pico Technology

PicoScope® 2000 Series

PC Oscilloscopes

Programmer's Guide

ps2000pg.en r11

Contents

1 Introduction

1.1 Overview

The PicoScope 2000 Series PC Oscilloscopes are low-cost, high-performance instruments that are fully USB 2.0-capable and also backwards-compatible with USB 1.1. There is no need for an additional power supply, as power is taken from the USB port.

This manual explains how to develop your own programs for collecting and analyzing data from the PicoScope 2000 Series oscilloscopes. This manual describes the application programming interface (API) for the devices shown below.

- PicoScope 2104
- PicoScope 2105
- PicoScope 2202
- PicoScope 2203
- PicoScope 2204
- PicoScope 2205
- PicoScope 2204A
- PicoScope 2205A

The Pico Technology software development kit (SDK) is available on the *Pico Technology Software and Reference CD-ROM* and for free download from www.picotech.com/downloads.

1.2 Minimum system requirements

To ensure that your PicoScope 2000 Series PC Oscilloscope operates correctly, you must have a computer with at least the minimum system requirements to run one of the supported operating systems, as shown in the following table. The performance of the oscilloscope will be better with a more powerful PC, and will benefit from a multi-core processor.

Please note the PicoScope software is not installed as part of the SDK.

Item	Specification
Operating system	Windows 7, Windows 8, Windows 10
	32 bit and 64 bit versions supported
Processor	As required by the operating system
Memory	
Free disk space	
Ports	USB 1.1 compliant port (absolute minimum)* USB 2.0 or USB 3.0 compliant port

* The oscilloscope will run slowly on a USB 1.1 port. This configuration is not recommended.

1.3 Legal information

The material contained in this release is licensed, not sold. Pico Technology Limited grants a license to the person who installs this software, subject to the conditions listed below.

Access. The licensee agrees to allow access to this software only to persons who have been informed of these conditions and agree to abide by them.

Usage. The software in this release is for use only with Pico Technology products or with data collected using Pico Technology products.

Copyright. Pico Technology Ltd. claims the copyright of, and retains the rights to, all material (software, documents, etc.) contained in this SDK except the example programs. You may copy and distribute the SDK without restriction, as long as you do not remove any Pico Technology copyright statements. The example programs in the SDK may be modified, copied and distributed for the purpose of developing programs to collect data using Pico products.

Liability. Pico Technology and its agents shall not be liable for any loss, damage or injury, howsoever caused, related to the use of Pico Technology equipment or software, unless excluded by statute.

Fitness for purpose. As no two applications are the same, Pico Technology cannot guarantee that its equipment or software is suitable for a given application. It is your responsibility, therefore, to ensure that the product is suitable for your application.

Mission-critical applications. This software is intended for use on a computer that may be running other software products. For this reason, one of the conditions of the license is that it excludes use in mission-critical applications, for example life support systems.

Viruses. This software was continuously monitored for viruses during production, but you are responsible for virus-checking the software once it is installed.

Support. If you are dissatisfied with the performance of this software, please contact our technical support staff, who will try to fix the problem within a reasonable time. If you are still dissatisfied, please return the product and software to your supplier within 14 days of purchase for a full refund.

Upgrades. We provide upgrades, free of charge, from our web site at www.picotech.com. We reserve the right to charge for updates or replacements sent out on physical media.

1.4 Trademarks

Pico Technology Limited and PicoScope are trademarks of Pico Technology Limited, registered in the United Kingdom and other countries.

PicoScope and Pico Technology are registered in the U.S. Patent and Trademark Office.

Windows is a registered trademark of Microsoft Corporation in the USA and other countries.

1.5 Warranty

Pico Technology warrants upon delivery, and for a period of 5 years unless otherwise stated from the date of delivery, that the Goods will be free from defects in material and workmanship.

Pico Technology shall not be liable for a breach of the warranty if the defect has been caused by fair wear and tear, willful damage, negligence, abnormal working conditions or failure to follow Pico Technology's spoken or written advice on the storage, installation, commissioning, use or maintenance of the Goods or (if no advice has been given) good trade practice; or if the Customer alters or repairs such Goods without the written consent of Pico Technology.

2 Programming the 2000 Series Oscilloscopes

2.1 General procedure

The ps2000.dll library in your PicoScope installation directory allows you to program a PicoScope 2000 Series oscilloscope using standard C function calls.

A typical program for capturing data consists of the following steps:

- Open the oscilloscope.
- Set up the input channels with the required voltage ranges and coupling mode.
- Set up triggering.
- Start capturing data. (See Sampling modes, where programming is discussed in more detail.)
- Wait until the oscilloscope is ready.
- Copy data to a buffer.
- Stop capturing data.
- Close the oscilloscope.

Numerous sample programs are included in the SDK. These show how to use the functions of the driver software in each of the modes available.

2.2 Driver

Your application will communicate with a PicoScope 2000 API driver called ps2000.dll, which is supplied in 32-bit and 64-bit versions. The driver exports the ps2000 function definitions in standard C format, but this does not limit you to programming in C. You can use the API with any programming language that supports standard C calls.

The API driver depends on a low-level driver called WinUsb.sys (supplied in 32-bit and 64-bit versions), which is installed by the SDK and configured when you plug the oscilloscope into each USB port for the first time. Your application does not call this driver directly.

2.3 Voltage ranges

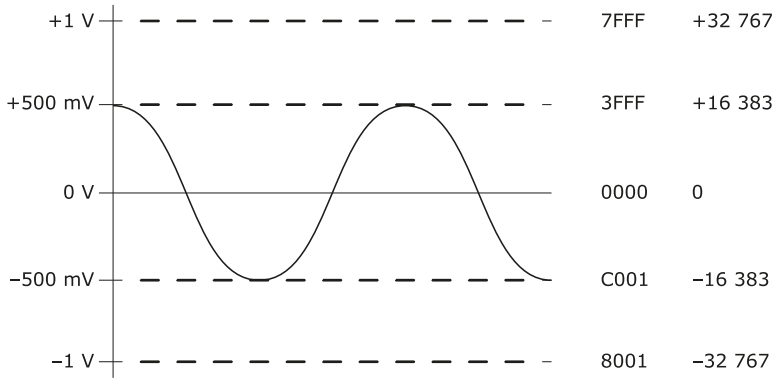
You can set the gain for each channel with the ps2000_set_channel function. The input voltage ranges available depend on the oscilloscope model.

The driver scales all readings to 16 bits, regardless of the voltage range the oscilloscope is set to. The following table shows the relationship between the reading from the driver and the signal level.

Constant	Reading		Voltage
	decimal	hex	
PS2000_LOST_DATA	-32 768	8000	Indicates a buffer overrun in fast streaming mode.
PS2000_MIN_VALUE	-32 767	8001	Negative full scale
0	0	0000	Zero volts
PS2000_MAX_VALUE	32 767	7FFF	Positive full scale

Example

1. Call ps2000_set_channel with range set to PS2000_1V.
2. Apply a sine wave input of 500 mV amplitude to the oscilloscope.
3. Capture some data using the desired sampling mode.
4. The data will be encoded as shown opposite.



2.4 Triggering

PicoScope 2000 Series PC Oscilloscopes can either start collecting data immediately, or be programmed to wait for a trigger event to occur. In both cases you need to use the ps2000_set_trigger function or, for scopes that support advanced triggering, the ps2000SetAdvTriggerChannelConditions and related functions. A trigger event can occur on any of the conditions available in the simple and advanced triggering modes.

Applicability	Available in block mode and fast streaming mode only. Calls to the ps2000_set_trigger and ps2000SetAdvTriggerChannelConditions functions have no effect in compatible streaming mode.

The triggering methods available for your oscilloscope are listed in the data sheet. Where available, the pulse width, delay and drop-out triggering methods additionally require the use of the pulse width qualifier function, ps2000SetPulseWidthQualifier.

2.5 Signal generator

The PicoScope 2203, 2204(A) and 2205(A) PC Oscilloscopes have a built-in signal generator, which is set up using ps2000_set_sig_gen_built_in. You can also use this signal generator to output arbitrary waveforms, using ps2000_set_sig_gen_arbitrary.

Applicability	PicoScope 2203, 2204(A) and 2205(A) oscilloscopes only.

2.6 AC/DC coupling

Using the ps2000_set_channel function, each channel can be set to either AC or DC coupling. When AC coupling is used, any component of the signal below about 1 Hz is filtered out.

2.7 Oversampling

When the oscilloscope is operating at sampling rates less than the maximum, it is possible to oversample. Oversampling is taking more than one measurement during a time interval and returning an average. If the signal contains a small amount of noise, this technique can increase the effective vertical resolution of the oscilloscope by the amount given by the equation below:

Increase in resolution (bits) = (log oversample) / (log 4)

Applicability	Available in block mode only.

3 Sampling modes

PicoScope 2000 Series PC Oscilloscopes can run in various sampling modes.

- Block mode. At the highest sampling rates, the oscilloscope collects data much faster than a PC can read it. In this case, the oscilloscope stores a block of data in an internal memory buffer, delaying transfer to the PC until the required number of data points have been sampled.

