



PicoScope 3000 Series PC Oscilloscopes

Programmer's Guide

Contents

| | |
|---|----|
| 1 Introduction | 1 |
| 1 Overview | 1 |
| 2 Minimum PC requirements | 1 |
| 3 Legal information | 2 |
| 4 Company details | 3 |
| 2 Technical information | 4 |
| 1 Driver | 4 |
| 2 Programming overview | 4 |
| 3 Device features | 4 |
| 1 AC/DC coupling | 4 |
| 2 Voltage ranges | 5 |
| 3 Oversampling | 5 |
| 4 Scaling | 5 |
| 5 Signal generator | 5 |
| 6 Triggering | 6 |
| 7 Combining oscilloscopes | 6 |
| 8 Sampling modes | 7 |
| 4 Functions | 13 |
| 1 ps3000_open_unit | 14 |
| 2 ps3000_open_unit_async | 15 |
| 3 ps3000_open_unit_progress | 16 |
| 4 ps3000_get_unit_info | 17 |
| 5 ps3000_set_channel | 18 |
| 6 ps3000_get_timebase | 19 |
| 7 ps3000_flash_led | 20 |
| 8 ps3000_set_siggen | 21 |
| 9 ps3000_set_ets | 23 |
| 10 ps3000_set_trigger | 24 |
| 11 ps3000_set_trigger2 | 25 |
| 12 ps3000SetAdvTriggerChannelProperties | 26 |
| 13 ps3000SetAdvTriggerChannelConditions | 28 |
| 14 ps3000SetAdvTriggerChannelDirections | 30 |
| 15 ps3000SetPulseWidthQualifier | 31 |
| 16 ps3000SetAdvTriggerDelay | 33 |
| 17 ps3000_run_block | 34 |
| 18 ps3000_run_streaming | 35 |
| 19 ps3000_ready | 36 |
| 20 ps3000_stop | 37 |
| 21 ps3000_get_values | 38 |
| 22 ps3000_get_times_and_values | 39 |
| 23 ps3000_run_streaming_ns | 41 |
| 24 ps3000_get_streaming_last_values | 42 |
| 25 Callback function to copy data to buffer | 43 |
| 26 ps3000_get_streaming_values | 45 |
| 27 ps3000_get_streaming_values_no_aggregation | 47 |
| 28 ps3000_save_streaming_data | 49 |
| 29 ps3000_overview_buffer_status | 50 |
| 30 Callback function to save data | 51 |
| 31 ps3000_close_unit | 52 |

| | |
|-------------------------------------|-----------|
| 5 Programming examples | 53 |
| 1 C | 53 |
| 2 C++ | 54 |
| 3 Visual Basic | 54 |
| 4 Delphi | 55 |
| 5 Excel | 55 |
| 6 Agilent VEE | 55 |
| 7 LabView | 55 |
| 6 Driver error codes | 57 |
| 3 Glossary | 58 |
| Index..... | 61 |

1 Introduction

1.1 Overview

The **PicoScope 3000 Series** is a range of [PC Oscilloscopes](#)^[59] 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](#)^[60]-capable and backwards-compatible with [USB 1.1](#)^[59]. 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 analyse 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):

| | |
|-------------------------|--|
| Processor | Pentium-class processor or equivalent minimum. |
| Memory | 256 MB minimum. |
| Disk space | 10 MB minimum. |
| Operating system | Microsoft Windows XP SP2, Vista or Windows 7. |
| Ports | USB 1.1 ^[59] compliant port minimum. USB 2.0 ^[60] compliant port recommended. Must be connected directly to the port or a powered USB hub. Will not work on a passive hub. |

1.3 Legal information

The material contained in this release is licensed, not sold. Pico Technology Limited grants a licence 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 products or with data collected using Pico 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 licence 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 28 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.

Trademarks

Windows is a trademark or registered trademark of Microsoft Corporation. Pico Technology Limited and PicoScope are internationally registered trademarks.

1.4 Company details

You can obtain technical assistance from Pico Technology at the following address:

Address: Pico Technology
James House,
Colmworth Business Park,
Eaton Socon,
St Neots,
Cambridgeshire PE19 8YP
United Kingdom

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

Email:
Technical Support: support@picotech.com
Sales: sales@picotech.com

Web site: www.picotech.com

2 Technical information

2.1 Driver

Important

You must install the PicoScope software, which includes the driver for the PicoScope 3000 Series scope devices, before plugging the scope device into your computer for the first time.

(If you do plug in a scope device before installing the driver, Windows will label the device as Unknown. You will then need to manually delete the device using the Device Manager before you can install the correct driver.)

The Windows XP/Vista/7 32-bit **driver**, `picopp.sys`, is installed under the control of an information file, `picopp.inf`.

Once you have installed the PicoScope software, Windows will automatically install the driver when you plug in the scope device for the first time.

2.2 Programming overview

The `ps3000.dll` library in your PicoScope installation directory allows you to program a PicoScope 3000 Series PC Oscilloscope using standard C [function calls](#).^[13]

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

- [Open](#)^[14] the scope unit.
- Set up the input channels with the required [voltage ranges](#)^[5] and [coupling mode](#)^[4].
- Set up [triggering](#)^[6].
- Start capturing data. (See [Sampling modes](#)^[7], 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](#)^[53] are installed with your PicoScope software. These show how to use the functions of the driver software in each of the modes available.

2.3 Device features

2.3.1 AC/DC coupling

Using the [ps3000_set_channel\(\)](#)^[18] 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.

2.3.2 Voltage ranges

It is possible to set the gain for each channel with the [ps3000_set_channel\(\)](#)^[18] function. This input **voltage ranges** available depend on the scope device that is connected.

2.3.3 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:

$$\text{Increase in resolution (bits)} = \log (\text{oversample}) / \log (4)$$

| | |
|---------------|--|
| Applicability | Available in block mode ^[8] only. |
|---------------|--|

2.3.4 Scaling

The PicoScope 3000 Series PC Oscilloscope has a resolution of 12 bits, but the oscilloscope driver normalises 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 |
|------------------|---------|--|
| PS3000_LOST_DATA | -32 768 | Indicates a buffer overrun in fast streaming ^[12] mode. |
| PS3000_MIN_VALUE | -32 767 | Negative full scale |
| 0 | 0 | Zero volts |
| PS3000_MAX_VALUE | 32 767 | Positive full scale |

2.3.5 Signal generator

The PicoScope 3204/5/6 PC Oscilloscopes have a built-in **signal generator** which is set using [ps3000_set_siggen\(\)](#)^[21]. 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 ^[59] 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. |
|---------------|--|

2.3.6 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\(\)](#)^[24] function. A trigger event can occur on any of the conditions available in the simple and advanced triggering modes.

| | |
|---------------|---|
| Applicability | Available in block mode ^[8] and fast streaming mode ^[12] only. Calls to the ps3000_set_trigger() ^[24] function have no effect in compatible streaming mode ^[11] . |
|---------------|---|

