Power Tracer SDK 1.4.1

Generated by Doxygen 1.8.0

Mon Jan 15 2018 09:34:51

Contents

1 Introduction

1.1 Using the SDK

The SDK contains an API that exports several C functions which can be used to control all features of the Power Tracer. The API can be used with any programming environment that supports standard C calls. This section describes how to use the API from Visual Studio 2008.

- Click *Project* → *Properties*
- In the *Configuration* combo box, select *All configurations*.
- Under *Configuration Properties* → *C/C++*, select *Additional Include Directories* and add the folder that contains the `powertracer_api.h` file (i.e. the `include` folder in this package).
- Under *Configuration Properties* → *Linker*, select *Additional Library Directories* and add the folder that contains the `powertracer.lib` file (i.e. the `lib\win32` folder or `lib\win64` folder in this package).
- Under *Configuration Properties* → *Linker* → *Input*, select *Additional Dependencies* and add `powertracer.lib` .

These instructions allow you to use the Power Tracer API functions in your project. Please refer to the following sections for more information about the functions available in the API.

1.2 Function overview

1.2.1 Opening and closing

In order to open a Power Tracer device, first call the function pt_device_list(). This function can be used to obtain the number of connected Power Tracer devices. Using this number, the pt_device_get_info() can be called once (or more if more than one Power Tracer devices are connected). After calling the pt_device_get_info() function, the returned pt_device pointer can be passed to the pt_open() function, after which the pt_device structure can be used to call all other functions. The function pt_close() should be used the close the connection to the Power Tracer.

1.2.2 Smart card control

Power Tracer features a configurable smart card voltage. This voltage can be set using the pt_set_smartcard_voltage() function. The current voltage can be obtained using the pt_get_smartcard_voltage(). In addition, the supported frequency range and stepsize can be obtained using the pt_get_smartcard_voltage_boundary() and pt_get_smartcard_voltage_stepsize() functions respectively. The default voltage is 1.8 volt.

The voltage supply to the smart card can be controlled using the pt_smartcard_powerdown() and pt_smartcard_powerup() functions. The pt_is_smartcard_powered() can be used to determine if the smart card is currently powered.

The smart card can be reset using the pt_smartcard_cold_reset() and pt_smartcard_warm_reset() functions.

Power Tracer features a configurable smart card clock frequency. This frequency can be set using the pt_set_smartcard_frequency() function. The current frequency can be obtained using the pt_get_smartcard_frequency(). In addition, the supported frequency range and step size can be obtained using the pt_get_smartcard_frequency_boundary() and pt_get_smartcard_frequency_stepsize() functions respectively. The default clock frequency is 1MHz.

Using the pt_is_smartcard_inserted() function, programs can check whether or not a smart card is inserted in the Power Tracer.

1.2.3 Smart card/RS232 communication

The Power Tracer supports multiple communication channels, such as the Smart card interface (default) and the external RS-232 interface. The active communication channel can be configured using the pt_set_communication_channel(). It can be obtained using the pt_get_communication_channel() channel.

In order to communicate with a target, communication parameters must be set correctly.

The F and D parameters, documented in the ISO/IEC 7816-3 standard, can be set using the pt_set_f_d() function. The current ETU (default value 372) can be obtained using the pt_get_etu() function.

The Character Guard Time (CGT) (default value 12), also documented in the ISO/IEC 7816-3 standard, can be set using the pt_set_cgt() function. The current CGT can be requested using the pt_get_cgt() function.

Power Tracer supports several error signaling features documented in the ISO/IEC 7816-3 standard. The pt_enable_rx_error_signaling() and pt_enable_tx_error_checking() functions defines whether Power Tracer handles the error signaling in character-level transmission. Correspondingly, pt_is_tx_error_checking_enabled() and pt_is_rx_error_signaling_enabled() returns the current error-signaling configuration of smart card transmitter and receiver of Power Tracer. When error signaling is enabled for either transmitter or receiver of Power Tracer, one can make use of pt_get_tx_error_status() or pt_get_rx_error_status() to check if parity error has happened during transmission or reception of characters. The pt_set_tx_error_handling() function defines the behavior of Power Tracer smart card transmitter when an parity error is signaled by smard card.

The RS232 channel can be configured using the pt_set_rs232_config() function. The current configuration can be obtained using the pt_get_rs232_config() function.

Regardless of the configured communication channel, the pt_write() and pt_read() functions can be used to communicate with the target. The read and write timeouts can be configured using the pt_set_read_timeout() and pt_set_write_timeout() function respectively.

1.2.4 Triggering

In order to generate a trigger signal after a command is sent, the Power Tracer needs to be armed before sending the command after which a trigger should be generated. Arming the Power Tracer can be done using the pt_set_armed() function. The status can be obtained using the pt_is_armed() function.

Measured from the trigger point, the generated trigger signal can be delayed using the pt_set_trigger_delay() function. The current trigger delay can be requested using the pt_set_trigger_delay() function.

1.2.5 Measurement control

Power Tracer features a configurable power measurement circuit. The gain of this circuit can be set using the pt_set_gain() function. The gain can be obtained using the pt_get_gain(). In addition, the supported gain range and step size can be obtained using the pt_get_gain_boundary() and pt_get_gain_stepsize() functions respectively. The default gain is 100%.

Power Tracer features a configurable power measurement circuit. The offset of this circuit can be set using the pt_set_offset_by_current() function. The offset current configured can be obtained using the pt_get_offset_by_current(). In addition, the supported offset current range and step size can be obtained using the pt_get_offset_boundary_by_current() and pt_get_offset_stepsize_by_current() functions respectively. The default offset current is -16 mA.

Power Tracer can stop the external smart card clock during the measurement. This feature can be enabled by setting the "clock off duration" to a non-zero value using the pt_set_clock_off_duration(). This value can be obtained by using the pt_get_clock_off_duration() function. The "clock off period" will be started after a delay (starting from the trigger point). This delay can be set using the pt_set_clock_off_delay() function. It can be obtained using the pt_get_clock_off_delay() function.

Power Tracer features a internal power supply using capacitors. In order to decrease noise the DC/DC converters used to charge the capacitors should be switched off during the measurement. This feature can be enabled by setting the "DC/DC off duration" to a non-zero value using the pt_set_dcdc_off_duration(). This value can be obtained by using the pt_get_dcdc_off_duration() function. The "DC/DC off period" will be started after a delay (starting from the trigger point). This delay can be set using the pt_set_dcdc_off_delay() function. It can be obtained using the pt_get_dcdc_off_delay() function.

1.2.6 Other functions

The SDK version can be requested using the pt_sdk_get_version() function. The version of the Power Tracer can be requested using the pt_get_version() function.

2 Deprecated List

Global pt_get_offset (pt_device ∗powertracer_device, double ∗offset)

Replacement function pt_get_offset_by_current()

Global pt_get_offset_boundary (pt_device ∗powertracer_device, double ∗min, double ∗max)

Replacement function pt_get_offset_boundary_by_current()

Global pt_get_offset_stepsize (pt_device ∗powertracer_device, double ∗step_size)

Replacement function pt_get_offset_stepsize_by_current()

Global pt_set_offset (pt_device ∗powertracer_device, double offset)

Replacement function pt_set_offset_by_current()

3 Module Index

3.1 Modules

Here is a list of all modules:

4 Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

5 Module Documentation

5.1 Opening/Closing

Data Structures

- struct pt_device

  Data structure for the Power Tracer device information.

Functions

- POWERTRACER_API PT_STATUS pt_device_list (unsigned int *count)

  Obtain the number of Power Tracer devices ready for use.
- POWERTRACER_API PT_STATUS pt_device_get_info (pt_device *powertracer_device, unsigned int index)

  Obtain the device information of the specified Power Tracer device and store it to a powertracer data structure.
- POWERTRACER_API PT_STATUS pt_open (pt_device *powertracer_device)

  Open the Power Tracer device described by pt_device structure.
- POWERTRACER_API PT_STATUS pt_close (pt_device *powertracer_device)

  Close the Power Tracer device described by pt_device structure.

5.1.1 Detailed Description

Opening must be performed before using any Power Tracer SDK functionality, and similarly you must not call any API functions after closing.

