PicoScope® 3000 Series

PC Oscilloscopes

Programmer's Guide

ps3000pg.en r5

Contents

1 Introduction

1.1 Overview

The **PicoScope 3000 Series is a range of** PC Oscilloscopes from Pico Technology. The range includes the following variants:

- General-purpose PicoScope 3204, 3205 and 3206
- High-precision PicoScope 3224 and 3424
- Differential PicoScope 3425

The scopes are fully USB 2.0-capable and are also backwards-compatible with USB 1.1, although they will run much more slowly over this connection. There is no need for an external power supply, as power is supplied from the USB port, making these oscilloscopes highly portable.

This manual explains how to use the API (application programming interface) functions, so that you can develop your own programs to collect and analyze data from the oscilloscope.

1.2 Minimum PC requirements

For the PicoScope 3000 Series PC Oscilloscope to operate correctly, you must connect it to a computer with the minimum requirements to run Windows or the following (whichever is the higher specification):

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 minimum*. USB 2.0 recommended. Must be connected directly to the port or a powered USB hub. Will not work on a passive hub.

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

1.3 License agreement

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 Limited claims the copyright of, and retains the rights to, all material (software, documents etc) contained in this release. You may copy and distribute the entire release in its original state, but must not copy individual items within the release other than for backup purposes.

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.

2 Programming the 3000 Series oscilloscopes

2.1 General procedure

The `ps3000.dll` library in the `lib` sub-directory of the Pico Technology SDK installation directory allows you to program a PicoScope 3000 Series PC Oscilloscope using standard C function calls.

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

- Open the scope unit.
- 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 scope unit is ready.
- Copy data to a buffer.
- Stop capturing data.
- Close the scope unit.

Numerous sample programs are installed as part of the software development kit (SDK). These show how to use the functions of the driver software in each of the modes available.

2.2 PicoScope 3000 Series driver

Your application will communicate with a PicoScope 3000 API driver called `ps3000.dll`, which is supplied in 32-bit and 64-bit versions. The driver exports the ps3000 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

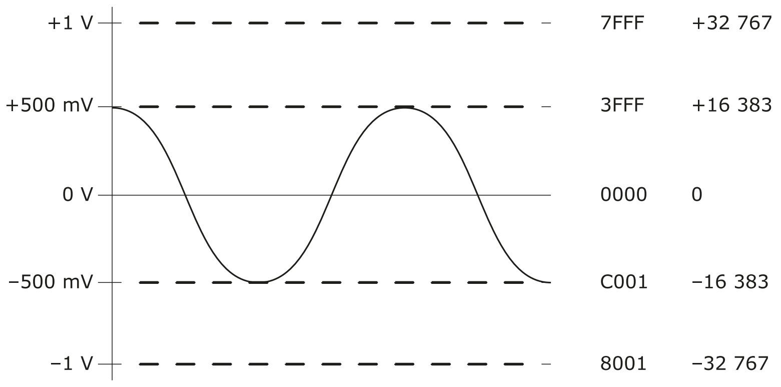
It is possible to set the gain for each channel with the `ps3000_set_channel` function. The input **voltage ranges** available depend on the oscilloscope model.

The PicoScope 3000 Series PC Oscilloscopes have a resolution of 12 bits, but the driver normalizes all readings to 16 bits. The following table shows the relationship between the reading from the driver and the voltage of the signal.

Constant	**Reading**		**Voltage**
	decimal	**hex**	
`PS3000_LOST_DATA`	-32 768	8000	Indicates a buffer overrun in fast streaming mode.
`PS3000_MIN_VALUE`	-32 767	8001	Negative full scale
`0`	0	0000	Zero volts
`PS3000_MAX_VALUE`	32 767	7FFF	Positive full scale

Example

1. Call `ps3000_set_channel` with `range` set to `PS3000_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 3000 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 `ps3000_set_trigger` function or, for scopes that support advanced triggering, `ps3000SetAdvTriggerChannelConditions` 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 `ps3000_set_trigger` and `ps3000SetAdvTriggerChannelConditions` 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, `ps3000SetPulseWidthQualifier`.

2.5 AC/DC coupling

Using the `ps3000_set_channel` function, each channel can be set to either **AC** or **DC** coupling. When AC coupling is used, any DC component of the signal is filtered out.

2.6 Signal generator

The PicoScope 3204/5/6 PC Oscilloscopes have a built-in **signal generator** which is set using `ps3000_set_siggen`. The output of the 3204 is a fixed-frequency square wave, while the 3205 and 3206 can produce a selection of accurate frequencies from 100 Hz to 1 MHz, and the waveform can be set to sine, square or triangle and swept back and forth in frequency. These options are selected under software control.

Applicability	Available only on the PicoScope 3204, 3205 and 3206 oscilloscopes. The signal generator output and external trigger input share the same connector, so these two functions cannot be used independently. It is possible, however, to use the output from the signal generator as a trigger.

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 3000 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. To compensate for this, 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 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 a PicoScope 3000 Series PC Oscilloscope to collect a block of data into its internal memory. When the oscilloscope has collected the whole block, it will signal that it is ready and then transfer 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 3000 Series scope can sample at a number of different rates. These rates 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 `ps3000_run_block`, `ps3000_ready`, `ps3000_stop` and `ps3000_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 `ps3000_open_unit`.
2. Select channel ranges and AC/DC coupling using `ps3000_set_channel`.
3. Using `ps3000_set_trigger`, set the trigger if required.
4. Using `ps3000_get_timebase`, select timebases until the required ns per sample is located.
5. Start the oscilloscope running using `ps3000_run_block`.
6. Poll the driver to find out when the oscilloscope has finished collecting data, using `ps3000_ready`.
7. Transfer the block of data from the oscilloscope using `ps3000_get_values` or `ps3000_get_times_and_values`.
8. Display the data.
9. Repeat steps 5 to 8.
10. Stop the oscilloscope using `ps3000_stop`.
11. Close the device using `ps3000_close_unit`.

