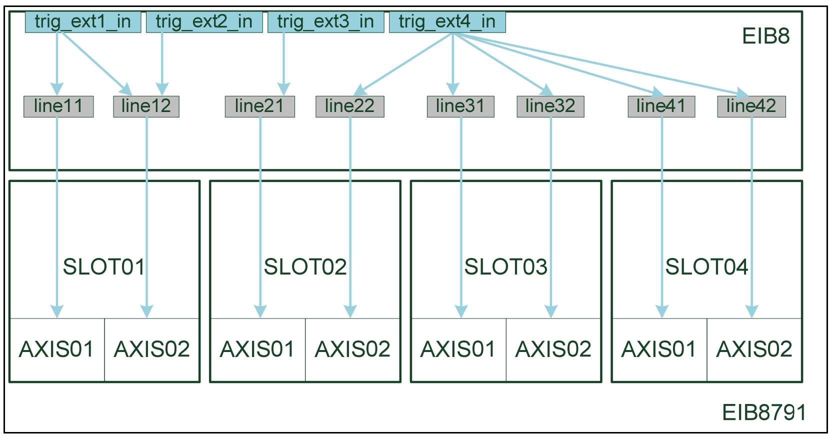
2.9.2.3 Multiple external triggers

The axes are to be triggered on different external trigger inputs (see Section 1.6.1).

To achieve this, the following procedure is recommended:

- Configure and activate all external triggers (or, if necessary, only those you will need)
- Configure each axis internally on one of the eight trigger lines.
- Assign the external triggers to the trigger lines. An external trigger can be assigned to multiple axes, or an axis can be configured on multiple external triggers.

  The figure below shows an example:
  SLOT01:AXIS01 → trig_line11_out → trig_ext1_in
  SLOT01:AXIS02 → trig_line12_out → trig_ext1_in, trig_ext2_in
  SLOT02:AXIS01 → trig_line21_out → trig_ext3_in
  SLOT02:AXIS02 → trig_line22_out → trig_ext4_in
  SLOT03:AXIS01 → trig_line31_out → trig_ext4_in
  SLOT03:AXIS02 → trig_line32_out → trig_ext4_in
  SLOT04:AXIS01 → trig_line41_out → trig_ext4_in
  SLOT04:AXIS02 → trig_line42_out → trig_ext4_in



The following table shows the configuration parameters that must be set:

Configuration			**Explanation**
Node	**Parameter**	**Values**	
EIB8	*trig_ext1_in:enable* *trig_ext2_in:enable* *trig_ext3_in:enable* *trig_ext4_in:enable*	*1*	Activation of the external trigger inputs
EIB8	*trig_ext1_in:terminate* *trig_ext2_in:terminate* *trig_ext3_in:terminate* *trig_ext4_in:terminate*	*0* *1*	Termination of the trigger inputs: – Off – On
EIB8	*trig_ext1_in:inverted* *trig_ext2_in:inverted* *trig_ext3_in:inverted* *trig_ext4_in:inverted*	*0* *1*	Inverting of the trigger inputs: – Off – On
EIB8	*trig_line11_out:enable* *trig_line12_out:enable* *trig_line21_out:enable* *trig_line22_out:enable* *trig_line31_out:enable* *trig_line32_out:enable* *trig_line41_out:enable* *trig_line42_out:enable*	*1*	Internal trigger routing in the EIB8 node for individual axis lines
EIB8	*trig_line11_out:inputs* *trig_line12_out:inputs* *trig_line21_out:inputs* *trig_line22_out:inputs* *trig_line31_out:inputs* *trig_line32_out:inputs* *trig_line41_out:inputs* *trig_line42_out:inputs*	*trig_ext1_in* *trig_ext1_in,trig_ext2_in* *trig_ext3_in* *trig_ext4_in* *trig_ext4_in* *trig_ext4_in* *trig_ext4_in* *trig_ext4_in*	Internal trigger routing: Assignment of the external triggers to the individual axis lines (depending on the application; the example from the figure above is used here)
SLOT00	*trig_line1_in:enable* *trig_line2_in:enable*	*1*	Internal trigger routing in the SLOT nodes for individual axis lines (identical for all slots)
SLOT00:AXIS01	*trigger:inputs*	*trig_line1_in*	Internal trigger routing in the AXIS nodes for individual axis lines (identical for all axes 1)
SLOT00:AXIS02	*trigger:inputs*	*trig_line2_in*	Internal trigger routing in the AXIS nodes for individual axis lines (identical for all axes 2)

See also the notes given in Section 2.9.2.2.

Please also keep in mind that, if all external triggers are to be activated simultaneously, the parameters
*EIB8; trig_**ext1**_in:enable;1*
*EIB8; trig_**ext2**_in:enable;1*
*EIB8; trig_**ext3**_in:enable;1*
*EIB8; trig_**ext4**_in:enable;1*
should be set with a function call. This is possible, for example, by using *eib8_set_config_file()* or *eib8_set_param_list_string()* (see also Section 2.6.3).

2.9.3 Configuration of the encoders

The following parameters can be set for an encoder:

Encoder configuration:

Configuration			**Explanation**
Node	**Parameter**	**Values**	
SLOT0X:AXIS0X	*encoder_config:type*	*linear* *rotary*	Encoder type: Linear encoder Rotary encoder
SLOT0X:AXIS0X	*encoder_config:interface*	*1V_pp_0_90*	Encoder interface: Currently only 1 V_{PP} (0 / 90° signals)
SLOT0X:AXIS0X	*encoder_config:* *reference_type*	*non* *single* *distance_coded*	Type of reference mark: None Single reference mark Distance-coded reference mark
SLOT0X:AXIS0X	*encoder_config:* *line_cnt*	*<signal periods>*	Number of signal periods for rotary encoders
SLOT0X:AXIS0X	*encoder_config:* *ref_increment*	*<signal periods>*	Nominal distance of distance-coded reference marks in signal periods
SLOT0X:AXIS0X	*encoder_config:* *limit_signal_1_present*	*0* *1*	Limit signal 1: Not present Present
SLOT0X:AXIS0X	*encoder_config:* *limit_signal_2_present*	*0* *1*	Limit signal 2: Not present Present
SLOT0X:AXIS0X	*encoder_config:* *homing_signal_present*	*0* *1*	Homing signal: Not present Present

Encoder supply voltage:

Configuration			**Explanation**
Node	**Parameter**	**Values**	
SLOT0X:AXIS0X	*encoder:supply_enable*	*0* *1*	Supply voltage and position calculation: Off On

Encoder compensation algorithms:

Configuration			**Explanation**
Node	**Parameter**	**Values**	
SLOT0X:AXIS0X	*encoder_processing:* *online_comp_enable*	*0* *1*	Online compensation: Off On
SLOT0X:AXIS0X	*encoder_processing:* *waveform_comp_enable*	*0* *1*	Waveform compensation: Off On

Encoder position value filter:

Configuration			Explanation
Node	**Parameter**	**Values**	
SLOT0X:AXIS0X	*pos_value_filter: characteristic*	*linear* *parabolic*	Filter characteristic: Linear Parabolic
SLOT0X:AXIS0X	*pos_value_filter: bandwidth*	*low* *medium* *high*	Filter bandwidth: Low (5 kHz) Medium (10 kHz) High (20 kHz)
SLOT0X:AXIS0X	*pos_value_filter: measure_time*	*<value>*	Relative measuring time in nanoseconds with respect to the trigger. Value range: [–5000 … 20000]
SLOT0X:AXIS0X	*pos_value_filter:active*	*0* *1*	Position value filter: Off On

Please note:

- The following procedure must be adhered to:
  1. Configure the encoder (*encoder_config*)
  2. Activate *encoder:supply_enable*
  3. Set *encoder_processing:online_comp_enable* (optional)
  4. Set *pos_value_filter:active* (optional)

  The driver follows the above sequence automatically when configuration files or configuration lists are used.
- *encoder_processing:waveform_comp_enable* can only be activated if
  – *encoder:supply_enable* is active
  – A valid reference position exists
- When the encoder is configured, the reference position becomes invalid.

