海洋儀器

致力于电子测试、维护领域!

PicoScope® 6000 Series

PC Oscilloscopes

Programmer's Guide

ps6000pg.en r10

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

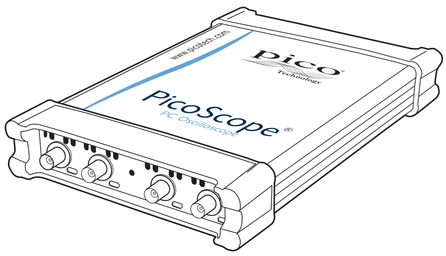
Contents

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

1 Introduction

1.1 Welcome



The **PicoScope 6000 Series** of oscilloscopes from Pico Technology is a range of compact high-performance units designed to replace traditional benchtop oscilloscopes and digitizers.

This manual explains how to use the Application Programming Interface (API) for the PicoScope 6000 Series scopes. For more information on the hardware, see the *PicoScope 6000 Series User's Guide* and *PicoScope 6000 A/B/C/D Series User's Guide* available separately.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

1.2 Software license conditions

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

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

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

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

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

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

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

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

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

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

1.3 Trademarks

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

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

Windows, **Excel** and **Visual Basic for Applications** are registered trademarks or trademarks of Microsoft Corporation in the USA and other countries. **LabVIEW** is a registered trademark of National Instruments Corporation. **MATLAB** is a registered trademark of The MathWorks, Inc.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

2 Programming overview

The `ps6000.dll` dynamic link library in the `lib` subdirectory of your Pico Technology SDK installation directory allows you to program a PicoScope 6000 Series oscilloscope using standard C function calls.

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

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

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

2.1 System requirements

Using with PicoScope for Windows

To ensure that your PicoScope 6000 Series PC Oscilloscope operates correctly, you must have a computer with at least the minimum system requirements to run one of the supported operating systems, as shown in the following table. The performance of the oscilloscope will be better with a more powerful PC, and will benefit from a multi-core processor.

Item	Specification
Operating system	Windows 7, Windows 8 or Windows 10 32-bit and 64-bit versions supported
Processor	As required by the operating system
Memory	
Free disk space	
Ports	USB 1.1 compliant port* USB 2.0 compliant port (recommended for 6000 and 6000A/B Series) USB 3.0 compliant port (recommended for 6000C/D Series)

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

Using with custom applications

32-bit and 64-bit drivers are available for Windows. The 32-bit drivers will also run in 32-bit mode on 64-bit operating systems.

USB

The PicoScope 6000 Series driver offers three different methods of recording data, all of which support USB 1.1, USB 2.0, and USB 3.0. Currently only the C and D models are able to make use of the fastest transfer rates via USB 3.0. For other models, either USB 2.0 or USB 3.0 can be used for optimal speed.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

2.2 Driver

Your application will communicate with a PicoScope 6000 API driver called `ps6000.dll`, which is supplied in 32-bit and 64-bit versions. The driver exports the PicoScope 6000 function definitions in standard C format, but this does not limit you to programming in C. You can use the API with any programming language that supports standard C calls.

The API driver depends on another DLL, `picoipp.dll`, which is supplied in 32-bit and 64-bit versions, and on a low-level driver, `WinUsb.sys`. These drivers are installed by the SDK and configured when you plug the oscilloscope into each USB port for the first time. Your application does not call these drivers directly.

2.3 Voltage ranges

Using the `ps6000SetChannel` function, you can set the oscilloscope input channels to the following voltage ranges:

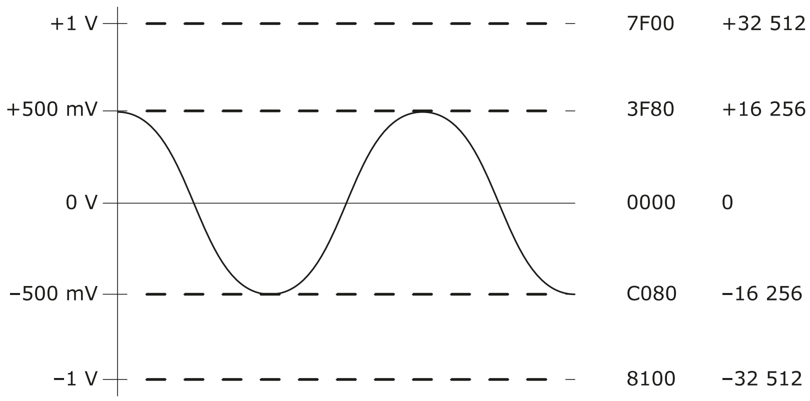
PicoScope 6407	±100 mV
All other PicoScope 6000 Series models	±50 mV to ±20 V (1 MΩ input) ±50 mV to ±5 V (50 Ω input)

Each sample is scaled to 16 bits so that the values returned to your application are as follows:

Constant	**Voltage**	**Value returned**	
		decimal	**hex**
`PS6000_MAX_VALUE`	maximum	32 512	7F00
	zero	0	0000
`PS6000_MIN_VALUE`	minimum	−32 512	8100

Example

1. Call `ps6000SetChannel` with `range` set to `PS6000_1V`.
2. Apply a sine wave input of 500 mV amplitude to the oscilloscope.
3. Capture some data using the desired sampling mode.
4. The data will be encoded as shown opposite.



Trigger thresholds for the channel inputs are also scaled as above. The AUX trigger input has a fixed range of −1 V to +1 V.

 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

2.4 Triggering

PicoScope 6000 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 trigger functions:

- `ps6000SetTriggerChannelConditions`
- `ps6000SetTriggerChannelDirections`
- `ps6000SetTriggerChannelProperties`
- `ps6000SetTriggerDelay` (optional)

These can be run collectively by calling `ps6000SetSimpleTrigger`, or singly.

A trigger event can occur when one of the input channels crosses a threshold voltage on either a rising or a falling edge. It is also possible to combine up to four inputs using the logic trigger function.

The driver supports these triggering methods:

- Simple edge
- Advanced edge
- Windowing
- Pulse width
- Logic
- Delay
- Drop-out
- Runt

The pulse width, delay and drop-out triggering methods additionally require the use of the pulse width qualifier function:

- `ps6000SetPulseWidthQualifierConditions`

2.5 Sampling modes

PicoScope 6000 Series oscilloscopes can run in various **sampling modes**.

- **Block mode.** In this mode, the scope stores data in its buffer memory and then transfers it to the PC. When the data has been collected it is possible to examine the data, with an optional downsampling factor. The data is lost when a new run is started in the same segment, the settings are changed, or the scope is powered down.

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

- **Rapid block mode.** This is a variant of block mode that allows you to capture more than one waveform at a time with a minimum of delay between captures. You can use downsampling in this mode if you wish.

- **Streaming mode.** In this mode, data is passed directly to the PC without being stored in the scope's buffer memory. This enables long periods of slow data collection for chart recorder and data-logging applications. Streaming mode also provides fast streaming at up to 13.33 MS/s (75 ns per sample) with USB 2.0 or 156.25 MS/s with USB 3.0. Downsampling and triggering are supported in this mode.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

In all sampling modes, the driver returns data asynchronously using a *callback*. This is a call to one of the functions in your own application. When you request data from the scope, you pass to the driver a pointer to your callback function. When the driver has written the data to your buffer, it makes a *callback* (calls your function) to signal that the data is ready. The callback function then signals to the application that the data is available.

Because the callback is called asynchronously from the rest of your application, in a separate thread, you must ensure that it does not corrupt any global variables while it runs.

In block mode, you can also poll the driver instead of using a callback.

2.5.1 Block mode

In **block mode**, the computer prompts a PicoScope 6000 series oscilloscope to collect a block of data into its internal memory. When the oscilloscope has collected the whole block, it signals that it is ready and then transfers the whole block to the computer's memory through the USB port.

- **Block size.** The maximum number of values depends upon the size of the oscilloscope's memory. The memory buffer is shared between the enabled channels, so if two channels are enabled, each receives half the memory. These features are handled transparently by the driver. The block size also depends on the number of memory segments in use (see `ps6000MemorySegments`).
- **Sampling rate.** A PicoScope 6000 Series oscilloscope can sample at a number of different rates according to the selected timebase and the combination of channels that are enabled. See the PicoScope 6000 Series User's Guide for the specifications that apply to your scope model.
- **Setup time.** The driver normally performs a number of setup operations, which can take up to 50 milliseconds, before collecting each block of data. If you need to collect data with the minimum time interval between blocks, use rapid block mode and avoid calling setup functions between calls to `ps6000RunBlock`, `ps6000Stop` and `ps6000GetValues`.
- **Downsampling.** When the data has been collected, you can set an optional downsampling factor and examine the data. Downsampling is a process that reduces the amount of data by combining adjacent samples. It is useful for zooming in and out of the data without having to repeatedly transfer the entire contents of the scope's buffer to the PC.
- **Memory segmentation.** The scope's internal memory can be divided into segments so that you can capture several waveforms in succession. Configure this using `ps6000MemorySegments`.
- **Data retention.** The data is lost when a new run is started in the same segment, the settings are changed, or the scope is powered down.

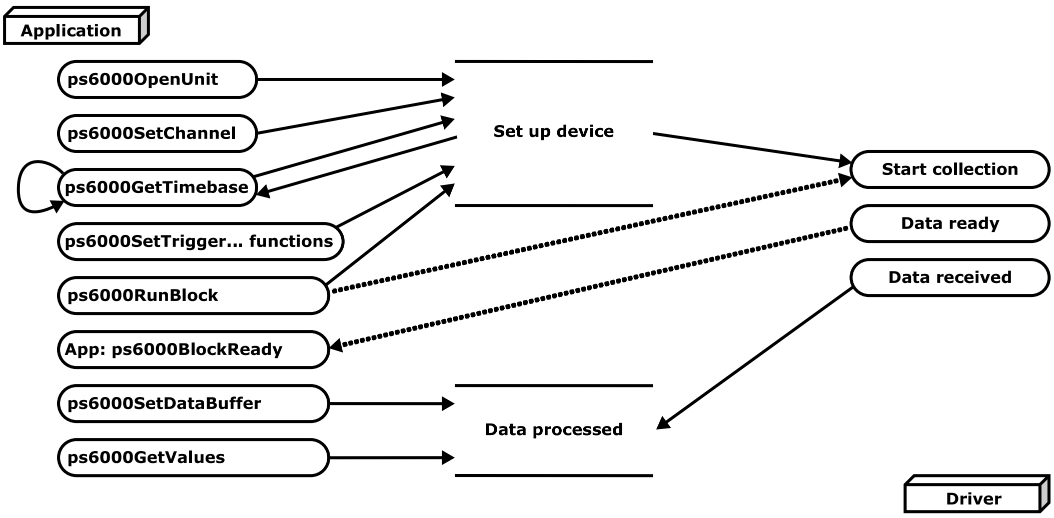
See Using block mode for programming details.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

2.5.1.1 Using block mode

This is the general procedure for reading and displaying data in block mode using a single memory segment:

1. Open the oscilloscope using `ps6000OpenUnit`.
2. Select channel ranges and AC/DC coupling using `ps6000SetChannel`.
3. Using `ps6000GetTimebase`, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions `ps6000SetTriggerChannelConditions`, `ps6000SetTriggerChannelDirections` and `ps6000SetTriggerChannelProperties` to set up the trigger if required.
5. Start the oscilloscope running using `ps6000RunBlock`.
6. Wait until the oscilloscope is ready using the `ps6000BlockReady` callback (or poll using `ps6000IsReady`).
7. Use `ps6000SetDataBuffer` to tell the driver where your memory buffer is. For greater efficiency with multiple captures, you can do this outside the loop after step 4.
8. Transfer the block of data from the oscilloscope using `ps6000GetValues`.
9. Display the data.
10. Repeat steps 5 to 9.
11. Stop the oscilloscope using `ps6000Stop`.
12. Request new views of stored data using different downsampling parameters: see Retrieving stored data.
13. Close the device using `ps6000CloseUnit`.



