Programming manual

BPS 202 SDK

version 1.2.0

Notice

Support

Our team provides uncomplicated support in case of problems, questions or requests concerning BPS 202 SDK. Please contact us by writing to sales@langer-emv.de.

Table of contents: Page

1 Introduction

The BPS 202 Software Development Kit (SDK) is a collection of resources designed to facilitate the development of applications that can control the Langer Burst Power Station (BPS 202) or integrate it into existing programs. The SDK supports both Windows and Linux environments, providing a cross-platform library (DLL for Windows and shared object file for Linux) for direct USB communication with the hardware.

The BPS202 can also be controlled through the provided GUI application "BPS202-Client". Both methods utilize the same underlying library file to communicate with the hardware, ensuring consistent behaviour across different control approaches.

2 System Requirements

2.1 Operating Systems

- Windows 10 Home/Pro (x64)
- Windows 11 Home/Pro (x64)
- Ubuntu 24.04 LTS (x64)

Equivalent systems that may be compatible include:

- Other Windows 10/11 versions
- Windows Server 2019 and 2022
- Ubuntu 22.04 LTS and newer
- Debian 11 and newer

2.2 Hardware Requirements

- Processor: Any x86 multi-core CPU
- RAM: 512 MB minimum
- USB port: Available USB port for communication with the BPS 202

2.3 Software Dependencies

- FTD2XX driver (installation files provided with the SDK or can be directly downloaded from https://ftdichip.com/drivers/d2xx-drivers/)

2.4 Additional Requirements

- Administrator privileges for initial setup and execution (on Linux)
- Note that before running a Linux D2XX application, the VCP driver must be unloaded

3 Compatibility Notes

- The SDK has been tested on Windows 10 Pro (x64), Windows 11 Pro (x64), and Ubuntu 24.04 LTS (64-bit x86, Kernel Version 6.8).
- While other Linux distributions may be compatible, they have not been officially assessed.
- For optimal performance and full feature support, it is recommended to use the latest version (currently 1.2) of the SDK with the most recent firmware updates for BPS202.
- The SDK described in this manual corresponds to BPS202-Client version 1.9.xx.