5.1.2 Function Documentation

5.1.2.1 POWERTRACER_API PT_STATUS pt_close (pt_device * *powertracer_device*)

Close the Power Tracer device described by pt_device structure.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_open()

5.1.2.2 POWERTRACER_API PT_STATUS pt_device_get_info (pt_device * *powertracer_device,* unsigned int *index*)

Obtain the device information of the specified Power Tracer device and store it to a powertracer data structure.

NOTE: The pt_device_get_info() should be preceeded by the pt_device_list() call.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
index	The index to the Power Tracer device from which the device information is collected. **NOTE:** Assigning ZERO to this parameter will indicate getting the information from the first detected device.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_device_list()
pt_open()

5.1.2.3 POWERTRACER_API PT_STATUS pt_device_list (unsigned int * *count*)

Obtain the number of Power Tracer devices ready for use.

Parameters

count	A pointer to an unsigned integer that stores the detected device count.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_device_get_info()

5.1.2.4 POWERTRACER_API PT_STATUS pt_open (pt_device * *powertracer_device*)

Open the Power Tracer device described by pt_device structure.

NOTE: The open call has to be preceeded by a pt_device_get_info() call which collects information neccessary for the opening operation.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_device_get_info

5.2 Smart Card Control

Functions

- POWERTRACER_API PT_STATUS pt_set_smartcard_voltage (pt_device *powertracer_device, double voltage)

  Set the voltage supplied on the VCC pin of smart card interface.
- POWERTRACER_API PT_STATUS pt_set_smartcard_frequency (pt_device *powertracer_device, double desired_frequency, double *actual_frequency)

  Configure the clock frequency(MHz) supplied on the smart card slot CLK pin.
- POWERTRACER_API PT_STATUS pt_smartcard_warm_reset (pt_device *powertracer_device)

  Perform a warm reset to the smart card.
- POWERTRACER_API PT_STATUS pt_smartcard_cold_reset (pt_device *powertracer_device)

  Perform a cold reset to the smart card.
- POWERTRACER_API PT_STATUS pt_smartcard_powerup (pt_device *powertracer_device)

  Power up the Power Tracer smart card slot.
- POWERTRACER_API PT_STATUS pt_smartcard_powerdown (pt_device *powertracer_device)

  Power down the Power Tracer smart card slot.
- POWERTRACER_API PT_STATUS pt_get_smartcard_voltage (pt_device *powertracer_device, double *voltage)

  Get the voltage that will be applied on smart card when it is powered.
- POWERTRACER_API PT_STATUS pt_get_smartcard_voltage_boundary (pt_device *powertracer_device, double *min, double *max)

  Get maximum and minimum supported voltage of smart card power.
- POWERTRACER_API PT_STATUS pt_get_smartcard_voltage_stepsize (pt_device *powertracer_device, double *step_size)

  Get the resolution of the smart card voltage setting.
- POWERTRACER_API PT_STATUS pt_get_smartcard_frequency (pt_device *powertracer_device, double *frequency)

  Get the smart card clock frequency (MHz) used by the Power Tracer.
- POWERTRACER_API PT_STATUS pt_get_smartcard_frequency_boundary (pt_device *powertracer_device, double *min, double *max)

  Get maximum and minimum smart card frequency supported by Power Tracer.
- POWERTRACER_API PT_STATUS pt_get_smartcard_frequency_stepsize (pt_device *powertracer_device, double *step_size)

  Get the resolution of between the maximum and minimum smart card frequency supported by Power Tracer.

5.2.1 Detailed Description

The smart cart control API calls hould be used to perform control actions to the smart card interface prior or post to a smart communication.

5.2.2 Function Documentation

5.2.2.1 POWERTRACER_API PT_STATUS pt_get_smartcard_frequency (pt_device * *powertracer_device,* double * *frequency*)

Get the smart card clock frequency (MHz) used by the Power Tracer.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
frequency	A pointer to a double variable that stores the obtained frequency value.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_smartcard_frequency()

5.2.2.2 POWERTRACER_API PT_STATUS pt_get_smartcard_frequency_boundary (pt_device * *powertracer_device,* double * *min,* double * *max*)

Get maximum and minimum smart card frequency supported by Power Tracer.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
min	Pointer to a double variable where the minimum frequency in MHz is returned.
max	Pointer to a double variable where the maximum frequency in MHz is returned.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.

See also

pt_set_smartcard_frequency()
pt_get_smartcard_frequency()
pt_get_smartcard_frequency_stepsize()

Definition at line 1942 of file powertracer_api.c.

5.2.2.3 POWERTRACER_API PT_STATUS pt_get_smartcard_frequency_stepsize (pt_device * *powertracer_device,* double * *step_size*)

Get the resolution of between the maximum and minimum smart card frequency supported by Power Tracer.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
step_size	Pointer to a double variable where the resolution of frequency in MHz is returned.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.

See also

pt_set_smartcard_frequency()
pt_get_smartcard_frequency()
pt_get_smartcard_frequency_boundary()

Definition at line 1962 of file powertracer_api.c.

5.2.2.4 POWERTRACER_API PT_STATUS pt_get_smartcard_voltage (pt_device * *powertracer_device,* double * *voltage*)

Get the voltage that will be applied on smart card when it is powered.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
voltage	A pointer to a double variable that stores the obtained voltage value.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_smartcard_voltage()
pt_get_smartcard_voltage_boundary()
pt_get_smartcard_voltage_stepsize()

5.2.2.5 POWERTRACER_API PT_STATUS pt_get_smartcard_voltage_boundary (pt_device * *powertracer_device,* double * *min,* double * *max*)

Get maximum and minimum supported voltage of smart card power.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
min	A pointer to a double variable bearing the returned minimum voltage allowed.
max	A pointer to a double variable bearing the returned maximum voltage allowed.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_smartcard_voltage()
pt_get_smartcard_voltage()
pt_get_smartcard_voltage_stepsize()

5.2.2.6 POWERTRACER_API PT_STATUS pt_get_smartcard_voltage_stepsize (pt_device * *powertracer_device,* double * *step_size*)

Get the resolution of the smart card voltage setting.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
step_size	A pointer to a double variable bearing the returned voltage resolution.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_smartcard_voltage()
pt_get_smartcard_voltage()
pt_get_smartcard_voltage_boundary()

5.2.2.7 POWERTRACER_API PT_STATUS pt_set_smartcard_frequency (pt_device * *powertracer_device,* double *desired_frequency,* double * *actual_frequency*)

Configure the clock frequency(MHz) supplied on the smart card slot CLK pin.

NOTE: The recommended frequency resolution is 0.05MHz, however, frequency setpoints with finer resoltions could also be achieved but without any guarantee.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
desired_frequency	The desired smart card clock frequency ranging from 1.0 to 10.0 in MHz. When Power Tracer is powered, the default smart card CLK frequency is 1MHz.
actual_frequency	The actualy smart card clock frequency achieved, also in MHz unit.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_smartcard_frequency()
pt_get_smartcard_frequency_boundary()
pt_get_smartcard_frequency_stepsize()

5.2.2.8 POWERTRACER_API PT_STATUS pt_set_smartcard_voltage (pt_device * *powertracer_device,* double *voltage*)

Set the voltage supplied on the VCC pin of smart card interface.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
voltage	The voltage value to be configured. Ranging from 1.8 volt to 6.0 volt, with resolution of 0.1 volt. When Power Tracer is powered up, the default voltage is 1.8 volt.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STAUTS.PT_VOLTAGE_OUT_RANGE if the voltage setpoint is out of the range.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_smartcard_voltage()
pt_get_smartcard_voltage_boundary()
pt_get_smartcard_voltage_stepsize()

5.2.2.9 POWERTRACER_API PT_STATUS pt_smartcard_cold_reset (pt_device * *powertracer_device*)

Perform a cold reset to the smart card.

NOTE: Cold reset will not change the ETU (F and D) setting, nor the parity ACK/NACK checking and signaling status. Restoring parameter to desired condition is up to the user.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_smartcard_warm_reset()

5.2.2.10 POWERTRACER_API PT_STATUS pt_smartcard_powerdown (pt_device * *powertracer_device*)

Power down the Power Tracer smart card slot.