- ETS mode. In this mode, it is possible to increase the effective sampling rate of the scope when capturing repetitive signals. It is a modified form of block mode.

- Streaming modes. At all but the highest sampling rates, these modes allow accurately timed data to be transferred back to the PC without gaps. The computer instructs the oscilloscope to start collecting data. The oscilloscope then transfers data back to the PC without storing it in its own memory, so the size of the data set is limited only by the size of the PC's memory. Sampling intervals from less than one microsecond (depending on model) to 60 seconds are possible. There are two streaming modes:

  - Compatible streaming mode
  - Fast streaming mode

3.1 Block mode

In block mode, the computer prompts the oscilloscope to collect a block of data in its internal memory. When the oscilloscope has collected the whole block, it signals that it is ready and then transfers the whole block to the computer's memory through the USB port.

The maximum number of values depends upon the size of the oscilloscope's memory. A PicoScope 2000 Series oscilloscope can sample at a number of different rates that correspond to the maximum sampling rate divided by 1, 2, 4, 8 and so on.

There is a separate memory buffer for each channel. When a channel is unused, its memory can be borrowed by the enabled channels. This feature is handled transparently by the driver.

The driver normally performs a number of setup operations before collecting each block of data. This can take up to 50 milliseconds. If it is necessary to collect data with the minimum time interval between blocks, avoid calling setup functions between calls to `ps2000_run_block`, `ps2000_ready`, `ps2000_stop` and `ps2000_get_values`.

See Using block mode for programming details.

3.1.1 Using block mode

This is the general procedure for reading and displaying data in block mode:

1. Open the oscilloscope using `ps2000_open_unit`.
2. Select channel ranges and AC/DC coupling using `ps2000_set_channel`.
3. Using `ps2000_set_trigger`, set the trigger if required.
4. Using `ps2000_get_timebase`, select timebases until you locate the required time interval per sample.
5. Start the oscilloscope running using `ps2000_run_block`.
6. Poll the driver to find out if the oscilloscope has finished collecting data, using `ps2000_ready`.
7. Transfer the block of data from the oscilloscope using `ps2000_get_values` or `ps2000_get_times_and_values`.

8. Display the data.
9. Repeat steps 5 to 8.
10. Stop the oscilloscope using ps2000_stop.
11. Close the device using ps2000_close_unit.

Note that if you call ps2000_get_values, ps2000_get_times_and_values or ps2000_stop before the oscilloscope is ready, no capture will be available and the driver will not return any samples.

3.2 Streaming mode

Streaming mode is an alternative to block mode that can capture data without gaps between blocks.

In streaming mode, the computer prompts the oscilloscope to start collecting data. The data is then transferred back to the PC without being stored in the oscilloscope's memory. Data can be sampled with a period between 1 µs or less and 60 s, and the maximum number of samples is limited only by the amount of free space on the PC's hard disk.

There are two varieties of streaming mode:

- Compatible streaming mode
- Fast streaming mode

3.2.1 Compatible streaming mode

Compatible streaming mode is a basic streaming mode that works at speeds from one sample per minute to a thousand samples per second.

The oscilloscope's driver transfers data to a computer program using either normal or windowed mode. In normal mode, any data collected since the last data transfer operation is returned in its entirety. Normal mode is useful if the computer program requires fresh data on every transfer. In windowed mode, a fixed number of samples is returned, where the oldest samples may have already been returned before. Windowed mode is useful when the program requires a constant time period of data.

Once the oscilloscope is collecting data in compatible streaming mode, any setup changes (for example, changing a channel range or AC/DC setting) will cause a restart of the data stream. The driver can buffer up to 32 kilosamples of data per channel, but the user must ensure that the ps2000_get_values function is called frequently enough to avoid buffer overrun.

For streaming with the PicoScope 2202, 2203, 2204, 2204A, 2205 and 2205A variants, we recommend you use fast streaming mode instead.

Applicability	Only recommended for use with PicoScope 2104 and 2105. Does not support triggering. The ps2000_get_times_and_values function always returns FALSE (0) in streaming mode.

See Using compatible streaming mode for programming details.

3.2.1.1 Using compatible streaming mode

This is the general procedure for reading and displaying data in compatible streaming mode:

1. Open the oscilloscope using ps2000_open_unit.
2. Select channel ranges and AC/DC coupling using ps2000_set_channel.
3. Start the oscilloscope running using ps2000_run_streaming.
4. Transfer the block of data from the oscilloscope using ps2000_get_values.
5. Display the data.
6. Repeat steps 4 and 5 as necessary.
7. Stop the oscilloscope using ps2000_stop.
8. Close the device using ps2000_close_unit.

3.2.2 Fast streaming mode

Fast streaming mode is an advanced streaming mode that can transfer data at speeds of a million samples per second or more, depending on the computer's performance. This makes it suitable for high-speed data acquisition, allowing you to capture very long data sets limited only by the computer's memory.

Fast streaming mode also provides data aggregation, which allows your application to zoom in and out of the data with the minimum of effort.

Applicability	PicoScope 2202, 2203, 2204, 2204A, 2205 and 2205A only. Works with triggering.

See Using fast streaming mode for programming details.

3.2.2.1 Using fast streaming mode

This is the general procedure for reading and displaying data in fast streaming mode:

1. Open the oscilloscope using ps2000_open_unit.
2. Select channel ranges and AC/DC coupling using ps2000_set_channel.
3. Set the trigger using ps2000_set_trigger.
4. Start the oscilloscope running using ps2000_run_streaming_ns.
5. Get a block of data from the oscilloscope using ps2000_get_streaming_last_values.
6. Display or process the data.
7. If required, check for overview buffer overruns by calling ps2000_overview_buffer_status.
8. Repeat steps 5 to 7 as necessary or until auto_stop is TRUE.
9. Stop fast streaming using ps2000_stop.
10. Retrieve any part of the data at any time scale by calling ps2000_get_streaming_values.
11. If you require raw data, retrieve it by calling ps2000_get_streaming_values_no_aggregation.
12. Repeat steps 10 to 11 as necessary.
13. Close the oscilloscope by calling ps2000_close_unit.

3.3 ETS (Equivalent Time Sampling) mode

ETS is a way of increasing the effective sampling rate when working with repetitive signals. It is controlled by the ps2000_set_trigger and ps2000_set_ets functions.

ETS works by capturing many instances of a repetitive waveform, then combining them to produce a composite waveform that has a higher effective sampling rate than the individual instances. The maximum effective sampling rates that can be achieved with this method are listed in the data sheet specifications for your oscilloscope.

Because of the high sensitivity of ETS mode to small time differences, you must set up the trigger to provide a stable waveform that varies as little as possible from one capture to the next.

Applicability	Block mode only. PicoScope 2104, 2105, 2203, 2204, 2204A, 2205 and 2205A oscilloscopes. As ETS returns random time intervals, the ps2000_get_times_and_values function must be used. The ps2000_get_values function will return FALSE (0). Stable, repetitive signals only.

3.3.1 Using ETS mode

This is the general procedure for reading and displaying data in ETS mode:

1. Open the oscilloscope using ps2000_open_unit.
2. Select channel ranges and AC/DC coupling using ps2000_set_channel.
3. Using ps2000_set_trigger, set the trigger if required.
4. Set ETS mode using ps2000_set_ets.
5. Start the oscilloscope running using ps2000_run_block.
6. Poll the driver to find out when the oscilloscope has finished collecting data, using ps2000_ready.
7. Transfer the block of data from the oscilloscope using ps2000_get_times_and_values.
8. Display the data.
9. Repeat steps 6 to 8 as necessary.
10. Stop the oscilloscope using ps2000_stop.
11. Close the device using ps2000_close_unit.

Note that if you call ps2000_get_values, ps2000_get_times_and_values or ps2000_stop before the oscilloscope is ready, no capture will be available and the driver will not return any samples.

4 Combining several oscilloscopes

The 2000 Series driver can collect data from up to 64 PicoScope 2000 Series PC oscilloscopes at the same time. Each oscilloscope must be connected to a separate USB port. If you use a USB hub, make sure it is a powered hub.

To begin, call ps2000_open_unit to obtain a handle for each oscilloscope. All the other functions require this handle for oscilloscope identification. For example, to collect data from two oscilloscopes at the same time:

```
handle1 = ps2000_open_unit()
handle2 = ps2000_open_unit()

ps2000_set_channel(handle1)
... set up unit 1
ps2000_run_block(handle1)

ps2000_set_channel(handle2)
... set up unit 2
ps2000_run_block(handle2)

ready = FALSE
while not ready
    ready = ps2000_ready(handle1)
    ready &= ps2000_ready(handle2)

ps2000_get_values(handle1)
ps2000_get_values(handle2)

ps2000_close_unit(handle1)
ps2000_close_unit(handle2)
```

It is not possible to synchronize the collection of data between oscilloscopes that are being used in combination.

5 API Functions

The PicoScope 2000 Series API exports the following functions for you to use in your own applications:

The following user-defined function is also described here:

5.1 ps2000_close_unit

```
int16_t ps2000_close_unit
(
  int16_t    handle
)
```

Shuts down a PicoScope 2000 Series oscilloscope.