2.5.1.2 Asynchronous calls in block mode

The `ps6000GetValues` function may take a long time to complete if a large amount of data is being collected. For example, it can take about a minute to retrieve the full 2 billion samples from a PicoScope 6404D over a USB 2.0 connection or a few seconds over USB 3.0. To avoid hanging the calling thread, it is possible to call `ps6000GetValuesAsync` instead. This immediately returns control to the calling thread, which then has the option of waiting for the data or calling `ps6000Stop` to abort the operation.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

2.5.2 Rapid block mode

In normal block mode, the PicoScope 6000 Series scopes collect one waveform at a time. You start the device running, wait until all samples are collected by the device, and then download the data to the PC or start another run. There is a time overhead of tens of milliseconds associated with starting a run, causing a gap between waveforms. When you collect data from the device, there is another minimum time overhead which is most noticeable when using a small number of samples.

Rapid block mode allows you to sample several waveforms at a time with the minimum time between waveforms. It reduces the gap from milliseconds to less than 1 microsecond.

See Using rapid block mode for details.

2.5.2.1 Using rapid block mode

You can use **rapid block mode** with or without aggregation. With aggregation, you need to set up two buffers for each channel, to receive the minimum and maximum values.

Without aggregation

1. Open the oscilloscope using `ps6000OpenUnit`.
2. Select channel ranges and AC/DC coupling using `ps6000SetChannel`.
3. Set the number of memory segments equal to or greater than the number of captures required using `ps6000MemorySegments`. Use `ps6000SetNoOfCaptures` before each run to specify the number of waveforms to capture.
4. Using `ps6000GetTimebase`, select timebases until the required nanoseconds per sample is located.
5. Use the trigger setup functions `ps6000SetTriggerChannelConditions`, `ps6000SetTriggerChannelDirections` and `ps6000SetTriggerChannelProperties` to set up the trigger if required.
6. Start the oscilloscope running using `ps6000RunBlock`.
7. Wait until the oscilloscope is ready using the `ps6000BlockReady` callback.
8. Use `ps6000SetDataBufferBulk` to tell the driver where your memory buffers are. Call the function once for each channel/segment combination for which you require data. For greater efficiency with multiple captures, you could do this outside the loop after step 5.
9. Transfer the blocks of data from the oscilloscope using `ps6000GetValuesBulk`.
10. Retrieve the time offset for each data segment using `ps6000GetValuesTriggerTimeOffsetBulk64`.
11. Display the data.
12. Repeat steps 6 to 11 if necessary.
13. Stop the oscilloscope using `ps6000Stop`.
14. Close the device using `ps6000CloseUnit`.

With aggregation

To use rapid block mode with aggregation, follow steps 1 to 7 above and then proceed as follows:

8a. Call `ps6000SetDataBuffersBulk` to set up one pair of buffers for every waveform segment required.
9a. Call `ps6000GetValuesBulk` for each pair of buffers.
10a. Retrieve the time offset for each data segment using `ps6000GetValuesTriggerTimeOffsetBulk64`.

Continue from step 11 above.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

2.5.2.2 Rapid block mode example 1: no aggregation

```
#define MAX_WAVEFORMS 100
#define MAX_SAMPLES 1000
```

Set up the device up as usual.

- Open the device
- Channels
- Trigger
- Number of memory segments (this should be equal or more than the no of captures required)

```
// set the number of waveforms to MAX_WAVEFORMS
ps6000SetNoOfCaptures (handle, MAX_WAVEFORMS);

pParameter = false;
ps6000RunBlock
(
  handle,
  0, // noOfPreTriggerSamples
  10000, // noOfPostTriggerSamples
  1, // timebase to be used
  1, // oversample
  &timeIndisposedMs,
  0, // segment index
  lpReady,
  &pParameter
);
```

Comment: these variables have been set as an example and can be any valid value. `pParameter` will be set true by your callback function `lpReady`.

```
while (!pParameter) Sleep (0);

for (int32_t i = 0; i < 10; i++)
{
  for (int32_t c = PS6000_CHANNEL_A; c <= PS6000_CHANNEL_D; c++)
  {
    ps6000SetDataBufferBulk
    (
      handle,
      c,
      buffer[c][i],
      MAX_SAMPLES,
      i
    );
  }
}
```

Comments: buffer has been created as a two-dimensional array of pointers to `uint16_t`, which will contain 1000 samples as defined by `MAX_SAMPLES`. There are only 10 buffers set, but it is possible to set up to the number of captures you have requested.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

```
ps6000GetValuesBulk
(
  handle,
  &noOfSamples, // set to MAX_SAMPLES on entering the function
  10, // fromSegmentIndex
  19, // toSegmentIndex
  1, // downsampling ratio
  PS6000_RATIO_MODE_NONE, // downsampling ratio mode
  overflow // indices 10 to 19 will be populated
)
```

Comments: the number of samples could be up to `noOfPreTriggerSamples + noOfPostTriggerSamples`, the values set in `ps6000RunBlock`. The samples are always returned from the first sample taken, unlike the `ps6000GetValues` function which allows the sample index to be set. This function does not support aggregation. The above segments start at 10 and finish at 19 inclusive. It is possible for the `fromSegmentIndex` to wrap around to the `toSegementIndex`, by setting the `fromSegmentIndex` to 98 and the `toSegmentIndex` to 7.

```
ps6000GetValuesTriggerTimeOffsetBulk64
(
  handle,
  times,
  timeUnits,
  10,
  19
)
```

Comments: the above segments start at 10 and finish at 19 inclusive. It is possible for the `fromSegmentIndex` to wrap around to the `toSegmentIndex`, if the `fromSegmentIndex` is set to 98 and the `toSegmentIndex` to 7.

2.5.2.3 Rapid block mode example 2: using aggregation

```
#define MAX_WAVEFORMS 100
#define MAX_SAMPLES 1000
```

Set up the device up as usual.

- Open the device
- Channels
- Trigger
- Number of memory segments (this should be equal or more than the number of captures required)

```
// set the number of waveforms to MAX_WAVEFORMS
ps6000SetNoOfCaptures (handle, MAX_WAVEFORMS);

pParameter = false;
ps6000RunBlock
(
```


Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

```
    handle,
    0, //noOfPreTriggerSamples,
    1000000, // noOfPostTriggerSamples,
    1, // timebase to be used,
    1, // oversample
    &timeIndisposedMs,
    0, // segmentIndex
    lpReady,
    &pParameter
  );
```

Comments: the set-up for running the device is exactly the same whether or not aggregation will be used when you retrieve the samples.

```
  for (int32_t c = PS6000_CHANNEL_A; c <= PS6000_CHANNEL_D; c++)
  {
    ps6000SetDataBuffers
    (
      handle,
      c,
      bufferMax[c],
      bufferMin[c]
      MAX_SAMPLES,
      PS6000_RATIO_MODE_AGGREGATE
    );
  }
```

Comments: since only one waveform will be retrieved at a time, you only need to set up one pair of buffers; one for the maximum samples and one for the minimum samples. Again, the buffer sizes are 1000 samples.

```
  for (int32_t segment = 10; segment < 20; segment++)
  {
    ps6000GetValues
    (
      handle,
      0,
      &noOfSamples, // set to MAX_SAMPLES on entering
      1000,
      &downSampleRatioMode, //set to RATIO_MODE_AGGREGATE
      index,
      overflow
    );

    ps6000GetTriggerTimeOffset64
    (
      handle,
      &time,
      &timeUnits,
      index
    )
  }
```

Comments: each waveform is retrieved one at a time from the driver with an aggregation of 1000.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

2.5.3 ETS (Equivalent Time Sampling)

ETS is a way of increasing the effective sampling rate of the scope when capturing repetitive signals. It is a modified form of block mode, and is controlled by the `ps6000SetTrigger` and `ps6000SetEts` functions.

- **Overview.** ETS works by capturing several cycles of a repetitive waveform, then combining them to produce a composite waveform that has a higher effective sampling rate than the individual captures. The scope hardware accurately measures the delay, which is a small fraction of a single sampling interval, between each trigger event and the subsequent sample. The driver then shifts each capture slightly in time and overlays them so that the trigger points are exactly lined up. The result is a larger set of samples spaced by a small fraction of the original sampling interval. The maximum effective sampling rates that can be achieved with this method are listed in the data sheet for the scope device.
- **Trigger stability.** Because of the high sensitivity of ETS mode to small time differences, the trigger must be set up to provide a stable waveform that varies as little as possible from one capture to the next.
- **Callback.** ETS mode returns data to your application using the `ps6000BlockReady` callback function.

Applicability	Available in block mode only. Not suitable for one-shot (non-repetitive) signals. Aggregation and oversampling are not supported. Edge-triggering only. Auto trigger delay (`autoTriggerMilliseconds`) is ignored. Only supports timebases 0, 1 and 2.