2.9.4 **Position data output**

For position data output, the following settings must be made (as shown in the above examples):

Configuration			Explanation
Node	**Parameter**	**Values**	
EIB8	*pdl_tx_from_slots_to_lane_1:slot_1* *pdl_tx_from_slots_to_lane_1:slot_1* *pdl_tx_from_slots_to_lane_1:slot_1* *pdl_tx_from_slots_to_lane_1:slot_1*	*1* *1* *1* *1*	Internal routing of the position data in the EIB 8791. Should always be set this way.
EIB8			
EIB8	*pdl_forwarding_ram_udp:slot_1* *pdl_forwarding_ram_udp:slot_2* *pdl_forwarding_ram_udp:slot_3* *pdl_forwarding_ram_udp:slot_4*	*"packet numbers"*	Packet numbers of the individual slots that are to be transferred via UDP or written to RAM. Format: "0x11,0x12,0x13"
EIB8	*pdl_forwarding_ram_udp:mode*	*"stop"* *"recording"* *"batch_soft_realtime"* *"direct_soft_realtime"*	Mode of the position data output (see Section 2.8.6)
SLOT00:AXIS00	*pdl:output_active*	*0* *1*	Activation of the position data output of the individual axes – Off – On

For UDP transfer, the following additional settings must be made:

Configuration			**Explanation**
Node	**Parameter**	**Values**	
EIB8	*udp_transfer:udp_dest_mac* *udp_transfer:udp_dest_ip* *udp_transfer:udp_dest_port*	*"IP address"* *"MAC address"* *"UDP port"*	UDP destination specified by MAC address, IP address, and port (see Section 2.8.6)

2.9.5 **Synchronous start of the position data packets**

Position data are generated in the EIB 8791 as a result of trigger events. In many applications, a single trigger source is used for all axes. For the evaluation of the position data packets, it is significant which data are associated with the same trigger event. It is also relevant whether packets from individual trigger events were lost because, for example, the host was temporarily overloaded during UDP transfer. The following section provides information on what to keep in mind when configuring the EIB 8791 and on how to detect errors.

In the case of a common trigger source for axes, the aim should be to generate all of the position data packets with the same frame counter. The counter is 0 when the first trigger event occurs. It is then incremented by 1 with each subsequent trigger event. Due to the value range, the counter overflows at 255 and then starts again from 0 (in polling mode, the frame counter is always 0; an incrementing frame counter can therefore be evaluated only in the "UDP" and "RAM recording" modes).

In the application, position data packets with the same frame counter can thus be allocated to a trigger event. Packet loss can be detected from gaps in the packet counter. For the reception of data via UDP, the *eib8_get_udp_synchronous_pdl()* function provides all of the position data packets with the same frame counter and indicates the loss of position data packets—that is, the occurrence of gaps in the frame counter (see Section 2.8.6.3).

In order for all position data packets to be triggered at the first trigger event and to start with the frame counter at 0, the following procedure is advisable (in exactly the given sequence):

No.	**Step**	**Explanation**	**Function**
1	Resetting the configuration	The configuration of the EIB 8791 is reset (deactivation of triggers, position data output, UDP, etc.) so that a new configuration can be loaded without restrictions.	*eib8_reset_idle()*
2	New configuration without activation of the triggers	As described in the previous sections, the desired configuration is loaded into the EIB 8791. The trigger that controls all of the axes is not yet activated. Section 2.9.2 describes which parameters activate the triggers. (e.g., *EIB8;trig_ptm_in:freq_config;0*)	See Section 2.6.3
3	Resetting the frame counters	The frame counter is set to 0 by command.	*eib8_reset_pdl_frame_counter (&myEIB8Handle, "SLOT00:AXIS00")*
4	Starting of UDP reception	Start of UDP reception in the driver: Position data packets are written to the position data memory	*eib8_start_udp_pdl_sampling()*
5	Activation of the triggers	With the activation of the trigger, position data are generated. (e.g., *EIB8;trig_ptm_in:freq_config;1000*)	See Section 2.6.3 e.g., *eib8_set_param_string()*

Note:

- The above procedure has also been implemented in the "example_udp.c" programming example (driver CD).
- If the above sequence is not followed, then it may, for example, occur that a trigger is already active while the axes are being configured. Since the driver configures the axes sequentially, one axis would already be starting the output of the position data while another axis wouldn't start until later. The frame counters of the different axes would thus differ for the same trigger event.

2.10 Other use cases

2.10.1 **Recording of position data**

Position data can be stored in the internal RAM of the EIB 8791 (see Section 2.7.1).

A maximum of 10 million position data packets can be stored. You can choose whether the memory area is written to until it is full, upon which the EIB 8791 automatically stops recording (single shot), or whether the memory area is operated as a ring buffer. When the memory is full in ring buffer mode, the oldest data are overwritten until the application stops recording. The application can also prematurely abort the recording in single-shot mode.

Before the data can be retrieved from the RAM of the EIB 8791, the recording must be stopped. The driver provides the corresponding functions for this purpose. The application receives the data in chronological order in both single-shot mode and ring-buffer mode.

Please note that the download speed of the recorded data is approx. 20 000 position data packets per second. For the maximum number of recorded position data packets, the transfer thus takes approx. 10 min.

2.10.1.1 **Configuration**

The configuration parameter settings for RAM recording are similar to those used for UDP transfer (e.g., the default configuration). The notes regarding a synchronous start of the position data packets are equally applicable (see Section 2.10.1).

Configuration			**Explanation**
Node	**Parameter**	**Values**	
EIB8	*udp_transfer:udp_dest_mac* *udp_transfer:udp_dest_ip* *udp_transfer:udp_dest_port*	*""* *""* *""*	The UDP destination is irrelevant here
EIB8	*pdl_forwarding_ram_udp:mode*	*"recording"*	Position data output mode set to recording; recording is thereby activated in the EIB 8791.
EIB8	*recording:mode_ringbuffer*	 *0* *1*	Ring-buffer mode – Off (single-shot mode) – On (ring-buffer mode)
EIB8	*recording:packets_buffer*	*<N>*	Number of position data packets to be recorded (since the EIB 8791 can only process multiples of 32, the value is reduced or set to at least 32 as needed)

2.10.1.2 **Stopping recording**

For the application to stop recording, the mode is reset in the EIB 8791:

Configuration			**Explanation**
Node	**Parameter**	**Values**	
EIB8	*pdl_forwarding_ram_udp:mode*	*"stop"*	Sets the position data output mode to "stop" (no UDP transfer, no RAM recording)

The mode *pdl_forwarding_ram_udp:mode* is automatically set to "stop" by the EIB 8791 when the number of position data packets to be recorded is reached in single-shot mode.

2.10.1.3 **Reading out the status**

The following parameters are available for querying the status of the current recording:

Status			**Explanation**
Node	**Parameter**	**Values**	
EIB8	*recording:active*	*0* *1*	Recording active – Not active (i.e., not active or stopped) – Active (i.e., not stopped yet)
EIB8	*recording:buffer_total*	*<N>*	Number of position packets to be recorded
EIB8	*recording:buffer_fill*	*<X>*	Number of position packets recorded In ring-buffer mode, the number increases from 0 to N and then remains at N until the recording is stopped.

2.10.1.4 **Retrieving data from the EIB 8791**

Recording must be stopped (*recording:active* must be 0) before the data can be retrieved from the EIB 8791.