NOTE: Powering down the smartcard will not change the ETU (F and D) setting, nor the partity ACK/NACK checking and signaling status.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.

PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_smartcard_powerup()
pt_is_smartcard_powered()

5.2.2.11 POWERTRACER_API PT_STATUS pt_smartcard_powerup (pt_device * *powertracer_device*)

Power up the Power Tracer smart card slot.

NOTE: The voltage applied on the VCC line of smart card slot depends on the value set by pt_set_smartcard_voltage().

NOTE: The smart card interface is in power-down status when Power Tracer is powered.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_smartcard_powerdown()
pt_is_smartcard_powered()

5.2.2.12 POWERTRACER_API PT_STATUS pt_smartcard_warm_reset (pt_device * *powertracer_device*)

Perform a warm reset to the smart card.

NOTE: Warm reset will not change the ETU (F and D) setting, nor the parity ACK/NACK checking and signaling status. Restoring parameters to desired condition is up to the user.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_smartcard_cold_reset()

5.3 Smart Card/RS232 I/O

Enumerations

- enum pt_tx_error_handling { CONTINUE = 0, HALT = 1 }

  Enumeration type for the smart card transmitter error handling.
- enum pt_com_channel { SMARTCARD = 0x1, EXT_SERIAL = 0x2 }

  Enumeration type for the Power Tracer communication channels.
- enum pt_rs232_baudrate {
  BAUD9600 = 0, BAUD19200 = 1, BAUD38400 = 2, BAUD57600 = 3,
  BAUD115200 = 4 }

  Enumeration type for supported RS232 baudrates.
- enum pt_rs232_parity { NONE = 0, ODD = 1, EVEN = 2 }

  Enumeration type for supported RS232 parity modes.
- enum pt_rs232_stop { ONE_STOP = 0, TWO_STOP = 1 }

  Enumeration type for supported RS232 stop bit length.

Functions

- POWERTRACER_API PT_STATUS pt_set_read_timeout (pt_device *powertracer_device, unsigned int timeout)

  Set the read timeout to the USB communication.
- POWERTRACER_API PT_STATUS pt_set_write_timeout (pt_device *powertracer_device, unsigned int timeout)

  Set the write timeout to the USB communication.
- POWERTRACER_API PT_STATUS pt_set_communication_channel (pt_device *powertracer_device, pt_com_channel channel_sel)

  Select the communication channel, through which data bytes are sent and received.
- POWERTRACER_API PT_STATUS pt_write (pt_device *powertracer_device, unsigned char *command, unsigned int command_length)

  Write bytes over the selected communication channel.
- POWERTRACER_API PT_STATUS pt_read (pt_device *powertracer_device, unsigned char *response, unsigned int read_length, unsigned int *actual_read_length)

  Read bytes over the selected communication channel.
- POWERTRACER_API PT_STATUS pt_set_f_d (pt_device *powertracer_device, unsigned int f, unsigned int d)

  Configure the clock rate conversion integer (F) and baudrate adjustment integer (D) required for smart card communication.
- POWERTRACER_API PT_STATUS pt_set_cgt (pt_device *powertracer_device, unsigned int cgt_value)

  Configure the character guard time required for smart card communication.
- POWERTRACER_API PT_STATUS pt_enable_tx_error_checking (pt_device *powertracer_device, BOOL enabled)

  Enable/Disable the smart card transmitter from checking an ACK/NACK from the smart card during a character frame.
- POWERTRACER_API PT_STATUS pt_enable_rx_error_signaling (pt_device *powertracer_device, BOOL enabled)

  Enable/Disable the smart card receiver from signaling an ACK/NACK to the smart card during a character frame.
- POWERTRACER_API PT_STATUS pt_set_tx_error_handling (pt_device *powertracer_device, pt_tx_error_handling error_handling)

  Configure the behavior of the transmitter when an parity error has been detected.
- POWERTRACER_API PT_STATUS pt_get_tx_error_status (pt_device *powertracer_device, BOOL *error_occured, unsigned int *error_index)

  Check if any smart card transmission error has occured. Report the index of the error byte if an error has been detected.

- POWERTRACER_API PT_STATUS pt_get_rx_error_status (pt_device *powertracer_device, BOOL *error_occured)

  Check if any smart card reception error has occured.
- POWERTRACER_API PT_STATUS pt_get_communication_channel (pt_device *powertracer_device, pt_com_channel *com_channel)

  Get the currently selected communication channel.
- POWERTRACER_API PT_STATUS pt_get_etu (pt_device *powertracer_device, unsigned int *etu)

  Get the elementary time unit (ETU) setting used to setup the communication between Power Tracer and smart card.
- POWERTRACER_API PT_STATUS pt_get_cgt (pt_device *powertracer_device, unsigned int *cgt)

  Get the character guard time (CGT) setting used to setup the communication between Power Tracer and smart card.
- POWERTRACER_API PT_STATUS pt_is_tx_error_checking_enabled (pt_device *powertracer_device, BOOL *enabled)

  Check if the transmitter will look at the ACK/NACK bit to determine the correctness of transmission.
- POWERTRACER_API PT_STATUS pt_is_rx_error_signaling_enabled (pt_device *powertracer_device, BOOL *enabled)

  Check if the receiver will signal the smart card a NACK bit when parity error is detected.
- POWERTRACER_API PT_STATUS pt_is_smartcard_powered (pt_device *powertracer_device, BOOL *powered)

  Check if the smart card is powered by the Power Tracer.
- POWERTRACER_API PT_STATUS pt_set_rs232_config (pt_device *powertracer_device, pt_rs232_baudrate baudrate, pt_rs232_parity parity, pt_rs232_stop stop)

  Configure the paramters of EXT_SERIAL channel.
- POWERTRACER_API PT_STATUS pt_get_rs232_config (pt_device *powertracer_device, pt_rs232_baudrate *baudrate, pt_rs232_parity *parity, pt_rs232_stop *stop)

  Obtain configured parameters of EXT_SERIAL channel.

5.3.1 Detailed Description

This group of API calls can be used to tune various communication related parameters for either smart card or RS232 serial communication channel.

5.3.2 Enumeration Type Documentation

5.3.2.1 enum pt_com_channel

Enumeration type for the Power Tracer communication channels.

Enumerator:

SMARTCARD Select smart card interface as primary communication channel.

EXT_SERIAL Select external serial interface as primary communication channel.

Definition at line 72 of file powertracer_api.h.

5.3.2.2 enum pt_rs232_baudrate

Enumeration type for supported RS232 baudrates.

Enumerator:

BAUD9600 9600 bps baudrate.

BAUD19200 19200 bps baudrate.

BAUD38400 38400 bps baudrate.

BAUD57600 57600 bps baudrate.

BAUD115200 115200 bps baudrate.

Definition at line 81 of file powertracer_api.h.

5.3.2.3 enum pt_rs232_parity

Enumeration type for supported RS232 parity modes.

Enumerator:

NONE Generate/Expect NO parity in the serial transmitter/receiver.

ODD Generate/Expect ODD parity in the serial transmitter/receiver.

EVEN Generate/Expect EVEN parity in the serial transmitter/receiver.

Definition at line 93 of file powertracer_api.h.

5.3.2.4 enum pt_rs232_stop

Enumeration type for supported RS232 stop bit length.

Enumerator:

ONE_STOP Generate/Expect ONE stop bit before transmission of another character can be started in the serial transmitter/receiver.

TWO_STOP Generate/Expect TWO stop bit before transmission of another character can be started in the serial transmitter/receiver.

Definition at line 103 of file powertracer_api.h.

5.3.2.5 enum pt_tx_error_handling

Enumeration type for the smart card transmitter error handling.

Enumerator:

CONTINUE Continue the transmission when a parity error has been detected, but the index of the first error charater will be returned (ZERO indexed).

HALT Halt the transmission if a parity error has been detected. The left-over data will be discarded and the index of the first error character will be returned (ZERO indexed).

Definition at line 55 of file powertracer_api.h.

5.3.3 Function Documentation

5.3.3.1 POWERTRACER_API PT_STATUS pt_enable_rx_error_signaling (pt_device * *powertracer_device,* BOOL *enabled*)

Enable/Disable the smart card receiver from signaling an ACK/NACK to the smart card during a character frame.