Note that if you call `ps3000_get_values`, `ps3000_get_times_and_values` or `ps3000_stop` before the oscilloscope is ready, no capture will be available and the driver will not return any samples.

3.2 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 ps3000_set_trigger and ps3000_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 scope hardware accurately measures the delay, which is a small fraction of a single sampling interval, between each trigger event and the subsequent sample. The driver then shifts each capture slightly in time and overlays them so that the trigger points are exactly lined up. The result is a much larger set of samples spaced by a small fraction of the original sampling interval. The maximum effective sampling rates that can be achieved with this method are listed in the data sheet specifications for your scope device.

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.

See Using ETS mode for programming details.

Applicability	Available in block mode only. Available only on the PicoScope 3204, 3205 and 3206 variants. As ETS will return random time intervals, the ps3000_get_times_and_values function must be used. The ps3000_get_values function will return FALSE (0). Not suitable for one-shot (non-repetitive) signals.

3.2.1 Using ETS mode

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

1. Open the oscilloscope using ps3000_open_unit.
2. Select channel ranges and AC/DC switches using ps3000_set_channel.
3. Using ps3000_set_trigger, set the trigger if required.
4. Set ETS mode using ps3000_set_ets.
5. Start the oscilloscope running using ps3000_run_block.
6. Poll the driver to find out when the oscilloscope has finished collecting data, using ps3000_ready.
7. Transfer the block of data from the oscilloscope using ps3000_get_times_and_values.
8. Display the data.
9. Repeat steps 6 to 8 as necessary.
10. Stop the oscilloscope using ps3000_stop.
11. Close the device using ps3000_close_unit.

Note that if you call ps3000_get_values, ps3000_get_times_and_values or ps3000_stop before the oscilloscope is ready, no capture will be available and the driver will not return any samples.

3.3 Streaming mode

The **streaming modes** are alternatives to block mode that can capture data without gaps between blocks.

In streaming mode, the computer prompts the PicoScope 3000 Series PC Oscilloscope to start collecting data. The data is then transferred back to the PC without being stored in oscilloscope memory. Data can be sampled with a period between 1 µs 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 streaming modes:

- Compatible streaming mode
- Fast streaming mode

3.3.1 Compatible streaming mode

Compatible streaming mode is a basic streaming mode that works with all scope units, 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 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 K samples of data per channel, but the user must ensure that the `ps3000_get_values` function is called frequently enough to avoid buffer overrun.

See Using compatible streaming mode for programming details.

Applicability	Does not support triggering. The `ps3000_get_times_and_values` function will always return FALSE (0) in streaming mode.

3.3.1.1 Using compatible streaming mode

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

1. Open the oscilloscope using `ps3000_open_unit`.
2. Select channel ranges and AC/DC switches using `ps3000_set_channel`.
3. Start the oscilloscope running using `ps3000_run_streaming`.
4. Transfer the block of data from the oscilloscope using `ps3000_get_values`.
5. Display the data.
6. Repeat steps 4 and 5 as necessary.
7. Stop the oscilloscope using `ps3000_stop`.
8. Close the device using `ps3000_close_unit`.

3.3.2 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 3224, 3424 and 3425 oscilloscopes only. Works with triggering.

See Using fast streaming mode for programming details.

3.3.2.1 Using fast streaming mode

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

1. Open the oscilloscope using `ps3000_open_unit`.
2. Select channel ranges and AC/DC switches using `ps3000_set_channel`.
3. Set the trigger using `ps3000_set_trigger`.
4. Start the oscilloscope running using `ps3000_run_streaming_ns`.
5. Get a block of data from the oscilloscope using `ps3000_get_streaming_last_values`.
6. Display or process the data.
7. If required, check for overview buffer overruns by calling `ps3000_overview_buffer_status`.
8. Repeat steps 5 to 7 as necessary or until auto_stop is TRUE.
9. Stop fast streaming using `ps3000_stop`.
10. Retrieve any part of the data at any time scale by calling `ps3000_get_streaming_values`.
11. If you require raw data, retrieve it by calling `ps3000_get_streaming_values_no_aggregation`.
12. Repeat steps 10 to 11 as necessary.
13. Close the oscilloscope by calling `ps3000_close_unit`.

4 Combining oscilloscopes

It is possible to collect data using up to 64 PicoScope 3000 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.

The ps3000_open_unit function returns a handle to an oscilloscope. All the other functions require this handle for oscilloscope identification. For example, to collect data from two oscilloscopes at the same time:

```
handle1 = ps3000_open_unit
handle2 = ps3000_open_unit

ps3000_set_channel(handle1)
... set up unit 1
ps3000_run_block(handle1)

ps3000_set_channel(handle2)
... set up unit 2
ps3000_run_block(handle2)

ready = FALSE
while not ready
   ready = ps3000_ready(handle1)
   ready &= ps3000_ready(handle2)

ps3000_get_values(handle1)
ps3000_get_values(handle2)

ps3000_close_unit(handle1)
ps3000_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 3000 Series API exports the following functions for you to use in your own applications:

ps3000_close_unit
ps3000_flash_led
ps3000_get_streaming_last_values
ps3000_get_streaming_values
ps3000_get_streaming_values_no_aggregation
ps3000_get_timebase
ps3000_get_times_and_values
ps3000_get_unit_info
ps3000_get_values
ps3000_open_unit
ps3000_open_unit_async
ps3000_open_unit_progress
ps3000_overview_buffer_status
ps3000PingUnit
ps3000_ready
ps3000_run_block
ps3000_run_streaming
ps3000_run_streaming_ns
ps3000_save_streaming_data
ps3000_set_channel
ps3000_set_trigger
ps3000_set_trigger2
ps3000_stop
ps3000SetAdvTriggerChannelConditions
ps3000SetAdvTriggerChannelDirections
ps3000SetAdvTriggerChannelProperties
ps3000SetAdvTriggerDelay
ps3000SetPulseWidthQualifier

The following user-defined functions are also described:

Callback function to copy data to buffer
Callback function to save data

5.1 ps3000_close_unit