2.5.3.1 Using ETS mode

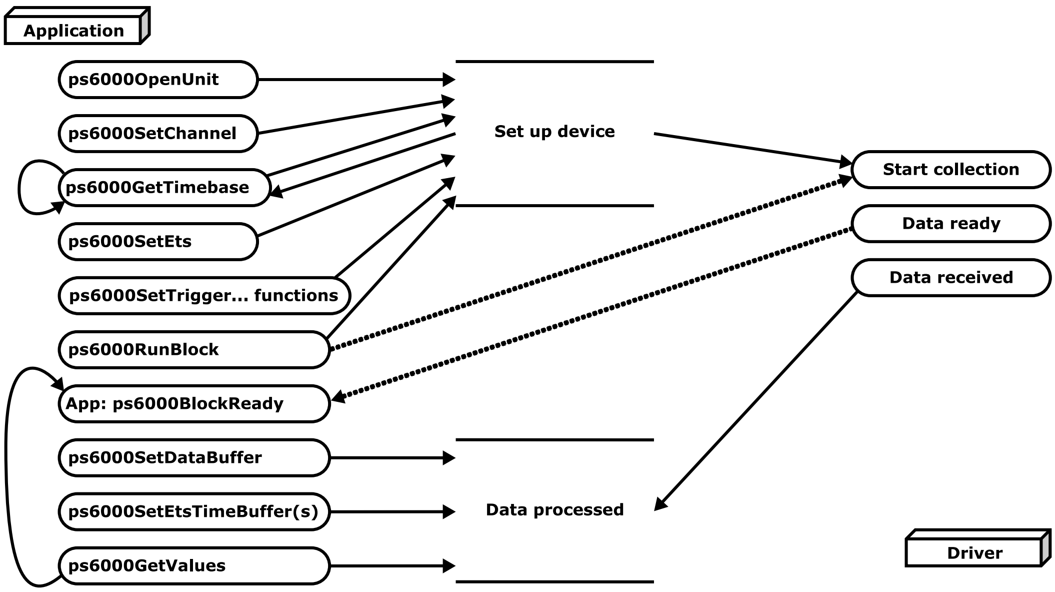
This is the general procedure for reading and displaying data in ETS mode using a single memory segment:

1. Open the oscilloscope using `ps6000OpenUnit`.
2. Select channel ranges and AC/DC coupling using `ps6000SetChannel`.
3. Use `ps6000GetTimebase` to verify the number of samples to be collected.
4. Set up ETS using `ps6000SetEts`.
5. Use the trigger setup functions `ps6000SetTriggerChannelConditions`, `ps6000SetTriggerChannelDirections` and `ps6000SetTriggerChannelProperties` to set up the trigger if required.
6. Start the oscilloscope running using `ps6000RunBlock`.
7. Wait until the oscilloscope is ready using the `ps6000BlockReady` callback (or poll using `ps6000IsReady`).
8. Use `ps6000SetDataBuffer` to tell the driver where to store sampled data.

8a. Use `ps6000SetEtsTimeBuffer` or `ps6000SetEtsTimeBuffers` to tell the driver where to store sample times.

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

9. Transfer the block of data from the oscilloscope using ps6000GetValues.
10. Display the data.
11. While you want to collect updated captures, repeat steps 7 to 10.
12. Stop the oscilloscope using ps6000Stop.
13. Repeat steps 6 to 12.
14. Close the device using ps6000CloseUnit.



2.5.4 Streaming mode

Streaming mode can capture data without the gaps that occur between blocks when using block mode.

With USB 2.0 it can transfer data to the PC at speeds of at least 13.33 million samples per second (75 nanoseconds per sample), depending on the computer's performance. With USB 3.0 this speed increases to 156.25 MS/s. This makes it suitable for **high-speed data acquisition**, allowing you to capture long data sets limited only by the computer's memory.

- **Aggregation.** The driver returns aggregated readings while the device is streaming. If aggregation is set to 1 then only one buffer is returned per channel. When aggregation is set above 1 then two buffers (maximum and minimum) per channel are returned.

- **Memory segmentation.** The memory can be divided into segments to reduce the latency of data transfers to the PC. However, this increases the risk of losing data if the PC cannot keep up with the device's sampling rate.

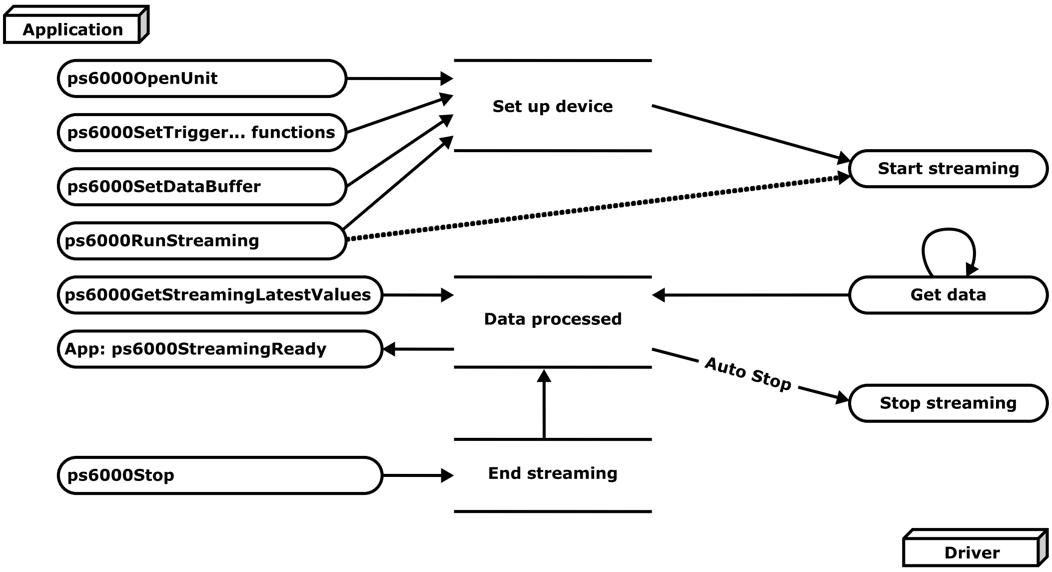
See Using streaming mode for programming details.

 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

2.5.4.1 Using streaming mode

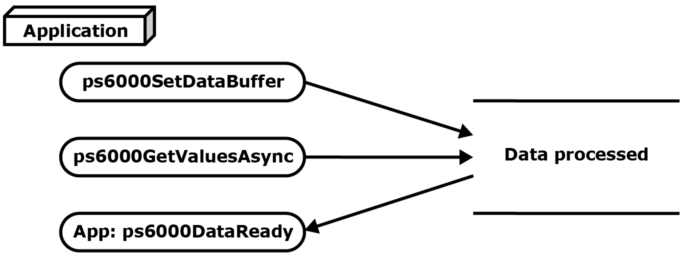
This is the general procedure for reading and displaying data in streaming mode using a single memory segment:

1. Open the oscilloscope using `ps6000OpenUnit`.
2. Select channels, ranges and AC/DC coupling using `ps6000SetChannel`.
3. Use the trigger setup functions `ps6000SetTriggerChannelConditions`, `ps6000SetTriggerChannelDirections` and `ps6000SetTriggerChannelProperties` to set up the trigger if required.
4. Call `ps6000SetDataBuffer` to tell the driver where your data buffer is.
5. Set up aggregation and start the oscilloscope running using `ps6000RunStreaming`.
6. Call `ps6000GetStreamingLatestValues` to get data.
7. Process data returned to your application's function. This example is using `autoStop`, so after the driver has received all the data points requested by the application, it stops the device streaming.
8. Call `ps6000Stop`, even if `autoStop` is enabled.
9. Request new views of stored data using different downsampling parameters: see Retrieving stored data.
10. Close the device using `ps6000CloseUnit`.



2.5.5 Retrieving stored data

You can collect data from the PicoScope 6000 driver with a different downsampling factor when `ps6000RunBlock` or `ps6000RunStreaming` has already been called and has successfully captured all the data. Use `ps6000GetValuesAsync`.



Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

2.6 Oversampling

Note: This feature is provided for backward compatibility only. The same effect can be obtained more efficiently with the PicoScope 6000 Series using the hardware averaging feature (see Downsampling modes).

When the oscilloscope is operating at sampling rates less than its maximum, it is possible to **oversample**. Oversampling is taking more than one measurement during a time interval and returning the average as one sample. The number of measurements per sample is called the oversampling factor. If the signal contains a small amount of wideband noise (strictly speaking, *Gaussian noise*), this technique can increase the effective vertical resolution of the oscilloscope by *n* bits, where *n* is given approximately by the equation below:

$$n = \log(\text{oversampling factor}) / \log 4$$

Conversely, for an improvement in resolution of *n* bits, the oversampling factor you need is given approximately by:

$$\text{oversampling factor} = 4^n$$

An oversample of 4, for example, would quadruple the time interval and quarter the maximum samples, and at the same time would increase the effective resolution by one bit.

Applicability	Available in block mode only. Cannot be used at the same time as downsampling.

2.7 Timebases

The API allows you to select any of 2^{32} different timebases based on a maximum sampling rate of 5 GHz. The timebases allow slow enough sampling in block mode to overlap the streaming sample intervals, so that you can make a smooth transition between block mode and streaming mode.

timebase	sample interval formula	sample interval examples
0 to 4	$2^{timebase}$ / 5 000 000 000	0 => 200 ps 1 => 400 ps 2 => 800 ps 3 => 1.6 ns 4 => 3.2 ns
5 to $2^{32}-1$	(timebase–4) / 156 250 000	5 => 6.4 ns ... $2^{32}-1$ => ~ 6.87 s

Applicability	Call either `ps6000GetTimebase` or `ps6000GetTimebase2`. Note that `ps6000GetTimebase` should not be used for timebases 0, 1 or 2. ETS mode only supports timebases 0, 1 and 2: see `ps6000SetEts` for more information.

Notes

1. The maximum possible sampling rate may depend on the number of enabled channels and on the sampling mode: please refer to the data sheet for details.
2. In streaming mode, the speed of the USB port may affect the rate of data transfer.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

2.8 Combining several oscilloscopes

It is possible to collect data using up to 64 PicoScope 6000 Series oscilloscopes at the same time, depending on the capabilities of the PC. Each oscilloscope must be connected to a separate USB port. The `ps6000OpenUnit` 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:

```
CALLBACK ps6000BlockReady(...)
// define callback function specific to application

handle1 = ps6000OpenUnit
handle2 = ps6000OpenUnit

ps6000SetChannel(handle1)
// set up unit 1
ps6000RunBlock(handle1)

ps6000SetChannel(handle2)
// set up unit 2
ps6000RunBlock(handle2)

// data will be stored in buffers
// and application will be notified using callback

ready = FALSE
while not ready
   ready = handle1_ready
   ready &= handle2_ready
```

Note: an external clock may be fed into the AUX input to provide some degree of synchronization between multiple oscilloscopes.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3 API functions

The PicoScope 6000 Series API exports the following functions for you to use in your own applications. All functions are C functions using the standard call naming convention (`__stdcall`). They are all exported with both decorated and undecorated names.