The driver provides two functions for this purpose:
eib8_get_rec_pdl_samples_buffer()
eib8_get_rec_pdl_samples_file()
One function writes the position data packets to a buffer (array), and the other one writes them to a text file. A timeout can be configured for both functions in order to prevent blocking of the function (the data retrieved before the timeout expires are valid). The data are provided in the standard format for position data *eib8_pdl_packet_t*.

Note:
The driver CD includes the "example_recording.c" example, which illustrates the procedure.

2.10.2 **Reference run**

Section 1.3 describes how the EIB 8791 handles reference marks. A reference run requires the following:

- Encoder configured:
  – *encoder_config:type*
  – *encoder_config:interface*
  – *encoder_config:reference_type*
  – *encoder_config:line_cnt*
- Additionally for distance-coded reference marks:
  – *encoder_config:ref_increment* is configured
- *encoder:supply_enable* is activated
- *encoder_processing:waveform_comp_enable* is deactivated

A reference run can be started for one or more axes at the same time with following function (the function can also abort the reference mark search):

- *eib8_reference_mark_search()*

The status of the reference mark search can be monitored with the following parameters:

Status			**Explanation**
Node	**Parameter**	**Values**	
SLOT0X:AXIS0X	*ref_mark:running*	*0* *1*	Reference mark search: Not active Active
SLOT0X:AXIS0X	*ref_mark:valid*	*0* *1*	Valid reference position exists: Not valid Valid
SLOT0X:AXIS0X	*ref_mark:ref_pos_1*	*<reference pos single>*	Reference position of a single reference mark
SLOT0X:AXIS0X	*ref_mark:ref_pos_dist_coded*	*<reference pos distance coded>*	Reference position of a distance-coded reference mark

Procedure in the application:

- Configuration of the axis
- Start the reference mark search with *eib8__reference_mark_search()*
  ref_mark:running returns the status "active" (1)
- Wait until *ref_mark:running* is "not active" (0). The reference mark search is finished.
- Check whether the reference mark search was successful: *ref_mark:valid* must be *"valid"* (1).
- Read out the reference position

Single reference mark:	*ref_mark:ref_pos_1*
Distance-coded reference mark:	*ref_mark:ref_pos_dist_coded*

  The reference position is a 32-bit value (*eib8_get_param_hex32()*) that indicates the period counter at which the reference mark was found. With distance-coded reference marks, the value of the period counter at the zero position is given.

The reference position can also be transmitted as a position data packet (see Section 2.7.3.3).

See also the programming example "example_ref_pos average.c".

2.10.3 **Initializing the position value**

The position value of the individual axes can be initialized with the

eib8_set_position_value()

function.

Example:
All axes on 0:

eib8_set_position_value(&myEIB8Handle, "SLOT00:AXIS00", 0)

In this example, only the period counter is set. The phase (interpolation value within the signal period) is maintained.
The desired position value must be specified in the format described in Section 2.7.2.

When setting the position counter, please note:

- The encoder must be configured (in the same way as for a reference run; see Section 2.10.2).
- Waveform compensation must not be active.
- After the position counter has been set, the reference position is invalid.

The position counter should be set before the reference run is performed.

2.10.4 **Clearing position errors**

If an error occurs during position calculation in the EIB 8791, then this is indicated in the position data packet (see Section 2.7.3.1).
The error will remain until it is cleared with

eib8_clear_position_error()

Example:
Slot 1 – Axis 1:

eib8_clear_position_error(&myEIB8Handle, "SLOT01:AXIS01")

All axes:

eib8_clear_position_error(&myEIB8Handle, "SLOT00:AXIS00")

The command can also be executed if no error exists.

2.10.5 **Waveform compensation run**

Before waveform compensation can be activated, a compensation run must be performed (see Section 1.4.4). The result of the compensation run is stored in the flash memory and is thus available again the next time the EIB 8791 is switched on.

2.10.5.1 **Configuration and requirements for the compensation run**

Before a compensation run can be performed, the following requirements must be met:

- The encoder must be configured (in the same way as for a reference run; see Section 2.10.2).
- Online compensation should be activated for the encoder.
- Valid referencing must exist for the encoder.
- Waveform compensation must be deactivated for both encoders (axes) of a slot. It is recommended that waveform compensation be deactivated for all encoders of the EIB 8791 if a compensation run is to be performed.

In addition, the following parameters must be configured for the compensation run:

Configuration			**Explanation**
Node	**Parameter**	**Values**	
SLOT0X:AXIS0X	*waveform_corr_run: start_value*	*<signal periods>*	Initial value of the compensation range in signal periods with respect to the reference position
SLOT0X:AXIS0X	*waveform_corr_run: basic_frequency*	*1* *2*	Fundamental frequency of signal: One signal period Two signal periods
SLOT0X:AXIS0X	*waveform_corr_run: direction*	*positive* *negative*	Traverse direction: Positive counting direction Negative counting direction
SLOT0X:AXIS0X	*waveform_corr_run: num_correction_points*	*<num>*	Number of compensation points
SLOT0X:AXIS0X	*waveform_corr_run: dist_correction_points*	*<signal periods>*	Distance (in signal periods) between the compensation points

2.10.5.2 **Starting and monitoring the compensation run**

When the requirements are met, the compensation run can be started with
eib8_exec_waveform_corr_run()
.

Please note:

- Only one compensation run can be started for a slot.
  The compensation run can be performed in parallel for multiple slots.
  For the individual encoders (axes) of a slot, the compensation run must be performed in sequence.
- A *StorageID* must be specified for the *eib8_exec_waveform_corr_run()* function. This corresponds to a flash memory location in the slot for which the compensation data are stored.

  The following memory locations should be selected:
  SLOT0X:AXIS01 → StorageID = 0
  SLOT0X:AXIS02 → StorageID = 1
- The compensation run can be started before or after *eib8_exec_waveform_corr_run()*. Make sure that, after the call, a sufficient number of samples are traversed before the start position is reached.

The following parameters are available for monitoring the compensation run:

Status			**Explanation**
Node	**Parameter**	**Values**	
SLOT0X:AXIS0X	*waveform_corr_status: improvement*	*<value>*	Value between 0 and 100 that indicates the improvement factor of the compensation.
SLOT0X:AXIS0X	*waveform_corr_status: successful*	*0* *1*	1 if the compensation run was completed successfully
SLOT0X:AXIS0X	*waveform_corr_status: failed*	*0* *1*	1 if the compensation run was completed and errors occurred
SLOT0X:AXIS0X	*waveform_corr_status: running*	*0* *1*	1 if the compensation run is currently being performed (data acquisition or data evaluation)
SLOT0X:AXIS0X	*waveform_corr_status: data_acquisition_done*	*0* *1*	1 if the compensation run is currently being performed. The traverse movement can already be stopped. Calculations are still being performed in the EIB 8791.
Error causes if *waveform_corr_status:failed* is set (1):			
SLOT0X:AXIS0X	*waveform_corr_status: error_underrun*	*0* *1*	1 if not enough samples were taken before waveform_corr_run:start_value.
SLOT0X:AXIS0X	*waveform_corr_status:error_direction*	*0* *1*	1 if the traverse direction was incorrect (waveform_corr_run:direction).
SLOT0X:AXIS0X	*waveform_corr_status: error_speed_low*	*0* *1*	1 if the traverse movement was performed at too low a speed.
SLOT0X:AXIS0X	*waveform_corr_status: error_speed_high*	*0* *1*	1 if the traverse movement was performed at too high a speed.
SLOT0X:AXIS0X	*waveform_corr_status: error_internal_bit*	*0* *1*	1 if an internal error occurred. Please contact HEIDENHAIN's encoder support.