2.3.7 Combining oscilloscopes

It is possible to collect data using up to four PicoScope 3000 Series PC Oscilloscopes at the same time. Each oscilloscope must be connected to a separate USB port. If a USB hub is used it must be a powered hub. The [ps3000_open_unit\(\)](#)^[14] 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)
```

Note 1: It is not possible to synchronise the collection of data between oscilloscopes that are being used in combination.

2.3.8 Sampling modes

PicoScope 3000 Series PC Oscilloscopes can run in various **sampling modes**.

- **Block mode.**^[8] 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.
- **Streaming modes.**^[10] 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.**^[11]
 - **Fast streaming mode.**^[12]

2.3.8.1 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\(\)](#)^[34], [ps3000_ready\(\)](#)^[36], [ps3000_stop\(\)](#)^[37] and [ps3000_get_values\(\)](#)^[38].

See [Using block mode](#)^[8] for programming details.

2.3.8.2 Using block mode

This is the general procedure for reading and displaying data in [block mode](#):^[58]

1. Open the oscilloscope using [ps3000_open_unit\(\)](#).^[14]
2. Select channel ranges and AC/DC coupling using [ps3000_set_channel\(\)](#).^[18]
3. Using [ps3000_set_trigger\(\)](#)^[24], set the trigger if required.
4. Using [ps3000_get_timebase\(\)](#)^[19], select timebases until the required ns per sample is located.
5. Start the oscilloscope running using [ps3000_run_block\(\)](#).^[34]
6. Wait until the oscilloscope says it is ready using [ps3000_ready\(\)](#).^[36]
7. Transfer the block of data from the oscilloscope using [ps3000_get_values\(\)](#)^[38] or [ps3000_get_times_and_values\(\)](#).^[39]
8. Display the data.
9. Repeat steps 5 to 8.
10. Stop the oscilloscope using [ps3000_stop\(\)](#).^[37]

2.3.8.3 ETS (Equivalent Time Sampling)

ETS is a way of increasing the effective sampling rate when working with repetitive signals. It is controlled by the [ps3000_set_trigger\(\)](#)^[24] and [ps3000_set_ets\(\)](#)^[23] 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 uses special circuitry to add a tiny variable delay, a small fraction of a single sampling interval, to each trigger event. This shifts each capture slightly in time so that the samples occur at slightly different times relative to those in the previous capture. 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 Specifications table 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.

| | |
|----------------------|--|
| Applicability | <p>Available in block mode^[8] only.</p> <p>Available only on the PicoScope 3204, 3205 and 3206 variants.</p> <p>As ETS will return random time intervals, the ps3000_get_times_and_values()^[39] function must be used. The ps3000_get_values()^[38] function will return FALSE (0).</p> <p>Not suitable for one-shot (non-repetitive) signals.</p> |
|----------------------|--|

2.3.8.4 Using ETS mode

This is the general procedure for reading and displaying data in [ETS](#)^[9] mode:

1. Open the oscilloscope using [ps3000_open_unit\(\)](#)^[14]
2. Select channel ranges and AC/DC switches using [ps3000_set_channel\(\)](#)^[18]
3. Using [ps3000_set_trigger\(\)](#)^[24], set the trigger if required.
4. Set ETS mode using [ps3000_set_ets\(\)](#)^[23]
5. Start the oscilloscope running using [ps3000_run_block\(\)](#)^[34]
6. Wait until the oscilloscope says it is ready using [ps3000_ready\(\)](#)^[36]
7. Transfer the block of data from the oscilloscope using [ps3000_get_times_and_values\(\)](#)^[39]
8. Display the data.
9. Repeat steps 6 to 8 as necessary.
10. Stop the oscilloscope using [ps3000_stop\(\)](#)^[37]

2.3.8.5 Streaming modes

The **streaming modes** are alternatives to [block mode](#)^[8] 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 μ s 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](#)^[11]
- [Fast streaming mode](#)^[12]

2.3.8.6 Compatible streaming mode

Compatible streaming mode is a basic [streaming mode](#)^[10] 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\(\)](#)^[38] function is called frequently enough to avoid buffer overrun.

See [Using compatible streaming mode](#)^[11] for programming details.

| | |
|----------------------|--|
| Applicability | Does not support triggering ^[6] . The ps3000_get_times_and_values() ^[39] function will always return FALSE (0) in streaming mode. |
|----------------------|--|

2.3.8.7 Using compatible streaming mode

This is the general procedure for reading and displaying data in [compatible streaming mode](#)^[11]:

1. Open the oscilloscope using [ps3000_open_unit\(\)](#)^[14]
2. Select channel ranges and AC/DC switches using [ps3000_set_channel\(\)](#)^[18]
3. Start the oscilloscope running using [ps3000_run_streaming\(\)](#)^[35]
4. Transfer the block of data from the oscilloscope using [ps3000_get_values\(\)](#)^[38]
5. Display the data.
6. Repeat steps 4 and 5 as necessary.
7. Stop the oscilloscope using [ps3000_stop\(\)](#)^[37]

2.3.8.8 Fast streaming mode

Fast streaming mode is an advanced [streaming mode](#)^[10] 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 | Works with triggering . ^[6] |
|----------------------|--|

See [Using fast streaming mode](#)^[12] for programming details.

2.3.8.9 Using fast streaming mode

This is the general procedure for reading and displaying data in [fast streaming mode](#):^[12]

1. Open the oscilloscope using [ps3000_open_unit\(\)](#).^[14]
2. Select channel ranges and AC/DC switches using [ps3000_set_channel\(\)](#).^[18]
3. Set the trigger using [ps3000_set_trigger\(\)](#).^[24]
4. Start the oscilloscope running using [ps3000_run_streaming_ns\(\)](#).^[41]
5. Get a block of data from the oscilloscope using [ps3000_get_streaming_last_values\(\)](#).^[42]
6. Display or process the data.
7. If required, check for overview buffer overruns by calling [ps3000_overview_buffer_status\(\)](#).^[50]
8. Repeat steps 5 to 7 as necessary or until `auto_stop` is TRUE.
9. Stop fast streaming using [ps3000_stop\(\)](#).^[37]
10. Retrieve any part of the data at any time scale by calling [ps3000_get_streaming_values\(\)](#).^[45]
11. If you require raw data, retrieve it by calling [ps3000_get_streaming_values_no_aggregation\(\)](#).^[47]
12. Repeat steps 10 to 11 as necessary.
13. Close the oscilloscope by calling [ps3000_close_unit\(\)](#).^[52]

2.4 Functions

The PicoScope 3000 Series API exports the following functions for you to use in your own applications:

[ps3000_open_unit](#)^[14]
[ps3000_open_unit_async](#)^[15]
[ps3000_open_unit_progress](#)^[16]
[ps3000_get_unit_info](#)^[17]
[ps3000_set_channel](#)^[18]
[ps3000_get_timebase](#)^[19]
[ps3000_flash_led](#)^[20]
[ps3000_set_trigger](#)^[24]
[ps3000_set_trigger2](#)^[25]
[ps3000SetAdvTriggerChannelProperties](#)^[26]
[ps3000SetAdvTriggerChannelConditions](#)^[28]
[ps3000SetAdvTriggerChannelDirections](#)^[30]
[ps3000SetPulseWidthQualifier](#)^[31]
[ps3000SetAdvTriggerDelay](#)^[33]
[ps3000_run_block](#)^[34]
[ps3000_run_streaming](#)^[35]
[ps3000_ready](#)^[36]
[ps3000_stop](#)^[37]
[ps3000_get_values](#)^[38]
[ps3000_get_times_and_values](#)^[39]
[ps3000_run_streaming_ns](#)^[41]
[ps3000_get_streaming_last_values](#)^[42]
[ps3000_get_streaming_values](#)^[45]
[ps3000_get_streaming_values_no_aggregation](#)^[47]
[ps3000_save_streaming_data](#)^[49]
[ps3000_overview_buffer_status](#)^[50]
[ps3000_close_unit](#)^[52]

The following user-defined functions are also described:

[Callback function to copy data to buffer](#)^[43]
[Callback function to save data](#)^[51]

2.4.1 ps3000_open_unit

```
short ps3000_open_unit (  
    void)
```

This function opens a PicoScope 3000 Series PC Oscilloscope. The driver can support up to four 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 |

2.4.2 ps3000_open_unit_async

```
short 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\(\)](#)^[16] 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 |

2.4.3 ps3000_open_unit_progress

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

This function checks on the progress of [ps3000_open_unit_async\(\)](#)¹⁵.