ps6000BlockReady	indicate when block-mode data ready
ps6000CloseUnit	close a scope device
ps6000DataReady	indicate when post-collection data ready
ps6000EnumerateUnits	find all connected oscilloscopes
ps6000FlashLed	flash the front-panel LED
ps6000GetAnalogueOffset	get min/max allowable analog offset
ps6000GetMaxDownSampleRatio	find out aggregation ratio for data
ps6000GetStreamingLatestValues	get streaming data while scope is running
ps6000GetTimebase	find out what timebases are available
ps6000GetTimebase2	find out what timebases are available
ps6000GetTriggerTimeOffset	find out when trigger occurred (32-bit)
ps6000GetTriggerTimeOffset64	find out when trigger occurred (64-bit)
ps6000GetUnitInfo	read information about scope device
ps6000GetValues	get block-mode data with callback
ps6000GetValuesAsync	get streaming data with callback
ps6000GetValuesBulk	get data in rapid block mode
ps6000GetValuesBulkAsync	get data in rapid block mode using callback
ps6000GetValuesOverlapped	set up data collection ahead of capture
ps6000GetValuesOverlappedBulk	set up data collection in rapid block mode
ps6000GetValuesTriggerTimeOffsetBulk	get rapid-block waveform timings (32-bit)
ps6000GetValuesTriggerTimeOffsetBulk64	get rapid-block waveform timings (64-bit)
ps6000IsReady	poll driver in block mode
ps6000IsTriggerOrPulseWidthQualifierEnabled	find out whether trigger is enabled
ps6000MemorySegments	divide scope memory into segments
ps6000NoOfStreamingValues	get number of samples in streaming mode
ps6000OpenUnit	open a scope device
ps6000OpenUnitAsync	open a scope device without waiting
ps6000OpenUnitProgress	check progress of OpenUnit call
ps6000RunBlock	start block mode
ps6000RunStreaming	start streaming mode
ps6000SetChannel	set up input channels
ps6000SetDataBuffer	register data buffer with driver
ps6000SetDataBufferBulk	set the buffers for each waveform
ps6000SetDataBuffers	register aggregated data buffers with driver
ps6000SetDataBuffersBulk	register data buffers for rapid block mode
ps6000SetEts	set up equivalent-time sampling
ps6000SetEtsTimeBuffer	set up buffer for ETS timings (64-bit)
ps6000SetEtsTimeBuffers	set up buffer for ETS timings (32-bit)
ps6000SetExternalClock	set AUX input to receive external clock
ps6000SetNoOfCaptures	set number of captures to collect in one run
ps6000SetPulseWidthQualifier	set up pulse width triggering
ps6000SetSigGenArbitrary	set up arbitrary waveform generator
ps6000SetSigGenBuiltIn	set up signal generator
ps6000SetSigGenBuiltInV2	set up signal generator (double precision)
ps6000SetSimpleTrigger	set up level triggers only
ps6000SetTriggerChannelConditions	specify which channels to trigger on
ps6000SetTriggerChannelDirections	set up signal polarities for triggering
ps6000SetTriggerChannelProperties	set up trigger thresholds
ps6000SetTriggerDelay	set up post-trigger delay
ps6000SigGenArbitraryMinMaxValues	get limits for AWG settings
ps6000SigGenFrequencyToPhase	calculate delta phase parameter for AWG setup
ps6000SigGenSoftwareControl	trigger the signal generator
ps6000Stop	stop data capture
ps6000StreamingReady	indicate when streaming-mode data ready

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

3.1 ps6000BlockReady

```
typedef void (CALLBACK *ps6000BlockReady)
(
  int16_t        handle,
  PICO_STATUS    status,
  void         * pParameter
)
```

This callback function is part of your application. You register it with the PicoScope 6000 Series driver using `ps6000RunBlock`, and the driver calls it back when block-mode data is ready. You can then download the data using the `ps6000GetValues` function.

Applicability	Block mode only
Arguments	`handle`, identifies the device `status`, indicates whether an error occurred during collection of the data. `pParameter`, a void pointer passed from `ps6000RunBlock`. Your callback function can write to this location to send any data, such as a status flag, back to your application.
Returns	nothing

3.2 ps6000CloseUnit

```
PICO_STATUS ps6000CloseUnit
(
  int16_t    handle
)
```

This function shuts down a PicoScope 6000 Series oscilloscope.

Applicability	All modes
Arguments	`handle`, the identifier, returned by `ps6000OpenUnit`, of the scope device to be closed.

Returns	`PICO_OK`
	`PICO_HANDLE_INVALID`
	`PICO_USER_CALLBACK`
	`PICO_DRIVER_FUNCTION`

3.3 ps6000DataReady

```
typedef void (CALLBACK *ps6000DataReady)
(
  int16_t        handle,
  PICO_STATUS    status,
  uint32_t       noOfSamples,
  int16_t        overflow,
  void         * pParameter
)
```

This is a callback function that you write to collect data from the driver. You supply a pointer to the function when you call `ps6000GetValuesAsync`, and the driver calls your function back when the data is ready.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

Applicability	All modes
Arguments	handle, identifies the device status, a PICO_STATUS code returned by the driver. noOfSamples, the number of samples collected. overflow, a set of flags that indicates whether an overvoltage has occurred and on which channels. It is a bit field with bit 0 representing Channel A. pParameter, a void pointer passed from ps6000GetValuesAsync. The callback function can write to this location to send any data, such as a status flag, back to the application. The data type is defined by the application programmer.
Returns	nothing

3.4 ps6000EnumerateUnits

```
PICO_STATUS ps6000EnumerateUnits
(
  int16_t   * count,
  int8_t    * serials,
  int16_t   * serialLth
)
```

This function counts the number of PicoScope 6000 units connected to the computer, and returns a list of serial numbers as a string. Note that this function will only detect devices that are not yet being controlled by an application.

Applicability	All modes
Arguments	* count, on exit, the number of PicoScope 6000 units found * serials, on exit, a list of serial numbers separated by commas and terminated by a final null. Example: AQ005/139,VDR61/356,ZOR14/107. Can be NULL on entry if serial numbers are not required. * serialLth, on entry, the length of the int8_t buffer pointed to by serials; on exit, the length of the string written to serials
Returns	PICO_OK PICO_BUSY PICO_NULL_PARAMETER PICO_FW_FAIL PICO_CONFIG_FAIL PICO_MEMORY_FAIL PICO_ANALOG_BOARD PICO_CONFIG_FAIL_AWG PICO_INITIALISE_FPGA

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.5 ps6000FlashLed

```
PICO_STATUS ps6000FlashLed
(
  int16_t    handle,
  int16_t    start
)
```

This function flashes the LED on the front of the scope without blocking the calling thread. Calls to ps6000RunStreaming and ps6000RunBlock cancel any flashing started by this function. It is not possible to set the LED to be constantly illuminated, as this state is used to indicate that the scope has not been initialized.

Applicability	All modes
Arguments	handle, identifies the device start, the action required: < 0 : flash the LED indefinitely. 0 : stop the LED flashing. > 0 : flash the LED start times. If the LED is already flashing on entry to this function, the flash count will be reset to start.
Returns	PICO_OK PICO_HANDLE_INVALID PICO_BUSY PICO_DRIVER_FUNCTION PICO_NOT_RESPONDING

 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.6 ps6000GetAnalogueOffset

```
PICO_STATUS ps6000GetAnalogueOffset
(
  int16_t              handle,
  PS6000_RANGE         range
  PS6000_COUPLING      coupling
  float              * maximumVoltage,
  float              * minimumVoltage
)
```

This function is used to get the maximum and minimum allowable analog offset for a specific voltage range.

Applicability	Not PicoScope 6407
Arguments	handle, identifies the device range, the voltage range for which minimum and maximum voltages are required coupling, the type of AC/DC coupling used * maximumVoltage, on output, the maximum analog offset voltage allowed for the range. Set to NULL if not required. * minimumVoltage, on output, the minimum analog offset voltage allowed for the range. Set to NULL if not required.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_INVALID_VOLTAGE_RANGE PICO_NULL_PARAMETER (if both maximumVoltage and minimumVoltage are NULL)

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.7 ps6000GetMaxDownSampleRatio

```
PICO_STATUS ps6000GetMaxDownSampleRatio
(
  int16_t               handle,
  uint32_t              noOfUnaggregatedSamples,
  uint32_t            * maxDownSampleRatio,
  PS6000_RATIO_MODE     downSampleRatioMode,
  uint32_t              segmentIndex
)
```

This function returns the maximum downsampling ratio that can be used for a given number of samples in a given downsampling mode.

Applicability	All modes
Arguments	`handle`, identifies the device `noOfUnaggregatedSamples`, the number of unprocessed samples to be downsampled `maxDownSampleRatio`, the maximum possible downsampling ratio `downSampleRatioMode`, the downsampling mode. See `ps6000GetValues`. `segmentIndex`, the memory segment where the data is stored
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_TOO_MANY_SAMPLES

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.8 ps6000GetNoOfCaptures

```
PICO_STATUS ps6000GetNoOfCaptures
(
  int16_t     handle,
  uint32_t  * nCaptures
)
```

This function returns the number of captures collected in one run of rapid block mode. You can call this function during device capture, after collection has completed or after interrupting waveform collection by calling ps6000Stop.

The returned value (nCaptures) can then be used to iterate through the number of segments using ps6000GetValues, or in a single call to ps6000GetValuesBulk where it is used to calculate the toSegmentIndex parameter.

Applicability	All modes
Arguments	handle, identifies the device nCaptures, on output, the number of available captures that has been collected from calling ps6000RunBlock
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_TOO_MANY_SAMPLES

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

3.9 ps6000GetNoOfProcessedCaptures

```
PICO_STATUS ps6000GetNoOfProcessedCaptures
(
  int16_t     handle,
  uint32_t  * nProcessedCaptures
)
```

This function gets the number of captures collected and processed in one run of rapid block mode. It enables your application to start processing captured data while the driver is still transferring later captures from the device to the computer.

The function returns the number of captures the driver has processed since you called `ps6000RunBlock`. It is for use in rapid block mode, alongside the `ps6000GetValuesOverlappedBulk` function, when the driver is set to transfer data from the device automatically as soon as the `ps6000RunBlock` function is called. You can call `ps6000GetNoOfProcessedCaptures` during device capture, after collection has completed or after interrupting waveform collection by calling `ps6000Stop`.

The returned value (`nProcessedCaptures`) can then be used to iterate through the number of segments using `ps6000GetValues`, or in a single call to `ps6000GetValuesBulk`, where it is used to calculate the `toSegmentIndex` parameter.

When capture is stopped

If `nProcessedCaptures` = 0, you will also need to call `ps6000GetNoOfCaptures`, in order to determine how many waveform segments were captured, before calling `ps6000GetValues` or `ps6000GetValuesBulk`.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the device. `* nProcessedCaptures`, on exit, the number of waveforms captured and processed.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.10 ps6000GetStreamingLatestValues

```
PICO_STATUS ps6000GetStreamingLatestValues
(
  int16_t                  handle,
  ps6000StreamingReady     lpPs6000Ready,
  void                   * pParameter
)
```

This function instructs the driver to return the next block of values to your ps6000StreamingReady callback function. You must have previously called ps6000RunStreaming beforehand to set up streaming.