Monitoring process:

- Check whether *waveform_corr_status:running* and *waveform_corr_status:data_acquisition_done* are active.
  – *waveform_corr_status:data_acquisition_done* = 0: Traverse can be stopped
  – *waveform_corr_status:running* = 0: Compensation run completed
- Check whether the compensation run was successful:
  – *waveform_corr_status:successful* = 1: Compensation run successful
  – *waveform_corr_status:failed* = 1: Compensation run failed
- If the compensation run failed, then evaluate the error states and repeat the compensation run:
  – *waveform_corr_status:error_underrun*
  – *waveform_corr_status:error_direction*
  – *waveform_corr_status:error_speed_low*
  – *waveform_corr_status:error_speed_high*
  – *waveform_corr_status:error_internal_bit*
  – *waveform_corr_status:error_acceleration_high*
  – *waveform_corr_status:error_position*
- If the compensation run was successful, read out and evaluate the improvement factor (*waveform_corr_status:improvement*):
  The improvement factor (0 – 100) indicates the level of accuracy improvement to be expected when waveform compensation is activated. If the factor is 0, then there is no improvement to the accuracy. The compensation should not be used.

2.10.5.3 **Concluding and storing the compensation run**

The memory location for the compensation run has already been defined with the *eib8_exec_waveform_corr_run()* function. This memory location must be assigned to the encoder (axis).

eib8_assign_storage_id()

In Section 2.10.5.2, the memory locations were defined as follows:

SLOT0X:AXIS01 → StorageID = 0
SLOT0X:AXIS02 → StorageID = 1

Thus, the function for SLOT01 must be called as follows:

/ SLOT01:AXIS0**1** */*
unit32_t u32*StorageID =* ***0****u;*
*eib8_assign_storage_id ((&myEIB8Handle , "SLOT01:AXIS0**1**",* u32*StorageID, 1u, 1000u);*
/ SLOT01:AXIS0**2** */*
unit32_t u32*StorageID =* ***1****u;*
*eib8_assign_storage_id ((&myEIB8Handle , "SLOT01:AXIS0**2**",* u32*StorageID, 1u, 1000u);*

The waveform compensation has now been stored and is available whenever the EIB 8791 is switched on again.

2.10.5.4 **Activating waveform compensation**

Requirements for activating waveform compensation:

- Encoder configured and reference run completed (see Section 2.10.2)
- Compensation run completed and assigned to the encoder (axis).

Waveform compensation is activated with:

encoder_processing:waveform_comp_enable = 1

2.10.6 **Setting the network parameters**

The network parameters of the EIB 8791 can be set as configuration parameters. They are saved in non-volatile memory in the device. Changed network parameters will not take effect until the EIB 8791 has been reset (*eib8_reset()*) or switched on again.

Configuration			**Explanation**
Node	**Parameter**	**Default**	
EIB8	*network_user:ip_address*	*192.168.168.2*	IP address of the EIB 8791
EIB8	*network_user:netmask*	*255.255.255.0*	Subnet mask
EIB8	*network_user:gateway*	*192.168.168.1*	Default gateway
EIB8	*network_user:dhcp_enabled*	*0*	Use DHCP (0: no, 1: yes)
EIB8	*network_user:dhcp_timeout*	*30*	Timeout for DHCP server discovery [sec] (only relevant when DHCP is used)
EIB8	*network_user:tcp_port*	*1050*	TCP port of the EIB 8791 (the host connects to this port with eib8_connect_tcp())
EIB8	*network_user:tcp_time*	*100*	Default. Should not be changed.
EIB8	*network_user:ftp_port*	*21*	FTP port of the EIB 8791 (e.g., for transfer of software packages)
EIB8	*network_user:host_name*	*EIB 8791 12345678*	Network name of the EIB 8791
EIB8	*network_user:ftp_user*	*anonymous*	User for FTP transfer
EIB8	*network_user:ftp_password*	***	Password for FTP transfer

Example of setting the IP address:

eib8_set_param_string(&myEIB8Handle, "EIB8", network_user:ip_address, "192.168.168.10")

2.10.7 **Checking the software update**

As described in Section *1.11 Firmware update*, new software packages are transmitted to the EIB 8791 via FTP. Upon successful completion of the transfer, the EIB 8791 starts processing the software package. The LAN LED on the front panel starts flashing.

Communication with the EIB 8791 is not possible while it is processing. The driver's functions issue the error
EIB8_ERROR_EIB8_BUSY_UPDATE (5014) or *"EIB8 busy with update"*

The EIB 8791 has completed the update when the LAN LED on the front panel is continuously on again or when the above error message no longer occurs.

Before resetting the EIB 8791, the following status parameters should be queried in order to check the update:

Status			**Explanation**
Node	**Parameter**	**Values**	
EIB8	*update_info:update_error_eib8* *update_info:update_error_slot_1* *update_info:update_error_slot_2* *update_info:update_error_slot_3* *update_info:update_error_slot_4*	*0* *1*	Update status: – No error occurred – Error occurred

If the update was successful, all five parameters have the value 0. If errors occurred, the update should be repeated first (retransmission of the update file via FTP to the EIB 8791).

If errors occur again, the events should be queried (see Section 2.8.2). Then please contact HEIDENHAIN's encoder support.

As the last step, a reset should be executed (*eib8_reset()*) and the software version of the EIB 8791 should be queried (the new software version is visible only after the reset).

Status			**Explanation**
Node	**Parameter**	**Values**	
EIB8	*system_info:device_user_software*	*<version>*	Version of EIB 8791 software e.g., 816477-04-00-A

Remark:
If the "EIB8 Application" is used for updating the firmware, errors are checked there automatically.

2.10.8 **Analog and digital inputs/outputs**

The analog and digital inputs are read and written as parameters in the driver.

2.10.8.1 **Analog inputs**

The four analog inputs (see Section *1.7.2 ANALOG IN*) can be queried with *eib8_get_param_hex32()*:

Status			**Explanation**
Node	**Parameter**	**Values**	
EIB8	*analog_in:analog_1* *analog_in:analog_2* *analog_in:analog_3* *analog_in:analog_4*	*<value>*	The analog values are in the range 0 – (2^{14} – 1)

2.10.8.2 **Digital inputs**

The eight digital inputs (see Section *1.7.1 DIGITAL IN*) can be queried with *eib8_get_param_hex32()*:

Status			**Explanation**
Node	**Parameter**	**Values**	
EIB8	*digital_in:bits8*	*<value>*	The digital inputs are output bit-encoded. Value range: 0x0000 0000 – 0x0000 00FF Digital In 1: Mask 0x0000 0001 Digital In 2: Mask 0x0000 0002 Digital In 3: Mask 0x0000 0004 … Digital In 8: Mask 0x0000 0080

2.10.8.3 **External trigger outputs as digital outputs**

The four external trigger outputs on the rear side of the EIB 8791 (see Section *1.6.1 Asynchronous operation with triggers)* can also be used as digital outputs.