Applicability	All modes
Arguments	handle: the handle, returned by ps2000_open_unit, of the oscilloscope being closed
Returns	non-zero: if a valid handle is passed 0: if handle is not valid

5.2 ps2000_flash_led

```
int16_t ps2000_flash_led
(
  int16_t    handle
)
```

Flashes the LED on the front of the oscilloscope (or in the pushbutton, for the PicoScope 2104 and 2105 oscilloscopes) three times and returns within one second.

Applicability	All modes
Arguments	handle: the handle of the PicoScope 2000 Series oscilloscope
Returns	non-zero: if a valid handle is passed 0: if handle is invalid

5.2 ps2000_flash_led

```
int16_t ps2000_flash_led
(
  int16_t    handle
)
```

Flashes the LED on the front of the oscilloscope (or in the pushbutton, for the PicoScope 2104 and 2105 oscilloscopes) three times and returns within one second.

Applicability	All modes
Arguments	`handle`: the handle of the PicoScope 2000 Series oscilloscope
Returns	non-zero: if a valid handle is passed 0: if handle is invalid

5.3 ps2000_get_streaming_last_values

```
int16_t ps2000_get_streaming_last_values
(
  int16_t                    handle
  GetOverviewBuffersMaxMin   lpGetOverviewBuffersMaxMin
)
```

This function is used to collect the next block of values while fast streaming is running. You must call `ps2000_run_streaming_ns` beforehand to set up fast streaming.

Applicability	Fast streaming mode only PicoScope 2202, 2203, 2204, 2204A, 2205 and 2205A only Not compatible with ETS triggering. Function has no effect in ETS mode.
Arguments	`handle`: the handle of the required oscilloscope `lpGetOverviewBuffersMaxMin`: a pointer to the `my_get_overview_buffers` callback function in your application that receives data from the streaming driver
Returns	1: if the callback will be called 0: if the callback will not be called, either because one of the inputs is out of range or because there are no samples available

5.4 ps2000_get_streaming_values

```
uint32_t ps2000_get_streaming_values
(
  int16_t     handle,
  double      *start_time,
  int16_t     *pbuffer_a_max,
  int16_t     *pbuffer_a_min,
  int16_t     *pbuffer_b_max,
  int16_t     *pbuffer_b_min,
  int16_t     *pbuffer_c_max,
  int16_t     *pbuffer_c_min,
  int16_t     *pbuffer_d_max,
  int16_t     *pbuffer_d_min,
  int16_t     *overflow,
  uint32_t    *triggerAt,
  int16_t     *triggered,
  uint32_t    no_of_values,
  uint32_t    noOfSamplesPerAggregate
)
```

This function is used after the driver has finished collecting data in fast streaming mode. It allows you to retrieve data with different aggregation ratios, and thus zoom in to and out of any region of the data.

Before calling this function, first capture some data in fast streaming mode, stop fast streaming by calling ps2000_stop, then allocate sufficient buffer space to receive the requested data. The function will store the data in your buffer with values in the range PS2000_MIN_VALUE to PS2000_MAX_VALUE. The special value PS2000_LOST_DATA is stored in the buffer when data could not be collected because of a buffer overrun. (See Voltage ranges for more on data values.)

Each sample of aggregated data is created by processing a block of raw samples. The aggregated sample is stored as a pair of values: the minimum and the maximum values of the block.

Applicability	Fast streaming mode only PicoScope 2202, 2203, 2204, 2204A, 2205 and 2205A only Not compatible with ETS triggering - function has no effect in ETS mode
Arguments	handle: the handle of the required oscilloscope start_time: the time in nanoseconds, relative to the trigger point, of the first data sample required pbuffer_a_max, pbuffer_a_min: pointers to two buffers into which the function will write the maximum and minimum aggregated sample values from channel A pbuffer_b_max, pbuffer_b_min: as above but for channel B (two-channel scopes only) pbuffer_c_max, pbuffer_c_min, pbuffer_d_max, pbuffer_d_min: not used overflow: on exit, the function writes a bit field here indicating whether the voltage on each of the input channels has overflowed: Bit 0: Ch A overflow Bit 1: Ch B overflow triggerAt: on exit, the function writes an index value here. This is the offset, from the start of the buffer, of the sample at the trigger reference point. It is valid only when triggered is TRUE. triggered: a pointer to a Boolean indicating that a trigger has occurred and triggerAt is valid no_of_values: the number of values required noOfSamplesPerAggregate: the number of samples that the driver should combine to form each aggregated value pair. The pair consists of the maximum and minimum values of all the samples that were aggregated. For channel A, the minimum value is stored in the buffer pointed to by pbuffer_a_min and the maximum value in the buffer pointed to by pbuffer_a_max.
Returns	The number of values written to each buffer, if successful 0: if a parameter was out of range

5.5 ps2000_get_streaming_values_no_aggregation

```
uint32_t ps2000_get_streaming_values_no_aggregation
(
  int16_t     handle,
  double      *start_time,
  int16_t     *pbuffer_a,
  int16_t     *pbuffer_b,
  int16_t     *pbuffer_c,
  int16_t     *pbuffer_d,
  int16_t     *overflow,
  uint32_t    *triggerAt,
  int16_t     *trigger,
  uint32_t    no_of_values
)
```

This function retrieves raw streaming data from the driver's data store after fast streaming has stopped.

Before calling the function, capture some data using fast streaming, stop streaming using ps2000_stop, and then allocate sufficient buffer space to receive the requested data. The function will store the data in your buffer with values in the range PS2000_MIN_VALUE to PS2000_MAX_VALUE. The special value PS2000_LOST_DATA is stored in the buffer when data could not be collected because of a buffer overrun. (See Voltage ranges for more details of data values.)

Applicability	Fast streaming mode only PicoScope 2203, 2204, 2204A, 2205 and 2205A only Not compatible with ETS triggering - has no effect in ETS mode
Arguments	`handle`: the handle of the required oscilloscope `start_time`: the time in nanoseconds of the first data sample required `pbuffer_a, pbuffer_b`: pointers to buffers into which the function will write the raw sample values from channels A (all scopes) and B (two-channel scopes only) `pbuffer_c, pbuffer_d`: not used `overflow`: on exit, the function will write a bit field here indicating whether the voltage on each of the input channels has overflowed. Bit 0 is the LSB. The bit assignments are as follows: Bit 0 - Ch A overflow Bit 1 - Ch B overflow `triggerAt`: on exit, the function writes an index into the buffers here. The index is the number of the sample at the trigger reference point. It is valid only when trigger is TRUE. `trigger`: on exit, the function writes a Boolean here indicating that a trigger has occurred and `triggerAt` is valid `no_of_values`: the number of values required
Returns	The number of values written to each buffer, if successful 0: if a parameter was out of range

5.6 ps2000_get_timebase

```
int16_t ps2000_get_timebase
(
  int16_t    handle,
  int16_t    timebase,
  int32_t    no_of_samples,
  int32_t    *time_interval,
  int16_t    *time_units,
  int16_t    oversample,
  int32_t    *max_samples
)
```

This function discovers which timebases are available on the oscilloscope. You should set up the channels using `ps2000_set_channel` and, if required, ETS mode using `ps2000_set_ets` first. Then call this function with increasing values of `timebase`, starting from 0, until you find a timebase with a sampling interval and sample count close enough to your requirements.

Applicability	All modes
Arguments	`handle`: the handle of the required oscilloscope `timebase`: a code between 0 and the maximum timebase (depending on the oscilloscope). Timebase 0 is the fastest timebase. Each successive timebase has twice the sampling interval of the previous one. `no_of_samples`: the number of samples that you require. The function uses this value to calculate the most suitable time unit to use. `time_interval`: on exit, this location will contain the time interval, in nanoseconds, between readings at the selected timebase. If `time_interval` is `NULL`, the function will write nothing. `time_units`: on exit, this location will contain an enumerated type indicating the most suitable unit for expressing sample times. You should pass this value to `ps2000_get_times_and_values`. If `time_units` is null, the function will write nothing. `oversample`: the amount of oversample required. For example, an oversample of 4 results in a `time_interval` 4 times larger and a `max_samples` 4 times smaller. At the same time it increases the effective resolution by one bit. See Oversampling for more details. `max_samples`: on exit, the maximum number of samples available. The scope allocates a certain amount of memory for internal overheads and this may vary depending on the number of channels enabled, the timebase chosen and the oversample multiplier selected. If `max_samples` is `NULL`, the function will write nothing.
Returns	non-zero: if all parameters are in range 0: on error

5.7 ps2000_get_times_and_values

```
int32_t ps2000_get_times_and_values
(
  int16_t    handle
  int32_t    *times,
  int16_t    *buffer_a,
  int16_t    *buffer_b,
  int16_t    *buffer_c,
  int16_t    *buffer_d,
  int16_t    *overflow,
  int16_t    time_units,
  int32_t    no_of_values
)
```

This function is used to get values and times in block mode after calling ps2000_run_block.