| | |
|----------------------|--|
| Applicability | All modes. Use only with ps3000_open_unit_async() ¹⁵ . |
| Arguments | <code>handle</code> , 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) <code>progress_percent</code> , 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 |

2.4.4 ps3000_get_unit_info

```
short ps3000_get_unit_info (
    short  handle,
    char * string,
    short  string_length,
    short  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 | <p><code>handle</code>, the handle to the device from which info is required. If an invalid handle is passed, the error code from the last unit that failed to open is returned.</p> <p><code>string</code>, a pointer to the character string buffer in the calling function where the unit information string (selected with <code>line</code>) will be stored. If a null pointer is passed, no information will be written.</p> <p><code>string_length</code>, the length of the character string buffer. If the string is not long enough to accept all of the information, only the first <code>string_length</code> characters are returned.</p> <p><code>line</code>, an enumerated type specifying what information is required from the driver.</p> |
| Returns | <p>The length of the string written to the character string buffer, <code>string</code>, by the function</p> <p>0 if one of the parameters is out of range, or a null pointer is passed for <code>string</code></p> |

| <code>line</code> | | 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 ⁵⁷ . | "4" |

2.4.5 ps3000_set_channel

```
short ps3000_set_channel (
    short handle,
    short channel,
    short enabled,
    short dc,
    short range )
```

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

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

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

2.4.6 ps3000_get_timebase

```

short ps3000_get_timebase (
    short    handle,
    short    timebase,
    long     no_of_samples,
    long *   time_interval,
    short *  time_units,
    short    oversample,
    long *   max_samples )

```

This function discovers which timebases are available on the oscilloscope. You should set up the channels using [ps3000_set_channel\(\)](#)^[18] and, if required, [ETS](#)^[9] mode using [ps3000_set_ets\(\)](#)^[23] first.

| | |
|----------------------|--|
| Applicability | All modes. |
| Arguments | <p><code>handle</code>, the handle of the required device.</p> <p><code>timebase</code>, 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.</p> <p><code>no_of_samples</code>, the number of samples required. This value is used to calculate the most suitable time unit to use.</p> <p><code>time_interval</code>, 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.</p> <p><code>time_units</code>, 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()^[39]. If a null pointer is passed, nothing will be written here.</p> <p><code>oversample</code>, 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^[59] for more details.</p> <p><code>max_samples</code>, a pointer to the maximum samples available. The maximum samples may vary depending on the number of channels enabled, the timebase chosen and the oversample selected. If this pointer is null, nothing will be written here.</p> |
| Returns | <p>1 if all parameters are in range</p> <p>0 on error</p> |

2.4.7 ps3000_flash_led

```
short ps3000_flash_led (  
    short handle )
```

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

| | |
|----------------------|--|
| Applicability | All modes. |
| Arguments | <code>handle</code> , the handle of the PicoScope 3000 Series PC Oscilloscope. |
| Returns | 1 if a valid handle is passed 0 if handle is invalid |

2.4.8 ps3000_set_siggen

```

long ps3000_set_siggen (
    short    handle,
    short    wave_type,
    long     start_frequency,
    long     stop_frequency,
    float    increment,
    short    dwell_time,
    short    repeat,
    short    dual_slope )

```

This function is used to enable or disable the [signal generator](#) ^[5] and sweep functions.

| | |
|----------------------|---|
| Applicability | <p>Sweep functions are not available if the oscilloscope is in streaming mode ^[10].</p> <p>The signal generator is available only on the PicoScope 3204, 3205 and 3206 PC Oscilloscope variants. See remarks below for more information.</p> |
| Arguments | <p><code>handle</code>, the handle of the required device.</p> <p><code>wave_type</code>, the type of wave. Choose <code>PS3000_SQUARE (0)</code>, <code>PS3000_TRIANGLE (1)</code> or <code>PS3000_SINE (2)</code>. <i>This argument has no effect if used with the PicoScope 3204 variant.</i></p> <p><code>start_frequency</code>, the required frequency, in the range $0 < \text{freq} < 1 \text{ MHz}$, to start the sweep or the frequency generated in a non-sweep mode. 0 switches the signal generator off.</p> <p><code>stop_frequency</code>, the required stop frequency of the sweep, in the range $0 < \text{freq} < 1 \text{ MHz}$ but not necessarily greater than <code>start_frequency</code>. If the start and stop frequencies are the same, the signal generator will be run with a constant frequency. <i>This argument has no effect if used with the PicoScope 3204 variant, or if run in streaming mode.</i> ^[10]</p> <p><code>increment</code>, 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 $0.1 \text{ Hz} < \text{increment} < \text{stop_frequency} - \text{start_frequency}$. It is not used in a non-sweep mode. <i>It has no effect if used with the PicoScope 3204 variant.</i></p> <p><code>dwell_time</code>, the time, in milliseconds, to wait before increasing the frequency by <code>increment</code> in a sweep mode. This is unused in a non-sweep mode. <i>This argument has no effect if used with the PicoScope 3204 variant.</i></p> <p><code>repeat</code>, if <code>TRUE</code> restarts the sweep when the <code>stop_frequency</code> is reached, if <code>FALSE</code> continues indefinitely at <code>stop_frequency</code> when it is reached. <i>This argument has no effect if used with the PicoScope 3204 variant.</i></p> <p><code>dual_slope</code>, if <code>repeat</code> is <code>TRUE</code> this specifies what to do at the</p> |

| | |
|----------------|--|
| | <code>stop_frequency</code> . TRUE will sweep back towards the <code>start_frequency</code> , FALSE will restart the sweep from <code>start_frequency</code> . <i>This argument has no effect if used with the PicoScope 3204 variant.</i> |
| 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.

2.4.9 ps3000_set_ets

```
long ps3000_set_ets (
    short handle,
    short mode,
    short ets_cycles,
    short ets_interleave )
```

This function is used to enable or disable [ETS](#)⁹⁴ (equivalent time sampling) and to set the ETS parameters.

| | |
|----------------------|---|
| Applicability | ETS ⁹⁴ is available only on the PicoScope 3204, 3205 and 3206 variants. |
| Arguments | <p><code>handle</code>, the handle of the required device.</p> <p><code>mode</code>,</p> <p><code>PS3000_ETS_OFF (0)</code> - disables ETS.</p> <p><code>PS3000_ETS_FAST (1)</code> - enables ETS and provides <code>ets_cycles</code> cycles of data, which may contain data from previously returned cycles,</p> <p><code>PS3000_ETS_SLOW (2)</code> - enables ETS and provides fresh data every <code>ets_cycles</code> cycles. <code>PS3000_ETS_SLOW</code> takes longer to provide each data set, but the data sets are more stable and unique.</p> <p><code>ets_cycles</code>, specifies the number of cycles to store: the computer can then select <code>ets_interleave</code> cycles to give the most uniform spread of samples. <code>ets_cycles</code> should be between two and five times the value of <code>ets_interleave</code>.</p> <p><code>ets_interleave</code>, 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.</p> |
| 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 |

2.4.10 ps3000_set_trigger

```
short ps3000_set_trigger (
    short handle,
    short source,
    short threshold,
    short direction,
    short delay,
    short auto_trigger_ms )
```

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

| | |
|----------------------|---|
| Applicability | Triggering is available in block mode ^[58] and fast streaming mode ^[12] . |
| Arguments | <p><code>handle</code>, the handle to the required device.</p> <p><code>source</code>, specifies where to look for a trigger. Use <code>PS3000_CHANNEL_A (0)</code>, <code>PS3000_CHANNEL_B (1)</code>, <code>PS3000_CHANNEL_C (2)</code>, <code>PS3000_CHANNEL_D (3)</code>, <code>PS3000_EXTERNAL(4)</code> or <code>PS3000_NONE(5)</code>. The number of channels available will depend on the scope variant.</p> <p><code>threshold</code>, 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.</p> <p><code>direction</code>, use <code>PS3000_RISING (0)</code> or <code>PS3000_FALLING (1)</code>.</p> <p><code>delay</code>, 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()^[25] instead.</p> <p><code>auto_trigger_ms</code>, 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.</p> |
| Returns | <p>0 if one of the parameters is out of range</p> <p>1 if successful</p> |

2.4.11 ps3000_set_trigger2