4 Using the SDK in External Scripts

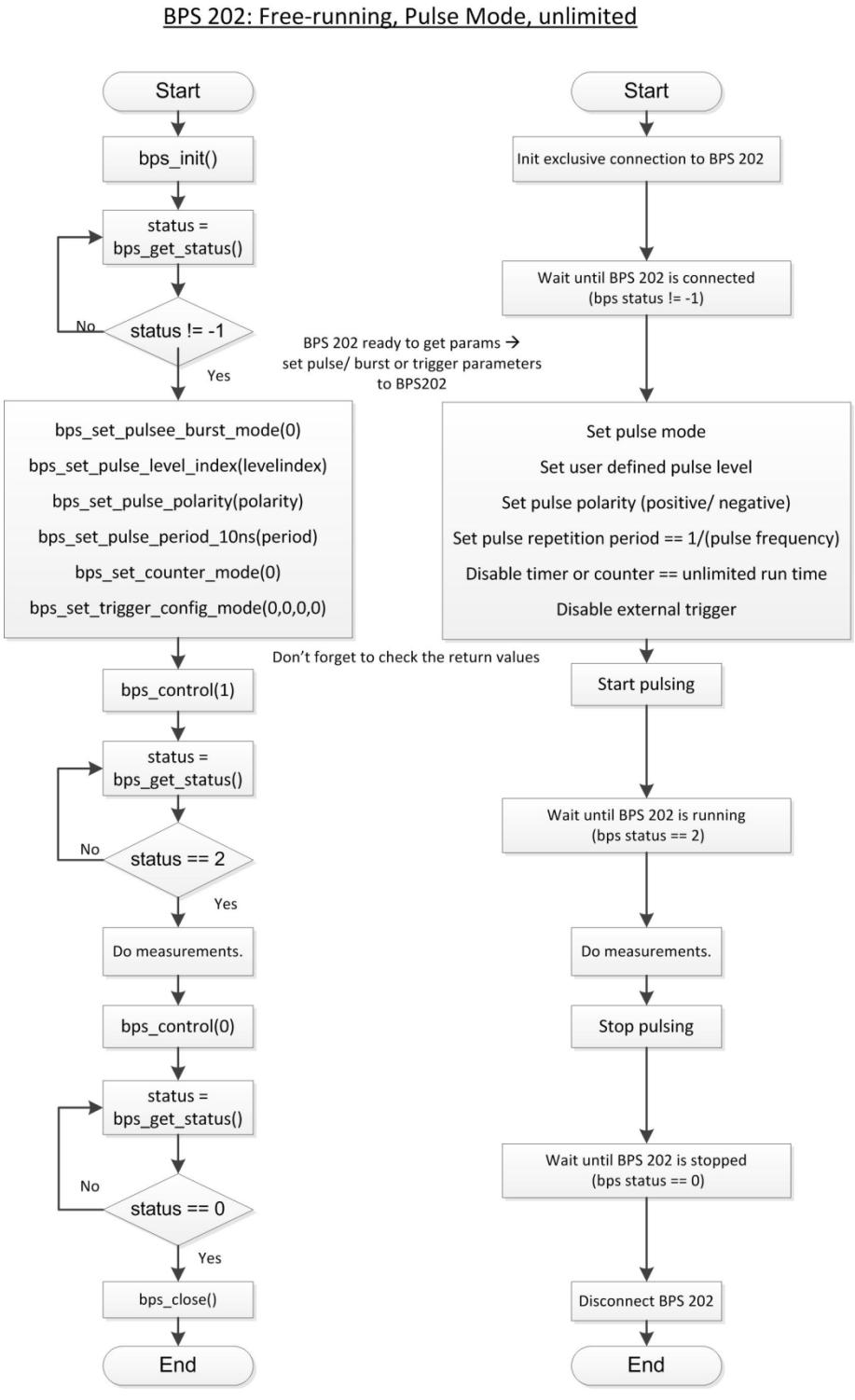

Figure 1 Flowchart of SDK usage in external scripts.

5 Commands

int bps_close()

Closes the exclusive USB connection to the BPS 202.

Remarks

- make sure to call this function after usage
- if not called, the USB connection is not closed which prevents the usage of the BPS 202 from other programs

Return values

-1 : error
0 : everything ok

int bps_control(int state)

Starts the high voltage generation of the BPS 202. Depending on the selected trigger mode, the BPS 202 will start generating the start pulses for the probes or waiting for the external trigger.

Remarks

- make sure to set the pulse parameters before running this function
- otherwise, the results are not defined
- wait for the bps status to be set correctly (1 / 2 / 3) before proceeding with other commands or external triggers
- if no probe is connected, the status will remain in “stop pulsing”

Parameter states

0: stop pulsing
1: start pulsing
2: single shot <- untested

Return values

-1 : bad parameter
0 : everything ok

int bps_get_alternating()

This command returns the current status of the alternating function. If enabled the polarity changes with every pulse.

Return values

-1 : error
0 : disabled
1 : enabled

char* bps_get_bpsfirmwareversion()

This command returns a pointer to the firmware version of the BPS.
Format of the Firmware Version: x.x.x.x

Return values
'---' : no bps connected
else : specific bps firmware version

char* bps_get_bpshardwareversion()

This command returns a pointer to the hardware version of the BPS.
Format of the Hardware Version: x.x.x.x

Return values
'---' : no bps connected
else : specific bps hardware version

char* bps_get_bpsmanufacturer()

This command returns a pointer to the string of the manufacturer of the BPS.

Return values
'---' : no bps connected
else : specific bps manufacturer – “Langer-EMV GmbH”

char* bps_get_bpsserial()

This command returns a pointer to the serial of the BPS.

Return values
'---' : no bps connected
else : specific bps serial

char* bps_get_bpstype()

This command returns a pointer to the string of the type of the BPS.