Note that if you are using block mode or ETS mode and call this function before the oscilloscope is ready, no capture will be available and the driver will not return any samples.

Applicability	Block mode only. It will not return any valid times if the oscilloscope is in streaming mode. Essential for ETS operation
Arguments	handle: the handle of the required oscilloscope times: a pointer to a buffer for the sample times in time_units. Each time is the interval between the trigger event and the corresponding sample. Times before the trigger event are negative, and times after the trigger event are positive. buffer_a, buffer_b: pointers to buffers that receive data from the channels A and B. A pointer will not be used if the oscilloscope is not collecting data from that channel. If a pointer is NULL, nothing will be written to it. buffer_c, buffer_d: not used overflow: a bit pattern indicating whether an overflow has occurred and, if so, on which channel. Bit 0 is the LSB. The bit assignments are as follows: Bit 0 - Ch A overflow Bit 1 - Ch B overflow time_units: can be one of the following: PS2000_FS (0), femtoseconds, PS2000_PS (1), picoseconds, PS2000_NS (2), nanoseconds [default] PS2000_US (3), microseconds, PS2000_MS (4), milliseconds, PS2000_S (5), seconds no_of_values: the number of data points to return. In streaming mode, this is the maximum number of values to return.
Returns	The actual number of data values per channel returned, which may be less than no_of_values if streaming 0: if one or more of the parameters are out of range, if the times will overflow with the time_units requested (use ps2000_get_timebase to acquire the most suitable time_units) or if the oscilloscope is in streaming mode

5.8 ps2000_get_unit_info

```
int16_t ps2000_get_unit_info
(
  int16_t    handle,
  int8_t     *string,
  int16_t    string_length,
  int16_t    line
)
```

This function writes oscilloscope information to a character string. If the oscilloscope failed to open, only `line` types 0 and 6 are available to explain why the last open unit call failed.

Applicability	All modes
Arguments	`handle`: the handle of the oscilloscope from which information is required. If an invalid handle is passed, the error code from the last oscilloscope that failed to open is returned. `string`: a pointer to the character string buffer in the calling function where the function will write the oscilloscope information string selected with `line`. If string is `NULL`, no information will be written. `string_length`: the length of the character string buffer. If the string is not long enough to accept all of the information, only the first `string_length` characters are returned. `line`: a value selected from enumerated type `PS2000_INFO` (see table below) specifying what information is required from the driver
Returns	The length of the string written to the `string` buffer 0: if one of the parameters is out of range or string is NULL

`PS2000_INFO` value	Example
`PS2000_DRIVER_VERSION (0)`, the version number of the DLL used by the oscilloscope driver.	`"1, 0, 0, 2"`
`PS2000_USB_VERSION (1)`, the type of USB connection that is being used to connect the oscilloscope to the computer.	`"1.1" or "2.0"`
`PS2000_HARDWARE_VERSION (2)`, the hardware version of the attached oscilloscope.	`"1"`
`PS2000_VARIANT_INFO (3)`, the variant of PicoScope 2000 PC Oscilloscope that is attached to the computer.	`"2203"`
`PS2000_BATCH_AND_SERIAL (4)`, the batch and serial number of the oscilloscope.	`"CMY66/052"`
`PS2000_CAL_DATE (5)`, the calibration date of the oscilloscope.	`"14Jan08"`
`PS2000_ERROR_CODE (6)`, one of the Error codes.	`"4"`
`PS2000_KERNEL_DRIVER_VERSION (7)`, the version number of the kernel driver.	`"1,1,2,4"`

5.9 ps2000_get_values

```
int32_t ps2000_get_values
(
  int16_t    handle
  int16_t    *buffer_a,
  int16_t    *buffer_b,
  int16_t    *buffer_c,
  int16_t    *buffer_d,
  int16_t    *overflow,
  int32_t    no_of_values
)
```

This function is used to get values in compatible streaming mode after calling ps2000_run_streaming, or in block mode after calling ps2000_run_block.

Note that if you are using block mode or ETS mode and call this function before the oscilloscope is ready, no capture will be available and the driver will not return any samples.

Applicability	Compatible streaming mode and block mode only Does nothing if ETS triggering is enabled. Use ps2000_get_times_and_values instead. Do not use in fast streaming mode. Use ps2000_get_streaming_last_values instead.
Arguments	handle: the handle of the required oscilloscope buffer_a, buffer_b: pointers to the buffers that receive data from the specified channels (A and B). A pointer is not used if the oscilloscope is not collecting data from that channel. If a pointer is NULL, nothing will be written to it. buffer_c, buffer_d: not used overflow: on exit, contains a bit pattern indicating whether an overflow has occurred and, if so, on which channel. Bit 0 is the least significant bit. The bit assignments are as follows: Bit 0 - Ch A overflow Bit 1 - Ch B overflow no_of_values: the number of data points to return. In streaming mode, this is the maximum number of values to return.
Returns	The actual number of data values per channel returned, which may be less than no_of_values if streaming 0: if one of the parameters is out of range or the oscilloscope is not in a suitable mode

5.10 ps2000_last_button_press

```
int16_t ps2000_last_button_press
(
    int16_t    handle
)
```

This function returns the last registered state of the pushbutton on the PicoScope 2104 or 2105 PC Oscilloscope and then resets the status to zero.

Applicability	PicoScope 2104 and 2105 only
Arguments	`handle`: handle of the oscilloscope
Returns	0: no button press registered 1: short button press registered 2: long button press registered

5.11 ps2000_open_unit

```
int16_t ps2000_open_unit
(
  void
)
```

This function opens a PicoScope 2000 Series oscilloscope. The driver can support up to 64 oscilloscopes.

Applicability	All modes
Arguments	None
Returns	-1: if the oscilloscope fails to open 0: if no oscilloscope is found >0 (oscilloscope handle): if the oscilloscope opened. Use this as the handle argument for all subsequent API calls for this oscilloscope.

5.12 ps2000_open_unit_async

```
int16_t ps2000_open_unit_async
(
  void
)
```

This function opens a PicoScope 2000 Series oscilloscope without waiting for the operation to finish. You can find out when it has finished by periodically calling `ps2000_open_unit_progress` until that function returns a non-zero value and a valid oscilloscope handle.

The driver can support up to 64 oscilloscopes.

Applicability	All modes
Arguments	None
Returns	0: if there is a previous open operation in progress non-zero: if the call has successfully initiated an open operation

5.13 ps2000_open_unit_progress

```
int16_t ps2000_open_unit_progress
(
  int16_t    *handle,
  int16_t    *progress_percent
)
```

This function checks on the progress of ps2000_open_unit_async.

Applicability	All modes Use only with ps2000_open_unit_async
Arguments	handle: a pointer to where the function should store the handle of the opened oscilloscope 0 if no oscilloscope is found or the oscilloscope fails to open, handle of oscilloscope (valid only if function returns 1) progress_percent: a pointer to an estimate of the progress towards opening the oscilloscope. The function will write a value from 0 to 100, where 100 implies that the operation is complete.
Returns	>0: if the driver successfully opens the oscilloscope 0: if opening still in progress -1: if the oscilloscope failed to open or was not found

5.14 ps2000_overview_buffer_status

```
int16_t ps2000_overview_buffer_status
(
  int16_t    handle,
  int16_t    *previous_buffer_overrun
)
```

This function indicates whether or not the overview buffers used by ps2000_run_streaming_ns have overrun. If an overrun occurs, you can choose to increase the overview_buffer_size argument that you pass in the next call to ps2000_run_streaming_ns.

Applicability	Fast streaming mode only PicoScope 2202, 2203, 2204, 2204A, 2205 and 2205A only Not compatible with ETS triggering - function has no effect in ETS mode
Arguments	handle: the handle of the required oscilloscope previous_buffer_overrun: a pointer to a Boolean indicating whether the overview buffers have overrun. The function will write a non-zero value to indicate a buffer overrun.
Returns	0: if the function was successful 1: if the function failed due to an invalid handle

5.15 ps2000PingUnit

```
int16_t ps2000PingUnit
(
  int16_t    handle
)
```

This function can be used to check that the already opened device is still connected to the USB port and communication is successful.

Applicability	All modes
Arguments	handle: the handle of the required device
Returns	0 if function fails: call ps2000_get_unit_info for further information Any non-zero value: communication successful

5.16 ps2000_ready

```
int16_t ps2000_ready
(
  int16_t    handle
)
```

This function polls the driver to see if the oscilloscope has finished the last data collection operation.

Applicability	Block mode only. Does nothing if the oscilloscope is in streaming mode.
Arguments	handle: the handle of the required oscilloscope
Returns	>0: if ready. The oscilloscope has collected a complete block of data or the auto trigger timeout has been reached. 0: if not ready. An invalid handle was passed, or the oscilloscope is in streaming mode, or the oscilloscope is still collecting data in block mode. -1: if the oscilloscope is not attached. The USB transfer failed, indicating that the oscilloscope may have been unplugged.

5.17 ps2000_run_block

```
int16_t ps2000_run_block
(
  int16_t    handle,
  int32_t    no_of_samples,
  int16_t    timebase,
  int16_t    oversample,
  int32_t    *time_indisposed_ms
)
```

This function tells the oscilloscope to start collecting data in block mode.