```

short ps3000_set_trigger2 (
    short handle,
    short source,
    short threshold,
    short direction,
    float delay,
    short auto_trigger_ms )

```

This function is used to enable or disable triggering and its parameters. It has the same behaviour as [ps3000_set_trigger\(\)](#)^[24], except that the `delay` parameter is a floating-point value.

| | |
|----------------------|---|
| Applicability | Triggering is available in block mode ^[58] and fast streaming mode ^[12] only. |
| Arguments | <p><code>handle</code>, the handle of the required device.</p> <p><code>source</code>, specifies where to look for a trigger. Use <code>PS3000_CHANNEL_A (0)</code>, <code>PS3000_CHANNEL_B (1)</code>, <code>PS3000_CHANNEL_C (2)</code>, <code>PS3000_CHANNEL_D (3)</code>, <code>PS3000_EXTERNAL(4)</code> or <code>PS3000_NONE (5)</code>. Channels C, D and External are not available on all models.</p> <p><code>threshold</code>, 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.</p> <p><code>direction</code>, use <code>PS3000_RISING (0)</code> or <code>PS3000_FALLING (1)</code>.</p> <p><code>delay</code>, 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()^[24] instead.</p> <p><code>auto_trigger_ms</code>, 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.</p> |
| Returns | <p>0 if one of the parameters is out of range</p> <p>1 if successful</p> |

2.4.12 ps3000SetAdvTriggerChannelProperties

```
short ps3000SetAdvTriggerChannelProperties(  
    short          handle,  
    TRIGGER_CHANNEL_PROPERTIES * channelProperties,  
    short          nChannelProperties,  
    long           autoTriggerMilliseconds);
```

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

| | |
|----------------------|--|
| Applicability | All modes. |
| Arguments | <p><code>handle</code>, the handle of the required device.</p> <p><code>channelProperties</code>, a pointer to a <code>TRIGGER_CHANNEL_PROPERTIES</code> structure describing the requested properties. If NULL, triggering is switched off.</p> <p><code>nChannelProperties</code>, should be set to 1 if <code>channelProperties</code> is non-null, otherwise 0.</p> <p><code>autoTriggerMilliseconds</code>, 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.</p> |
| Returns | <p>0 if unsuccessful, or if one or more of the arguments are out of range</p> <p>1 if successful</p> |

2.4.12.1 TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to [ps3000SetAdvTriggerChannelProperties\(\)](#)^[26] in the `channelProperties` argument to specify the trigger mechanism, and is defined as follows: -

```
typedef struct tTriggerChannelProperties
{
    short          thresholdMajor;
    short          thresholdMinor;
    unsigned short hysteresis;
    short          channel;
    THRESHOLD_MODE thresholdMode;
} TRIGGER_CHANNEL_PROPERTIES;
```

| | |
|----------------------|---|
| Applicability | All modes. |
| Members | <p><code>thresholdMajor</code>, 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.</p> <p><code>thresholdMinor</code>, 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.</p> <p><code>hysteresis</code>, the hysteresis that the trigger has to exceed before it will fire. It is scaled in 16-bit counts.</p> <p><code>channel</code>, the channel to which the properties apply.</p> <p><code>thresholdMode</code>, either a level or window trigger. Use one of these constants: -</p> <ul style="list-style-type: none"> LEVEL (0) WINDOW(1) |

2.4.13 ps3000SetAdvTriggerChannelConditions

```
short ps3000SetAdvTriggerChannelConditions(  
    short          handle,  
    TRIGGER_CONDITIONS * conditions,  
    short          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 | All modes. |
| Arguments | <p><code>handle</code>, the handle of the required device.</p> <p><code>conditions</code>, a pointer to a <code>TRIGGER_CONDITIONS</code> structure specifying the conditions that should be applied to the current trigger channel. If NULL, triggering is switched off.</p> <p><code>nConditions</code>, should be set to 1 if conditions is non-null, otherwise 0.</p> |
| Returns | <p>0 if unsuccessful, or if one or more of the arguments are out of range</p> <p>1 if successful</p> |

2.4.13.1 TRIGGER_CONDITIONS structure

A structure of this type is passed to [ps3000SetAdvTriggerChannelConditions\(\)](#)^[28] 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 | All modes |
| Members | <p>channelA, channelB, channelC, channelD, pulseWidthQualifier: the type of condition that should be applied to each channel. Use these constants: -</p> <pre>CONDITION_DONT_CARE (0) CONDITION_TRUE (1) CONDITION_FALSE (2)</pre> <p>external, not used</p> |

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.

2.4.14 ps3000SetAdvTriggerChannelDirections

```

short ps3000SetAdvTriggerChannelDirections(
    short 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 | All modes. |
| Arguments | <p><code>handle</code>, the handle of the required device</p> <p><code>channelA</code>, <code>channelB</code>, <code>channelC</code>, <code>channelD</code>, all specify the direction in which the signal must pass through the threshold to activate the trigger. The allowable values for a <code>THRESHOLD_DIRECTION</code> variable are listed in the table below.</p> <p><code>ext</code>, not used</p> |
| Returns | <p>0 if unsuccessful, or if one or more of the arguments are out of range</p> <p>1 if successful</p> |

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 |

2.4.15 ps3000SetPulseWidthQualifier

```

short ps3000SetPulseWidthQualifier(
    short handle,
    PNQ_CONDITIONS * conditions,
    short nConditions,
    THRESHOLD_DIRECTION direction,
    unsigned long lower,
    unsigned long 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 other triggering to produce more complex triggers. The pulse width qualifier is set by defining a `conditions` structure.

| | | | | | | | | | | | |
|-----------------------------------|---|---------------------------|--------------------------------------|--------------------------------|-----------------------------|-----------------------------------|--------------------------------|-------------------------------|-------------------------------------|-----------------------------------|---|
| Applicability | All modes | | | | | | | | | | |
| Arguments | <p><code>handle</code>, the handle of the required device</p> <p><code>conditions</code>, a pointer to a <code>PWQ_CONDITIONS</code> structure specifying the conditions that should be applied to the trigger channel. If <code>conditions</code> is NULL then the pulse width qualifier is not used.</p> <p><code>nConditions</code>, should be set to 1 if <code>conditions</code> is non-null, otherwise 0.</p> <p><code>direction</code>, the direction of the signal required to trigger the pulse.</p> <p><code>lower</code>, the lower limit of the pulse width counter.</p> <p><code>upper</code>, the upper limit of the pulse width counter. This parameter is used only when the <code>type</code> is set to <code>PW_TYPE_IN_RANGE</code> or <code>PW_TYPE_OUT_OF_RANGE</code>.</p> <p><code>type</code>, the pulse width type, one of these constants: -</p> <table> <tr> <td><code>PW_TYPE_NONE</code></td><td>do not use the pulse width qualifier</td></tr> <tr> <td><code>PW_TYPE_LESS_THAN</code></td><td>pulse width less than lower</td></tr> <tr> <td><code>PW_TYPE_GREATER_THAN</code></td><td>pulse width greater than lower</td></tr> <tr> <td><code>PW_TYPE_IN_RANGE</code></td><td>pulse width between lower and upper</td></tr> <tr> <td><code>PW_TYPE_OUT_OF_RANGE</code></td><td>pulse width not between lower and upper</td></tr> </table> | <code>PW_TYPE_NONE</code> | do not use the pulse width qualifier | <code>PW_TYPE_LESS_THAN</code> | pulse width less than lower | <code>PW_TYPE_GREATER_THAN</code> | pulse width greater than lower | <code>PW_TYPE_IN_RANGE</code> | pulse width between lower and upper | <code>PW_TYPE_OUT_OF_RANGE</code> | pulse width not between lower and upper |
| <code>PW_TYPE_NONE</code> | do not use the pulse width qualifier | | | | | | | | | | |
| <code>PW_TYPE_LESS_THAN</code> | pulse width less than lower | | | | | | | | | | |
| <code>PW_TYPE_GREATER_THAN</code> | pulse width greater than lower | | | | | | | | | | |
| <code>PW_TYPE_IN_RANGE</code> | pulse width between lower and upper | | | | | | | | | | |
| <code>PW_TYPE_OUT_OF_RANGE</code> | pulse width not between lower and upper | | | | | | | | | | |
| Returns | <p>0 if unsuccessful, or if one or more of the arguments are out of range</p> <p>1 if successful</p> | | | | | | | | | | |