NOTE: This configuration only affects the SMARTCARD communication channel.

NOTE: When the parity ACK/NACK signaling is disabled, the Power Tracer smart card receiver will not inspect the parity correctness of a received character and always ACK on character.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
enabled	A boolean variable serving as a control switch. TRUE/FALSE for on/off.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_is_rx_error_signaling_enabled()

5.3.3.2 POWERTRACER_API PT_STATUS pt_enable_tx_error_checking (pt_device * *powertracer_device,* BOOL *enabled*)

Enable/Disable the smart card transmitter from checking an ACK/NACK from the smart card during a character frame.

NOTE: When the parity ACK/NACK check is disabled, the smart card transmitter will assume all characters are correctly received by the smart card.

NOTE: To support character retransmission with T=0 card, the transmitter error handling has to be set to 'HALT' mode with pt_set_tx_error_handling().

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
enabled	A boolean variable serving as a control switch. TRUE/FALSE for on/off.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_is_tx_error_checking_enabled()
pt_set_tx_error_handling()

5.3.3.3 POWERTRACER_API PT_STATUS pt_get_cgt (pt_device * *powertracer_device,* unsigned int * *cgt*)

Get the character guard time (CGT) setting used to setup the communication between Power Tracer and smart card.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
cgt	A pointer to an unsigned integer that stores the obtained CGT setting.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_cgt()

5.3.3.4 POWERTRACER_API PT_STATUS pt_get_communication_channel (pt_device * *powertracer_device*, pt_com_channel * *com_channel*)

Get the currently selected communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
com_channel	A pointer to pt_com_channel enumeration type bearing the returned selected channel. Possible returned values are: **SMARTCARD**, if data bytes are to be sent through the Power Tracer smart card serial interface, **EXT_SERIAL**, if data bytes are to be sent through the Power Tracer RS232 serial interface.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_communication_channel()

5.3.3.5 POWERTRACER_API PT_STATUS pt_get_etu (pt_device * *powertracer_device,* unsigned int * *etu*)

Get the elementary time unit (ETU) setting used to setup the communication between Power Tracer and smart card.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
etu	A pointer to an unsigned integer that stores the obtained ETU setting.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_f_d()

5.3.3.6 POWERTRACER_API PT_STATUS pt_get_rs232_config (pt_device * *powertracer_device,* pt_rs232_baudrate * *baudrate,* pt_rs232_parity * *parity,* pt_rs232_stop * *stop*)

Obtain configured parameters of EXT_SERIAL channel.

NOTE: This function does not affect SMARTCARD communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
baudrate	A pointer to the pt_rs232_baudrate enumeration structure. Possible values are: **BAUD9600**, for 9600 bps baudrate, **BAUD19200**, for 19200 bps baudrate, **BAUD38400**, for 38400 bps baudrate, **BAUD57600**, for 57600 bps baudrate, **BAUD115200**, for 115200 bps baudrate,
parity	A pointer to the pt_rs232_parity enumeration structure. Possible values are: **NONE**, if parity bit will not be included, **ODD**, if the odd-parity bit will be included, **EVEN**, if the even-parity bit will be included.
stop	A pointer to the pt_rs232_stop enumeration structure Possible values are: **ONE_STOP**, if one stop bit is attached to the end of a character frame, **TWO_STOP**, if two stop bits are attached to the end of a character frame.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_rs232_config()

5.3.3.7 POWERTRACER_API PT_STATUS pt_get_rx_error_status (pt_device * *powertracer_device,* BOOL * *error_occured*)

Check if any smart card reception error has occured.

NOTE: The regisers storing error infomation will be cleared to 'Error never happend' status after the function call.

NOTE: This function only reports Rx error status of SMARTCARD channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
error_occured	A boolean variable indicating if any error has occured.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_enable_rx_error_signaling ()

5.3.3.8 POWERTRACER_API PT_STATUS pt_get_tx_error_status (pt_device * *powertracer_device,* BOOL * *error_occured,* unsigned int * *error_index*)

Check if any smart card transmission error has occured. Report the index of the error byte if an error has been detected.

NOTE: It is recommended to configure the transmitter error handling to HALT if the detection of transmission error is critical. Since only the latest transmssion error is reported.

NOTE: The regisers storing error infomation will be cleared to 'Error never happend' status after the function call.

NOTE: This function only reports Tx error status of SMARTCARD channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
error_occured	A pointer to a boolean variable indicating if any error has occured.
error_index	A pointer to an unsigned integer that stores the index to the error character. (0xffff if no error occurred)

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_enable_tx_error_checking()

5.3.3.9 POWERTRACER_API PT_STATUS pt_is_rx_error_signaling_enabled (pt_device * *powertracer_device*, BOOL * *enabled*)

Check if the receiver will signal the smart card a NACK bit when parity error is detected.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
enabled	A pointer to a boolean variable that stores the obtained inquiry result.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_enable_rx_error_signaling ()

5.3.3.10 POWERTRACER_API PT_STATUS pt_is_smartcard_powered (pt_device * *powertracer_device*, BOOL * *powered*)

Check if the smart card is powered by the Power Tracer.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
powered	A pointer to a boolean variable that stores the obtained inquiry result.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_smartcard_powerup()
pt_smartcard_powerdown()

5.3.3.11 POWERTRACER_API PT_STATUS pt_is_tx_error_checking_enabled (pt_device * *powertracer_device,* BOOL * *enabled*)

Check if the transmitter will look at the ACK/NACK bit to determine the correctness of transmission.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
enabled	A pointer to a boolean variable that stores the obtained inquiry result.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_enable_tx_error_checking()

5.3.3.12 POWERTRACER_API PT_STATUS pt_read (pt_device * *powertracer_device,* unsigned char * *response,* unsigned int *read_length,* unsigned int * *actual_read_length*)

Read bytes over the selected communication channel.

NOTE: The read timeout specified pt_set_read_timeout() also applies to this function.

NOTE: The function will try to read the indicated number of bytes from the communication channel set by pt_set_communication_channel().

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
response	A pointer to a byte array buffer where fetched data bytes are stored.
read_length	The desired length of bytes to read into the byte array. The length should not exceed 1024. **NOTE:** Assigning ZERO to this parameter cause the function to return any bytes (no more than 1024 bytes) that are currently available from the selected communication channel.
actual_read_length	The number of bytes being actually fetched out.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_LENGTH_OUT_RANGE if the read length is out of range.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_write()
pt_set_communication_channel()

5.3.3.13 POWERTRACER_API PT_STATUS pt_set_cgt (pt_device * *powertracer_device*, unsigned int *cgt_value*)

Configure the character guard time required for smart card communication.

NOTE: The default CGT=12 when the Power Tracer is powered up.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
cgt_value	The character guard time (CGT) to be configured, ranging from 11 to 511.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_cgt()

5.3.3.14 POWERTRACER_API PT_STATUS pt_set_communication_channel (pt_device * *powertracer_device,* pt_com_channel *channel_sel*)

Select the communication channel, through which data bytes are sent and received.

NOTE: When the Power Tracer is powered up, the default communication channel is SMARTCARD.

NOTE: When a communication channel is de-selected, its transceivers are put into reset and unfetched data is lost.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
channel_sel	The new communication channel to be selected and used. Possible values are: **SMARTCARD**, if data bytes are to be sent through the Power Tracer smart card serial interface, **EXT_SERIAL**, if data bytes are to be sent through the Power Tracer RS232 serial interface.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_INVALID_COM_CHANNEL if an invalid communication channel was given to the function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_communication_channel()

5.3.3.15 POWERTRACER_API PT_STATUS pt_set_f_d (pt_device * *powertracer_device,* unsigned int *f,* unsigned int *d*)

Configure the clock rate conversion integer (F) and baudrate adjustment integer (D) required for smart card communication.

NOTE: The default F=372 and D=1 when the Power Tracer is powered up.