```
int16_t ps3000_close_unit
(
  int16_t    handle
)
```

Shuts down a PicoScope 3000 Series PC Oscilloscope.

Applicability	All modes
Arguments	`handle`, the handle, returned by `ps3000_open_unit`, identifying the oscilloscope being closed
Returns	`1` if a valid handle is passed `0` if handle is not valid

5.2 ps3000_flash_led

```
int16_t ps3000_flash_led
(
  int16_t    handle
)
```

Flashes the LED on the front of the oscilloscope three times and returns within one second.

Applicability	All modes
Arguments	`handle,` identifies the scope device
Returns	`1` if a valid handle is passed `0` if handle is invalid

5.3 ps3000_get_streaming_last_values

```
int16_t ps3000_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 have called `ps3000_run_streaming_ns` beforehand to set up fast streaming.

Applicability	Fast streaming mode only PicoScope 3224, 3424 and 3425 variants only Not compatible with ETS triggering - function has no effect in ETS mode
Arguments	`handle`, identifies the scope device `lpGetOverviewBuffersMaxMin`, a pointer to the callback function in your application that receives data from the streaming driver
Returns	The actual number of data values returned per channel, which may be less than `max_samples` if streaming, where `max_samples` is a parameter passed to `ps3000_run_streaming_ns` `FALSE` if one of the parameters is out of range

5.4 ps3000_get_streaming_values

```
uint32_t ps3000_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 `ps3000_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 `PS3000_MIN_VALUE` to `PS3000_MAX_VALUE`. The special value `PS3000_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 of raw samples.

Applicability	Fast streaming mode only PicoScope 3224, 3424 and 3425 variants only Not compatible with ETS triggering - function has no effect in ETS mode
Arguments	`handle,` identifies the scope device `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, pbuffer_c_max, pbuffer_c_min, pbuffer_d_max, pbuffer_d_min,` as the two parameters above but for channels B, C and D `overflow,` where the function will write a bit field indicating whether the voltage on each of the input channels has overflowed:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				D	C	B	A					D	C	B	A
				common-mode overflow								differential overflow*			

*3425 variant only

	`triggerAt,` a pointer to where the function will write an index into the buffers. The index is the number of the sample at the trigger reference point. 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 ps3000_get_streaming_values_no_aggregation

```
uint32_t ps3000_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 `ps3000_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 `PS3000_MIN_VALUE` to `PS3000_MAX_VALUE`. The special value `PS3000_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 3224, 3424 and 3425 variants only Not compatible with ETS triggering - has no effect in ETS mode
Arguments	`handle,` identifies the scope device `start_time,` the time in nanoseconds of the first data sample required `pbuffer_a, pbuffer_b, pbuffer_c, pbuffer_d,` pointers to buffers into which the function will write the raw sample values from channels A, B, C and D `overflow,` where the function will write a bit field indicating whether the voltage on each of the input channels has overflowed:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				D	C	B	A					D	C	B	A
				common-mode overflow								differential overflow*			

*3425 variant only

	`triggerAt,` where the function will write an index into the buffers. The index is the number of the sample at the trigger reference point. Valid only when trigger is TRUE. `trigger,` where the function will write a Boolean 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 ps3000_get_timebase

```
int16_t ps3000_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 `ps3000_set_channel` and, if required, ETS mode using `ps3000_set_ets` first.

Applicability	All modes
Arguments	`handle,` identifies the scope device `timebase,` a code between 0 and the maximum timebase (dependent on variant). Timebase 0 is the fastest timebase, timebase 1 is twice the time per sample of timebase 0, timebase 2 is four times, etc. `no_of_samples,` the number of samples required. This value is used to calculate the most suitable time unit to use. `time_interval,` a pointer to the time interval, in nanoseconds, between readings at the selected timebase. If a null pointer is passed, nothing will be written here. `time_units,` a pointer to the most suitable units that the results should be measured in. This value should also be passed when calling `ps3000_get_times_and_values`. If a null pointer is passed, nothing will be written here. `oversample,` the amount of oversample required. An oversample of 4 would quadruple the time interval and quarter the maximum samples. At the same time it would increase 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 this pointer is null, nothing will be written here.
Returns	1 if all parameters are in range 0 on error

5.7 ps3000_get_times_and_values

```
int32_t ps3000_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 `ps3000_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`, identifies the scope device `times`, a pointer to the buffer for the 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, buffer_c, buffer_d,` pointers to the buffers that receive data from the specified channels (A, B, C or D). A pointer is unused if the oscilloscope is not collecting data from that channel. If a pointer is NULL, nothing will be written to it. `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:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				D	C	B	A					D	C	B	A
				common-mode overflow								differential overflow*			

*3425 variant only

	`time_units`, which can be one of: `PS3000_FS` (0, femtoseconds), `PS3000_PS` (1, picoseconds), `PS3000_NS` (2, nanoseconds, default), `PS3000_US` (3, microseconds), `PS3000_MS` (4, milliseconds) or `PS3000_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 or if the times will overflow with the time_units requested. Use `ps3000_get_timebase` to acquire the most suitable `time_units`.