Configuration for the use as a digital output (table for trigger output 1: **ext1**; **ext2**, **ext3** or **ext4** can be used in the same way):

Configuration			**Explanation**
Node	**Parameter**	**Value**	
EIB8	*trig_**ext1**_out:enable*	*0*	Deactivation of the trigger function
EIB8	*trig_**ext1**_out:gpio_mode*	*1*	Activation of the output function

The output is then set with the following parameter:

Configuration			**Explanation**
Node	**Parameter**	**Value**	
EIB8	*trig_**ext1**_out:gpio_set*	*0* *1*	Status of the output: – Logical 0 (low level) – Logical 1 (high level)

3 Appendix

3.1 Example programs in C

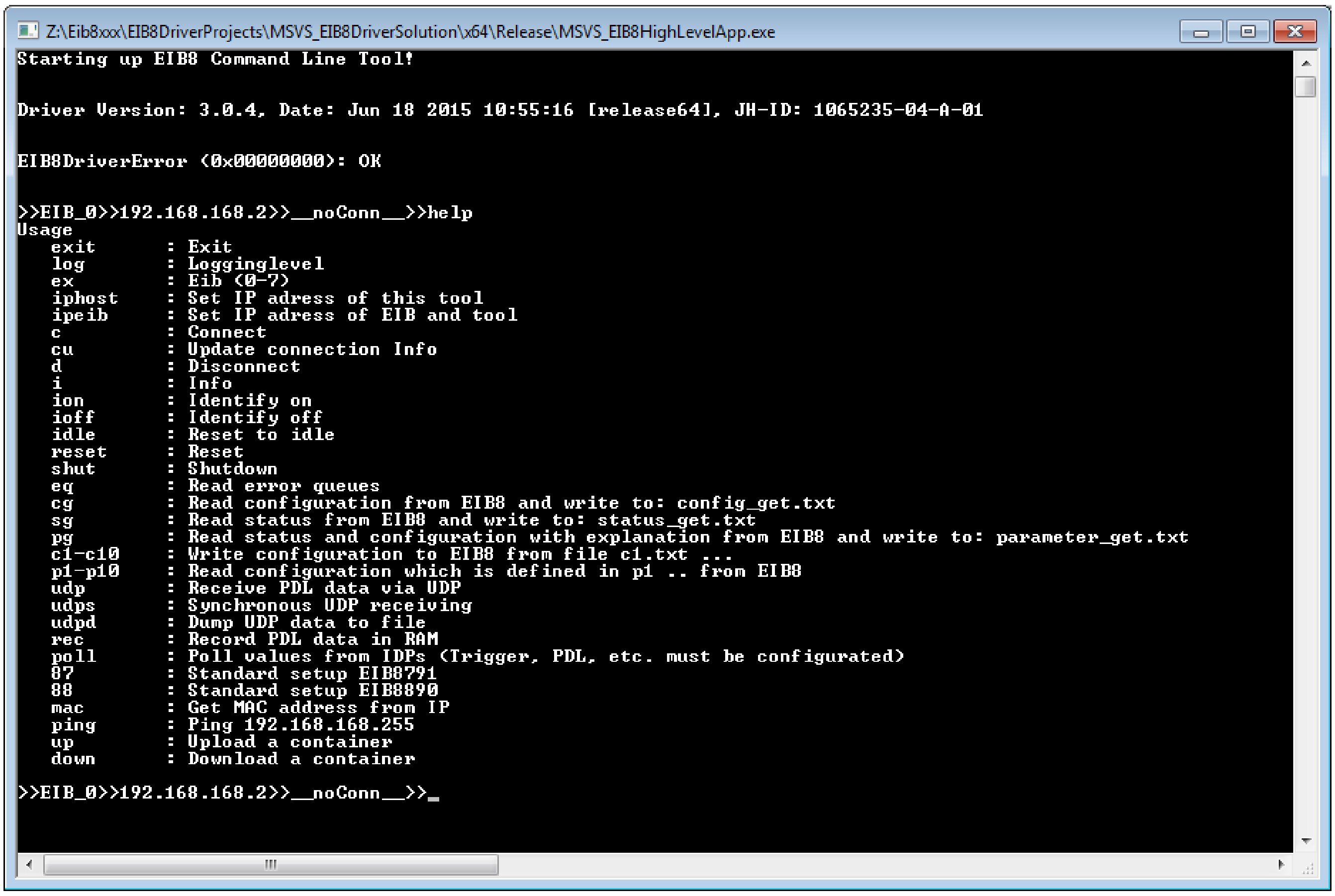
The driver CD includes various examples that illustrate how to use the driver.

Example	Explanation
example_identify.c	• Activation and deactivation of the blinking mode for the LAN LED of the EIB 8791
example_poll.c	• Generation of position data with a default configuration • Querying of the position data in polling mode.
example_udp.c	• Generation of position data with a default configuration • Transferring of packets via UDP • Reading of the packets from the driver's position data memory
example_recording.c	• Generation of position data with a default configuration • Storing of the position data packets in the internal RAM of the EIB 8791 • Retrieval of the packets by the driver
example_ref_pos_average.c	• Generation of position data with a default configuration • Setting of the position value to 0 • Reference run • Transferring of packets via UDP • Reading of the packets from the driver's position data memory • Calculation of the mean value from 4 axes
example_multi_head_system.c	• Generation of position data with a default configuration • Setting of the position value to 0 • Reference run • Transferring of packets via UDP • Reading of the packets from the driver's position data memory • Calculation of the mean value from *n* axes • Compensation run and waveform compensation

In each example, the event queues are checked and default configurations are loaded into the EIB 8791.

3.2 Command-line tool

The command-line tools EIB8AppXX.exe (Windows) and EIB8App (Linux), which are provided on the driver CD, do not require installation and can be used for simple tests with the EIB 8791. Entering "help" displays the commands of the tool.


Z:\Eib8xxx\EIB8DriverProjects\MSVS_EIB8DriverSolution\x64\Release\MSVS_EIB8HighLevelApp.exe

```
Starting up EIB8 Command Line Tool!


Driver Version: 3.0.4, Date: Jun 18 2015 10:55:16 [release64], JH-ID: 1065235-04-A-01


EIB8DriverError (0x00000000): OK


>>EIB_0>>192.168.168.2>>__noConn__>>help
Usage
   exit       : Exit
   log        : Logginglevel
   ex         : Eib (0-7)
   iphost     : Set IP adress of this tool
   ipeib      : Set IP adress of EIB and tool
   c          : Connect
   cu         : Update connection Info
   d          : Disconnect
   i          : Info
   ion        : Identify on
   ioff       : Identify off
   idle       : Reset to idle
   reset      : Reset
   shut       : Shutdown
   eq         : Read error queues
   cg         : Read configuration from EIB8 and write to: config_get.txt
   sg         : Read status from EIB8 and write to: status_get.txt
   pg         : Read status and configuration with explanation from EIB8 and write to: parameter_get.txt
   c1-c10     : Write configuration to EIB8 from file c1.txt ...
   p1-p10     : Read configuration which is defined in p1 .. from EIB8
   udp        : Receive PDL data via UDP
   udps       : Synchronous UDP receiving
   udpd       : Dump UDP data to file
   rec        : Record PDL data in RAM
   poll       : Poll values from IDPs (Trigger, PDL, etc. must be configurated)
   87         : Standard setup EIB8791
   88         : Standard setup EIB8890
   mac        : Get MAC address from IP
   ping       : Ping 192.168.168.255
   up         : Upload a container
   down       : Download a container

>>EIB_0>>192.168.168.2>>__noConn__>>_
```

The following table explains the commands that are relevant to standard applications:

Command	**Explanation**
exit	Exits the command-line tool
log	Defines the logging level of the tool. • 0: Logging deactivated • 2: Warnings and errors (default) • 5: Warnings, errors and the entire communication with the EIB 8791 The command-line tool automatically writes logging messages to the log server via UDP. To display the messages, start EIB8LogServerXX.exe.
e0 *e1* *…* *e7*	The command-line tool can operate a maximum of eight EIB 8791 units. For example, with the command *e1* you can switch to the EIB 8791_1. The active EIB 8791 is displayed in the prompt.
iphost	Sets the IP address that the tool uses to connect to the EIB 8791. (Default 192.168.168.2).
ipeib8	Configures the IP address of the EIB 8791. The IP address only takes effect after the device has been restarted (*reset* command) and is stored in non-volatile memory in the device.
c	Establishes a connection to the EIB 8791. Displays the basic system states (*eib8_connect_tcp()*).
cu	Updates the basic states (*eib8_connect_result_update()*).
d	Disconnects from the EIB 8791 (*eib8_disconnect_tcp()*).
i	Reads out the software version of the EIB 8791 modules.
ion	Activates the blinking mode of the LAN LED (*eib8_identify()*).
ioff	Deactivates the blinking mode of the LAN LED (*eib8_identify()*).
idle	Resets the configuration of the EIB 8791 (*eib8_reset_idle()*).
Reset	Restarts the EIB 8791 (*eib8_reset()*)
shut	Initiates shutdown of the EIB 8791 (*eib8_shutdown()*).
eq	Reads out and displays the event queues. In addition, the events are stored in events.txt in the directory of the tool (if the file already exists, the events are appended to the existing ones).
cg	Reads out and displays the overall configuration of the EIB 8791. In addition, the configuration is stored in config_get.txt in the directory of the tool.
sg	Reads out and displays the overall status of the EIB 8791. In addition, the status is stored in status_get.txt in the directory of the tool.
pg	Reads out and displays all parameters of the EIB 8791. In addition, the parameters are stored in parameter_get.txt in the directory of the tool.
c1 *c2* *…* *c10*	Transmits the configuration file to the EIB 8791. During this process, *c1* sends, for example, c1.txt in the directory of the tool to the EIB 8791. The file names c1.txt – c10.txt are permanently defined.
udp	Reception of position data packets via UDP (reads the driver's position data memory). To apply the default values, press Enter. To cancel, press CTRL + C. The EIB 8791 must be configured for this beforehand (e.g., with the *87* command).
poll	Queries position data in polling mode. Press CTRL + C to cancel. The EIB 8791 must be configured for this beforehand (e.g., with the *87* command).
87	Transmits the default configuration to the EIB 8791. Position packets are created and can be received via UDP. The IP address and the port of the computer to which the UDP data are to be transmitted must be entered (usually, the computer running the tool).

3.3 LabVIEW™ EIB 8791 Application

The EIB 8791 Application is on the driver CD and can be started directly (\labview\32Bit\EIB8Application\EIB8 Application EXE\ EIB8Application.exe) if LabVIEW™ (2012 version, service pack 1, 32-bit) is on the computer.

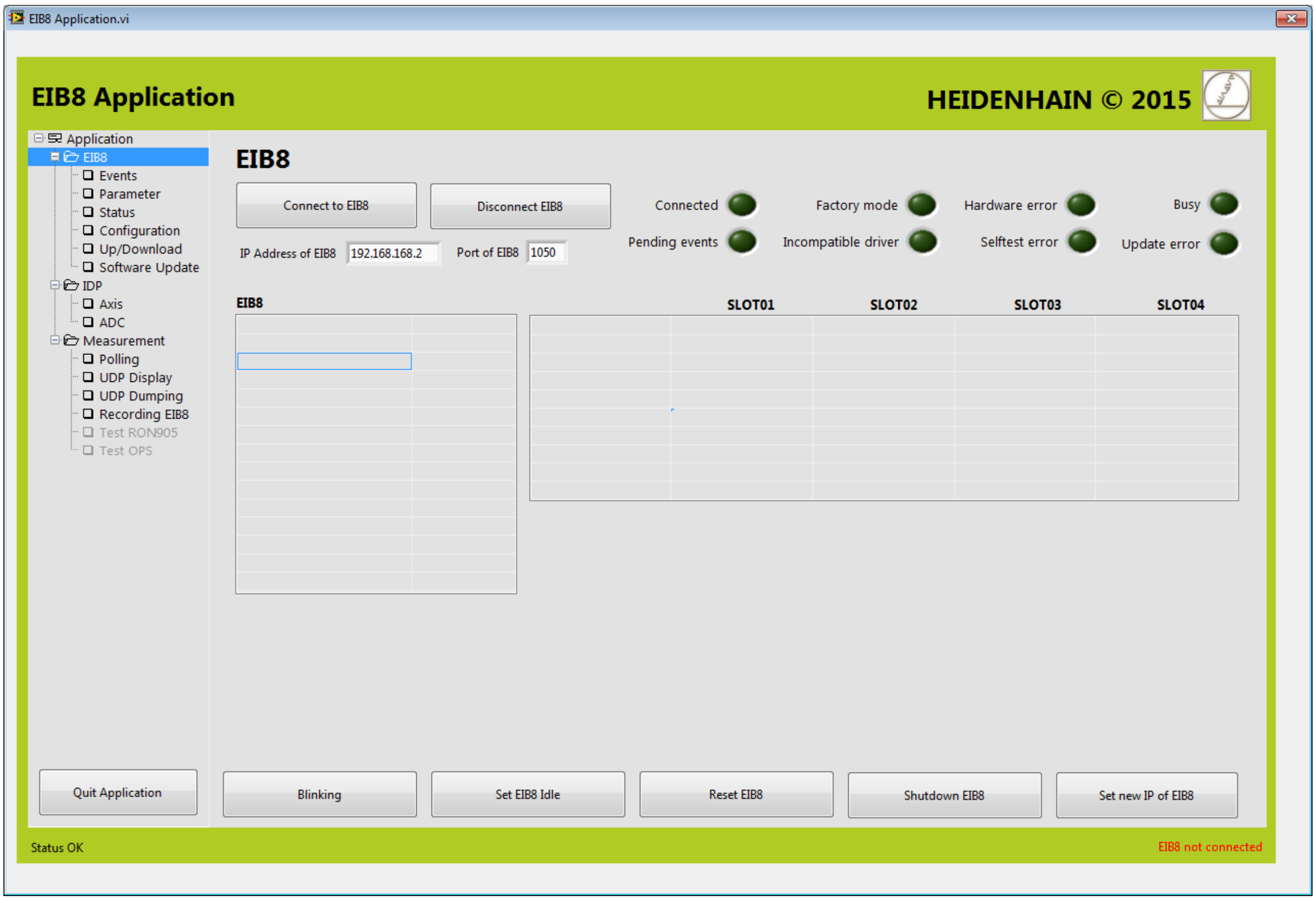
If LabVIEW™ is not on the computer, you can install the tool with a free runtime version of LabVIEW™ (\labview\32Bit\EIB8Application\EIB8 Application Setup\Volume\setup.exe).

Moreover, the tool is also on the driver CD as a virtual instrument (VI) (\labview\32Bit\EIB8Driver\ApplicationVI\ EIB8 Application Project.lvproj).

This section describes the functions that are relevant for typical applications.