NOTE: The rounded up quotient of F divided by D should be equal or larger than 5 and should not exceed 4095.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
f	The desired clock rate conversion interger value.
d	The desired baudrate adjustment integer value.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_etu()

5.3.3.16 POWERTRACER_API PT_STATUS pt_set_read_timeout (pt_device * *powertracer_device,* unsigned int *timeout*)

Set the read timeout to the USB communication.

NOTE: This parameter is also used as read timeout of pt_read() function.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
timeout	The timeout length in miliseconds.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_write_timeout()

5.3.3.17 POWERTRACER_API PT_STATUS pt_set_rs232_config (pt_device * *powertracer_device,* pt_rs232_baudrate *baudrate,* pt_rs232_parity *parity,* pt_rs232_stop *stop*)

Configure the paramters of EXT_SERIAL channel.

NOTE: This function does not affect SMARTCARD communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.

baudrate	Enumeration type defining a set of supported baudrates. Possible values are: **BAUD9600**, for 9600 bps baudrate, **BAUD19200**, for 19200 bps baudrate, **BAUD38400**, for 38400 bps baudrate, **BAUD57600**, for 57600 bps baudrate, **BAUD115200**, for 115200 bps baudrate,
parity	Enumeration type defining supported parity mode. Possible values are: **NONE**, if parity bit will not be included, **ODD**, if the odd-parity bit will be included, **EVEN**, if the even-parity bit will be included.
stop	Enumeration type defining supported stop bit length. Possible values are: **ONE_STOP**, if one stop bit is attached to the end of a character frame, **TWO_STOP**, if two stop bits are attached to the end of a character frame.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_INVALID_BAUDRATE if an invalid baudrate was chosen.
PT_STATUS.PT_INVALID_PARITY if an invalid parity mode was chosen.
PT_STATUS.PT_INVALID_STOP if an invalid stop bit length was chosen.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_rs232_config()

5.3.3.18 POWERTRACER_API PT_STATUS pt_set_tx_error_handling (pt_device * *powertracer_device,* pt_tx_error_handling *error_handling*)

Configure the behavior of the transmitter when an parity error has been detected.

NOTE: This configuration only affects the SMARTCARD communication channel.

NOTE: The transmitter error checking has to be enabled, with pt_enable_tx_error_checking(), in order to support 'HALT' error handling.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
error_handling	The desired error handling behaviour. Possible values are COUNTINUE and HALT.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_enable_tx_error_checking()

5.3.3.19 POWERTRACER_API PT_STATUS pt_set_write_timeout (pt_device * *powertracer_device,* unsigned int *timeout*)

Set the write timeout to the USB communication.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
timeout	The timeout length in miliseconds.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_read_timeout()

5.3.3.20 POWERTRACER_API PT_STATUS pt_write (pt_device * *powertracer_device,* unsigned char * *command,* unsigned int *command_length*)

Write bytes over the selected communication channel.

NOTE: The function call will be blocked until the actual transmission is complete.

NOTE: Data bytes sent by this function are directed to the communication channel set by pt_set_communication_channel().

NOTE: Upon finishing sending data bytes, the trigger status of Power Tracer is forced to 'disarmed'.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
command	A pointer to a byte array containing the bytes to be transmitted.
command_length	The total length of the command frame in byte. The length should not exceed 1024 bytes.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_LENGTH_OUT_RANGE if an invalid command length was specified.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_read()
pt_set_communication_channel()

5.4 Triggering

Functions

- POWERTRACER_API PT_STATUS pt_set_armed (pt_device *powertracer_device, BOOL arm)

  Arm/Disarm the Power Tracer so that it could generate a series of events such as signaling a trigger, powering the card with capacitor(SMARTCARD channel only) and temporarily cutting the card clock supply(SMARTCARD channel only) immediatly after the transmission of the last byte with an optional delay.
- POWERTRACER_API PT_STATUS pt_set_trigger_delay (pt_device *powertracer_device, unsigned int delay)

  Set the trigger delay.
- POWERTRACER_API PT_STATUS pt_set_dcdc_off_delay (pt_device *powertracer_device, unsigned int delay)

  Set the delay before the capacitor array are turned on and supply power to the smart card.
- POWERTRACER_API PT_STATUS pt_set_dcdc_off_duration (pt_device *powertracer_device, unsigned int duration)

  Set the duration of using the capacitor array for the smart card power supply.
- POWERTRACER_API PT_STATUS pt_set_clock_off_delay (pt_device *powertracer_device, unsigned int delay)

  Set the delay before turning of the smart card clock supply.
- POWERTRACER_API PT_STATUS pt_set_clock_off_duration (pt_device *powertracer_device, unsigned int duration)

  Set the duration of turning off the smart card clock supply.
- POWERTRACER_API PT_STATUS pt_get_trigger_delay (pt_device *powertracer_device, unsigned int *delay)

  Get the trigger delay setting.
- POWERTRACER_API PT_STATUS pt_is_armed (pt_device *powertracer_device, BOOL *armed)

  Check if the Power Tracer is armed to generate timed events(i.e. triggering, switching to capacitor power supply and clock gating).
- POWERTRACER_API PT_STATUS pt_get_dcdc_off_delay (pt_device *powertracer_device, unsigned int *delay)

  Get the delay before the capacitor array is switched on for the smart card power supply.
- POWERTRACER_API PT_STATUS pt_get_dcdc_off_duration (pt_device *powertracer_device, unsigned int *duration)

  Get the duration of using the capacitor array as the smart card power supply.
- POWERTRACER_API PT_STATUS pt_get_clock_off_delay (pt_device *powertracer_device, unsigned int *delay)

  Get the delay before the smart card clock supply is switched off.
- POWERTRACER_API PT_STATUS pt_get_clock_off_duration (pt_device *powertracer_device, unsigned int *duration)

  Get the duration of the smart card clock supply being switched off.

5.4.1 Detailed Description

This group of API calls defines the behavior of triggering mechanism of Power Tracer device.

5.4.2 Function Documentation

5.4.2.1 POWERTRACER_API PT_STATUS pt_get_clock_off_delay (pt_device * *powertracer_device,* unsigned int * *delay*)

Get the delay before the smart card clock supply is switched off.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
delay	A pointer to an unsigned integer that stores the obtained delay value.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_clock_off_delay()
pt_set_clock_off_duration()
pt_get_clock_off_duration()

5.4.2.2 POWERTRACER_API PT_STATUS pt_get_clock_off_duration (pt_device * *powertracer_device*, unsigned int * *duration*)

Get the duration of the smart card clock supply being switched off.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
duration	A pointer to an unsigned integer that stores the obtained duration value.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_clock_off_duration()
pt_set_clock_off_delay()
pt_get_clock_off_delay()

5.4.2.3 POWERTRACER_API PT_STATUS pt_get_dcdc_off_delay (pt_device * *powertracer_device,* unsigned int * *delay*)

Get the delay before the capacitor array is switched on for the smart card power supply.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
delay	A pointer to an unsigned integer that stores the obtained delay value.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_dcdc_off_delay()
pt_set_dcdc_off_duration()
pt_get_dcdc_off_duration()

5.4.2.4 POWERTRACER_API PT_STATUS pt_get_dcdc_off_duration (pt_device * *powertracer_device,* unsigned int * *duration*)

Get the duration of using the capacitor array as the smart card power supply.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
duration	A pointer to an unsigned integer that stores the obtained duration value.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_dcdc_off_duration()
pt_set_dcdc_off_delay()
pt_get_dcdc_off_delay()

5.4.2.5 POWERTRACER_API PT_STATUS pt_get_trigger_delay (pt_device * *powertracer_device,* unsigned int * *delay*)

Get the trigger delay setting.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
delay	A pointer to an unsigned integer that stores the obtained delay value.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_trigger_delay()

5.4.2.6 POWERTRACER_API PT_STATUS pt_is_armed (pt_device * *powertracer_device,* BOOL * *armed*)

Check if the Power Tracer is armed to generate timed events(i.e. triggering, switching to capacitor power supply and clock gating).

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
armed	A pointer to a boolean variable that stores the obtained inquiry result.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_armed()

5.4.2.7 POWERTRACER_API PT_STATUS pt_set_armed (pt_device * *powertracer_device,* BOOL *arm*)

Arm/Disarm the Power Tracer so that it could generate a series of events such as signaling a trigger, powering the card with capacitor(SMARTCARD channel only) and temporarily cutting the card clock supply(SMARTCARD channel only) immediatly after the transmission of the last byte with an optional delay.