2.4.15.1 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 | <p><code>channelA</code>, <code>channelB</code>, <code>channelC</code>, <code>channelD</code>: the type of condition that should be applied to each channel. Use these constants: -</p> <pre>CONDITION_DONT_CARE (0) CONDITION_TRUE (1) CONDITION_FALSE (2)</pre> <p><code>external</code>, not used</p> |

2.4.16 ps3000SetAdvTriggerDelay

```
short ps3000SetAdvTriggerDelay(  
    short      handle,  
    unsigned long delay,  
    float      preTriggerDelay);
```

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

| | |
|----------------------|---|
| Applicability | All modes |
| Arguments | <p><code>handle</code>, the handle of the required device</p> <p><code>delay</code>, 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.</p> |
| Returns | <p>0 if unsuccessful, or if one or more of the arguments are out of range</p> <p>1 if successful</p> |

2.4.17 ps3000_run_block

```

short ps3000_run_block (
    short    handle,
    long     no_of_samples,
    short    timebase,
    short    oversample,
    long *   time_indisposed_ms )

```

This function tells the oscilloscope to start collecting data in [block mode](#)^[8].

| | |
|----------------------|---|
| Applicability | Block mode ^[8] only. |
| Arguments | <p><code>handle</code>, the handle to the required device.</p> <p><code>no_of_samples</code>, the number of samples to return.</p> <p><code>timebase</code>, 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.</p> <p><code>oversample</code>, the oversampling factor, a number between 1 and 256. See Oversampling^[5] for details.</p> <p><code>time_indisposed_ms</code>, 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.</p> |
| Returns | <p>0 if one of the parameters is out of range</p> <p>1 if successful</p> |

2.4.18 ps3000_run_streaming

```
short ps3000_run_streaming (
    short handle,
    short sample_interval_ms,
    long max_samples,
    short windowed )
```

This function tells the oscilloscope to start collecting data in [compatible streaming mode](#)^[11]. 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\(\)](#)^[41] instead.

| | |
|----------------------|--|
| Applicability | Compatible streaming ^[11] mode only. |
| Arguments | <p><code>handle</code>, the handle to the required device.</p> <p><code>sample_interval_ms</code>, the time interval, in milliseconds, between data points. This can be no shorter than 1 ms.</p> <p><code>max_samples</code>, 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.</p> <p><code>windowed</code>, if this is 0, only the values taken since the last call to ps3000_get_values()^[38] are returned. If this is 1, the number of values requested by ps3000_get_values()^[38] are returned, even if they have already been read by ps3000_get_values()^[38].</p> |
| Returns | <p>1 if streaming has been enabled correctly,</p> <p>0 if a problem occurred or a value was out of range.</p> |

2.4.19 ps3000_ready

```
short ps3000_ready (  
    short handle )
```

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

| | |
|----------------------|--|
| Applicability | Block mode ^[8] only. Does nothing if the oscilloscope is in streaming mode ^[10] . |
| Arguments | <code>handle</code> , the handle to the required 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. |

2.4.20 ps3000_stop

```
short ps3000_stop (  
    short handle )
```

Call this function to stop the oscilloscope sampling data. If this function is called before a trigger event occurs, the oscilloscope may not contain valid data.

| | |
|----------------------|--|
| Applicability | All modes. |
| Arguments | <code>handle</code> , the handle to the required device. |
| Returns | 0 if an invalid handle is passed 1 if successful |

2.4.21 ps3000_get_values

```

long ps3000_get_values (
    short  handle
    short * buffer_a,
    short * buffer_b,
    short * buffer_c,
    short * buffer_d,
    short * overflow,
    long   no_of_values)

```

This function is used to get values in [compatible streaming mode](#)^[11] after calling [ps3000_run_streaming\(\)](#)^[35], or in [block mode](#)^[8] after calling [ps3000_run_block\(\)](#)^[34].

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|--|----|----|----|----|----|----|----------------------|---|---|---|------------------------|---|---|---|---|---|--|--|--|--|---|---|---|---|--|--|--|--|---|---|---|---|--|--|--|--|--|--|--|--|----------------------|--|--|--|------------------------|--|--|--|
| Applicability | <p>Compatible streaming mode^[11] and block mode^[8] only.</p> <p>Does nothing if ETS^[9] triggering is enabled.</p> <p>Do not use in fast streaming mode^[12] - use ps3000_get_streaming_last_values()^[42] instead.</p> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Arguments | <p><code>handle</code>, the handle of the required device.</p> <p><code>buffer_a</code>, <code>buffer_b</code>, <code>buffer_c</code>, <code>buffer_d</code>, 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.</p> <p><code>overflow</code>, 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:</p> <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="4"></td><td>D</td><td>C</td><td>B</td><td>A</td><td colspan="4"></td><td>D</td><td>C</td><td>B</td><td>A</td></tr><tr><td colspan="8"></td><td colspan="4">common-mode overflow</td><td colspan="4">differential overflow*</td></tr></table> <p>*3425 variant only</p> <p><code>no_of_values</code>, the number of data points to return. In streaming mode, this is the maximum number of values to return.</p> | 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* | | | |
| 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* | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Returns | <p>The actual number of data values per channel returned, which may be less than <code>no_of_values</code> if streaming <code>FALSE</code> if one of the parameters is out of range</p> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

2.4.22 ps3000_get_times_and_values

```
long ps3000_get_times_and_values (
    short    handle
    long *   times,
    short *   buffer_a,
    short *   buffer_b,
    short *   buffer_c,
    short *   buffer_d,
    short *   overflow,
    short    time_units,
    long     no_of_values )
```

This function is used to get values and times in [block mode](#)^[8] after calling [ps3000_run_block\(\)](#)^[34].