Applicability	Streaming mode only
Arguments	handle, identifies the device lpPs6000Ready, a pointer to your ps6000StreamingReady callback function pParameter, a void pointer that will be passed to the ps6000StreamingReady callback function. The callback function may optionally use this pointer to return information to the application.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_INVALID_CALL PICO_BUSY PICO_NOT_RESPONDING PICO_DRIVER_FUNCTION PICO_STARTINDEX_INVALID

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

3.11 ps6000GetTimebase

```
PICO_STATUS ps6000GetTimebase
(
  int16_t      handle,
  uint32_t     timebase,
  uint32_t     noSamples,
  int32_t    * timeIntervalNanoseconds,
  int16_t      oversample,
  uint32_t   * maxSamples
  uint32_t     segmentIndex
)
```

This function calculates the sampling rate and maximum number of samples for a given timebase under the specified conditions. The result will depend on the number of channels enabled by the last call to `ps6000SetChannel`.

This function is provided for use with programming languages that do not support the `float` data type. The value returned in the `timeIntervalNanoseconds` argument is restricted to integers. If your programming language supports the `float` type, then we recommend that you use `ps6000GetTimebase2` instead.

To use `ps6000GetTimebase` or `ps6000GetTimebase2`, first estimate the timebase number that you require using the information in the timebase guide. Pass this timebase to the GetTimebase function and check the returned `timeIntervalNanoseconds` argument. If necessary, repeat until you obtain the time interval that you need.

Note that `ps6000GetTimebase` should not be called for timebases 0, 1 or 2, as they will return values smaller than 1 nanosecond.

 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

Applicability	All modes.
Arguments	`handle`, identifies the device. `timebase`, see timebase guide. In ETS mode the driver selects its own timebase and this argument is ignored. `noSamples`, the number of samples required. This value is used to calculate the most suitable time interval. `timeIntervalNanoseconds`, on exit, the time interval between readings at the selected timebase. Use `NULL` if not required. In ETS mode this argument is not valid; use the sample time returned by `ps6000SetEts` instead. `oversample`, the amount of oversample required. Range: 0 to `PS6000_MAX_OVERSAMPLE_8BIT`. `maxSamples`, on exit, the maximum number of samples available. The scope allocates a certain amount of memory for internal overheads and this may vary depending on the number of segments, number of channels enabled, and the timebase chosen. Use `NULL` if not required. `segmentIndex`, the index of the memory segment to use.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_TOO_MANY_SAMPLES`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_TIMEBASE`
	`PICO_INVALID_PARAMETER`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_DRIVER_FUNCTION`

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.12 ps6000GetTimebase2

```
PICO_STATUS ps6000GetTimebase2
(
  int16_t      handle,
  uint32_t     timebase,
  uint32_t     noSamples,
  float      * timeIntervalNanoseconds,
  int16_t      oversample,
  uint32_t   * maxSamples
  uint32_t     segmentIndex
)
```

This function is an upgraded version of ps6000GetTimebase, and returns the time interval as a float rather than a uint32_t. This allows it to return sub-nanosecond time intervals. See ps6000GetTimebase for a full description.

Note that ps6000GetTimebase should not be called for timebases 0, 1 or 2, as they will return values smaller than 1 nanosecond.

Applicability	All modes
Arguments	timeIntervalNanoseconds, a pointer to the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. All other arguments: see ps6000GetTimebase
Returns	See ps6000GetTimebase

 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.13 ps6000GetTriggerTimeOffset

```
PICO_STATUS ps6000GetTriggerTimeOffset
(
  int16_t             handle
  uint32_t          * timeUpper
  uint32_t          * timeLower
  PS6000_TIME_UNITS * timeUnits
  uint32_t            segmentIndex
)
```

This function gets the trigger time offset for waveforms obtained in block mode or rapid block mode. The trigger time offset is an adjustment value used for correcting jitter in the waveform, and is intended mainly for applications that wish to display the waveform with reduced jitter. The offset is zero if the waveform crosses the threshold at the trigger sampling instant, or a positive or negative value if jitter correction is required. The value should be added to the nominal trigger time to get the corrected trigger time.

Call this function after data has been captured or when data has been retrieved from a previous capture.

This function is provided for use in programming environments that do not support 64-bit integers. Another version of this function, `ps6000GetTriggerTimeOffset64`, is available that returns the time as a single 64-bit value.

Applicability	Block mode, rapid block mode

Arguments	`handle`, identifies the device
	
	`timeUpper`, on exit, the upper 32 bits of the time at which the trigger point occurred
	
	`timeLower`, on exit, the lower 32 bits of the time at which the trigger point occurred
	
	`timeUnits`, returns the time units in which `timeUpper` and `timeLower` are measured. The allowable values are:
	`PS6000_FS`
	`PS6000_PS`
	`PS6000_NS`
	`PS6000_US`
	`PS6000_MS`
	`PS6000_S`
	
	`segmentIndex`, the number of the memory segment for which the information is required.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_DEVICE_SAMPLING`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NULL_PARAMETER`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_DRIVER_FUNCTION`

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

3.14 ps6000GetTriggerTimeOffset64

```
PICO_STATUS ps6000GetTriggerTimeOffset64
(
  int16_t              handle,
  int64_t            * time,
  PS6000_TIME_UNITS  * timeUnits,
  uint32_t             segmentIndex
)
```

This function gets the trigger time offset for a waveform. It is equivalent to `ps6000GetTriggerTimeOffset` except that the time offset is returned as a single 64-bit value instead of two 32-bit values.

Applicability	Block mode, rapid block mode

Arguments	`handle`, identifies the device
	
	`time`, on exit, the time at which the trigger point occurred
	
	`timeUnits`, on exit, the time units in which time is measured. The possible values are:
	`PS6000_FS`
	`PS6000_PS`
	`PS6000_NS`
	`PS6000_US`
	`PS6000_MS`
	`PS6000_S`
	
	`segmentIndex`, the number of the memory segment for which the information is required

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_DEVICE_SAMPLING`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NULL_PARAMETER`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_DRIVER_FUNCTION`

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.15 ps6000GetUnitInfo

```
PICO_STATUS ps6000GetUnitInfo
(
  int16_t      handle,
  int8_t     * string,
  int16_t      stringLength,
  int16_t    * requiredSize
  PICO_INFO    info
)
```

This function retrieves information about the specified oscilloscope. If the device fails to open, only the driver version and error code are available to explain why the last open unit call failed.

Applicability	All modes
Arguments	handle, identifies the device from which information is required. If an invalid handle is passed, the error code from the last unit that failed to open is returned. string, on exit, the unit information string selected specified by the info argument. If string is NULL, only requiredSize is returned. stringLength, the maximum number of int8_t values that may be written to string. requiredSize, on exit, the required length of the string array. info, a number specifying what information is required. The possible values are listed in the table below.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_INVALID_INFO PICO_INFO_UNAVAILABLE PICO_DRIVER_FUNCTION

info		**Example**
0	PICO_DRIVER_VERSION - Version number of PicoScope 6000 DLL	1,0,0,1
1	PICO_USB_VERSION - Type of USB connection to device: 1.1, 2.0 or 3.0	3.0
2	PICO_HARDWARE_VERSION - Hardware version of device	1
3	PICO_VARIANT_INFO - Model number of device	6403
4	PICO_BATCH_AND_SERIAL - Batch and serial number of device	KJL87/6
5	PICO_CAL_DATE - Calibration date of device	30Sep09
6	PICO_KERNEL_VERSION - Version of kernel driver	1,1,2,4
7	PICO_DIGITAL_HARDWARE_VERSION - Hardware version of the digital section	1
8	PICO_ANALOGUE_HARDWARE_VERSION - Hardware version of the analog section	1
9	PICO_FIRMWARE_VERSION_1 - Version information of Firmware 1	1,0,0,1
A	PICO_FIRMWARE_VERSION_2 - Version information of Firmware 2	1,0,0,1

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.16 ps6000GetValues

```
PICO_STATUS ps6000GetValues
(
  int16_t               handle,
  uint32_t              startIndex,
  uint32_t            * noOfSamples,
  uint32_t              downSampleRatio,
  PS6000_RATIO_MODE     downSampleRatioMode,
  uint32_t              segmentIndex,
  int16_t             * overflow
)
```

This function returns block-mode data, with downsampling if requested, starting at the specified sample number. It is used to get the stored data from the oscilloscope after data collection has stopped.

Applicability	Block mode, rapid block mode

Arguments	`handle`, identifies the device.
	
	`startIndex`, a zero-based index that indicates the start point for data collection. It is measured in sample intervals from the start of the buffer.
	
	`noOfSamples`, on entry, the number of samples required. On exit, the actual number retrieved. The number of samples retrieved will not be more than the number requested, and the data retrieved always starts with the first sample captured.
	
	`downSampleRatio`, the downsampling factor that will be applied to the raw data. Must be greater than zero.
	
	`downSampleRatioMode`, which downsampling mode to use. The available values are:
	`PS6000_RATIO_MODE_NONE` (`downSampleRatio` is ignored)
	`PS6000_RATIO_MODE_AGGREGATE`
	`PS6000_RATIO_MODE_AVERAGE`
	`PS6000_RATIO_MODE_DECIMATE`
	
	`PS6000_RATIO_MODE_AGGREGATE`, `PS6000_RATIO_MODE_AVERAGE`, and `PS6000_RATIO_MODE_DECIMATE` are single-bit constants that can be ORed to apply multiple downsampling modes to the same data.
	
	`segmentIndex`, the zero-based number of the memory segment where the data is stored.
	
	`overflow`, on exit, a set of flags that indicate whether an overvoltage has occurred on any of the channels. It is a bit field with bit 0 denoting Channel A.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_DEVICE_SAMPLING PICO_NULL_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_PARAMETER PICO_TOO_MANY_SAMPLES PICO_DATA_NOT_AVAILABLE PICO_STARTINDEX_INVALID PICO_INVALID_SAMPLERATIO PICO_INVALID_CALL PICO_NOT_RESPONDING PICO_MEMORY PICO_RATIO_MODE_NOT_SUPPORTED PICO_DRIVER_FUNCTION

3.16.1 Downsampling modes

Various methods of data reduction, or **downsampling**, are possible with the PicoScope 6000 Series oscilloscopes. The downsampling is done at high speed by dedicated hardware inside the scope, making your application faster and more responsive than if you had to do all the data processing in software.