NOTE: To manipulate the timing of the triggering, capacitor powering and clock gating, see pt_set_trigger_delay(), pt_set_dcdc_off_delay(), pt_set_dcdc_off_duration(), pt_set_clock_off_delay(), and pt_set_clock_off_duration() functions.

NOTE: This function affects the trigger behavior under both SMARTCARD channel and EXT_SERIAL channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
arm	if 'TRUE' then the function will try to arm the trigger, disarm it otherwise.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_CAPACITOR_LOW_VOLTAGE if the internal capacitors are not sufficiently charged.
PT_TRIGGER_NOT_READY if the trigger is still busy and not ready for the next operation. PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_is_armed()

5.4.2.8 POWERTRACER_API PT_STATUS pt_set_clock_off_delay (pt_device * *powertracer_device,* unsigned int *delay*)

Set the delay before turning of the smart card clock supply.

NOTE: The delay referred here is relative to the start of transmission of the last command byte.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
delay	delay The delay in microsecond. It should not exceed 4,995,904 microseconds. **NOTE:** Assigning 0 to this parameter to indicate that no delay should apply on the generation of this event.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_LENGTH_OUT_RANGE if the specified **delay** length is out of valid range.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_clock_off_delay()
pt_set_clock_off_duration()
pt_get_clock_off_duration()

5.4.2.9 POWERTRACER_API PT_STATUS pt_set_clock_off_duration (pt_device * *powertracer_device,* unsigned int *duration*)

Set the duration of turning off the smart card clock supply.

NOTE: The duration will apply after the delay setting to this event expires.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
duration	The duration in microsecond. Duration = 0 indicates the clock should not be turned off and delay value will be ignored.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_LENGTH_OUT_RANGE if the specified **duration** length is out of valid range.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_clock_off_duration()
pt_set_clock_off_delay()
pt_get_clock_off_duration()

5.4.2.10 POWERTRACER_API PT_STATUS pt_set_dcdc_off_delay (pt_device * *powertracer_device,* unsigned int *delay*)

Set the delay before the capacitor array are turned on and supply power to the smart card.

NOTE: The delay referred here is relative to the start of transmission of the last command byte.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
delay	The delay in microsecond. It should not exceed 4,995,904 microseconds. **NOTE:** Assigning 0 to this parameter to indicate that no delay should apply on the generation of this event.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_LENGTH_OUT_RANGE if the specified **delay** length is out of valid range.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_dcdc_off_delay()
pt_set_dcdc_off_duration()
pt_get_dcdc_off_duration()

5.4.2.11 POWERTRACER_API PT_STATUS pt_set_dcdc_off_duration (pt_device * *powertracer_device,* unsigned int *duration*)

Set the duration of using the capacitor array for the smart card power supply.

NOTE: The duration will apply after the delay setting to this event expires.

NOTE: This function does not affect EXT_SERIAL communication channel.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
duration	The duration in microsecond. It should not exceed 5,000,000 microseconds. **NOTE:** Assigning 0 to this parameter to indicate that this event should not be generated and ignore its delay setting.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_LENGTH_OUT_RANGE if the specified **duration** length is out of valid range.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_dcdc_off_duration()
pt_set_dcdc_off_delay()
pt_get_dcdc_off_delay()

5.4.2.12 POWERTRACER_API PT_STATUS pt_set_trigger_delay (pt_device * *powertracer_device,* unsigned int *delay*)

Set the trigger delay.

NOTE: The delay referred here is relative to the end of the last byte transmission.

NOTE: This function affects the trigger behavior under both SMARTCARD channel and EXT_SERIAL channel

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
delay	The delay in microsecond. It should not exceed 4,995,904 microseconds. **NOTE:** Assigning 0 to this parameter to indicate that no delay should apply on the generation of this event.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_LENGTH_OUT_RANGE if the specified **delay** length is out of valid range.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_trigger_delay()

5.5 Measurement Control

Functions

- POWERTRACER_API PT_STATUS pt_set_offset (pt_device *powertracer_device, double offset)

  Set the offset of the Power Tracer Signal port.
- POWERTRACER_API PT_STATUS pt_set_offset_by_current (pt_device *powertracer_device, int offset_current)

  Set the offset current to the Power Tracer Signal port.
- POWERTRACER_API PT_STATUS pt_set_gain (pt_device *powertracer_device, unsigned int gain)

  Set the gain of the Power Tracer "signal" port.
- POWERTRACER_API PT_STATUS pt_get_offset (pt_device *powertracer_device, double *offset)

  Get the offset voltage applied on the Power Tracer Signal port.
- POWERTRACER_API PT_STATUS pt_get_offset_by_current (pt_device *powertracer_device, int *offset_current)

  Get the offset current applied on the Power Tracer Signal port.
- POWERTRACER_API PT_STATUS pt_get_gain (pt_device *powertracer_device, unsigned int *gain)

  Get the gain applied on the Power Tracer 'signal' port.
- POWERTRACER_API PT_STATUS pt_get_offset_boundary (pt_device *powertracer_device, double *min, double *max)

  Get maximum and minimum supported offset voltage.
- POWERTRACER_API PT_STATUS pt_get_offset_boundary_by_current (pt_device *powertracer_device, int *min, int *max)

  Get maximum and minimum supported offset current.
- POWERTRACER_API PT_STATUS pt_get_gain_boundary (pt_device *powertracer_device, unsigned int *min, unsigned int *max)

  Get maximum and minimum supported gain percentage.
- POWERTRACER_API PT_STATUS pt_get_offset_stepsize (pt_device *powertracer_device, double *step_size)

  Get the resolution of the Power Tracer 'signal' port offset setting.
- POWERTRACER_API PT_STATUS pt_get_offset_stepsize_by_current (pt_device *powertracer_device, int *step_size)

  Get the offset current setpoint resolution.
- POWERTRACER_API PT_STATUS pt_get_gain_stepsize (pt_device *powertracer_device, unsigned int *step_size)

  Get the resolution of the Power Tracer 'signal' port gain setting.

5.5.1 Detailed Description

This group of API calls can be used to tune analog parameter that affects the smart card power consumption measurment.

5.5.2 Function Documentation

5.5.2.1 POWERTRACER_API PT_STATUS pt_get_gain (pt_device * *powertracer_device,* unsigned int * *gain*)

Get the gain applied on the Power Tracer 'signal' port.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
gain	A pointer to a double variable that stores the obtained gain value.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_gain()
pt_get_gain_boundary()
pt_get_gain_stepsize()

5.5.2.2 POWERTRACER_API PT_STATUS pt_get_gain_boundary (pt_device * *powertracer_device,* unsigned int * *min,* unsigned int * *max*)

Get maximum and minimum supported gain percentage.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
min	A pointer to an unsigned integer variable bearing the returned minimum gain allowed.
max	A pointer to an unsigned integer variable bearing the returned maximum gain allowed.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_gain()
pt_get_gain()
pt_get_gain_stepsize()

5.5.2.3 POWERTRACER_API PT_STATUS pt_get_gain_stepsize (pt_device * *powertracer_device,* unsigned int * *step_size*)

Get the resolution of the Power Tracer 'signal' port gain setting.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
step_size	A pointer to an unsigned integer variable bearing the returned gain resolution.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_gain()
pt_get_gain()
pt_get_gain_boundary()

5.5.2.4 POWERTRACER_API PT_STATUS pt_get_offset (pt_device * *powertracer_device,* double * *offset*)

Get the offset voltage applied on the Power Tracer Signal port.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
offset	A pointer to a double variable that stores the obtained offset value.

Warning

Not thread safe !

Deprecated Replacement function pt_get_offset_by_current()

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_offset()
pt_get_offset_boundary()
pt_get_offset_stepsize()

5.5.2.5 POWERTRACER_API PT_STATUS pt_get_offset_boundary (pt_device * *powertracer_device,* double * *min,* double * *max*)

Get maximum and minimum supported offset voltage.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
min	A pointer to a double variable bearing the returned minimum offset allowed.
max	A pointer to a double variable bearing the returned maximum offset allowed.