5.8 ps3000_get_unit_info

```
int16_t ps3000_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 fails to open, only `line` types 0 and 6 are available to explain why the last open unit call failed.

Applicability	All modes
Arguments	`handle`, identifies the device from which information is required. If an invalid handle is passed, the error code from the last unit that failed to open is returned. `string`, a pointer to the character string buffer in the calling function where the unit information string (selected with line) will be stored. If a null pointer is passed, 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`, an enumerated type specifying what information is required from the driver.
Returns	The length of the string written to the character string buffer, `string`, by the function 0 if one of the parameters is out of range, or a null pointer is passed for string

`line`		**String returned**	**Example**
0	`PS3000_DRIVER_VERSION`	The version number of the DLL used by the oscilloscope driver.	`"1, 0, 0, 2"`
1	`PS3000_USB_VERSION`	The type of USB connection that is being used to connect the oscilloscope to the computer.	`"1.1" or "2.0"`
2	`PS3000_HARDWARE_VERSION`	The hardware version of the attached oscilloscope.	`"1"`
3	`PS3000_VARIANT_INFO`	The variant of PicoScope 3000 PC Oscilloscope that is attached to the computer.	`"3425"`
4	`PS3000_BATCH_AND_SERIAL`	The batch and serial number of the oscilloscope.	`"CMY66/052"`
5	`PS3000_CAL_DATE`	The calibration date of the oscilloscope.	`"21Oct07"`
6	`PS3000_ERROR_CODE`	One of the Error codes.	`"4"`

5.9 ps3000_get_values

```
int32_t ps3000_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 `ps3000_run_streaming`, or in block mode after calling `ps3000_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 Do not use in fast streaming mode - use `ps3000_get_streaming_last_values` instead
Arguments	`handle`, identifies the scope device `buffer_a, buffer_b, buffer_c, buffer_d`, pointers to the buffers that receive data from the specified channels (A, B, C or D). A pointer is unused if the oscilloscope is not collecting data from that channel. If a pointer is NULL, nothing will be written to it. `overflow`, 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:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				D	C	B	A					D	C	B	A
				common-mode overflow								differential overflow*			

*3425 variant only

	`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 `FALSE` if one of the parameters is out of range

5.10 ps3000_open_unit

```
int16_t ps3000_open_unit (void)
```

This function opens a PicoScope 3000 Series PC 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` (device handle) if the device opened

5.11 ps3000_open_unit_async

```
int16_t ps3000_open_unit_async (void)
```

This function opens a PicoScope 3000 Series PC Oscilloscope without waiting for the operation to finish. You can find out when it has finished by periodically calling `ps3000_open_unit_progress` until that function returns a non-zero value.

The driver can support up to four oscilloscopes.

Applicability	All modes
Arguments	None
Returns	0 if there is a previous open operation in progress 1 if the call has successfully initiated an open operation

5.12 ps3000_open_unit_progress

```
int16_t ps3000_open_unit_progress
(
  int16_t    *handle,
  int16_t    *progress_percent
)
```

This function checks on the progress of `ps3000_open_unit_async`.

Applicability	All modes Use only with `ps3000_open_unit_async`
Arguments	`handle,` a pointer to a location in which the function will store the handle of the opened device `0` if no unit is found or the unit fails to open, handle of device (valid only if function returns TRUE) `progress_percent,` a pointer to an estimate of the progress towards opening the unit, from 0 to 100. 100 implies that the operation is complete.
Returns	`1` if the driver successfully opens the unit `0` if opening still in progress `-1` if the unit failed to open or was not found

5.13 ps3000_overview_buffer_status

```
int16_t ps3000_overview_buffer_status
(
  int16_t    handle,
  int16_t    *previous_buffer_overrun
)
```

This function indicates whether or not the overview buffers used by `ps3000_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 `ps3000_run_streaming_ns`.

Applicability	Fast streaming mode only PicoScope 3224, 3424 and 3425 variants only Not compatible with ETS triggering – function has no effect in ETS mode
Arguments	`handle,` identifies the scope device `previous_buffer_overrun,` a pointer to a Boolean indicating whether the overview buffers have overrun. Any non-zero value indicates a buffer overrun.
Returns	`0` if the function failed due to an invalid handle `1` if the function was successful

5.14 ps3000PingUnit

```
int16_t ps3000PingUnit
(
  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,` identifies the scope device
Returns	0 if function fails: call `ps3000_get_unit_info` for further information Any non-zero value: communication successful

5.15 ps3000_ready

```
int16_t ps3000_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,` identifies the scope device
Returns	`1` 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 is passed, or the oscilloscope is in streaming mode, or the scope is still collecting data in block mode. `-1` if device not attached. The endpoint transfer fails, indicating that the unit may well have been unplugged.

5.16 ps3000_run_block

```
int16_t ps3000_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`, identifies the scope device `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 scope device. Note that 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, over which the ADC will 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. Note: The actual time may differ from computer to computer, depending on how fast the computer can respond to I/O requests.
Returns	`0` if one of the parameters is out of range `1` if successful

5.17 ps3000_run_streaming

```
int16_t ps3000_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 faster streaming with the PicoScope 3224, 3424 and 3425 variants, use `ps3000_run_streaming_ns` instead.