3.3.1 **Start-up screen and connecting to the EIB 8791**

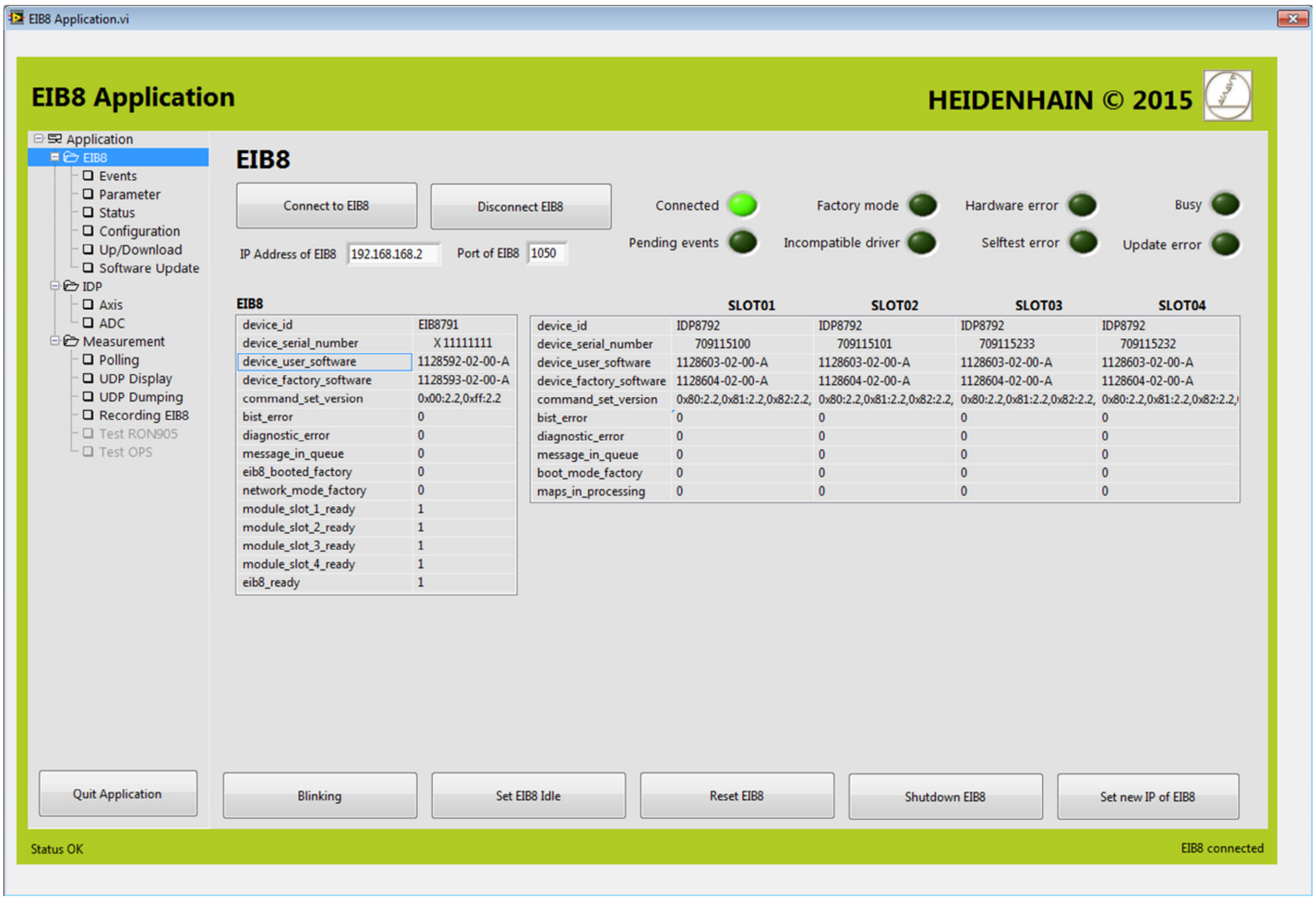
The EIB 8791 Application starts up with the *EIB8* page. In the tree on the left of the screen, you can select the different pages. The status bar at the bottom displays a text in red type if an error has occurred (this is usually the text output by the *eib8_get_last_error()* function of the driver).



IP Address of EIB8 192.168.168.2 Port of EIB8 1050

Status OK EIB8 not connected

If the correct IP address of the EIB 8791 has been entered in "IP Address of EIB8," the EIB 8791 can be connected by pressing the "Connect to EIB8" button. If no errors have occurred, the following window appears:



IP Address of EIB8: 192.168.168.2
Port of EIB8: 1050

EIB8

device_id	EIB8791
device_serial_number	X 11111111
device_user_software	1128592-02-00-A
device_factory_software	1128593-02-00-A
command_set_version	0x00:2.2,0xff:2.2
bist_error	0
diagnostic_error	0
message_in_queue	0
eib8_booted_factory	0
network_mode_factory	0
module_slot_1_ready	1
module_slot_2_ready	1
module_slot_3_ready	1
module_slot_4_ready	1
eib8_ready	1

	SLOT01	SLOT02	SLOT03	SLOT04
device_id	IDP8792	IDP8792	IDP8792	IDP8792
device_serial_number	709115100	709115101	709115233	709115232
device_user_software	1128603-02-00-A	1128603-02-00-A	1128603-02-00-A	1128603-02-00-A
device_factory_software	1128604-02-00-A	1128604-02-00-A	1128604-02-00-A	1128604-02-00-A
command_set_version	0x80:2.2,0x81:2.2,0x82:2.2,	0x80:2.2,0x81:2.2,0x82:2.2,	0x80:2.2,0x81:2.2,0x82:2.2,	0x80:2.2,0x81:2.2,0x82:2.2,
bist_error	0	0	0	0
diagnostic_error	0	0	0	0
message_in_queue	0	0	0	0
boot_mode_factory	0	0	0	0
maps_in_processing	0	0	0	0

Status OK

EIB8 connected

3.3.2 ***EIB8* page**

After the EIB 8791 has been connected, the following basic functions are available on the *EIB8* page:

Button	**Explanation**
Connect to EIB8	Connects to EIB 8791 and updates the states. Can be repeated at any time.
Disconnect EIB8	Disconnects from the EIB 8791 (*eib8_disconnect_tcp()*).
Blinking	Activates / deactivates the blinking mode of the LAN LED (*eib8_identify()*).
Set EIB8 Idle	Resets the configuration of the EIB 8791 (*eib8_reset_idle()*).
Reset EIB8	Restarts the EIB 8791 (*eib8_reset()*)
Shutdown EIB8	Initiates shutdown of the EIB 8791 (*eib8_shutdown()*).
Set new IP of EIB8	Configures the IP address of the EIB 8791. After the EIB 8791 has been connected, enter the desired address into the *IP Address of EIB8* field and press *Set new IP of EIB8.* The IP address takes effect after the device has been restarted (*Reset EIB8* button) and is stored in non-volatile memory in the device.

3.3.3 **Application page**

The *Application* page displays the version of the driver. In addition, the logging level that is used by the EIB 8791 Application for logging on UDP can be configured (see Section 2.5.4.4).

UDP destination:
Under *Destination IP*, *Port* and *MAC*, the destination of the UDP transfer is set for the default configuration of the EIB 8791 (see Section 3.3.7). During the program start, the EIB 8791 Application determines the IP address of the computer as well as the associated MAC address, and enters them into the respective input fields (not possible if the network connection is inactive). Depending on the application, the values can be edited if, for example, the UDP packets are to be transmitted to another computer.

3.3.4 ***Events* page**

Button	Explanation
Get Events	Reads out and displays the event queues. New events are appended to the list.
Clear Event table	Clears the list of events in the tool

3.3.5 ***Parameter* page**

Button	Explanation
Get Parameter List from EIB8	Reads out all current parameters of the EIB 8791 along with a description and example. The parameter list is stored in the Windows Temp directory under temp_eib8_params_get.txt.

3.3.6 ***Status* page**

Button	Explanation
Get Status List from EIB8	Reads out all current status parameters of the EIB 8791. The status list is saved to the Windows temp directory under temp_eib8_status_get.txt.

3.3.7 ***Configuration* page**

Button	Explanation
Get Config List from EIB8	Reads out all current configuration parameters of the EIB 8791. Displayed in the *List of current Configuration*. In addition, the configuration is stored in the Windows Temp directory under temp_eib8_config_get.txt.
Write Config List to EIB8	Transmits the *List of current Configuration* to the EIB 8791.
Get Config List from File	Reads a configuration file and displays it in *List of current Configuration* (to transmit it to the EIB 8791, press the *Write Config List to EIB8* button).
Write Config List to File	Writes the *List of current Configuration* to a file.
PTM	Activates / deactivates the internal PTM generator with *Frequency Hz* (corresponds to the configuration parameter *EIB8; trig_ptm_in:freq_config*). The value is effective only if the EIB 8791 is operated with internal PTM.
c1.txt to EIB8 *c2.txt to EIB8* *c3.txt to EIB8* *c4.txt to EIB8*	For quick configuration, the configuration file c1.txt (or c2.txt, etc.), which must be located in the EIB8Application.exe directory, is displayed in the *List of current Configuration* and immediately transmitted to the EIB 8791.
Set standard Configuration EIB 8791	Writes the default configuration to the EIB 8791. The *List of current Configuration* remains unchanged. For UDP transfer, the settings from the *Application* page are used.
Set EIB8 Idle	Resets the configuration of the EIB 8791 (*eib8_reset_idle()*).
Reset PDL frame counter	Resets the frame counters of all axes (*eib8_reset_pdl_frame_counter()*).