Applicability	Block mode only.
Arguments	`handle`: the oscilloscope of the required oscilloscope `no_of_samples`: the number of samples to return `timebase`: a code between 0 and the maximum timebase available (consult the driver header file). Timebase 0 gives the maximum sample rate available, timebase 1 selects a sample rate half as fast, timebase 2 is half as fast again and so on. For the maximum sample rate, see the specifications for your oscilloscope. The number of channels enabled may affect the availability of the fastest timebases. `oversample`: the oversampling factor, a number between 1 and 256. See Oversampling for details. `time_indisposed_ms`: a pointer to the approximate time, in milliseconds, that the ADC will take to collect data. If a trigger is set, it is the amount of time the ADC takes to collect a block of data after a trigger event, calculated as (sample interval) x (number of points required). The actual time may differ from computer to computer, depending on how quickly the computer can respond to I/O requests.
Returns	0: if one of the parameters is out of range non-zero: if successful

5.18 ps2000_run_streaming

```
int16_t ps2000_run_streaming
(
  int16_t    handle,
  int16_t    sample_interval_ms,
  int32_t    max_samples,
  int16_t    windowed
)
```

This function tells the oscilloscope to start collecting data in compatible streaming mode. If this function is called when a trigger has been enabled, the trigger settings will be ignored.

For streaming with the PicoScope 2202, 2203, 2204, 2204A, 2205 and 2205A variants, we recommend you use ps2000_run_streaming_ns instead: this will allow much faster data transfer.

Applicability	Only recommended for use with PicoScope 2104 and 2105
Arguments	handle: the handle of the required oscilloscope sample_interval_ms: the time interval, in milliseconds, between data points. This can be no shorter than 1 ms. max_samples: the maximum number of samples that the driver is to store. This can be no greater than 60 000. It is the application's responsibility to retrieve data before the oldest values are overwritten. windowed: if this is 0, only the values taken since the last call to ps2000_get_values are returned. If this is 1, the number of values requested by ps2000_get_values are returned, even if they have already been read by ps2000_get_values.
Returns	non-zero: if streaming has been enabled correctly 0: if a problem occurred or a value was out of range

5.19 ps2000_run_streaming_ns

```
int16_t ps2000_run_streaming_ns
(
  int16_t              handle,
  uint32_t             sample_interval,
  PS2000_TIME_UNITS    time_units,
  uint32_t             max_samples,
  int16_t              auto_stop,
  uint32_t             noOfSamplesPerAggregate,
  uint32_t             overview_buffer_size
)
```

This function tells the oscilloscope to start collecting data in fast streaming mode. It returns immediately without waiting for data to be captured. After calling it, you should next call ps2000_get_streaming_last_values to copy the data to your application's buffer.

Applicability	PicoScope 2202, 2203, 2204, 2204A, 2205 and 2205A only
Arguments	handle: the handle of the required oscilloscope sample_interval: the time interval, in time_units, between data points time_units: the units in which sample_interval is measured max_samples: the maximum number of samples that the driver should store from each channel. Your computer must have enough physical memory for this many samples, multiplied by the number of channels in use, multiplied by the number of bytes per sample. auto_stop: a Boolean to indicate whether streaming should stop automatically when max_samples is reached. Set to any non-zero value for TRUE. noOfSamplesPerAggregate: the number of incoming samples that the driver will merge together (or aggregate: see aggregation) to create each value pair passed to the application. The value must be between 1 and max_samples. overview_buffer_size: the size of the overview buffers, temporary buffers used by the driver to store data before passing it to your application. You can check for overview buffer overruns using the ps2000_overview_buffer_status function and adjust the overview buffer size if necessary. The maximum allowable value is 1,000,000. We recommend using an initial value of 15,000 samples.
Returns	non-zero: if streaming has been enabled correctly 0: if a problem occurred or a value was out of range

5.20 ps2000SetAdvTriggerChannelConditions

```
int16_t ps2000SetAdvTriggerChannelConditions
(
  int16_t                       handle,
  PS2000_TRIGGER_CONDITIONS     *conditions,
  int16_t                       nConditions
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is defined by a PS2000_TRIGGER_CONDITIONS structure.

Applicability	Available in block mode and fast streaming mode only PicoScope 2202, 2204, 2204A, 2205 and 2205A only
Arguments	handle: the handle of the required oscilloscope conditions: a pointer to a PS2000_TRIGGER_CONDITIONS structure specifying the conditions that should be applied to the current trigger channel. If NULL, triggering is switched off. nConditions: should be set to 1 if conditions is non-null, otherwise 0
Returns	0: if unsuccessful, or if one or more of the arguments are out of range non-zero: if successful

5.20.1 PS2000_TRIGGER_CONDITIONS structure

A structure of this type is passed to ps2000SetAdvTriggerChannelConditions in the conditions argument to specify the trigger conditions, and is defined as follows:

```
typedef struct tPS2000TriggerConditions
{
  PS2000_TRIGGER_STATE channelA;
  PS2000_TRIGGER_STATE channelB;
  PS2000_TRIGGER_STATE channelC;
  PS2000_TRIGGER_STATE channelD;
  PS2000_TRIGGER_STATE external;
  PS2000_TRIGGER_STATE pulseWidthQualifier;
} PS2000_TRIGGER_CONDITIONS;
```

Applicability	See ps2000SetAdvTriggerChannelConditions
Members	channelA, channelB: the type of condition that should be applied to each channel. Use these constants: CONDITION_DONT_CARE (0) CONDITION_TRUE (1) CONDITION_FALSE (2) channelC, channelD: not used external: not used pulseWidthQualifier: the type of condition to apply to the pulse width qualifier. Choose from the same list of constants given under channelA, channelB.

Remarks

The channels that are set to CONDITION_TRUE or CONDITION_FALSE must all meet their conditions simultaneously to produce a trigger. Channels set to CONDITION_DONT_CARE are ignored.

The PicoScope 2202 oscilloscope can use only a single input channel (either channel A or channel B) for the trigger source. Therefore you may define CONDITION_TRUE or CONDITION_FALSE for only one of these channels at a time. You can, optionally, set up the pulse width qualifier in combination with one of the input channels.

The PicoScope 2204, 2204A, 2205 and 2205A models can all trigger from both channel A and channel B, and therefore all support logic triggering.

5.21 ps2000SetAdvTriggerChannelDirections

```
int16_t ps2000SetAdvTriggerChannelDirections
(
  int16_t handle,
  PS2000_THRESHOLD_DIRECTION channelA,
  PS2000_THRESHOLD_DIRECTION channelB,
  PS2000_THRESHOLD_DIRECTION channelC,
  PS2000_THRESHOLD_DIRECTION channelD,
  PS2000_THRESHOLD_DIRECTION ext
)
```

This function sets the direction of the trigger for each channel.

Applicability	Available in block mode and fast streaming mode only PicoScope 2202, 2204, 2204A, 2205 and 2205A only
Arguments	handle: the handle of the required oscilloscope channelA, channelB: specify the direction in which the signal must pass through the threshold to activate the trigger. The allowable values for a PS2000_THRESHOLD_DIRECTION variable are listed in the table below. channelC, channelD: not used ext: not used
Returns	0: if unsuccessful, or if one or more of the arguments are out of range non-zero: if successful

PS2000_THRESHOLD_DIRECTION constants

PS2000_ABOVE	for gated triggers: above a threshold
PS2000_BELOW	for gated triggers: below a threshold
PS2000_ADV_RISING	for threshold triggers: rising edge
PS2000_ADV_FALLING	for threshold triggers: falling edge
PS2000_RISING_OR_FALLING	for threshold triggers: either edge
PS2000_INSIDE	for window-qualified triggers: inside window
PS2000_OUTSIDE	for window-qualified triggers: outside window
PS2000_ENTER	for window triggers: entering the window
PS2000_EXIT	for window triggers: leaving the window
PS2000_ENTER_OR_EXIT	for window triggers: either entering or leaving the window
PS2000_ADV_NONE	no trigger

5.22 ps2000SetAdvTriggerChannelProperties

```
int16_t ps2000SetAdvTriggerChannelProperties
(
  int16_t                                handle,
  PS2000_TRIGGER_CHANNEL_PROPERTIES    *channelProperties,
  int16_t                                nChannelProperties,
  int32_t                                autoTriggerMilliseconds
)
```

This function is used to enable or disable triggering and set its parameters.

Applicability	Available in block mode and fast streaming mode only PicoScope 2202, 2204, 2204A, 2205 and 2205A only
Arguments	`handle`: the handle of the required oscilloscope `channelProperties`: a pointer to a `PS2000_TRIGGER_CHANNEL_PROPERTIES` structure describing the requested properties. If NULL, triggering is switched off. `nChannelProperties`: should be set to 1 if `channelProperties` is non-null, otherwise 0 `autoTriggerMilliseconds`: the time in milliseconds for which the oscilloscope will wait before collecting data if no trigger event occurs. If this is set to zero, the oscilloscope will wait indefinitely for a trigger.
Returns	0: if unsuccessful, or if one or more of the arguments are out of range non-zero: if successful

5.22.1 PS2000_TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to ps2000SetAdvTriggerChannelProperties in the channelProperties argument to specify the trigger mechanism, and is defined as follows:

```
typedef struct tPS2000TriggerChannelProperties
{
int16_t                 thresholdMajor;
int16_t                 thresholdMinor;
uint16_t                hysteresis;
int16_t                 channel;
PS2000_THRESHOLD_MODE   thresholdMode;
} PS2000_TRIGGER_CHANNEL_PROPERTIES
```

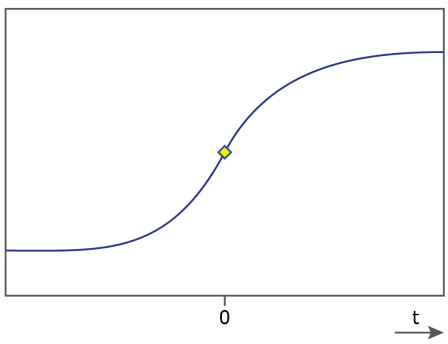
Applicability	See ps2000SetAdvTriggerChannelProperties
Members	thresholdMajor: the upper threshold at which the trigger event is to take place. This is scaled in 16-bit ADC counts at the currently selected range for that channel. thresholdMinor: the lower threshold at which the trigger event is to take place. This is scaled in 16-bit ADC counts at the currently selected range for that channel. hysteresis: the hysteresis that the trigger has to exceed before it will fire. It is scaled in 16-bit counts. channel: the channel to which the properties apply thresholdMode: either a level or window trigger. Use one of these constants: LEVEL (0) WINDOW (1)

5.23 ps2000SetAdvTriggerDelay

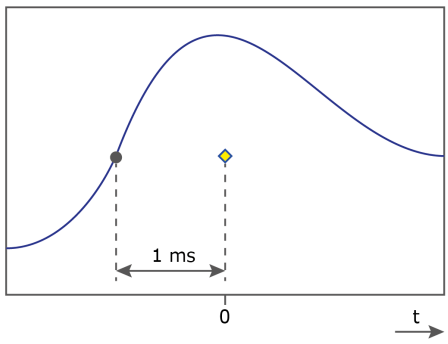
```
int16_t ps2000SetAdvTriggerDelay
(
  int16_t     handle,
  uint32_t    delay,
  float       preTriggerDelay
)
```

This function sets the pre-trigger and post-trigger delays. The default action, when both these delays are zero, is to start capturing data beginning with the trigger event and to stop a specified time later. The start of capture can be delayed by using a non-zero value of `delay`. Alternatively, the start of capture can be advanced to a time before the trigger event by using a negative value of `preTriggerDelay`. If both arguments are non-zero then their effects are added together.

Applicability	Block mode only PicoScope 2202, 2204, 2204A, 2205 and 2205A only
Arguments	`handle`: the handle of the required oscilloscope `delay`: the post-trigger delay, measured in sample periods. This is the time between the trigger event and the sample at time t = 0. For example, at a timebase of 50 MS/s, or 20 ns per sample, and with `delay = 100`, the post-trigger delay would be 100 × 20 ns = 2 µs. Range: [0, 2^{32}–1] `preTriggerDelay`: the location of the sample at time t = 0 within the data block, as a percentage of the data block length. 0% places t = 0 at the start of the block, –50% places it in the middle, and –100% places it at the end. Positive values can also be used to place t = 0 before the beginning of the data block, but the `delay` argument is more convenient for this purpose as it has a wider range. Range: [–100%, +100%]
Returns	0: if unsuccessful, or if one or more of the arguments are out of range non-zero: if successful



Example 1:
`delay` = 0,
`preTriggerDelay` = –50%



Example 2:
`delay` = 1 ms,
`preTriggerDelay` = –50%

5.24 ps2000_set_channel

```
int16_t ps2000_set_channel
(
  int16_t    handle,
  int16_t    channel,
  int16_t    enabled,
  int16_t    dc,
  int16_t    range
)
```

Specifies if a channel is to be enabled, the AC/DC coupling mode and the input range.

Note: The channels are not configured until capturing starts.

Applicability	All modes
Arguments	`handle`: the handle of the required oscilloscope `channel`: an enumerated type specifying the channel. Use `PS2000_CHANNEL_A (0)` or `PS2000_CHANNEL_B (1)`. `enabled`: specifies if the channel is active: `TRUE` = active `FALSE` = inactive `dc`: specifies the AC/DC coupling mode: `TRUE`: DC coupling `FALSE`: AC coupling `range`: a code between 1 and 10. See the table below, but note that each oscilloscope variant supports only a subset of these ranges.
Returns	0: if unsuccessful, or if one or more of the arguments are out of range non-zero: if successful

Code	Enumeration	Range
1	PS2000_20MV	±20 mV
2	PS2000_50MV	±50 mV
3	PS2000_100MV	±100 mV
4	PS2000_200MV	±200 mV
5	PS2000_500MV	±500 mV
6	PS2000_1V	±1 V
7	PS2000_2V	±2 V
8	PS2000_5V	±5 V
9	PS2000_10V	±10 V
10	PS2000_20V	±20 V

5.25 ps2000_set_ets

```
int32_t ps2000_set_ets
(
  int16_t    handle,
  int16_t    mode,
  int16_t    ets_cycles,
  int16_t    ets_interleave

)
```

This function is used to enable or disable ETS mode and to set the ETS parameters.

Applicability	Not PicoScope 2202
Arguments	`handle`: the handle of the required oscilloscope `mode`: `PS2000_ETS_OFF (0)` - disables ETS `PS2000_ETS_FAST (1)` - enables ETS and provides `ets_cycles` cycles of data, which may contain data from previously returned cycles `PS2000_ETS_SLOW (2)` - enables ETS and provides fresh data every `ets_cycles` cycles. `PS2000_ETS_SLOW` takes longer to provide each data set, but the data sets are more stable and unique `ets_cycles`: the number of cycles to store. The computer can then select `ets_interleave` cycles to give the most uniform spread of samples. `ets_cycles` should be between two and five times the value of `ets_interleave`. `ets_interleave`: the number of ETS interleaves to use. If the sample time is 20 ns and the interleave 10, the approximate time per sample will be 2 ns.
Returns	The effective sample time in picoseconds, if ETS is enabled 0: if ETS is disabled or one of the parameters is out of range

5.26 ps2000_set_light

```
int16_t ps2000_set_light
(
    int16_t    handle,
    int16_t    state
)
```

This function controls the white light that illuminates the probe tip on a handheld oscilloscope.

Applicability	PicoScope 2104 and 2105 handheld oscilloscopes only
Arguments	`handle`: handle of the oscilloscope `state`: `0`: light off 1: light on
Returns	0: the function failed, for example if a PicoScope 2000 Series oscilloscope was not found non-zero: success

5.27 ps2000_set_led

```
int16_t ps2000_set_led
(
    int16_t    handle,
    int16_t    state
)
```

This function turns the LED on the oscilloscope on and off, and controls its color.

Applicability	PicoScope 2104 and 2105 handheld oscilloscopes only
Arguments	`handle`: handle of the oscilloscope `state`: 3: off 1: red 2: green
Returns	0: the function failed, for example if a PicoScope 2000 Series oscilloscope was not found non-zero: success

5.28 ps2000SetPulseWidthQualifier

```
int16_t ps2000SetPulseWidthQualifier
(
  int16_t                       handle,
  PS2000_PWQ_CONDITIONS         *conditions,
  int16_t                       nConditions,
  PS2000_THRESHOLD_DIRECTION    direction,
  uint32_t                      lower,
  uint32_t                      upper,
  PS2000_PULSE_WIDTH_TYPE       type
)
```

This function sets up pulse width qualification, which can be used on its own for pulse width triggering or combined with other triggering to produce more complex triggers. The pulse width qualifier is set by defining a `conditions` structure.

Applicability	Available in block mode and fast streaming mode only PicoScope 2202, 2204, 2204A, 2205 and 2205A only
Arguments	`handle`: the handle of the required oscilloscope `conditions`: a pointer to a `PS2000_PWQ_CONDITIONS` structure specifying the conditions that should be applied to the trigger channel. If `conditions` is NULL then the pulse width qualifier is not used. `nConditions`: should be set to 1 if `conditions` is non-null, otherwise 0 `direction`: the direction of the signal required to trigger the pulse `lower`: the lower limit of the pulse width counter `upper`: the upper limit of the pulse width counter. This parameter is used only when the `type` is set to `PW_TYPE_IN_RANGE` or `PW_TYPE_OUT_OF_RANGE`. `type`: the pulse width type, one of these constants: `PW_TYPE_NONE` — do not use the pulse width qualifier `PW_TYPE_LESS_THAN` — pulse width less than lower `PW_TYPE_GREATER_THAN` — pulse width greater than lower `PW_TYPE_IN_RANGE` — pulse width between lower and upper `PW_TYPE_OUT_OF_RANGE` — pulse width not between lower and upper
Returns	0: if unsuccessful, or if one or more of the arguments are out of range non-zero: if successful

5.28.1 PS2000_PWQ_CONDITIONS structure

A structure of this type is passed to ps2000SetPulseWidthQualifier in the conditions argument to specify the pulse-width qualifier conditions, and is defined as follows:

```
typedef struct tPS2000PwqConditions
{
  PS2000_TRIGGER_STATE channelA;
  PS2000_TRIGGER_STATE channelB;
  PS2000_TRIGGER_STATE channelC;
  PS2000_TRIGGER_STATE channelD;
  PS2000_TRIGGER_STATE external;
} PS2000_PWQ_CONDITIONS
```

Applicability	Pulse-width-qualified triggering
Members	channelA, channelB: the type of condition that should be applied to each channel. Choose from these constants: CONDITION_DONT_CARE (0) CONDITION_TRUE (1) CONDITION_FALSE (2) channelC, channelD, external: not used

5.29 ps2000_set_sig_gen_arbitrary

```
int16_t ps2000_set_sig_gen_arbitrary
(
  int16_t              handle,
  int32_t              offsetVoltage,
  uint32_t             pkToPk,
  uint32_t             startDeltaPhase,
  uint32_t             stopDeltaPhase,
  uint32_t             deltaPhaseIncrement,
  uint32_t             dwellCount,
  uint8_t              *arbitraryWaveform,
  int32_t              arbitraryWaveformSize,
  PS2000_SWEEP_TYPE    sweepType,
  uint32_t             sweeps
)
```

This function programs the signal generator to produce an arbitrary waveform.

The arbitrary waveform generator uses direct digital synthesis (DDS). It maintains a 32-bit phase accumulator that indicates the present location in the waveform. The top bits of the phase accumulator are used as an index into a buffer containing the arbitrary waveform. The remaining bits act as the fractional part of the index, enabling high-resolution control of output frequency and allowing the generation of lower frequencies.

The generator steps through the waveform by adding a *deltaPhase* value between 1 and *phaseAccumulatorSize-1* to the phase accumulator every *ddsPeriod* (*1 / ddsFrequency*). If the *deltaPhase* is constant, the generator produces a waveform at a constant frequency that can be calculated as follows:

$$outputFrequency = ddsFrequency \times \left(\frac{deltaPhase}{phaseAccumulatorSize}\right) \times \left(\frac{awgBufferSize}{arbitraryWaveformSize}\right)$$

where:

outputFrequency = repetition rate of the complete arbitrary waveform
ddsFrequency = clock rate of phase accumulator (not the same as the DAC update rate)
deltaPhase = user-specified delta phase value
phaseAccumulatorSize = 2^{32} for all models
awgBufferSize = AWG buffer size
arbitraryWaveformSize = length in samples of the user-defined waveform

Parameter	Value
phaseAccumulatorSize	2^{32}
awgBufferSize	4096
ddsFrequency	48 MHz
ddsPeriod (= 1/*ddsFrequency*)	20.833 ns (= 1/48 MHz)

It is also possible to sweep the frequency by continually modifying the *deltaPhase*. This is done by setting up a `deltaPhaseIncrement` that the oscilloscope adds to the *deltaPhase* at intervals specified by `dwellCount`.

Applicability	All modes. PicoScope 2203, 2204, 2204A, 2205 and 2205A only
Arguments	

`handle`: the handle of the required oscilloscope
 	
`offsetVoltage`: the voltage offset, in microvolts, to be applied to the waveform
 	
`pkToPk`: the peak-to-peak voltage, in microvolts, of the waveform signal
 	
`startDeltaPhase`: the initial value added to the phase counter as the generator begins to step through the waveform buffer
 	
`stopDeltaPhase`: the final value added to the phase counter before the generator restarts or reverses the sweep
 	
`deltaPhaseIncrement`: the amount added to the delta phase value every time the `dwellCount` period expires. This determines the amount by which the generator sweeps the output frequency in each dwell period.
 	
`dwellCount`: the time, in multiples of *ddsPeriod*, between successive additions of `deltaPhaseIncrement` to the delta phase counter. This determines the rate at which the generator sweeps the output frequency.
 	
`arbitraryWaveform`: a pointer to a buffer that holds the waveform pattern as a set of samples equally spaced in time
 	
`arbitraryWaveformSize`: the size of the arbitrary waveform buffer
 	
`sweepType`: determines whether the `startDeltaPhase` is swept up to the `stopDeltaPhase`, or down to it, or repeatedly swept up and down. Use one of the following values:
`UP`
`DOWN`
`UPDOWN`
`DOWNUP`
 	
`sweeps`: the number of times to sweep the frequency after a trigger event, according to `sweepType`.	

Returns	0: if one of the parameters is out of range non-zero: if successful

5.30 ps2000_set_sig_gen_built_in

```
int16_t ps2000_set_sig_gen_built_in
(
  int16_t              handle,
  int32_t              offsetVoltage,
  uint32_t             pkToPk,
  PS2000_WAVE_TYPE     waveType,
  float                startFrequency,
  float                stopFrequency,
  float                increment,
  float                dwellTime,
  PS2000_SWEEP_TYPE    sweepType,
  uint32_t             sweeps
)
```

This function sets up the signal generator to produce a signal from a list of built-in waveforms. If different start and stop frequencies are specified, the oscilloscope will sweep either up, down or up and down.

Applicability	PicoScope 2203, 2204, 2204A, 2205 and 2205A only
Arguments	

`handle`: the handle of the required oscilloscope
 	
`offsetVoltage`: the voltage offset, in microvolts, to be applied to the waveform
 	
`pkToPk`: the peak-to-peak voltage, in microvolts, of the waveform signal
 	
`waveType`: the type of waveform to be generated by the oscilloscope. See the table below.
 	
`startFrequency`: the frequency at which the signal generator should begin. For allowable values see `ps2000.h`.
 	
`stopFrequency`: the frequency at which the sweep should reverse direction or return to the start frequency
 	
`increment`: the amount by which the frequency rises or falls every `dwellTime` seconds in sweep mode
 	
`dwellTime`: the time in seconds between frequency changes in sweep mode
 	
`sweepType`: specifies whether the frequency should sweep from `startFrequency` to `stopFrequency`, or in the opposite direction, or repeatedly reverse direction. Use one of these values of the enumerated type `enPS2000SweepType`:
`PS2000_UP`
`PS2000_DOWN`
`PS2000_UPDOWN`
`PS2000_DOWNUP`
 	
`sweeps`: the number of times to sweep the frequency	

Returns	0: if one of the parameters is out of range non-zero: if successful

waveType values

PS2000_SINE	sine wave
PS2000_SQUARE	square wave
PS2000_TRIANGLE	triangle wave
PS2000_RAMPUP	rising sawtooth
PS2000_RAMPDOWN	falling sawtooth
PS2000_DC_VOLTAGE	DC voltage
PS2000_GAUSSIAN	Gaussian
PS2000_SINC	sin(x)/x
PS2000_HALF_SINE	half (full-wave rectified) sine

5.31 ps2000_set_trigger

```
int16_t ps2000_set_trigger
(
  int16_t    handle,
  int16_t    source,
  int16_t    threshold,
  int16_t    direction,
  int16_t    delay,
  int16_t    auto_trigger_ms
)
```

This function is used to enable or disable basic triggering and set its parameters.

For oscilloscopes that support advanced triggering, see ps2000SetAdvTriggerChannelConditions, ps2000SetAdvTriggerDelay and related functions.

Applicability	Triggering is available in block mode and fast streaming mode
Arguments	handle: the handle of the required oscilloscope source: where to look for a trigger. Use PS2000_CHANNEL_A (0), PS2000_CHANNEL_B (1) or PS2000_NONE(5). The number of channels available depends on the oscilloscope. threshold: the threshold for the trigger event. This is scaled in 16-bit ADC counts at the currently selected range. direction: use PS2000_RISING (0) or PS2000_FALLING (1) delay: the delay, as a percentage of the requested number of data points, between the trigger event and the start of the block. It should be in the range -100% to +100%. Thus, 0% means that the trigger event is at the first data value in the block, and -50% means that it is in the middle of the block. If you wish to specify the delay as a floating-point value, use ps2000_set_trigger2 instead. Note that if delay = 0 and you call ps2000_stop before a trigger event occurs, the device will return no data. auto_trigger_ms: the delay in milliseconds after which the oscilloscope will collect samples if no trigger event occurs. If this is set to zero the oscilloscope will wait for a trigger indefinitely.
Returns	0: if one of the parameters is out of range non-zero: if successful

5.32 ps2000_set_trigger2