Return values
'---' : no bps connected
else : specific bps type (e.g. “BPS202-1”).

int bps_get_burst_counter()

This command returns the current burst counter value.

Return values
-1 : error
else : counter value

int bps_get_burst_counter_init()

This command returns the initial burst counter value.

Return values
-1 : error
0 : disabled
else : initial counter value

int bps_get_burst_period_max_ms()

This command returns the maximum possible burst period in milliseconds.

Return values
-1 : error
else : current burst period value in milliseconds

int bps_get_burst_period_min_ms()

This command returns the minimum possible burst period in milliseconds.

Return values
-1 : error
else : current burst period value in milliseconds

int bps_get_burst_period_ms()

Returns the currently set burst period in milliseconds.

Return values
-1 : error
else : current burst period value in milliseconds

int bps_get_counter_mode()

This command returns the current counter mode.

Return values
-1 : error
0 : disabled
1 : counter
2 : timer

char* bps_get_error_msg()

This command retrieves the error message for the last error.

Return values
'message' : error message
‘’ : empty string if no error was found

char* bps_get_level_unit()

This command returns a pointer to the unit of the pulse level values.

Return values
'V' : volt
'kV': kilovolt

int bps_get_low_jitter_trigger_delay()

Get low jitter trigger delay.

Remarks
- check probe features to determine if probe supports low jitter trigger delay

Return values
-1 : error
else : current low jitter trigger delay value

int bps_get_probefeatures()

This command returns integer with the encoded features of the connected probe.

Remarks
- the following probe features exist (bit flags)
  0x1 → probe supports low jitter trigger delay when bypass of trigger logic is enabled
  0x2 → probe supports contact measurement
  0x4 → probe supports different pulse level ranges (internal use)
  0x10 → probe requires pulse level index (internal use only)

Return values
0 : no probe connected
else : specific probe features

char* bps_get_probefirmwareversion()

This command returns a pointer to the probe firmware version of the probe connected to the BPS202.
Format of the firmware version: x.x.x.x

Return values
'---' : no probe connected
else : specific probe firmware version

char* bps_get_probehardwareversion()

This command returns a pointer to the probe hardware version of the probe connected to the BPS202.
Format of the hardware version: x.x.x.x

Return values
'---' : no probe connected
else : specific probe hardware version

int bps_get_probeid()

This command returns the probe ID.
Format of the hardware version: 4Byte number

Return values
-1 : no probe connected
else : specific probe id

char* bps_get_probemanufacturer()

This command returns a pointer to a string containing the probe manufacturer.

Return values
'---' : no probe connected
else : specific probe manufacturer → “Langer-EMV GmbH”

char* bps_get_probename()

This command returns a pointer string that contains the probe name of the probe connected to the BPS202.

Return values
'No Probe': no probe connected
else : current probes name

char* bps_get_probeserial()

This command returns a pointer to the string of the serial number of the probe connected to the BPS202.

Return values
'---' : no probe connected
else : specific probe serial

char* bps_get_probetablesignature()

This command returns a pointer to the probe table signature of the probe connected to the BPS202. The probe table signature is the firmware version of the parameter set of the probe.
Format of the probe table signature: x.x.x.x

Return values
'---' : no probe connected
else : specific probe table signature

char* bps_get_probetype()

This command returns a pointer string that contains the probe type of the probe connected to the BPS202.

Return values
'---' : no probe connected
else : specific probe type

int bps_get_pulse_burst_mode()

This command returns the flag indicating whether the BPS is set to pulse mode or burst mode.

Return values
-1: error
0 : pulse mode
1 : burst mode

int bps_get_pulse_counter()

This command returns the current pulse counter value.

Return values
-1 : error
else : counter value

int bps_get_pulse_counter_init()

This command returns the initial counter value.

Return values
-1 : error
0 : disabled
else : initial counter value

int bps_get_pulse_level_count()

The available pulse levels (voltage levels) are probe series dependant. This command retrieves the number of pulse levels. If bps_get_pulse_range_count () > 1, this command returns the available pulse levels of to the active pulse range.