Applicability	Compatible streaming mode only
Arguments	`handle`, identifies the scope device `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 caller'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 `ps3000_get_values` are returned. If this is 1, the number of values requested by `ps3000_get_values` are returned, even if they have already been read by `ps3000_get_values`.
Returns	`1` if streaming has been enabled correctly `0` if a problem occurred or a value was out of range

5.18 ps3000_run_streaming_ns

```
int16_t ps3000_run_streaming_ns
(
  int16_t             handle,
  uint32_t            sample_interval,
  PS3000_TIME_UNITS   time_units,
  uint32_t            max_samples,
  int16_t             auto_stop,
  uint32_t            noOfSamplesPerAggregate,
  uint32_t            overview_buffer_size
)
```

This function tells the scope unit to start collecting data in fast streaming mode. The function returns immediately without waiting for data to be captured. After calling this function, you should next call `ps3000_get_streaming_last_values` to copy the data to your application's buffer.

Applicability	Fast streaming mode only PicoScope 3224, 3424 and 3425 variants only
Arguments	`handle,` identifies the scope device `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. The maximum allowed value is 2^{21} (2,097,152). `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. You can check for overview buffer overruns using the `ps3000_overview_buffer_status` function and adjust the overview buffer size if necessary. The minimum and maximum overview buffer sizes are 15,000 and 1,000,000 samples for the PicoScope 3424 and 3425, and 7500 and 500,000 samples for the PicoScope 3224.
Returns	`1` if streaming has been enabled correctly `0` if a problem occurred or a value was out of range

5.19 ps3000_save_streaming_data

```
int16_t ps3000_save_streaming_data
(
  int16_t                   handle,
  PS3000_CALLBACK_FUNC      lpCallbackFunc,
  int16_t                   *dataBuffers,
  int16_t                   dataBufferSize
)
```

This function sends all available streaming data to the `my_save_streaming_data` callback function in your application. Your callback function decides what to do with the data.

Applicability	Fast streaming mode only PicoScope 3224, 3424 and 3425 variants only Not compatible with ETS triggering - function has no effect in ETS mode
Arguments	`handle,` identifies the scope device `lpCallbackFunc,` a pointer to the `my_save_streaming_data` callback function in your application that handles the saving of streaming data `dataBuffers,` a pointer to the data `dataBufferSize,` the size of the buffer, in samples
Returns	undefined

5.20 ps3000_set_channel

```
int16_t ps3000_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.

Applicability	All modes
Arguments	`handle`, identifies the scope device `channel`, an enumerated type. Use `PS3000_CHANNEL_A (0)`, `PS3000_CHANNEL_B (1)`, `PS3000_CHANNEL_C (2)` or `PS3000_CHANNEL_D (3)`. Channels C and D are not available on all models. `enabled`, specifies if the channel is active: TRUE = active, FALSE = inactive `dc`, specifies the AC/DC coupling mode: TRUE = DC, FALSE = AC `range`, a code between 1 and 10. See the table below, but note that each scope variant supports only a subset of these ranges.
Returns	`0` if unsuccessful, or if one or more of the arguments are out of range `1` if successful

Code	Enumeration	Range
1	PS3000_20MV	±20 mV
2	PS3000_50MV	±50 mV
3	PS3000_100MV	±100 mV
4	PS3000_200MV	±200 mV
5	PS3000_500MV	±500 mV
6	PS3000_1V	±1 V
7	PS3000_2V	±2 V
8	PS3000_5V	±5 V
9	PS3000_10V	±10 V
10	PS3000_20V	±20 V
11	PS3000_50V	±50 V
12	PS3000_100V	±100 V
13	PS3000_200V	±200 V
14	PS3000_400V	±400 V

5.21 ps3000_set_ets

```
int32_t ps3000_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 (equivalent time sampling) and to set the ETS parameters.

Applicability	ETS is available only on the PicoScope 3204, 3205 and 3206 variants
Arguments	`handle,` identifies the scope device `mode,` `PS3000_ETS_OFF (0)` - disables ETS `PS3000_ETS_FAST (1)` - enables ETS and provides `ets_cycles` cycles of data, which may contain data from previously returned cycles `PS3000_ETS_SLOW (2)` - enables ETS and provides fresh data every `ets_cycles` cycles. `PS3000_ETS_SLOW` takes longer to provide each data set, but the data sets are more stable and unique. `ets_cycles,` specifies 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,` specifies 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

5.22 ps3000_set_siggen

```
int32_t ps3000_set_siggen
(
  int16_t    handle,
  int16_t    wave_type,
  int32_t    start_frequency,
  int32_t    stop_frequency,
  float      increment,
  int16_t    dwell_time,
  int16_t    repeat,
  int16_t    dual_slope
)
```

This function is used to enable or disable the signal generator and sweep functions.

Applicability	Sweep functions are not available if the oscilloscope is in streaming mode The signal generator is available only on the PicoScope 3204, 3205 and 3206 PC Oscilloscope variants. See remarks below for more information.
Arguments	`handle`, identifies the scope device `wave_type`, the type of wave. Choose `PS3000_SQUARE (0)`, `PS3000_TRIANGLE (1)` or `PS3000_SINE (2)`. *This argument has no effect if used with the PicoScope 3204 variant.* `start_frequency`, the required frequency, in the range 0 < freq < 1 MHz, to start the sweep or the frequency generated in a non-sweep mode. `0` switches the signal generator off. `stop_frequency`, the required stop frequency of the sweep, in the range 0 < freq < 1 MHz but not necessarily greater than `start_frequency`. If the start and stop frequencies are the same, the signal generator will be run with a constant frequency. *This argument has no effect if used with the PicoScope 3204 variant, or if run in streaming mode.* `increment`, the size of the steps to increment or decrement the frequency by in a sweep mode. It must always be positive. The start and stop frequencies determine whether to increment or decrement. It must be a frequency in the range 0.1 Hz < `increment` < `|stop_frequency - start_frequency|`. It is not used in a non-sweep mode. *It has no effect if used with the PicoScope 3204 variant.* `dwell_time`, the time, in milliseconds, to wait before increasing the frequency by `increment` in a sweep mode. This is unused in a non-sweep mode. *This argument has no effect if used with the PicoScope 3204 variant.* `repeat`, if TRUE restarts the sweep when the `stop_frequency` is reached, if FALSE continues indefinitely at `stop_frequency` when it is reached. *This argument has no effect if used with the PicoScope 3204 variant.*