You specify the downsampling mode when you call one of the data collection functions, such as ps6000GetValues. The following modes are available:

PS6000_RATIO_MODE_NONE	No downsampling. Returns the raw data values.
PS6000_RATIO_MODE_AGGREGATE	Reduces every block of *n* values to just two values: a minimum and a maximum. The minimum and maximum values are returned in two separate buffers.
PS6000_RATIO_MODE_AVERAGE	Reduces every block of *n* values to a single value representing the average (arithmetic mean) of all the values.
PS6000_RATIO_MODE_DECIMATE	Reduces every block of *n* values to just the first value in the block, discarding all the other values.
PS6000_RATIO_MODE_DISTRIBUTION	Not implemented.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.17 ps6000GetValuesAsync

```
PICO_STATUS ps6000GetValuesAsync
(
  int16_t                handle,
  uint32_t               startIndex,
  uint32_t               noOfSamples,
  uint32_t               downSampleRatio,
  PS6000_RATIO_MODE      downSampleRatioMode,
  uint32_t               segmentIndex,
  void                 * lpDataReady,
  void                 * pParameter
)
```

This function returns data, with downsampling if requested, starting at the specified sample number. In streaming mode it retrieves stored data from the driver after data collection has stopped. In block mode it retrieves data from the oscilloscope. It returns the data using a callback.

Applicability	Streaming mode and block mode
Arguments	`handle`, `startIndex`, `noOfSamples`, `downSampleRatio`, `downSampleRatioMode`, `segmentIndex`: see `ps6000GetValues` `lpDataReady`, a pointer to the user-supplied function that will be called when the data is ready. This will be a `ps6000DataReady` function for block-mode data or a `ps6000StreamingReady` function for streaming-mode data. `pParameter`, a void pointer that will be passed to the callback function. The data type is determined by the application.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_DEVICE_SAMPLING`
	`PICO_NULL_PARAMETER`
	`PICO_STARTINDEX_INVALID`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_INVALID_PARAMETER`
	`PICO_DATA_NOT_AVAILABLE`
	`PICO_INVALID_SAMPLERATIO`
	`PICO_INVALID_CALL`
	`PICO_DRIVER_FUNCTION`

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.18 ps6000GetValuesBulk

```
PICO_STATUS ps6000GetValuesBulk
(
  int16_t              handle,
  uint32_t           * noOfSamples,
  uint32_t             fromSegmentIndex,
  uint32_t             toSegmentIndex,
  uint32_t             downSampleRatio,
  PS6000_RATIO_MODE    downSampleRatioMode,
  int16_t            * overflow
)
```

This function retrieves waveforms captured using rapid block mode. The waveforms must have been collected sequentially and in the same run.

Applicability	Rapid block mode
Arguments	handle, identifies the device * noOfSamples, on entry, the number of samples required; on exit, the actual number retrieved. The number of samples retrieved will not be more than the number requested. The data retrieved always starts with the first sample captured. fromSegmentIndex, the first segment from which the waveform should be retrieved toSegmentIndex, the last segment from which the waveform should be retrieved downSampleRatio, downSampleRatioMode: see ps6000GetValues * overflow, an array of integers equal to or larger than the number of waveforms to be retrieved. Each segment index has a corresponding entry in the overflow array, with overflow[0] containing the flags for the segment numbered fromSegmentIndex and the last element in the array containing the flags for the segment numbered toSegmentIndex. Each element in the array is a bit field as described under ps6000GetValues.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE PICO_STARTINDEX_INVALID PICO_NOT_RESPONDING PICO_DRIVER_FUNCTION PICO_INVALID_SAMPLERATIO

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.19 ps6000GetValuesBulkAsync

```
PICO_STATUS ps6000GetValuesBulkAsync
(
  int16_t                     handle,
  uint32_t                    startIndex,
  uint32_t                  * noOfSamples,
  uint32_t                    downSampleRatio,
  PS6000_RATIO_MODE           downSampleRatioMode,
  uint32_t                    fromSegmentIndex,
  uint32_t                    toSegmentIndex,
  int16_t                   * overflow
)
```

This function retrieves more than one waveform at a time from the driver in rapid block mode after data collection has stopped. The waveforms must have been collected sequentially and in the same run. The data is returned using a callback.

Applicability	Rapid block mode
Arguments	handle, startIndex, * noOfSamples, downSampleRatio, downSampleRatioMode: see ps6000GetValues fromSegmentIndex, toSegmentIndex, * overflow: see ps6000GetValuesBulk
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE PICO_STARTINDEX_INVALID PICO_NOT_RESPONDING PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.20 ps6000GetValuesOverlapped

```
PICO_STATUS ps6000GetValuesOverlapped
(
  int16_t               handle,
  uint32_t              startIndex,
  uint32_t            * noOfSamples,
  uint32_t              downSampleRatio,
  PS6000_RATIO_MODE     downSampleRatioMode,
  uint32_t              segmentIndex,
  int16_t             * overflow
)
```

This function allows you to make a deferred data-collection request in block mode. The request will be executed, and the arguments validated, when you call ps6000RunBlock. The advantage of this function is that the driver makes contact with the scope only once, when you call ps6000RunBlock, compared with the two contacts that occur when you use the conventional ps6000RunBlock, ps6000GetValues calling sequence. This slightly reduces the dead time between successive captures in block mode.

After calling ps6000RunBlock, you can optionally use ps6000GetValues to request further copies of the data. This might be required if you wish to display the data with different data reduction settings.

For more information, see Using the GetValuesOverlapped functions.

Applicability	Block mode
Arguments	handle, startIndex, * noOfSamples, downSampleRatio, downSampleRatioMode, segmentIndex: see ps6000GetValues * overflow: see ps6000GetValuesBulk
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

3.20.1 Using the GetValuesOverlapped functions

1. Open the oscilloscope using ps6000OpenUnit.
2. Select channel ranges and AC/DC coupling using ps6000SetChannel.
3. Using ps6000GetTimebase, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions ps6000SetTriggerChannelConditions, ps6000SetTriggerChannelDirections and ps6000SetTriggerChannelProperties to set up the trigger if required.
5. Use ps6000SetDataBuffer to tell the driver where your memory buffer is.
6. Set up the transfer of the block of data from the oscilloscope using ps6000GetValuesOverlapped.
7. Start the oscilloscope running using ps6000RunBlock.
8. Wait until the oscilloscope is ready using the ps6000BlockReady callback (or poll using ps6000IsReady).
9. Display the data.
10. Repeat steps 7 to 9 if needed.

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

11. Stop the oscilloscope by calling `ps6000Stop`.

A similar procedure can be used with rapid block mode using the `ps6000GetValuesOverlappedBulk` function.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.21 ps6000GetValuesOverlappedBulk

```
PICO_STATUS ps6000GetValuesOverlappedBulk
(
  int16_t              handle,
  uint32_t             startIndex,
  uint32_t           * noOfSamples,
  uint32_t             downSampleRatio,
  PS6000_RATIO_MODE    downSampleRatioMode,
  uint32_t             fromSegmentIndex,
  uint32_t             toSegmentIndex,
  int16_t            * overflow
)
```

This function allows you to make a deferred data-collection request in rapid block mode. The request will be executed, and the arguments validated, when you call ps6000RunBlock. The advantage of this method is that the driver makes contact with the scope only once, when you call ps6000RunBlock, compared with the two contacts that occur when you use the conventional ps6000RunBlock, ps6000GetValues calling sequence. This slightly reduces the dead time between successive captures in rapid block mode.

After calling ps6000RunBlock, you can optionally use ps6000GetValues to request further copies of the data. This might be required if you wish to display the data with different data reduction settings.

For more information, see Using the GetValuesOverlapped functions.

Applicability	Rapid block mode
Arguments	handle, startIndex, * noOfSamples, downSampleRatio, downSampleRatioMode: see ps6000GetValues fromSegmentIndex, toSegmentIndex, * overflow, see ps6000GetValuesBulk
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

3.22 ps6000GetValuesTriggerTimeOffsetBulk

```
PICO_STATUS ps6000GetValuesTriggerTimeOffsetBulk
(
  int16_t               handle,
  uint32_t            * timesUpper,
  uint32_t            * timesLower,
  PS6000_TIME_UNITS   * timeUnits,
  uint32_t              fromSegmentIndex,
  uint32_t              toSegmentIndex
)
```

This function retrieves the trigger time offset for multiple waveforms obtained in block mode or rapid block mode. It is a more efficient alternative to calling `ps6000GetTriggerTimeOffset` once for each waveform required. See `ps6000GetTriggerTimeOffset` for an explanation of trigger time offsets.

There is another version of this function, `ps6000GetValuesTriggerTimeOffsetBulk64`, that returns trigger time offsets as 64-bit values instead of pairs of 32-bit values.

 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

Applicability	Rapid block mode
Arguments	handle, identifies the device * timesUpper, an array of integers. On exit, the most significant 32 bits of the time offset for each requested segment index. times[0] will hold the fromSegmentIndex time offset and the last times index will hold the toSegmentIndex time offset. The array must be long enough to hold the number of requested times. * timesLower, an array of integers. On exit, the least-significant 32 bits of the time offset for each requested segment index. times[0] will hold the fromSegmentIndex time offset and the last times index will hold the toSegmentIndex time offset. The array size must be long enough to hold the number of requested times. * timeUnits, an array of integers. The array must be long enough to hold the number of requested times. On exit, timeUnits[0] will contain the time unit for fromSegmentIndex and the last element will contain the time unit for toSegmentIndex. PS6000_TIME_UNITS values are listed under ps6000GetTriggerTimeOffset. fromSegmentIndex, the first segment for which the time offset is required toSegmentIndex, the last segment for which the time offset is required. If toSegmentIndex is less than fromSegmentIndex then the driver will wrap around from the last segment to the first.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.23 ps6000GetValuesTriggerTimeOffsetBulk64

```
PICO_STATUS ps6000GetValuesTriggerTimeOffsetBulk64
(
  int16_t               handle,
  int64_t             * times,
  PS6000_TIME_UNITS   * timeUnits,
  uint32_t              fromSegmentIndex,
  uint32_t              toSegmentIndex
)
```

This function retrieves the 64-bit time offsets for waveforms captured in rapid block mode.

A 32-bit version of this function, ps6000GetValuesTriggerTimeOffsetBulk, is available for use with programming languages that do not support 64-bit integers. See that function for an explanation of waveform time offsets.

Applicability	Rapid block mode
Arguments	handle, identifies the device * times, an array of integers. On exit, this will hold the time offset for each requested segment index. times[0] will hold the time offset for fromSegmentIndex, and the last times index will hold the time offset for toSegmentIndex. The array must be long enough to hold the number of times requested. * timeUnits, see ps6000GetValuesTriggerTimeOffsetBulk. fromSegmentIndex, the first segment for which the time offset is required. The results for this segment will be placed in times[0] and timeUnits[0]. toSegmentIndex, the last segment for which the time offset is required. The results for this segment will be placed in the last elements of the times and timeUnits arrays. If toSegmentIndex is less than fromSegmentIndex then the driver will wrap around from the last segment to the first.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.24 ps6000IsReady

```
PICO_STATUS ps6000IsReady
(
  int16_t    handle,
  int16_t  * ready
)
```

This function may be used instead of a callback function to receive data from ps6000RunBlock. To use this method, pass a NULL pointer as the lpReady argument to ps6000RunBlock. You must then poll the driver to see if it has finished collecting the requested samples.