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------|---|----|----|----------------------|----|----|----|---|---|---|---|------------------------|---|---|---|---|---|--|--|--|--|---|---|---|---|--|--|--|--|---|---|---|---|--|--|--|--|----------------------|--|--|--|--|--|--|--|------------------------|--|--|--|
| Applicability | <p>Block mode^[8] only. It will not return any valid times if the oscilloscope is in streaming mode^[11].</p> <p>Essential for ETS^[9] operation.</p> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Arguments | <p><code>handle</code>, the handle to the required device.</p> <p><code>times</code>, a pointer to the buffer for the times in <code>time_units</code>. 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.</p> <p><code>buffer_a</code>, <code>buffer_b</code>, <code>buffer_c</code>, <code>buffer_d</code>, 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.</p> <p><code>overflow</code>, 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:</p> <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="4"></td><td>D</td><td>C</td><td>B</td><td>A</td><td colspan="4"></td><td>D</td><td>C</td><td>B</td><td>A</td></tr><tr><td colspan="4"></td><td colspan="4">common-mode overflow</td><td colspan="4"></td><td colspan="4">differential overflow*</td></tr></table> <p>*3425 variant only</p> <p><code>time_units</code>, which can be one of: <code>PS3000_FS</code> (0, femtoseconds), <code>PS3000_PS</code> (1, picoseconds), <code>PS3000_NS</code> (2, nanoseconds, default), <code>PS3000_US</code> (3, microseconds), <code>PS3000_MS</code> (4, milliseconds) or <code>PS3000_S</code> (5, seconds).</p> <p><code>no_of_values</code>, the number of data points to return. In streaming mode, this is the maximum number of values to return.</p> | 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* | | | |
| 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* | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Returns | <p>The actual number of data values per channel returned, which may be less than <code>no_of_values</code> if streaming</p> <p>0 if one or more of the parameters are out of range or if the times will overflow with the <code>time_units</code> requested. Use ps3000_get_timebase()^[19] to acquire the most suitable <code>time_units</code>.</p> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

2.4.23 ps3000_run_streaming_ns

```

short ps3000_run_streaming_ns (
    short          handle,
    unsigned long   sample_interval,
    PS3000_TIME_UNITS time_units,
    unsigned long   max_samples,
    short          auto_stop,
    unsigned long   noOfSamplesPerAggregate,
    unsigned long   overview_buffer_size )

```

This function tells the scope unit to start collecting data in [fast streaming mode](#)^[12]. The function returns immediately without waiting for data to be captured. After calling this function, you should next call [ps3000_get_streaming_last_values\(\)](#)^[42] to copy the data to your application's buffer.

| | |
|----------------------|---|
| Applicability | Fast streaming ^[12] mode only. |
| Arguments | <p><code>handle</code>, the handle to the required device.</p> <p><code>sample_interval</code>, the time interval, in <code>time_units</code>, between data points.</p> <p><code>time_units</code>, the units in which <code>sample_interval</code> is measured.</p> <p><code>max_samples</code>, 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.</p> <p><code>auto_stop</code>, a Boolean to indicate whether streaming should stop automatically when <code>max_samples</code> is reached. Set to any non-zero value for TRUE.</p> <p><code>noOfSamplesPerAggregate</code>, the number of incoming samples that the driver will merge together (or aggregate: see aggregation^[58]) to create each value pair passed to the application. The value must be between 1 and <code>max_samples</code>.</p> <p><code>overview_buffer_size</code>, 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 ps3000_overview_buffer_status()^[50] function and adjust the overview buffer size if necessary. We recommend using an initial value of 15,000 samples.</p> |
| Returns | <p>1 if streaming has been enabled correctly,</p> <p>0 if a problem occurred or a value was out of range</p> |

2.4.24 ps3000_get_streaming_last_values

```
short ps3000_get_streaming_last_values (
    short          handle
    GetOverviewBuffersMaxMin lpGetOverviewBuffersMaxMin )
```

This function is used to collect the next block of values while [fast streaming](#)^[12] is running. You must have called [ps3000_run_streaming_ns\(\)](#)^[41] beforehand to set up fast streaming.

| | |
|----------------------|---|
| Applicability | Fast streaming ^[12] mode only. PicoScope 3224, 3424 and 3425 variants only. Not compatible with ETS ^[9] triggering - function has no effect in ETS mode. |
| Arguments | <code>handle</code> , the handle of the required device. <code>lpGetOverviewBuffersMaxMin</code> , a pointer to the callback function ^[43] 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 <code>max_samples</code> if streaming, where <code>max_samples</code> is a parameter passed to ps3000_run_streaming_ns() ^[41] . <code>FALSE</code> if one of the parameters is out of range |

2.4.25 Callback function to copy data to buffer

```
void my_get_overview_buffers (
    short          ** overviewBuffers,
    short          overflow,
    unsigned long   triggeredAt,
    short          triggered,
    short          auto_stop,
    unsigned long   nValues )
```

This is the callback function in your application that receives data from the driver in [fast streaming](#)^[12] mode. You pass a pointer to this function to [ps3000_get_streaming_last_values\(\)](#)^[42], 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\(\)](#)^[42].

For an example of a suitable callback function, see the [C++ sample code](#)^[54] included in your PicoScope installation.

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------|--|----|----|----------------------|----|----|----|---|---|---|---|------------------------|---|---|---|---|---|--|--|--|--|---|---|---|---|--|--|--|--|---|---|---|---|--|--|--|--|----------------------|--|--|--|--|--|--|--|------------------------|--|--|--|
| Applicability | Fast streaming ^[12] mode only. PicoScope 3224, 3424 and 3425 variants only. Not compatible with ETS ^[9] triggering - has no effect in ETS mode. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Arguments | <p><code>overviewBuffers</code>, a pointer to a location where ps3000_get_streaming_last_values()^[42] 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()^[41] to start fast streaming.</p> <p><code>overflow</code>, a bit field that indicates whether there has been a voltage overflow and, if so, on which channel. The bit assignments are as follows:</p> <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="4"></td><td>D</td><td>C</td><td>B</td><td>A</td><td colspan="4"></td><td>D</td><td>C</td><td>B</td><td>A</td></tr><tr><td colspan="4"></td><td colspan="4">common-mode overflow</td><td colspan="4"></td><td colspan="4">differential overflow*</td></tr></table> <p>*3425 variant only</p> <p><code>triggeredAt</code>, an index into the overview buffers, indicating the sample at the trigger event. Valid only when triggered is <code>TRUE</code>.</p> <p><code>triggered</code>, a Boolean indicating whether a trigger event has occurred and <code>triggeredAt</code> is valid. Any non-zero value signifies <code>TRUE</code>.</p> <p><code>auto_stop</code>, a Boolean indicating whether streaming data capture has automatically stopped. Any non-zero value signifies <code>TRUE</code>.</p> <p><code>nValues</code>, the number of values in each overview buffer.</p> | 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* | | | |
| 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* | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Returns | nothing | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

2.4.26 ps3000_get_streaming_values

```

unsigned long ps3000_get_streaming_values (
    short      handle,
    double     * start_time,
    short      * pbuffer_a_max,
    short      * pbuffer_a_min,
    short      * pbuffer_b_max,
    short      * pbuffer_b_min,
    short      * pbuffer_c_max,
    short      * pbuffer_c_min,
    short      * pbuffer_d_max,
    short      * pbuffer_d_min,
    short      * overflow,
    unsigned long * triggerAt,
    short      * triggered,
    unsigned long no_of_values,
    unsigned long noOfSamplesPerAggregate )

```

This function is used after the driver has finished collecting data in [fast streaming mode](#).^[12] It allows you to retrieve data with different [aggregation](#)^[58] 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\(\)](#).^[37] 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 [Scaling](#)^[5] 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 ^[12] mode only. PicoScope 3224, 3424 and 3425 variants only. Not compatible with ETS ^[9] triggering - function has no effect in ETS mode. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Arguments | <p><code>handle</code>, the handle of the required device.</p> <p><code>start_time</code>, the time in nanoseconds, relative to the trigger point, of the first data sample required.</p> <p><code>pbuffer_a_max</code>, <code>pbuffer_a_min</code>, pointers to two buffers into which the function will write the maximum and minimum aggregated sample values from channel A.</p> <p><code>pbuffer_b_max</code>, <code>pbuffer_b_min</code>, <code>pbuffer_c_max</code>, <code>pbuffer_c_min</code>, <code>pbuffer_d_max</code>, <code>pbuffer_d_min</code>, as the two parameters above but for channels B, C and D</p> <p><code>overflow</code>, where the function will write a bit field indicating whether the voltage on each of the input channels has overflowed:</p> <table border="1"><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="4"></td><td>D</td><td>C</td><td>B</td><td>A</td><td colspan="4"></td><td>D</td><td>C</td><td>B</td><td>A</td></tr><tr><td colspan="4"></td><td colspan="4">common-mode overflow</td><td colspan="4"></td><td colspan="4">differential overflow*</td></tr></table> <p>*3425 variant only</p> <p><code>triggerAt</code>, 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.</p> <p><code>triggered</code>, a pointer to a Boolean indicating that a trigger has occurred and <code>triggerAt</code> is valid.</p> <p><code>no_of_values</code>, the number of values required.</p> <p><code>noOfSamplesPerAggregate</code>, 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 <code>pbuffer_a_min</code> and the maximum value in the buffer pointed to by <code>pbuffer_a_max</code>.</p> | 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* | | | |
| 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* | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Returns | the number of values written to each buffer, if successful 0 if a parameter was out of range | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