```
int16_t ps2000_set_trigger2
(
  int16_t    handle,
  int16_t    source,
  int16_t    threshold,
  int16_t    direction,
  float      delay,
  int16_t    auto_trigger_ms
)
```

This function is used to enable or disable triggering and set its parameters. It has the same behavior as ps2000_set_trigger, except that the delay parameter is a floating-point value.

For oscilloscopes that support advanced triggering, see ps2000SetAdvTriggerChannelConditions and related functions.

Applicability	Triggering is available in block mode and fast streaming mode only
Arguments	handle: the handle of the required oscilloscope source: specifies where to look for a trigger. Use PS2000_CHANNEL_A (0), PS2000_CHANNEL_B (1) or PS2000_NONE (5). threshold: the threshold for the trigger event. This is scaled in 16-bit ADC counts at the currently selected range. direction: use PS2000_RISING (0) or PS2000_FALLING (1) delay: specifies the delay, as a percentage of the requested number of data points, between the trigger event and the start of the block. It should be in the range -100% to +100%. Thus, 0% means that the trigger event is at the first data value in the block, and -50% means that it is in the middle of the block. If you wish to specify the delay as an integer, use ps2000_set_trigger instead. Note that if delay = 0 and you call ps2000_stop before a trigger event occurs, the device will return no data. auto_trigger_ms: the delay in milliseconds after which the oscilloscope will collect samples if no trigger event occurs. If this is set to zero the oscilloscope will wait for a trigger indefinitely.
Returns	0: if one of the parameters is out of range non-zero: if successful

5.33 ps2000_stop

```
int16_t ps2000_stop
(
  int16_t    handle
)
```

Call this function to stop the oscilloscope sampling data.

When running the device in streaming mode, you should always call this function after the end of a capture to ensure that the scope is ready for the next capture.

When running the device in block mode or ETS mode, you can call this function to interrupt data capture.

Note that if you are using block mode or ETS mode and call this function before the oscilloscope is ready, no capture will be available and the driver will not return any samples.

Applicability	All modes
Arguments	`handle`: the handle of the required oscilloscope
Returns	0: if an invalid handle is passed non-zero: if successful

5.34 my_get_overview_buffers

```
void my_get_overview_buffers
(
  int16_t     **overviewBuffers,
  int16_t     overflow,
  uint32_t    triggeredAt,
  int16_t     triggered,
  int16_t     auto_stop,
  uint32_t    nValues
)
```

This is the callback function in your application that receives data from the driver in fast streaming mode. You pass a pointer to this function to `ps2000_get_streaming_last_values`, which then calls it back when the data is ready. Your callback function should do nothing more than copy the data to another buffer within your application. To maintain the best application performance, the function should return as quickly as possible without attempting to process or display the data.

The function name `my_get_overview_buffers` is arbitrary. When you write this function, you can give it any name you wish. The PicoScope driver does not need to know your function's name, as it refers to it only by the pointer that you pass to `ps2000_get_streaming_last_values`.

For an example of a suitable callback function, see the programming examples included in the Pico Technology SDK.

Applicability	Fast streaming mode only PicoScope 2202, 2203, 2204, 2204A, 2205 and 2205A only Not compatible with ETS triggering - has no effect in ETS mode
Arguments	overviewBuffers: a pointer to a location where ps2000_get_streaming_last_values will store a pointer to its overview buffers that contain the sampled data. The driver creates the overview buffers when you call ps2000_run_streaming_ns to start fast streaming. overviewBuffers is a two dimensional array containing an array of length nValues for each channel (overviewBuffers[4][nValues]). Disabled channels return a null pointer resulting in four overview pointers whether all channels are enabled or not. overviewBuffer [0] ch_a_max overviewBuffer [1] ch_a_min overviewBuffer [2] ch_b_max overviewBuffer [3] ch_b_min overflow: a bit field that indicates whether there has been a voltage overflow and, if so, on which channel. The bit assignments are as follows: Bit 0 - Ch A overflow Bit 1 - Ch B overflow triggeredAt: an index into the overview buffers, indicating the sample at the trigger event. Valid only when triggered is TRUE. triggered: a Boolean indicating whether a trigger event has occurred and triggeredAt is valid. Any non-zero value signifies TRUE. auto_stop: a Boolean indicating whether streaming data capture has automatically stopped. Any non-zero value signifies TRUE. nValues: the number of values in each overview buffer
Returns	nothing

6 Programming examples

Your SDK installation includes programming examples in several languages and development environments. Please refer to the SDK for details.

7 Driver error codes

Code	Name	Description
0	PS2000_OK	The oscilloscope is functioning correctly.
1	PS2000_MAX_UNITS_OPENED	Attempts have been made to open more than PS2000_MAX_UNITS oscilloscopes.
2	PS2000_MEM_FAIL	Not enough memory could be allocated on the host machine.
3	PS2000_NOT_FOUND	An oscilloscope could not be found.
4	PS2000_FW_FAIL	Unable to download firmware.
5	PS2000_NOT_RESPONDING	The oscilloscope is not responding to commands from the PC.
6	PS2000_CONFIG_FAIL	The configuration information in the oscilloscope has become corrupt or is missing.
7	PS2000_OS_NOT_SUPPORTED	The operating system is not supported by this driver.

8 Glossary

Aggregation. In fast streaming mode, the PicoScope 2000 driver can use a method called aggregation to reduce the amount of data your application needs to process. This means that for every block of consecutive samples, it stores only the minimum and maximum values. You can set the number of samples in each block, called the aggregation parameter, when you call `ps2000_run_streaming_ns` for real-time capture, and when you call `ps2000_get_streaming_values` to obtain post-processed data.

Analog bandwidth. The input frequency at which the signal amplitude has fallen by 3 dB, or by half the power, from its nominal value.

Block mode. A sampling mode in which the computer prompts the oscilloscope to collect a block of data into its internal memory before stopping the oscilloscope and transferring the whole block into computer memory. This is the best mode to use when the input signal being sampled contains high frequencies. To avoid aliasing effects, the sampling rate must be greater than twice the maximum frequency in the input signal.

Buffer size. The size of the oscilloscope's buffer memory. The oscilloscope uses this to store data temporarily so that it can sample data independently of the speed at which it can transfer data to the computer.

Coupling mode. This mode selects either AC or DC coupling in the oscilloscope's input path. Use AC mode for small signals that may be superimposed on a DC level. Use DC mode for measuring absolute voltage levels. Set the coupling mode using `ps2000_set_channel`.

Driver. A piece of software that controls a hardware device. The driver for the PicoScope 2000 Series PC Oscilloscopes is supplied in the form of a 32-bit Windows DLL, which contains functions that you can call from your application.

ETS. Equivalent time sampling. Some PicoScope 2000 Series oscilloscopes can collect data over a number of cycles of a repetitive waveform to give a higher effective sampling rate than is possible for a single cycle. Equivalent time sampling allows the oscilloscope to use faster timebases than those available in real-time mode.

Maximum sampling rate. A figure indicating the maximum number of samples the oscilloscope is capable of acquiring per second. Maximum sample rates are usually given in MS/s (megasamples per second) or GS/s (gigasamples per second). The higher the sampling speed of the oscilloscope, the more accurate the representation of the high-frequency details in a fast signal.

Oversampling. A method of increasing the effective resolution of a measurement by sampling faster than the required sampling rate, then averaging the extra samples. An oversampling factor of four increases the effective resolution by one bit, but this increased resolution comes at the expense of reducing the maximum sampling rate by the same factor.

Overview buffer. A buffer in the PC's memory in which the PicoScope 2000 Series driver temporarily stores data on its way from the oscilloscope to the application's buffer.

PC Oscilloscope. A virtual instrument consisting of a PicoScope PC Oscilloscope and a software application.

PicoScope 2000 Series. A range of low-cost PC Oscilloscopes that includes the PicoScope 2202, 2203, 2204 and 2205 two-channel oscilloscopes and the PicoScope 2104 and 2105 handheld oscilloscopes.

PicoScope software. This is an application that accompanies all our PC Oscilloscopes. Although you do not need it if you are writing your own application, you should install it anyway, because it includes the drivers that you will need to control the oscilloscope.

Real-time continuous mode. A sampling mode in which the software repeatedly requests single samples from the oscilloscope. This mode is suitable for low sampling rates when you require the latest sample to be displayed as soon as it is captured.

Streaming mode. A sampling mode in which the oscilloscope samples data and returns it to the computer in an unbroken stream. This mode of operation is suitable when the input signal being sampled contains only low frequencies.

Timebase. A number that is supplied to the driver to specify a sampling rate for the oscilloscope. Each oscilloscope model has a different range of possible sampling frequencies, as specified in the User's Guide for that model.

USB 1.1. An early version of the Universal Serial Bus standard found on older PCs. Although your PicoScope will work with a USB 1.1 port, it will operate much more slowly than with a USB 2.0 or 3.0 port.

USB 2.0. Universal Serial Bus (High Speed). A standard port used to connect external devices to PCs. The high-speed data connection provided by a USB 2.0 port enables your PicoScope to achieve its maximum performance.

USB 3.0. A faster version of the Universal Serial Bus standard. Your PicoScope is fully compatible with USB 3.0 ports and will operate with the same performance as on a USB 2.0 port.

Vertical resolution. A value, in bits, that indicates the number of input voltage levels that the oscilloscope can distinguish. Calculation techniques can improve the effective resolution.

Voltage range. The range of input voltages that the oscilloscope will measure in a given mode.

Index

A

B

C

D

E

F

H

L

M

N

O

P

R

S

T

U

V

W