Applicability	Block mode
Arguments	handle, identifies the device ready, output: indicates the state of the collection. If zero, the device is still collecting. If non-zero, the device has finished collecting and ps6000GetValues can be used to retrieve the data.
Returns	

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.25 ps6000IsTriggerOrPulseWidthQualifierEnabled

```
PICO_STATUS ps6000IsTriggerOrPulseWidthQualifierEnabled
(
  int16_t    handle,
  int16_t  * triggerEnabled,
  int16_t  * pulseWidthQualifierEnabled
)
```

This function discovers whether a trigger, or pulse width triggering, is enabled.

Applicability	Call after setting up the trigger, just before calling either `ps6000RunBlock` or `ps6000RunStreaming`
Arguments	`handle`, identifies the device `triggerEnabled`, on exit, indicates whether the trigger will successfully be set when `ps6000RunBlock` or `ps6000RunStreaming` is called. A non-zero value indicates that the trigger is set, zero that the trigger is not set. `pulseWidthQualifierEnabled`, on exit, indicates whether the pulse width qualifier will successfully be set when `ps6000RunBlock` or `ps6000RunStreaming` is called. A non-zero value indicates that the pulse width qualifier is set, zero that the pulse width qualifier is not set.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`
	`PICO_DRIVER_FUNCTION`

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.26 ps6000MemorySegments

```
PICO_STATUS ps6000MemorySegments
(
  int16_t     handle
  uint32_t    nSegments,
  uint32_t  * nMaxSamples
)
```

This function sets the number of memory segments that the scope will use.

When the scope is opened, the number of segments defaults to 1, meaning that each capture fills the scope's available memory. This function allows you to divide the memory into a number of segments so that the scope can store several waveforms sequentially.

Applicability	All modes
Arguments	`handle`, identifies the device `nSegments`, the number of segments required: **Model** / **Min** / **Max** PicoScope 6402 / 1 / 32 768 PicoScope 6402A / 1 / 125 000 PicoScope 6402B / 1 / 250 000 PicoScope 6402C / 1 / 250 000 PicoScope 6402D / 1 / 500 000 PicoScope 6403 / 1 / 1 000 000 PicoScope 6403A / 1 / 250 000 PicoScope 6403B / 1 / 500 000 PicoScope 6403C / 1 / 500 000 PicoScope 6403D / 1 / 1 000 000 PicoScope 6404 / 1 / 1 000 000 PicoScope 6404A / 1 / 500 000 PicoScope 6404B / 1 / 1 000 000 PicoScope 6404C / 1 / 1 000 000 PicoScope 6404D / 1 / 2 000 000 PicoScope 6407 / 1 / 1 000 000 `* nMaxSamples`, on exit, the number of samples available in each segment. This is the total number over all channels, so if more than one channel is in use then the number of samples available to each channel is `nMaxSamples` divided by the number of channels.

Returns	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_TOO_MANY_SEGMENTS`
	`PICO_MEMORY`
	`PICO_DRIVER_FUNCTION`

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.27 ps6000NoOfStreamingValues

```
PICO_STATUS ps6000NoOfStreamingValues
(
  int16_t     handle,
  uint32_t  * noOfValues
)
```

This function returns the number of samples available after data collection in streaming mode. Call it after calling `ps6000Stop`.

Applicability	Streaming mode
Arguments	`handle`, identifies the device `* noOfValues`, on exit, the number of samples

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_NOT_USED`
	`PICO_BUSY`
	`PICO_DRIVER_FUNCTION`

 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.28 ps6000OpenUnit

```
PICO_STATUS ps6000OpenUnit
(
  int16_t  * handle,
  int8_t   * serial
)
```

This function opens a PicoScope 6000 Series scope attached to the computer. The maximum number of units that can be opened depends on the operating system, the kernel driver and the computer.

Applicability	All modes
Arguments	`* handle`, on exit, the result of the attempt to open a scope: -1 : if the scope fails to open 0 : if no scope is found > 0 : a number that uniquely identifies the scope If a valid handle is returned, it must be used in all subsequent calls to API functions to identify this scope. `serial`, on entry, a null-terminated string containing the serial number of the scope to be opened. If `serial` is NULL then the function opens the first scope found; otherwise, it tries to open the scope that matches the string.
Returns	PICO_OK PICO_OS_NOT_SUPPORTED PICO_OPEN_OPERATION_IN_PROGRESS PICO_EEPROM_CORRUPT PICO_KERNEL_DRIVER_TOO_OLD PICO_FW_FAIL PICO_MAX_UNITS_OPENED PICO_NOT_FOUND (if the specified unit was not found) PICO_NOT_RESPONDING PICO_MEMORY_FAIL PICO_ANALOG_BOARD PICO_CONFIG_FAIL_AWG PICO_INITIALISE_FPGA

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

3.29 ps6000OpenUnitAsync

```
PICO_STATUS ps6000OpenUnitAsync
(
  int16_t  * status,
  int8_t   * serial
)
```

This function opens a scope without blocking the calling thread. You can find out when it has finished by periodically calling ps6000OpenUnitProgress until that function returns a non-zero value.

Applicability	All modes
Arguments	* status, a status code: 0 if the open operation was disallowed because another open operation is in progress 1 if the open operation was successfully started * serial: see ps6000OpenUnit
Returns	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_OPERATION_FAILED

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.30 ps6000OpenUnitProgress

```
PICO_STATUS ps6000OpenUnitProgress
(
  int16_t  * handle,
  int16_t  * progressPercent,
  int16_t  * complete
)
```

This function checks on the progress of a request made to ps6000OpenUnitAsync to open a scope.

Applicability	Use after ps6000OpenUnitAsync
Arguments	* handle: see ps6000OpenUnit. This handle is valid only if the function returns PICO_OK. * progressPercent, on exit, the percentage progress towards opening the scope. 100% implies that the open operation is complete. * complete, set to 1 when the open operation has finished
Returns	PICO_OK PICO_NULL_PARAMETER PICO_OPERATION_FAILED

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.31 ps6000PingUnit

```
PICO_STATUS ps6000PingUnit
(
  int16_t    handle
)
```

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

Applicability	All modes
Arguments	handle, the handle of the required device
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_BUSY PICO_NOT_RESPONDING

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.32 ps6000RunBlock

```
PICO_STATUS ps6000RunBlock
(
  int16_t              handle,
  uint32_t             noOfPreTriggerSamples,
  uint32_t             noOfPostTriggerSamples,
  uint32_t             timebase,
  int16_t              oversample,
  int32_t            * timeIndisposedMs,
  uint32_t             segmentIndex,
  ps6000BlockReady     lpReady,
  void               * pParameter
)
```

This function starts collecting data in block mode. For a step-by-step guide to this process, see Using block mode.

The number of samples is determined by noOfPreTriggerSamples and noOfPostTriggerSamples (see below for details). The total number of samples must not be more than the size of the segment referred to by segmentIndex.

Note that ETS mode only supports timebases 0, 1 and 2.

Applicability	Block mode, rapid block mode
Arguments	handle, identifies the device noOfPreTriggerSamples, the number of samples to return before the trigger event. If no trigger has been set, then this argument is added to noOfPostTriggerSamples to give the maximum number of data points (samples) to collect. noOfPostTriggerSamples, the number of samples to return after the trigger event. If no trigger event has been set, then this argument is added to noOfPreTriggerSamples to give the maximum number of data points to collect. If a trigger condition has been set, this specifies the number of data points to collect after a trigger has fired, and the number of samples to be collected is: noOfPreTriggerSamples + noOfPostTriggerSamples timebase, a number in the range 0 to 2^{32}–1. See the guide to calculating timebase values. oversample, the oversampling factor, a number in the range 1 to 256. * timeIndisposedMs, on exit, the time in milliseconds that the scope will spend collecting samples. This does not include any auto trigger timeout. If this pointer is null, nothing will be written here. segmentIndex, zero-based, specifies which memory segment to use.

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

	lpReady, a pointer to the ps6000BlockReady callback function that the driver will call when the data has been collected. To use the ps6000IsReady polling method instead of a callback function, set this pointer to NULL. * pParameter, a void pointer that is passed to the ps6000BlockReady callback function. The callback can use this pointer to return arbitrary data to the application.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_CHANNEL PICO_INVALID_TRIGGER_CHANNEL PICO_INVALID_CONDITION_CHANNEL PICO_TOO_MANY_SAMPLES PICO_INVALID_TIMEBASE PICO_NOT_RESPONDING PICO_CONFIG_FAIL PICO_INVALID_PARAMETER PICO_NOT_RESPONDING PICO_TRIGGER_ERROR PICO_DRIVER_FUNCTION PICO_EXTERNAL_FREQUENCY_INVALID PICO_FW_FAIL PICO_NOT_ENOUGH_SEGMENTS (in Bulk mode) PICO_TRIGGER_AND_EXTERNAL_CLOCK_CLASH PICO_PWQ_AND_EXTERNAL_CLOCK_CLASH PICO_PULSE_WIDTH_QUALIFIER PICO_SEGMENT_OUT_OF_RANGE (in Overlapped mode) PICO_STARTINDEX_INVALID (in Overlapped mode) PICO_INVALID_SAMPLERATIO (in Overlapped mode) PICO_CONFIG_FAIL PICO_SIGGEN_GATING_AUXIO_ENABLED (signal generator is set to trigger on AUX input with incompatible trigger type)

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.33 ps6000RunStreaming

```
PICO_STATUS ps6000RunStreaming
(
  int16_t               handle,
  uint32_t            * sampleInterval,
  PS6000_TIME_UNITS     sampleIntervalTimeUnits
  uint32_t              maxPreTriggerSamples,
  uint32_t              maxPostTriggerSamples,
  int16_t               autoStop,
  uint32_t              downSampleRatio,
  PS6000_RATIO_MODE     downSampleRatioMode,
  uint32_t              overviewBufferSize
)
```

This function tells the oscilloscope to start collecting data in streaming mode. When data has been collected from the device it is downsampled if necessary and then delivered to the application. Call `ps6000GetStreamingLatestValues` to retrieve the data. See Using streaming mode for a step-by-step guide to this process.

When a trigger is set, the total number of samples stored in the driver is the sum of `maxPreTriggerSamples` and `maxPostTriggerSamples`. If `autoStop` is false then this will become the maximum number of samples without downsampling.

Applicability	Streaming mode

Arguments	`handle,` identifies the device
	
	`* sampleInterval,` on entry, the requested time interval between samples; on exit, the actual time interval used
	
	`sampleIntervalTimeUnits,` the unit of time used for `sampleInterval`. Use one of these values:
	`PS6000_FS`
	`PS6000_PS`
	`PS6000_NS`
	`PS6000_US`
	`PS6000_MS`
	`PS6000_S`
	
	`maxPreTriggerSamples,` the maximum number of raw samples before a trigger event for each enabled channel. If no trigger condition is set this argument is ignored.
	