	dual_slope, if repeat is TRUE this specifies what to do at the stop_frequency. TRUE will sweep back towards the start_frequency, FALSE will restart the sweep from start_frequency. *This argument has no effect if used with the PicoScope 3204 variant.*
Returns	The actual frequency or start frequency, in hertz, that is generated 0 if one of the parameters is not in range

Remarks

The PicoScope 3204 variant has a simple 1 kHz square wave signal generator for scope probe calibration. With this variant, therefore, only two arguments of this function have any effect:

To switch the square wave on, use a valid handle and set start_frequency to a non-zero value. To switch the square wave off, use a valid handle and set start_frequency to 0.

5.23 ps3000_set_trigger

```
int16_t ps3000_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 triggering and its parameters.

Applicability	Triggering is available in block mode and fast streaming mode
Arguments	`handle,` identifies the scope device `source,` specifies where to look for a trigger. Use `PS3000_CHANNEL_A (0), PS3000_CHANNEL_B (1), PS3000_CHANNEL_C (2), PS3000_CHANNEL_D (3), PS3000_EXTERNAL(4)` or `PS3000_NONE(5)`. The number of channels available will depend on the scope variant. `threshold,` the threshold for the trigger event. This is scaled in 16-bit ADC counts at the currently selected range. If an external trigger is enabled the range is fixed at +/-20V. `direction,` use `PS3000_RISING (0)` or `PS3000_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 a floating-point value, use `ps3000_set_trigger2` instead. `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 `1` if successful

5.24 ps3000_set_trigger2

```
int16_t ps3000_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 its parameters. It has the same behavior as `ps3000_set_trigger`, except that the `delay` parameter is a floating-point value.

Applicability	Triggering is available in block mode and fast streaming mode only
Arguments	`handle,` identifies the scope device `source,` specifies where to look for a trigger. Use `PS3000_CHANNEL_A (0), PS3000_CHANNEL_B (1), PS3000_CHANNEL_C (2), PS3000_CHANNEL_D (3), PS3000_EXTERNAL(4)` or `PS3000_NONE (5)`. Channels C, D and External are not available on all models. `threshold,` the threshold for the trigger event. This is scaled in 16-bit ADC counts at the currently selected range. If an external trigger is enabled the range is fixed at ±20 V. `direction,` use `PS3000_RISING (0)` or `PS3000_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 `ps3000_set_trigger` instead. `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 1 if successful

5.25 ps3000_stop

```
int16_t ps3000_stop
(
  int16_t    handle
)
```

Call this function to stop the oscilloscope sampling data.

When running the device in streaming mode, 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. No capture will be available and the driver will not return any samples.

Applicability	All modes
Arguments	`handle`, identifies the scope device
Returns	`0` if an invalid handle is passed `1` if successful

5.26 ps3000SetAdvTriggerChannelConditions

```
int16_t ps3000SetAdvTriggerChannelConditions
(
  int16_t               handle,
  TRIGGER_CONDITIONS    *conditions,
  int16_t               nConditions
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is set up by defining a TRIGGER_CONDITIONS structure. Each structure is the AND of the states of one scope input.

Applicability	Available in block mode and fast streaming mode only
Arguments	handle, identifies the scope device conditions, a pointer to a 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 1 if successful

5.26.1 TRIGGER_CONDITIONS structure

A structure of this type is passed to ps3000SetAdvTriggerChannelConditions in the conditions argument to specify the trigger conditions, and is defined as follows:

```
typedef struct tTriggerConditions
{
  TRIGGER_STATE channelA;
  TRIGGER_STATE channelB;
  TRIGGER_STATE channelC;
  TRIGGER_STATE channelD;
  TRIGGER_STATE external;
  TRIGGER_STATE pulseWidthQualifier;
} TRIGGER_CONDITIONS;
```

Applicability	See ps3000SetAdvTriggerChannelConditions
Members	channelA, channelB, channelC, channelD, pulseWidthQualifier: the type of condition that should be applied to each channel. Use these constants: CONDITION_DONT_CARE (0) CONDITION_TRUE (1) CONDITION_FALSE (2) external, not used

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 oscilloscope can only use a single channel for the trigger source. Therefore you must define CONDITION_TRUE or CONDITION_FALSE, and the pulse width qualifier if required, for only one channel at a time.

5.27 ps3000SetAdvTriggerChannelDirections