2.4.27 ps3000_get_streaming_values_no_aggregation

```
unsigned long ps3000_get_streaming_values_no_aggregation (
    short      handle,
    double      * start_time,
    short      * pBuffer_a,
    short      * pBuffer_b,
    short      * pBuffer_c,
    short      * pBuffer_d,
    short      * overflow,
    unsigned long * triggerAt,
    short      * trigger,
    unsigned long no_of_values )
```

This function retrieves raw streaming data from the driver's data store after [fast streaming](#)^[12] has stopped.

Before calling the function, capture some data using fast streaming, stop streaming using [ps3000_stop\(\)](#),^[37] 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 [Scaling](#)^[5] for more details of data values.)

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------|---|----|----|----------------------|----|----|----|---|---|---|---|------------------------|---|---|---|---|---|--|--|--|--|---|---|---|---|--|--|--|--|---|---|---|---|--|--|--|--|----------------------|--|--|--|--|--|--|--|------------------------|--|--|--|
| Applicability | Fast streaming ^[12] mode only. PicoScope 3224, 3424 and 3425 variants only. Not compatible with ETS ^[9] triggering - has no effect in ETS mode. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Arguments | <p><code>handle</code>, the handle of the required device.</p> <p><code>start_time</code>, the time in nanoseconds of the first data sample required.</p> <p><code>pbuffer_a</code>, <code>pbuffer_b</code>, <code>pbuffer_c</code>, <code>pbuffer_d</code>, pointers to buffers into which the function will write the raw sample values from channels A, B, C and D.</p> <p><code>overflow</code>, where the function will write a bit field indicating whether the voltage on each of the input channels has overflowed:</p> <table><tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td colspan="4"></td><td>D</td><td>C</td><td>B</td><td>A</td><td colspan="4"></td><td>D</td><td>C</td><td>B</td><td>A</td></tr><tr><td colspan="4"></td><td colspan="4">common-mode overflow</td><td colspan="4"></td><td colspan="4">differential overflow*</td></tr></table> <p>*3425 variant only</p> <p><code>triggerAt</code>, where the function will write an index into the buffers. The index is the number of the the sample at the trigger reference point. Valid only when trigger is TRUE.</p> <p><code>trigger</code>, where the function will write a Boolean indicating that a trigger has occurred and <code>triggerAt</code> is valid.</p> <p><code>no_of_values</code>, the number of values required.</p> | 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* | | | |
| 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* | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Returns | the number of values written to each buffer, if successful 0 if a parameter was out of range | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

2.4.28 ps3000_save_streaming_data

```

short ps3000_save_streaming_data (
    short          handle,
    PS3000_CALLBACK_FUNC lpCallbackFunc,
    short          * dataBuffers,
    short          dataBufferSize )

```

This function sends all available streaming data to the [my_save_streaming_data\(\)](#) ^[51] callback function in your application. Your callback function decides what to do with the data.

| | |
|----------------------|---|
| Applicability | Fast streaming ^[12] mode only. PicoScope 3224, 3424 and 3425 variants only. Not compatible with ETS ^[9] triggering - function has no effect in ETS mode. |
| Arguments | <p><code>handle</code>, the handle of the required device.</p> <p><code>lpCallbackFunc</code>, a pointer to the my_save_streaming_data() ^[51] callback function in your application that handles the saving of streaming data.</p> <p><code>dataBuffers</code>, a pointer to the data.</p> <p><code>dataBufferSize</code>, the size of the buffer, in samples.</p> |
| Returns | undefined |

2.4.29 ps3000_overview_buffer_status

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

This function indicates whether or not the overview buffers used by [ps3000_run_streaming_ns\(\)](#)^[41] 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\(\)](#)^[41].

| | |
|----------------------|--|
| Applicability | Fast streaming ^[12] mode only. PicoScope 3224, 3424 and 3425 variants only. Not compatible with ETS ^[9] triggering - function has no effect in ETS mode. |
| Arguments | <code>handle</code> , the handle of the required device. <code>previous_buffer_overrun</code> , 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 was successful 1 if the function failed due to an invalid handle |

2.4.30 Callback function to save data

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

This is a callback function in your application that receives data from [ps3000_save_streaming_data\(\)](#)^[49].

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\(\)](#)^[42].

| | |
|----------------------|--|
| Applicability | Fast streaming ^[12] mode only. PicoScope 3224, 3424 and 3425 variants only. Not compatible with ETS ^[9] triggering - function has no effect in ETS mode. |
| Arguments | <code>dataBuffer</code> , a pointer to the buffer where the values are stored. <code>noOfBuffers</code> , tells your function how many buffers there are. |
| Returns | undefined |

2.4.31 ps3000_close_unit

```
short ps3000_close_unit (  
    short handle )
```

Shuts down a PicoScope 3000 Series PC Oscilloscope.

| | |
|----------------------|--|
| Applicability | All modes. |
| Arguments | <code>handle</code> , the handle, returned by ps3000_open_unit() ¹⁴ , of the oscilloscope being closed. |
| Returns | 1 if a valid handle is passed 0 if handle is not valid |

2.5 Programming examples

Programming examples are optionally available when you install PicoScope 5. Select the **Custom** installation option and then enable **Programming Examples**, selecting either the whole tree or just the examples you are interested in. There are examples for the following languages and development environments:

2.5.1 C

There are two C example programs: one is a simple GUI application, and the other is a more comprehensive console mode program that demonstrates all of the facilities of the driver.

The GUI example program is a generic Windows application - that is, it does not use Borland AppExpert or Microsoft AppWizard. To compile the program, create a new project for an Application containing the following files from the `Examples\ps3000\` subdirectory of your PicoScope installation:

- `ps3000.c`
- `ps3000.rc`

and

- `ps3000bc.lib` (Borland 32-bit applications)

or

- `ps3000.lib` (Microsoft Visual C 32-bit applications)

The following files must be in the compilation directory:

- `ps3000.rch`
- `ps3000.h`

and the following file must be in the same directory as the executable.

- `ps3000.dll`

The console example program is a generic windows application - that is, it does not use Borland AppExpert or Microsoft AppWizard. To compile the program, create a new project for an Application containing the following files:

- `ps3000con.c`

and

- `ps3000bc.lib` (Borland 32-bit applications)

or

- `ps3000.lib` (Microsoft Visual C 32-bit applications).

The following file must be in the compilation directory:

- `ps3000.h`

and the following file must be in the same directory as the executable:

- `ps3000.dll`

2.5.2 C++

The C++ example program shows how to use the [fast streaming mode](#)^[12] in the driver, with and without [triggering](#)^[6], and demonstrates the `auto_stop` feature. It runs in console mode and requires a PicoScope 3224 or 3424 scope unit..