3.3.8 *Software Update (FTP)* page

First, select the software package (bin file) in the file selection dialog (File button on the right). Leave *Destination* set to the default (*update).*

Press the *Start Update* button to start the update process. The following steps are executed (see also Sections 1.11 and 2.10.7):

- Sending of the software package to the EIB 8791 via FTP (renaming the file is not necessary here).
- Waiting until the update has been processed.
- Checking whether the update was successful.

If errors occurred, the update should initially be repeated.

If errors occur again, the events should be read out (see Section 3.3.4). Then please contact encoder support at HEIDENHAIN.

Upon successful completion of the update, a reset should be triggered on the *EIB8* page. After the EIB 8791 has reconnected, use the *EIB8; device_user_software* parameter to check whether the new software has been loaded (the parameter is read out and displayed automatically when the EIB 8791 is connected).

3.3.9 *Axis* page

The commands on the *Axis* page always apply to a single axis. This axis is selected in the *Slot and Axis for actions* selection fields (the EIB 8791 has two axes per slot). Generally, a valid configuration (i.e., configuration of position data packet, encoder configuration, and triggers) must also be loaded into the EIB 8791 prior to execution of a command.

Button	**Explanation**
Referencing / Compensation Check	Updates the *Referencing Done* status as well as the *Online Compensation* and *Waveform Compensation* configuration parameters
Position	Display of period counter and phase (interpolation value within the signal period) of the axis
Referencing Start	Starts the reference run of the axis. *Referencing Done* starts to flash yellow. When the reference mark is found, *Referencing Done* turns green.
Waveform correction run start	Starts the compensation run for waveform compensation (see Section 2.10.5).
Set Compensation	Activates online compensation and/or waveform compensation
Clear Position error	Resets the error bit in the position data (see Section 2.10.4).
Set Position	Sets the position value of the axis to the desired value (see Section 2.10.3).

3.3.10 *ADC* page

On the *ADC* page, the analog values of the two incremental signals A and B can be displayed for all eight axes.

This requires a valid configuration with UDP output in the EIB 8791, as can be created, for example, with *Set standard Configuration EIB 8791* on the *Configuration* page.

The *ADC* button automatically changes the configuration for the output of ADC position data packets and starts UDP reception in the driver. Please note that the configuration is not reset to the original values upon completion of ADC output.

The other settings for the ADC display, with the exception of *Scaling*, should not be changed.

3.3.11 *Polling* page

On the Polling page, all of the configured position data packets can be read out and displayed (see Section 2.8.5). The left three columns are for all of the position data types, whereas all of the other columns apply only to packets of the data types "position," "speed," and "reference."

This requires a valid configuration of the EIB 8791 such as can be created with *Set standard Configuration EIB 8791* on the *Configuration* page.

3.3.12 *UDP Display* page

The *UDP Display* page shows all of the position data packets received by the driver on the selected port (*UDP receiving Port*). The left three columns are for all of the position data types, whereas all of the other columns apply only to packets of the data types "position," "speed," and "reference." *Overrun occurred* indicates that the driver failed to process all packets.

Note:
The display uses the *eib8_get_udp_pdl_latest()* driver function, meaning that the most recent value of each position data type is displayed with every query of the position data buffer.

3.3.13 *UDP Dumping* page

All of the position data packets that the driver receives on the selected port (*UDP receiving Port*) are written to a file (*File for dumping data*).

The position data can be stored as a string (*dumping format*: *EIB8_PDL_DUMP_STD_TXT_ CHECKS*) or in binary format (*dumping format*: *EIB8_PDL_DUMP_RAW_BINARY*) (see Section 2.7.2). Binary means that the position data packet is stored with 12 bytes.

The following states are indicated:

- *Total packets:*
  Number of packets that have been written to the file.
- *Load (%):*
  Processing power required by the driver.
- *Internal queue empty:*
  The driver could immediately empty the entire position data memory, or no position data packets were received.
- *Overrun occurred*:
  An overflow of the position data memory occurred. Packets cannot be processed quickly enough (data rate of EIB 8791 too high).
- *PD frame counter with gaps*:
  Gaps were detected in the frame counter, indicating that packets were lost (usually because the data rate of the EIB 8791 was too high).
- *CRC errors in PD packets*:
  CRC errors were detected in the packets.
- Aborted by driver:
  Dumping was aborted due to an error.

3.3.14 *Recording EIB8* page

The *Recording EIB8* page supports the use during the reading out of position data that have been recorded in the RAM of the EIB 8791. For a description on how to configure the EIB 8791 and start recording the position data, please refer to Section 2.10.1.

Button	**Explanation**
Recording state	Continuously checks the status of recording.
Transfer Recorded PD from EIB8	Retrieves the position data packets from the EIB 8791 and saves them to a file. This feature can be used only after recording has stopped. By setting an appropriate timeout, you can limit the time for retrieving the packets. If all of the packets are to be retrieved, then the timeout period must be set so as to be sufficiently long. Depending on the computer, the retrieval of 1 million packets will take about 60 seconds.

3.4 LabVIEW™ examples

The driver CD provides virtual instruments (VIs) that calls the driver functions (\labview\32Bit\EIB8Driver\ EIB8ApiVI). The names of the VIs are similar to those of the driver functions. For a description of the functions and data types, refer to the interface definition ("eib8.h") or the supplied HTML documentation.

The LabVIEW™ examples on the driver CD show how to use the abovementioned VIs. These examples are gathered together in a project (labview\32Bit\EIB8Driver\ExampleVI\Eib8Examples.lvproj).

Example	**Explanation**
ConnectEvents.vi	• Evaluation of status information during Connect • Reading out and displaying of the events
DefaultConfig.vi	• Default configuration of the EIB 8791: Enter the IP address of the host and determine the MAC address with *DISCOVER MAC HOST*. Pressing the *Default EIB 8791* button sends the configuration to the EIB 8791 • Reset • Reset Idle • Shutdown
Identify.vi	• Activation and deactivation of the blinking mode for the LAN LED of the EIB 8791
MultiHeadSystem.vi *(with configuration file config_multi_head_system.txt)*	• Writing the configuration file *(config_multi_head_system.txt)* • Reference run • Compensation run and waveform compensation • Display of position and speed as a mean value from up to eight axes
Configuration.vi	• Reading and writing of configuration files • Reading of the entire parameter list from the EIB 8791
SetIPAdress	• Configuration of the IP address of the EIB 8791
PollPD.vi	• Reading out of position data in polling mode
UDPReceive.vi	• Reading out of position data from the driver's position data memory
UDPSynchReceive.vi	• Reading out of position data from the driver's position data memory, with the position data being expected synchronously (see Section 2.8.6.3)

HEIDENHAIN

DR. JOHANNES HEIDENHAIN GmbH
Dr.-Johannes-Heidenhain-Straße 5
83301 Traunreut, Germany
☏ +49 8669 31-0
FAX +49 8669 32-5061
E-mail: info@heidenhain.de

Technical support	FAX	+49 8669 32-1000
Measuring systems	☏	+49 8669 31-3104
E-mail: service.ms-support@heidenhain.de		
NC support	☏	+49 8669 31-3101
E-mail: service.nc-support@heidenhain.de		
NC programming	☏	+49 8669 31-3103
E-mail: service.nc-pgm@heidenhain.de		
PLC programming	☏	+49 8669 31-3102
E-mail: service.plc@heidenhain.de		
APP programming	☏	+49 8669 31-3106
E-mail: service.app@heidenhain.de		

www.heidenhain.de

1210768 · 01 · A · 02 · 11/2018 · PDF