```
int16_t ps3000SetAdvTriggerChannelDirections
(
  int16_t                  handle,
  THRESHOLD_DIRECTION      channelA,
  THRESHOLD_DIRECTION      channelB,
  THRESHOLD_DIRECTION      channelC,
  THRESHOLD_DIRECTION      channelD,
  THRESHOLD_DIRECTION      ext
)
```

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

Applicability	Available in block mode and fast streaming mode only
Arguments	`handle,` identifies the scope device `channelA, channelB, channelC, channelD,` all specify the direction in which the signal must pass through the threshold to activate the trigger. The allowable values for a `THRESHOLD_DIRECTION` variable are listed in the table below. `ext,` not used
Returns	`0` if unsuccessful, or if one or more of the arguments are out of range `1` if successful

THRESHOLD_DIRECTION constants

`ABOVE`	for gated triggers: above a threshold
`BELOW`	for gated triggers: below a threshold
`RISING`	for threshold triggers: rising edge
`FALLING`	for threshold triggers: falling edge
`RISING_OR_FALLING`	for threshold triggers: either edge
`INSIDE`	for window-qualified triggers: inside window
`OUTSIDE`	for window-qualified triggers: outside window
`ENTER`	for window triggers: entering the window
`EXIT`	for window triggers: leaving the window
`ENTER_OR_EXIT`	for window triggers: either entering or leaving the window
`NONE`	no trigger

5.28 ps3000SetAdvTriggerChannelProperties

```
int16_t ps3000SetAdvTriggerChannelProperties
(
  int16_t                       handle,
  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
Arguments	`handle,` identifies the scope device `channelProperties,` a pointer to a `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 scope device will wait before collecting data if no trigger event occurs. If this is set to zero, the scope device will wait indefinitely for a trigger.
Returns	`0` if unsuccessful, or if one or more of the arguments are out of range `1` if successful

5.28.1 TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to ps3000SetAdvTriggerChannelProperties in the channelProperties argument to specify the trigger mechanism, and is defined as follows:

```
typedef struct tTriggerChannelProperties
{
  int16_t          thresholdMajor;
  int16_t          thresholdMinor;
  uint16_t         hysteresis;
  int16_t          channel;
  THRESHOLD_MODE   thresholdMode;
} TRIGGER_CHANNEL_PROPERTIES;
```

Applicability	See ps3000SetAdvTriggerChannelProperties
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)

5.29 ps3000SetAdvTriggerDelay

```
int16_t ps3000SetAdvTriggerDelay
(
  int16_t     handle,
  uint32_t    delay,
  float       preTriggerDelay
)
```

This function sets the post-trigger delay, which causes capture to start a defined time after the trigger event.

Applicability	Block mode only
Arguments	`handle,` identifies the scope device `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%. For example, 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.
Returns	`0` if unsuccessful, or if one or more of the arguments are out of range `1` if successful

5.30 ps3000SetPulseWidthQualifier

```
int16_t ps3000SetPulseWidthQualifier
(
  int16_t                  handle,
  PWQ_CONDITIONS           *conditions,
  int16_t                  nConditions,
  THRESHOLD_DIRECTION      direction,
  uint32_t                 lower,
  uint32_t                 upper,
  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 threshold triggering, level triggering or window 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
Arguments	`handle,` identifies the scope device `conditions,` a pointer to a `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, in samples `upper,` the upper limit of the pulse width counter, in samples. 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 `1` if successful

5.30.1 PWQ_CONDITIONS structure

A structure of this type is passed to `ps3000SetPulseWidthQualifier` in the `conditions` argument to specify the pulse-width qualifier conditions, and is defined as follows:

```
typedef struct tPwqConditions
{
  TRIGGER_STATE channelA;
  TRIGGER_STATE channelB;
  TRIGGER_STATE channelC;
  TRIGGER_STATE channelD;
  TRIGGER_STATE external;
} PWQ_CONDITIONS;
```

Applicability	Pulse-width qualified triggering

Members	`channelA, channelB, channelC, channelD:` the type of condition that should be applied to each channel. Use these constants:
	
	`CONDITION_DONT_CARE (0)`
	`CONDITION_TRUE (1)`
	`CONDITION_FALSE (2)`
	
	`external,` not used

5.31 Callback function to copy data to buffer

```
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 `ps3000_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 just for illustration. 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 address that you pass to `ps3000_get_streaming_last_values`.

For an example of a suitable callback function, see the programming examples included in your PicoScope installation.

Do not call `ps3000_stop` within this function: this will delete the overview buffer and cause an assertion error if you later try to access it.

Applicability	Fast streaming mode only PicoScope 3224, 3424 and 3425 variants only Not compatible with ETS triggering - has no effect in ETS mode

Arguments

`overviewBuffers`, a pointer to a location where `ps3000_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 `ps3000_run_streaming_ns` to start fast streaming. `overviewBuffers` is a two dimensional array containing an array of length `nValues` for each channel (`overviewBuffers[8][nValues]`). Disabled channels return a null pointer resulting in eight 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
`overviewBuffer [4]`	ch_c_max
`overviewBuffer [5]`	ch_c_min
`overviewBuffer [6]`	ch_d_max
`overviewBuffer [7]`	ch_d_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:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				D	C	B	A					D	C	B	A
				common-mode overflow								differential overflow*			

*3425 variant only

`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

5.32 Callback function to save data

```
int16_t my_save_streaming_data
(
  int16_t    *dataBuffer,
  int16_t    noOfBuffers
)
```

This is a callback function in your application that receives data from ps3000_save_streaming_data.

The function name my_save_streaming_data is just for illustration. 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; it refers to it only by the address that you pass to ps3000_save_streaming_data.

Do not call ps3000_stop within this function: this will delete the overview buffer and cause an assertion error if you later try to access it.