Remarks
- when the return value is “1” there is an error with the probe connection, or no probe is attached

Return values
-1 : error
else : number of pulse levels

int bps_get_pulse_level_index()

This command returns the currently set pulse level index starting from 0. This index represents the current set pulse voltage level.

Return values
-1 : error
else : current level Index

int bps_get_pulse_levels(double* levels, int size)

This command returns an array containing the possible pulse levels. If *bps_get_pulse_range_count ()* > 1, this command returns an array containing the possible pulse levels of to the active pulse range.

Return values
-1 : buffer too small (use *bps_get_pulse_level_count*)
0 : ok

int bps_get_pulse_period_10ns()

This command returns the currently set pulse period in 10 nano seconds.

Return values
-1 : error
else : current pulse period value in 10 nano seconds

int bps_get_pulse_period_max_10ns()

This command returns the maximum possible pulse period (minimum pulse repetition frequency) 10 nano seconds.

Return values
-1 : error
else : maximum pulse period value in 10 nano seconds

int bps_get_pulse_period_min_10ns()

This command returns the minimum possible pulse period (maximum pulse repetition frequency) 10 nano seconds.

Return values
-1 : error
else : minimum pulse period value in 10 nano seconds

int bps_get_pulse_polarity()

Get Pulse Polarity.

Return values
-1 : error
0 : negative
1 : positive

int bps_get_pulse_range_count()

The available pulse level ranges are probe series dependant. This command retrieves the number of available pulse level ranges.

Return values
-1 : error
else : number of pulse levels

int bps_get_pulses_per_burst()

Returns number of pulses for burst mode.

Return values
-1 : error
else : pulses per burst

int bps_get_status()

Get status of BPS 202.

Remarks

- the BPS status is updated every 100 ms – significant faster function recalls won't have an effect

Return values

-2 : error
-1 : not connected
0 : stopped
1 : single
2 : running
3 : waiting for trigger

int bps_get_timer_init_ms()

This command returns the timer initial value in milliseconds.

Return values

-1 : error
0 : disabled
<=65535: initial timer value

int bps_get_timer_ms()

This command returns the current timer value in milliseconds.

Return values

-1 : error
else : current timer value in milliseconds

int bps_get_trigger_delay_10ns()

Get trigger delay in 10 nanoseconds.

Return values

-1 : error
else : current trigger delay value in 10 nanoseconds

int bps_get_trigger_delay_max_10ns()

Get trigger delay maximum in 10 nanoseconds.

Return values
-1 : error
else : current trigger delay maximum value in 10 nanoseconds

int bps_get_trigger_delay_min_10ns()

Get trigger delay minimum in 10 nanoseconds.

Return values
-1 : error
else : current trigger delay minimum value in 10 nanoseconds

int bps_init()

Initializes an exclusive USB connection to the BPS 202 and initializes the BPS 202.

Remarks
- wait at least 2 seconds to make sure the initialization routine has finished
- make sure to call this function before using any of the other functions, otherwise they will return invalid information
- check BPS status with bps_get_status() before proceeding with other commands

Return values
-1 : error
0 : everything ok

int bps_library_version(int* major, int* minor, int* patch)

Retrieves BPS library version.

Parameter
major: pointer to major version number
minor: pointer to minor version number
patch: pointer to patch version number

Return values
0: everything ok

int bps_reinit_remaining_counters()

This command reinitializes burst/pulse remain counter values set by the *bps_set_burst_counter_init* command.

Return values
-1 : error
0 : everything ok

int bps_reinit_remaining_timer()

This command reinitializes burst/pulse remain counter values set by the *bps_set_timer_init_ms* command.

Return values
-1 : error
0 : everything ok

int bps_set_alternating(int alternate)

This command enables the alternating function. If enabled the polarity changes with every pulse.