Warning

Not thread safe !

Deprecated Replacement function pt_get_offset_boundary_by_current()

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_offset()
pt_get_offset()
pt_get_offset_stepsize()

5.5.2.6 POWERTRACER_API PT_STATUS pt_get_offset_boundary_by_current (pt_device * *powertracer_device*, int * *min*, int * *max*)

Get maximum and minimum supported offset current.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
min	A pointer to a double variable bearing the returned minimum offset current allowed.
max	A pointer to a double variable bearing the returned maximum offset current allowed.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_offset_by_current()
pt_get_offset_by_current()
pt_get_offset_stepsize_by_current()

5.5.2.7 POWERTRACER_API PT_STATUS pt_get_offset_by_current (pt_device * *powertracer_device,* int * *offset_current*)

Get the offset current applied on the Power Tracer Signal port.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
offset_current	A pointer to a integer variable that stores the obtained offset current value.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STAUTS.PT_OFFSET_OUT_RANGE if the offset current cannot be determined.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_offset_by_current()
pt_get_offset_boundary_by_current()
pt_get_offset_stepsize_by_current()

Definition at line 1461 of file powertracer_api.c.

5.5.2.8 POWERTRACER_API PT_STATUS pt_get_offset_stepsize (pt_device * *powertracer_device,* double * *step_size*)

Get the resolution of the Power Tracer 'signal' port offset setting.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
step_size	A pointer to a double variable bearing the returned offset resolution.

Warning

Not thread safe !

Deprecated Replacement function pt_get_offset_stepsize_by_current()

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_offset()
pt_get_offset()
pt_get_offset_boundary()

5.5.2.9 POWERTRACER_API PT_STATUS pt_get_offset_stepsize_by_current (pt_device * *powertracer_device,* int * *step_size*)

Get the offset current setpoint resolution.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
step_size	A pointer to a double variable bearing the returned offset resolution.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_set_offset_by_current()
pt_get_offset_by_current()
pt_get_offset_boundary_by_current()

5.5.2.10 POWERTRACER_API PT_STATUS pt_set_gain (pt_device * *powertracer_device,* unsigned int *gain*)

Set the gain of the Power Tracer "signal" port.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
gain	The desired gain value to be configured. The gain is expressed by an integer ranging from 100 $\sim$ 200 to represent 100% $\sim$ 200%, with resolution of 1%. When Power Tracer is powered up, the default gain value is 100.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.

PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STAUTS.PT_GAIN_OUT_RANGE if the gain to be configured is out of the range.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_gain()
pt_get_gain_boundary()
pt_get_gain_stepsize()

5.5.2.11 POWERTRACER_API PT_STATUS pt_set_offset (pt_device * *powertracer_device,* double *offset*)

Set the offset of the Power Tracer Signal port.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
offset	The desired offset value to be configured. Ranging from -2.0 volt to 2.0 volt, with resolution of 0.1 volt. When Power Tracer is powered up, the default offset is 0.0 volt.

Warning

Not thread safe !

Deprecated Replacement function pt_set_offset_by_current()

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STAUTS.PT_OFFSET_OUT_RANGE if the offset to be configured is out of the range.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_offset()
pt_get_offset_boundary()
pt_get_offset_stepsize()

5.5.2.12 POWERTRACER_API PT_STATUS pt_set_offset_by_current (pt_device * *powertracer_device,* int *offset_current*)

Set the offset current to the Power Tracer Signal port.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
offset_current	The offset current(mA) for the Power Tracer Signal port, ranging from -30 to 0.

Warning

Not thread safe ! PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STAUTS.PT_OFFSET_OUT_RANGE if the offset to be configured is out of the range.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_offset_by_current()
pt_get_offset_boundary_by_current()
pt_get_offset_stepsize_by_current()

Definition at line 941 of file powertracer_api.c.

5.6 Miscellaneous

Data Structures

- struct pt_version

  Data structure for the Power Tracer version information.

Enumerations

- enum PT_STATUS {
  PT_OK = 0, PT_NOT_FOUND = 1, PT_NOT_COMPATIBLE = 2, PT_VOLTAGE_OUT_RANGE = 3,
  PT_OFFSET_OUT_RANGE = 4, PT_GAIN_OUT_RANGE = 5, PT_CAPACITOR_LOW_VOLTAGE = 6, PT_IO_ERROR = 7,
  PT_INVALID_HANDLE = 8, PT_INVALID_COM_CHANNEL = 9, PT_INVALID_BAUDRATE = 10, PT_INVALID_PARITY = 11,
  PT_INVALID_STOP = 12, PT_FAILED = 13, PT_ALREADY_OPENED = 14, PT_NOT_OPENED = 15,
  PT_LENGTH_OUT_RANGE = 16, PT_TRIGGER_NOT_READY = 17 }

  Enumeration type of API function return status.
- enum pt_pcb_version { PT_PCB_VERSION_4B = 0x31 }

  Enumeration type for the Power Tracer PCB version.

Functions

- POWERTRACER_API PT_STATUS pt_get_version (pt_device *powertracer_device, unsigned char *version_buffer, unsigned int buffer_size)

  Obtain the device version, and print it into a byte array in the format of XX.YY.ZZ, where X stands for PCB version, Y for bitstream version and Z for firmware version.
- POWERTRACER_API PT_STATUS pt_is_smartcard_inserted (pt_device *powertracer_device, BOOL *inserted)

  Check if the smart card is inserted.
- POWERTRACER_API PT_STATUS pt_get_hw_build_time (pt_device *powertracer_device, unsigned int *build_time)

  Fetch the Power Tracer bitstream build time as 32-bit integer representing seconds since the epoch (Jan. 1st, 01:00:00, 1970).
- POWERTRACER_API PT_STATUS pt_get_hw_build_id (pt_device *powertracer_device, unsigned char *id_buffer, unsigned int buf_len)

  Fetch the Power Tracer bitstream build ID in the form of ASCII string.
- POWERTRACER_API PT_STATUS pt_test (pt_device *powertracer_device, void *data)

  Undocumented test function for Riscure internal use only.
- POWERTRACER_API PT_STATUS pt_sdk_get_cached_firmware_count (unsigned int *n_cached_fw)

  Obtain the number of firmware binaries cached in SDK library.
- POWERTRACER_API PT_STATUS pt_sdk_get_cached_firmware_build_time (unsigned int cache_index, pt_version *version, unsigned int *build_time)

  Obtain the build time of the indexed firmware cache entry, together with its version.
- POWERTRACER_API PT_STATUS pt_sdk_get_cached_firmware_build_id (unsigned int cache_index, pt_version *version, unsigned char *buf_id, unsigned int len_buf)

  Obtain the build if of the indexed firmware cache entry, together with its version.
- POWERTRACER_API char * pt_sdk_get_build_time ()

  Obtain the SDK build time as ASCII string with the format of yyyy-MM-dd-HH:mm:ss.
- POWERTRACER_API char * pt_sdk_get_version ()

  Get the SDK version number.
- POWERTRACER_API BOOL pt_sdk_is_snapshot_version ()

  Checks if this version of the SDK is a snapshot (development) version. Make sure to always use a final version.

5.6.1 Detailed Description

This group of API calls provide information that could be found useful.

5.6.2 Enumeration Type Documentation

5.6.2.1 enum pt_pcb_version

Enumeration type for the Power Tracer PCB version.

Enumerator:

PT_PCB_VERSION_4B value assigned to PCB version 4B.

Definition at line 64 of file powertracer_api.h.

5.6.2.2 enum PT_STATUS

Enumeration type of API function return status.

Only critical failures are enumerated into separate items, other minor issues are enumerated by PT_FAILED.

Enumerator:

PT_OK Operation completed succesfully.

PT_NOT_FOUND Power Tracer could not be found.

PT_NOT_COMPATIBLE Function not compatible with this version Power Tracer.

PT_VOLTAGE_OUT_RANGE The voltage setting for the potmeter is out of range.

PT_OFFSET_OUT_RANGE The offset setting for the potmeter is out of range.

PT_GAIN_OUT_RANGE The gain setting for the potmeter is out of range.