Applicability	Fast streaming mode only PicoScope 3224, 3424 and 3425 variants only Not compatible with ETS triggering - function has no effect in ETS mode
Arguments	dataBuffer, a pointer to the buffer where the values are stored noOfBuffers, tells your function how many buffers there are
Returns	undefined

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	PS3000_OK	The oscilloscope is functioning correctly.
1	PS3000_MAX_UNITS_OPENED	Attempts have been made to open more than PS3000_MAX_UNITS.
2	PS3000_MEM_FAIL	Not enough memory could be allocated on the host machine.
3	PS3000_NOT_FOUND	An oscilloscope could not be found.
4	PS3000_FW_FAIL	Unable to download firmware.
5	PS3000_NOT_RESPONDING	The oscilloscope is not responding to commands from the PC.
6	PS3000_CONFIG_FAIL	The configuration information in the oscilloscope has become corrupt or is missing.
7	PS3000_OS_NOT_SUPPORTED	The operating system is not supported by this driver.

8 Glossary

AC/DC control. Each channel can be set to either AC coupling or DC coupling. With DC coupling, the voltage displayed on the screen is equal to the true voltage of the signal across the differential inputs. With AC coupling, any DC component of the signal is filtered out, leaving only the variations in the signal (the AC component).

Aggregation. In fast streaming mode, the ps3000 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 `ps3000_run_streaming_ns` for real-time capture, and when you call `ps3000_get_streaming_values` to obtain post-processed data.

Aliasing. An effect that can cause digital oscilloscopes to display fast-moving waveforms incorrectly, by showing spurious low-frequency signals ("aliases") that do not exist in the input. To avoid this problem, choose a sampling rate that is at least twice the frequency of the fastest-changing input signal.

Analog bandwidth. All oscilloscopes have an upper limit to the range of frequencies at which they can measure accurately. The analog bandwidth of an oscilloscope is defined as the frequency at which a displayed sine wave has half the power of the input sine wave (or, equivalently, about 71% of the amplitude).

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 mode of operation is effective when the input signal being sampled is high frequency. Note: To avoid aliasing effects, the maximum input frequency must be less than half the sampling rate.

Buffer size. The size, in samples, of the oscilloscope buffer memory. The buffer memory is used by the oscilloscope to temporarily store data before transferring it to the PC.

Common-mode voltage. The common-mode voltage of two points is the average voltage of the two points with respect to ground. A differential oscilloscope accurately measures the voltage difference between its two inputs and ignores their common-mode voltage, as long as the common-mode voltage remains within a defined range. Outside this range the accuracy of the measurement cannot be guaranteed.

Differential oscilloscope. A differential oscilloscope measures the voltage difference between two points, regardless of the voltage of either point with respect to ground. This is unlike a conventional oscilloscope, which requires one of the two points to be at ground potential.

Differential voltage limit. The differential voltage (the voltage difference between the positive and negative inputs on one channel) must not exceed this limit, or the oscilloscope may be permanently damaged.

ETS. Equivalent Time Sampling. ETS constructs a picture of a repetitive signal by accumulating information over many similar wave cycles. This means the oscilloscope can capture fast-repeating signals that have a higher frequency than the maximum sampling rate. Note: ETS should not be used for one-shot or non-repetitive signals.

External trigger. This is the BNC socket marked **E** on the PicoScope 3204, 3205 and 3206 PC Oscilloscopes. It can be used to start a data collection run but cannot be used to record data. As it shares the same connector as the signal generator output, these two functions cannot be used at the same time. It is possible, however, to use the output from the signal generator as a trigger.

Maximum sampling rate. A figure indicating the maximum number of samples the oscilloscope is capable of acquiring per second. Maximum sample rates are given in MS/s (megasamples per second). The higher the sampling capability of the oscilloscope, the more accurate the representation of the high frequencies in a fast signal.

Oversampling. 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.

Overview buffer. A temporary buffer used by the driver to store data before passing it to your application.

Overvoltage. Any input voltage to the oscilloscope must not exceed the overvoltage limit, measured with respect to ground, otherwise the oscilloscope may be permanently damaged.

PC Oscilloscope. A measuring instrument consisting of a Pico Technology scope device and the PicoScope software. It provides all the functions of a bench-top oscilloscope without the cost of a display, hard disk, network adapter and other components that your PC already has.

Signal generator. This is a feature of some oscilloscopes which allows a signal to be generated without an external input device being present. The signal generator output is the BNC socket marked **E** on the oscilloscope. If you connect a BNC cable between this and one of the channel inputs, you can send a signal into one of the channels. On some units, the signal generator can generate a simple TTL square wave, while on others it can generate a sine, square or triangle wave that can be swept back and forth.

Note: The signal generator output is on the same connector as the external trigger input, so these two functions cannot be used at the same time. It is possible, however, to use the output from the signal generator as a trigger.

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 effective when the input signal being sampled contains only low frequencies.

Timebase. The sampling rate that the scope uses to acquire data. The timebase can be set to any value returned by the `ps3000_get_timebase` function.

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.

Vertical resolution. A value, in bits, indicating the degree of precision with which the oscilloscope can turn input voltages into digital values. Calculation techniques can improve the effective resolution.

Voltage range. The voltage range is the difference between the maximum and minimum voltages that can be accurately captured by the oscilloscope.

Index

United Kingdom headquarters

Pico Technology
James House
Colmworth Business Park
St. Neots
Cambridgeshire
PE19 8YP
United Kingdom

Tel: +44 (0) 1480 396 395
Fax: +44 (0) 1480 396 296

sales@picotech.com
support@picotech.com

www.picotech.com

United States headquarters

Pico Technology
320 N Glenwood Blvd
Tyler
Texas 75702
United States of America

Tel: +1 800 591 2796
Fax: +1 620 272 0981

ps3000pg.en r5 2016-06-30