Parameter** alternate **values
0: disable alternating
1: enable alternating

Return values
-1 : error
0 : everything ok

int bps_set_burst_counter_init(int counter)

This command sets an initial burst counter value. The value is written to the BPS after it is started, or the *bps_reinit_remaining_counters* command is executed.

Parameter** counter **values
0 : disable counter
<=65535 : set initial counter value to *counter*

Return values
-1 : error
0 : everything ok

int bps_set_burst_period_ms(int period)

This command sets the time between each burst packet (burst period) in milliseconds.

Return values
-1 : parameter exceeds limits
0 : everything ok

int bps_set_counter_mode(int mode)

This command enables / disables the counter or timer feature in the BPS.

Parameter** mode **value
0: disabled
1: counter
2: timer

Return values
-1 : bad counter mode
0 : everything ok

int bps_set_low_jitter_trigger_delay(int delay)

Set low jitter trigger delay.

Remark
- check probe features to determine if probe supports low jitter trigger delay

Parameter** delay **values
Range: 0...510

Return values
-1 : error
0 : everything ok

int bps_set_probe_pin_contact(int enabled)

Set probe tip contact detection.

Remark
- check probe features to determine if probe supports contact detection

Parameter** enabled **values
0: disable detection
1: enabled detection

Return values
-1 : error
0 : everything ok

int bps_set_pulse_burst_mode(int mode)

This command sets the BPS to pulse mode or to burst mode.

Parameter** mode **values
0: pulse mode
1: burst mode

Return values
-1 : error
0 : everything ok

int bps_set_pulse_counter_init(int counter)

This command sets an initial pulse counter value. The value is written to the BPS when it is started or when *bps_reinit_remaining_counters* is executed.

Parameter** counter **values
0 : disable counter
<=65535 : set counter to value

Return values
-1 : error
0 : everything ok

int bps_set_pulse_level_index(int levelindex)

This command sets the pulse voltage level according to its index. The index starts at 0.

0 <= *levelindex* < *bps_get_pulse_level_count()*

Return values
-1 : parameter exceeds limits
0 : everything ok

int bps_set_pulse_period_10ns(int period)

This command sets the desired pulse period in 10ns. In free running mode the probe will generate pulses with the set period.

Remark
- set this parameter to the probe maximum when using the external trigger feature

Return values
-1 : parameter exceeds limits
0 : everything ok

int bps_set_pulse_polarity(int polarity)

This command sets the pulse polarity.

Parameter** polarity **values
0: negative
1: positive

Return values
-1 : error
0 : everything ok

int bps_set_pulse_range_index(int rangeindex)

This command sets the pulse range according to its index. The index starts at 0.

0 <= *rangeindex* < *bps_get_pulse_range_count ()*

Return values
-1 : parameter exceeds limits
0 : everything ok

int bps_set_pulses_per_burst(int pulses)

This command sets the number of pulses in each burst packet. This only applies in burst mode.

Return values
-1 : error
0 : everything ok

int bps_set_timer_init_ms(int timer)

This command sets the timer initial value in milliseconds. The value is written to the BPS after it is started or the *bps_reinit_remaining_timer* command is executed.

Return values
-1 : error
0 : everything ok

int bps_set_trigger_config_mode(int enabled, int edge, int action, int bypasslogic)

This command configures the trigger.

Remarks
- if bypass is enabled, parameter *edge* has to be 1 and parameter *action* has to be 0
- if parameter *action* is set to 0, set pulse period to minimum (maximum pulse frequency) → *bps_set_pulse_period_10ns(bps_get_pulse_period_min_10ns())*

Parameter** enabled **values
0: disable trigger
1: enabled trigger

Parameter** edge **values
0: falling edge
1: rising edge

Parameter** action **values
0: start pulsing or bursting (depending on pulse burst mode);
1: singe pulse or burst (depending on pulse burst mode);

Parameter** bypasslogic **values
0: bypass disabled
1: bypass enabled

Return values
-1 : error
0 : everything ok

int bps_set_trigger_delay_10ns(int delay)

This command sets trigger delay in 10 nanoseconds.

Return values
-1 : error
0 : everything ok