PT_CAPACITOR_LOW_VOLTAGE The capacitors are not charged sufficiently.

PT_IO_ERROR USB Communication error.

PT_INVALID_HANDLE An invalid Power Tracer handle was specified.

PT_INVALID_COM_CHANNEL An invalid communication channel was selected.

PT_INVALID_BAUDRATE An invalid RS232 baudrate was specified.

PT_INVALID_PARITY An invalid RS232 parity mode was specified.

PT_INVALID_STOP An invalid RS232 stop bit length was specified.

PT_FAILED The operation failed (reason unknown).

PT_ALREADY_OPENED The Power Tracer has already been opened.

PT_NOT_OPENED The Power Tracer has not been opened.

PT_LENGTH_OUT_RANGE An invalid length was specified.

PT_TRIGGER_NOT_READY The trigger is not yet ready to be armed.

Definition at line 30 of file powertracer_api.h.

5.6.3 Function Documentation

5.6.3.1 POWERTRACER_API PT_STATUS pt_get_hw_build_id (pt_device * *powertracer_device,* unsigned char * *id_buffer,* unsigned int *buf_len*)

Fetch the Power Tracer bitstream build ID in the form of ASCII string.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
id_buffer	A pointer to a byte array where the build ID will be stored.
buf_len	The size of byte array. Minimum size of 14 bytes is required.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_hw_build_time()

Definition at line 1982 of file powertracer_api.c.

5.6.3.2 POWERTRACER_API PT_STATUS pt_get_hw_build_time (pt_device * *powertracer_device,* unsigned int * *build_time*)

Fetch the Power Tracer bitstream build time as 32-bit integer representing seconds since the epoch (Jan. 1st, 01:00:00, 1970).

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information. A pointer to a 32-bit integer where the time stamp will be stored.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_get_hw_build_id()

Definition at line 2002 of file powertracer_api.c.

5.6.3.3 POWERTRACER_API PT_STATUS pt_get_version (pt_device * *powertracer_device,* unsigned char * *version_buffer,* unsigned int *buffer_size*)

Obtain the device version, and print it into a byte array in the format of XX.YY.ZZ, where X stands for PCB version, Y for bitstream version and Z for firmware version.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
version_buffer	Pointer to the byte array where the version information is stored.
buffer_size	The length of the byte array, which should be at least 9 bytes large.(i.e. XX.YY.ZZ + '\0')

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_FAILED if an error occured during the action.

5.6.3.4 POWERTRACER_API PT_STATUS pt_is_smartcard_inserted (pt_device * *powertracer_device,* BOOL * *inserted*)

Check if the smart card is inserted.

Parameters

powertracer_device	A pointer to the Power Tracer data structure that stores the device information.
inserted	A pointer to a boolean variable that indicates whether the card is inserted.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_INVALID_HANDLE if an invalid handle was passed to the function.
PT_STATUS.PT_NOT_FOUND if the Power Tracer was not found.
PT_STATUS.PT_IO_ERROR if a communication error happened.
PT_STATUS.PT_NOT_COMPATIBLE if the Power Tracer in use does not support this function.
PT_STATUS.PT_FAILED if an error occured during the action.

5.6.3.5 POWERTRACER_API char* pt_sdk_get_build_time ( )

Obtain the SDK build time as ASCII string with the format of yyyy-MM-dd-HH:mm:ss.

Warning

Not thread safe !

Returns

SDK build timestamp.

See also

pt_sdk_get_version()

Definition at line 2130 of file powertracer_api.c.

5.6.3.6 POWERTRACER_API PT_STATUS pt_sdk_get_cached_firmware_build_id (unsigned int *cache_index,* pt_version * *version,* unsigned char * *buf_id,* unsigned int *len_buf*)

Obtain the build if of the indexed firmware cache entry, together with its version.

Parameters

cache_index	The index to the firmware cache entries, index valid from 0 to (n_cached_fw-1).
version	A pointer to pt_version structure, where version information is stored.
buf_id	A pointer to a bytearray where the build id is stored.
len_buf	The size of the bytearray.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_sdk_get_cached_firmware_count()
pt_sdk_get_cached_firmware_build_time()

Definition at line 2065 of file powertracer_api.c.

5.6.3.7 POWERTRACER_API PT_STATUS pt_sdk_get_cached_firmware_build_time (unsigned int *cache_index,* pt_version * *version,* unsigned int * *build_time*)

Obtain the build time of the indexed firmware cache entry, together with its version.

Parameters

cache_index	The index to the firmware cache entries, index valid from 0 to (n_cached_fw-1).
version	A pointer to pt_version structure, where version information is stored.
build_time	A pointer to an integer where a 32-bit time stamp is stored.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_sdk_get_cached_firmware_count()
pt_sdk_get_cached_firmware_build_id()

Definition at line 2036 of file powertracer_api.c.

5.6.3.8 POWERTRACER_API PT_STATUS pt_sdk_get_cached_firmware_count (unsigned int * n_cached_fw)

Obtain the number of firmware binaries cached in SDK library.

Parameters

n_cached_fw	A pointer to an integer value where the firmware count is stored.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.
PT_STATUS.PT_FAILED if an error occured during the action.

See also

pt_sdk_get_cached_firmware_build_time()

Definition at line 2022 of file powertracer_api.c.

5.6.3.9 POWERTRACER_API char* pt_sdk_get_version ( )

Get the SDK version number.

Warning

Not thread safe !

Returns

SDK version number.

See also

pt_sdk_get_build_time()

Definition at line 2122 of file powertracer_api.c.

5.6.3.10 POWERTRACER_API BOOL pt_sdk_is_snapshot_version ( )

Checks if this version of the SDK is a snapshot (development) version. Make sure to always use a final version.

Warning

Not thread safe !

Returns

TRUE if the SDK is a snapshot (development) version.
FALSE if the SDK is a final version.

Definition at line 2126 of file powertracer_api.c.

5.6.3.11 POWERTRACER_API PT_STATUS pt_test (pt_device * *powertracer_device,* void * *data*)

Undocumented test function for Riscure internal use only.

Warning

Not thread safe !

Returns

PT_STATUS.PT_OK if the action was performed successfully.

6 Data Structure Documentation

6.1 pt_device Struct Reference

Data structure for the Power Tracer device information.

`#include <powertracer_api.h>`

Data Fields

- DWORD locationId
- char serialNumber [16]
- char description [64]
- pt_handle handle
- pt_version version
- void * internal_config

6.1.1 Detailed Description

Data structure for the Power Tracer device information.

Definition at line 122 of file powertracer_api.h.

6.1.2 Field Documentation

6.1.2.1 char pt_device::description[64]

Description of the Power Tracer device.

Definition at line 125 of file powertracer_api.h.

6.1.2.2 pt_handle pt_device::handle

Handle of the Power Tracer device.

Definition at line 126 of file powertracer_api.h.

6.1.2.3 void* pt_device::internal_config

Structure for internal use.

Definition at line 128 of file powertracer_api.h.

6.1.2.4 DWORD pt_device::locationId

Location ID of the Power Tracer device (Obsolete).

Definition at line 123 of file powertracer_api.h.

6.1.2.5 char pt_device::serialNumber[16]

Serial number of the Power Tracer device.

Definition at line 124 of file powertracer_api.h.

6.1.2.6 pt_version pt_device::version

Version information.

Definition at line 127 of file powertracer_api.h.

6.2 pt_version Struct Reference

Data structure for the Power Tracer version information.

`#include <powertracer_api.h>`

Data Fields

- pt_pcb_version pcb_version
- unsigned char bitstream_version
- unsigned char firmware_version

6.2.1 Detailed Description

Data structure for the Power Tracer version information.

Definition at line 112 of file powertracer_api.h.

6.2.2 Field Documentation

6.2.2.1 unsigned char pt_version::bitstream_version

Bitstream version of Power Tracer.

Definition at line 114 of file powertracer_api.h.

6.2.2.2 unsigned char pt_version::firmware_version

Firmware version of Power Tracer.

Definition at line 115 of file powertracer_api.h.

6.2.2.3 pt_pcb_version pt_version::pcb_version

PCB version of Power Tracer.

Definition at line 113 of file powertracer_api.h.

Index