You will need to compile the following files that are supplied in the `Examples\ps3000\` subdirectory of your PicoScope installation:

- `ps3000.h`
- `small.ico`
- `streamingTests.cpp`
- `streamingTests.ico`
- `streamingTests.rc`
- `streamingTestsResource.h` (rename to `resource.h` before compiling)

You will also need the following library for Microsoft C++:

- `ps3000.lib` (Microsoft Visual C 32-bit applications)

Ensure that the program directory contains a copy of:

- `ps3000.dll`

from the PicoScope installation directory.

A Visual Studio 2005 (VC8) project file, `faststreaming.vcproj`, is provided.

2.5.3 Visual Basic

The `Examples\ps3000\` subdirectory of your PicoScope installation contains the following files:

- `ps3000.vbp` - project file
- `ps3000.bas` - procedure prototypes
- `ps3000.frm` - form and program

Note: The functions which return a TRUE/FALSE value, return 0 for FALSE and 1 for TRUE, whereas Visual Basic expects 65,535 for TRUE. Check for `>0` rather than `=TRUE`.

2.5.4 Delphi

The program:

• `ps3000.dpr`

in the `Examples\ps3000\` subdirectory of your PicoScope installation demonstrates how to operate PicoScope 3000 Series PC Oscilloscopes. The file:

• `ps3000.inc`

contains procedure prototypes that you can include in your own programs. Other required files are:

• `ps3000fm.res`

• `ps3000fm.dfm`

• `ps3000fm.pas`

This has been tested with Delphi version 3.

2.5.5 Excel

1. Load the spreadsheet `ps3000.xls`
2. Select **Tools | Macro**
3. Select **GetData**
4. Select **Run**

Note: The Excel macro language is similar to Visual Basic. The functions which return a TRUE/FALSE value, return 0 for FALSE and 1 for TRUE, whereas Visual Basic expects 65,535 for TRUE. Check for >0 rather than =TRUE.

2.5.6 Agilent VEE

The example function `ps3000.vee` is in the `Examples\ps3000\` subdirectory of your PicoScope installation. It uses procedures that are defined in `ps3000.vh`. It was tested using Agilent VEE version 5.

2.5.7 LabView

The VI example in the `Examples\ps3000\` subdirectory of your PicoScope installation shows how to access the driver functions using LabVIEW. It was tested using version 6.1 of LabVIEW for Windows. To use the example, copy these files to your LabVIEW directory:

• `ps3000_fastStream.vi`


• `ps3000_runBlock.vi`

• `ps3000_runStream.vi`

• `ps3000wrap.c`

• `ps3000wrap.dll`

You will also need this file from the installation directory:

 ps3000.dll

2.6 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 Windows XP SP2 or Vista. |

3 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](#)^[12], the PicoScope 3000 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\(\)](#)^[41] for real-time capture, and when you call [ps3000_get_streaming_values\(\)](#)^[45] 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.

Analogue 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](#)^[58] 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.

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.

PicoScope software. This is a software product that accompanies all our oscilloscopes. It turns your PC into an oscilloscope, spectrum analyser, and meter display.

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.

Spectrum analyser. An instrument that measures the energy content of a signal in each of a large number of frequency bands. It displays the result as a graph of energy (on the vertical axis) against frequency (on the horizontal axis). The PicoScope software includes a spectrum analyser.

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 timebase controls the time interval across the scope display. There are ten divisions across the screen and the timebase is specified in units of time per division, so the total time interval is ten times the timebase.

USB 1.1. USB (Universal Serial Bus) is a standard port that enables you to connect external devices to PCs. A typical USB 1.1 port supports a data transfer rate of 12 Mbps (12 megabits per second), and is much faster than a serial port.

USB 2.0. USB (Universal Serial Bus) is a standard port that enables you to connect external devices to PCs. A typical USB 2.0 port supports a data transfer rate that is 40 times faster than that supported by USB 1.1. USB 2.0 is backwards-compatible with USB 1.1.

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

A

AC/DC control 58
 AC/DC coupling 4, 18
 Access 2
 Address 3
 Advanced triggering 26, 28, 30, 31, 33
 Aggregation 12, 41, 45
 Agilent VEE 55
 Aliasing 5, 58
 Analogue bandwidth 58
 API 13

B

Block mode 5, 6, 7, 8, 9, 34, 58
 Buffer size 58

C

C programming 53
 C++ programming 54
 Callback 43
 Channel 4, 5, 18, 24, 25
 Closing a unit 52
 Common-mode voltage 58
 Compatible streaming mode 11
 Contact details 3
 Copyright 2

D

Data acquisition 12
 Delphi programming 55
 Driver 4
 error codes 57

E

Email 3
 Error codes 57
 ETS 9, 23
 Excel macros 55
 External trigger 5, 6, 24, 25

F

Fast streaming mode 12
 Fax 3
 Fitness for purpose 2

Functions 13

ps3000_close_unit 52
 ps3000_flash_led 20
 ps3000_get_streaming_last_values 42
 ps3000_get_streaming_values 45
 ps3000_get_streaming_values_no_aggregation 47
 ps3000_get_timebase 19
 ps3000_get_times_and_values 39
 ps3000_get_unit_info 17
 ps3000_get_values 38
 ps3000_open_unit 14
 ps3000_open_unit_async 15
 ps3000_open_unit_progress 16
 ps3000_overview_buffer_status 50
 ps3000_ready 36
 ps3000_run_block 34
 ps3000_run_streaming 35
 ps3000_run_streaming_ns 41
 ps3000_save_streaming_data 49
 ps3000_set_channel 18
 ps3000_set_ets 23
 ps3000_set_siggen 21
 ps3000_set_trigger 24
 ps3000_set_trigger2 25
 ps3000_stop 37
 ps3000SetAdvTriggerChannelConditions 28
 ps3000SetAdvTriggerChannelDirections 30
 ps3000SetAdvTriggerChannelProperties 26
 ps3000SetAdvTriggerDelay 33
 ps3000SetPulseWidthQualifier 31
 save streaming data callback 51
 streaming data buffer callback 43

G

Gain 5

H

High-precision scopes 12
 High-speed sampling 7

I

Intended use 1

L

LED 20
 Legal information 2
 Liability 2

M

Macros in Excel 55
Memory in scope 8
Meter 1
Mission-critical applications 2
Multi-unit operation 6

N

Normal mode 10, 11

O

One-shot signal 9
Opening a unit 14, 15, 16
Oversampling 5
Overview buffer 50

P

PC Oscilloscope 1, 58
PC requirements 1
PicoLog software 4, 57
picopp.inf 4
picopp.sys 4
PicoScope 3000 Series 1, 6, 57
PicoScope software 1, 4, 57, 58
Pre-trigger 6
Programming
 C 53
 C++ 54
 Dephi 55
 Visual Basic 54
PWQ_CONDITIONS structure 32

R

Resolution, vertical 5, 58

S

Sampling rate 9, 58
Signal generator 5, 6, 8, 21
Spectrum analyser 1, 58
Stopping sampling 37
Streaming mode 7, 58
 compatible 11
 fast 12
 normal 10, 11
 windowed 10, 11
Support 2
Sweep 5

T

Technical assistance 3
Telephone 3
Threshold voltage 6
Time interval 5, 9
Timebase 19, 34, 58
Trademarks 2
TRIGGER_CHANNEL_PROPERTIES structure 27
TRIGGER_CONDITIONS structure 29
Triggering 6, 9, 24, 25

U

Upgrades 2
Usage 2
USB 1, 58
 hub 6

V

Vertical resolution 5
Viruses 2
Visual Basic programming 54
Voltage range 58

W

Website 3
Windowed mode 10, 11



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
www.picotech.com

ps3000pg.en-2

23.09.10

Copyright © 2007-2010 Pico Technology Limited. All rights reserved.