	`maxPostTriggerSamples,` the maximum number of raw samples after a trigger event for each enabled channel. If no trigger condition is set, this argument states the maximum number of samples to be stored.
	
	`autoStop,` a flag that specifies if the streaming should stop when all of `maxSamples` have been captured.
	
	`downSampleRatio,`
	`downSampleRatioMode:` see `ps6000GetValues`

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

	overviewBufferSize, the size of the overview buffers. These are temporary buffers used for storing the data before returning it to the application. The size is the same as the bufferLth value passed to ps6000SetDataBuffer.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_STREAMING_FAILED PICO_NOT_RESPONDING PICO_TRIGGER_ERROR PICO_INVALID_SAMPLE_INTERVAL PICO_INVALID_BUFFER PICO_DRIVER_FUNCTION PICO_EXTERNAL_FREQUENCY_INVALID PICO_FW_FAIL PICO_TRIGGER_AND_EXTERNAL_CLOCK_CLASH PICO_PWQ_AND_EXTERNAL_CLOCK_CLASH PICO_MEMORY PICO_SIGGEN_GATING_AUXIO_ENABLED (signal generator is set to trigger on AUX input with incompatible trigger type)

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.34 ps6000SetChannel

```
PICO_STATUS ps6000SetChannel
(
  int16_t                     handle,
  PS6000_CHANNEL              channel,
  int16_t                     enabled,
  PS6000_COUPLING             type,
  PS6000_RANGE                range,
  float                       analogueOffset,
  PS6000_BANDWIDTH_LIMITER    bandwidth
)
```

This function specifies whether an input channel is to be enabled, its input coupling type, voltage range, analog offset and bandwidth limit. Some of the arguments within this function have model-specific values. Please consult the relevant section below according to the model you have.

Applicability	All modes

Arguments

handle, identifies the device

channel, the channel to be configured. The values are:
- PS6000_CHANNEL_A: Channel input
- PS6000_CHANNEL_B: Channel input
- PS6000_CHANNEL_C: Channel input
- PS6000_CHANNEL_D: Channel input

enabled, whether or not to enable the channel. The values are:
- TRUE: enable
- FALSE: do not enable

type, the impedance and coupling type. The values supported are:

PicoScope 6402/6403/6404 (all model variants)

PS6000_AC, 1 MΩ impedance, AC coupling. The channel accepts input frequencies from about 1 hertz up to its maximum -3 dB analog bandwidth.

PS6000_DC_1M, 1 MΩ impedance, DC coupling. The scope accepts all input frequencies from zero (DC) up to its maximum -3 dB analog bandwidth.

PS6000_DC_50R, DC coupling, 50 Ω impedance. In this mode the ±10 volt and ±20 volt input ranges are not available.

PicoScope 6407

PS6000_DC_50R, DC coupling, 50 Ω impedance.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

range, the input voltage range:

PicoScope 6402/6403/6404 (all model variants)

PS6000_50MV: ±50 mV
PS6000_100MV: ±100 mV
PS6000_200MV: ±200 mV
PS6000_500MV: ±500 mV
PS6000_1V: ±1 V
PS6000_2V: ±2 V
PS6000_5V: ±5 V
PS6000_10V: ±10 V *
PS6000_20V: ±20 V *
* not available when type = PS6000_DC_50R

PicoScope 6407

PS6000_100MV: ±100 mV

analogueOffset, a voltage to add to the input channel before digitization.

PicoScope 6402/6403 (all model variants)

The allowable range of offsets depends on the input range selected for the channel, as follows:

50 mV to 200 mV: MIN_ANALOGUE_OFFSET_50MV_200MV to MAX_ANALOGUE_OFFSET_50MV_200MV

500 mV to 2 V: MIN_ANALOGUE_OFFSET_500MV_2V to MAX_ANALOGUE_OFFSET_500MV_2V

5 V to 20 V: MIN_ANALOGUE_OFFSET_5V_20V to MAX_ANALOGUE_OFFSET_5V_20V. (When type = PS6000_DC_50R, the allowable range is reduced to that of the 50 mV to 200 mV input range, i.e. MIN_ANALOGUE_OFFSET_50MV_200MV to MAX_ANALOGUE_OFFSET_50MV_200MV).

Allowable range of offsets can also be returned by ps6000GetAnalogueOffset for the device currently connected.

PicoScope 6404 (all model variants)

Allowable range of offsets is returned by ps6000GetAnalogueOffset for the device currently connected.

PicoScope 6407

analogueOffset, Not used. Set to 0.

bandwidth, the bandwidth limiter setting:

PicoScope 6402/6403 (all model variants)

PS6000_BW_FULL: the connected scope's full specified bandwidth
PS6000_BW_20MHZ: –3 dB bandwidth limited to 20 MHz

PicoScope 6404 (all model variants)

PS6000_BW_FULL: the scope's full specified bandwidth
PS6000_BW_25MHZ: –3 dB bandwidth limited to 25 MHz

PicoScope 6407

PS6000_BW_FULL: the scope's full specified bandwidth

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

Returns	PICO_OK PICO_USER_CALLBACK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_INVALID_VOLTAGE_RANGE PICO_INVALID_COUPLING PICO_COUPLING_NOT_SUPPORTED PICO_INVALID_ANALOGUE_OFFSET PICO_INVALID_BANDWIDTH PICO_BANDWIDTH_NOT_SUPPORTED PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.35 ps6000SetDataBuffer

```
PICO_STATUS ps6000SetDataBuffer
(
  int16_t                handle,
  PS6000_CHANNEL         channel,
  int16_t              * buffer,
  uint32_t               bufferLth,
  PS6000_RATIO_MODE      downSampleRatioMode
)
```

This function tells the driver where to store the data, either unprocessed or downsampled, that will be returned after the next call to one of the GetValues functions. The function allows you to specify only a single buffer, so for aggregation mode, which requires two buffers, you must call ps6000SetDataBuffers instead.

The buffer remains persistent between captures until it is replaced with another buffer or the buffer is set to NULL. The buffer can be replaced at any time between calls to ps6000GetValues.

You must allocate memory for the buffer before calling this function.

Applicability	Block, rapid block and streaming modes. All downsampling modes except aggregation.
Arguments	handle, identifies the device channel, the channel you want to use with the buffer. Use one of these values: PS6000_CHANNEL_A PS6000_CHANNEL_B PS6000_CHANNEL_C PS6000_CHANNEL_D buffer, the location of the buffer bufferLth, the size of the buffer array downSampleRatioMode, the downsampling mode. See ps6000GetValues for the available modes, but note that a single call to ps6000SetDataBuffer can only associate one buffer with one downsampling mode. If you intend to call ps6000GetValues with more than one downsampling mode activated, then you must call ps6000SetDataBuffer several times to associate a separate buffer with each downsampling mode.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_RATIO_MODE_NOT_SUPPORTED PICO_DRIVER_FUNCTION PICO_INVALID_PARAMETER

 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.36 ps6000SetDataBufferBulk

```
PICO_STATUS ps6000SetDataBufferBulk
(
int16_t               handle,
PS6000_CHANNEL        channel,
int16_t             * buffer,
uint32_t              bufferLth,
uint32_t              waveform,
PS6000_RATIO_MODE     downSampleRatioMode
)
```

This function allows you to associate a buffer with a specified waveform number and input channel in rapid block mode. The number of waveforms captured is determined by the nCaptures argument sent to ps6000SetNoOfCaptures. There is only one buffer for each waveform because the only downsampling mode that requires two buffers, aggregation mode, is not available in rapid block mode. Call one of the GetValues functions to retrieve the data after capturing.

Applicability	Rapid block mode without aggregation.
Arguments	handle, identifies the device channel, the input channel to use with this buffer buffer, an array in which the captured data is stored bufferLth, the size of the buffer waveform, an index to the waveform number. Range: 0 to nCaptures - 1 downSampleRatioMode: see ps6000GetValues
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_INVALID_PARAMETER PICO_RATIO_MODE_NOT_SUPPORTED PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.37 ps6000SetDataBuffers

```
PICO_STATUS ps6000SetDataBuffers
(
  int16_t             handle,
  PS6000_CHANNEL      channel,
  int16_t           * bufferMax,
  int16_t           * bufferMin,
  uint32_t            bufferLth,
  PS6000_RATIO_MODE   downSampleRatioMode
)
```

This function tells the driver the location of one or two buffers for receiving data. You need to allocate memory for the buffers before calling this function. If you do not need two buffers, because you are not using aggregate mode, then you can optionally use ps6000SetDataBuffer instead.

Applicability	Block and streaming modes with aggregation.
Arguments	handle, identifies the device channel, the channel for which you want to set the buffers. Use one of these constants: PS6000_CHANNEL_A PS6000_CHANNEL_B PS6000_CHANNEL_C PS6000_CHANNEL_D * bufferMax, a buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise. * bufferMin, a buffer to receive the minimum aggregated data values. Not used in other downsampling modes. bufferLth, the size of the bufferMax and bufferMin arrays. downSampleRatioMode: see ps6000GetValues
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_RATIO_MODE_NOT_SUPPORTED PICO_DRIVER_FUNCTION PICO_INVALID_PARAMETER

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.38 ps6000SetDataBuffersBulk

```
PICO_STATUS ps6000SetDataBuffersBulk
(
   int16_t              handle,
   PS6000_CHANNEL       channel,
   int16_t            * bufferMax,
   int16_t            * bufferMin,
   uint32_t             bufferLth,
   uint32_t             waveform,
   PS6000_RATIO_MODE    downSampleRatioMode
)
```

This function tells the driver where to find the buffers for aggregated data for each waveform in rapid block mode. The number of waveforms captured is determined by the nCaptures argument sent to ps6000SetNoOfCaptures. Call one of the GetValues functions to retrieve the data after capture. If you do not need two buffers, because you are not using aggregate mode, then you can optionally use ps6000SetDataBufferBulk instead.

Applicability	Rapid block mode with aggregation
Arguments	handle, identifies the device channel, the input channel to use with the buffer * bufferMax, a buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise * bufferMin, a buffer to receive the minimum data values in aggregate mode. Not used in other downsampling modes. bufferLth, the size of the buffer waveform, an index to the waveform number between 0 and nCaptures-1 downSampleRatioMode: see ps6000GetValues
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_INVALID_PARAMETER PICO_RATIO_MODE_NOT_SUPPORTED PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.39 ps6000SetEts

```
PICO_STATUS ps6000SetEts
(
  int16_t           handle,
  PS6000_ETS_MODE   mode,
  int16_t           etsCycles,
  int16_t           etsInterleave,
  int32_t         * sampleTimePicoseconds
)
```

This function is used to enable or disable ETS (equivalent-time sampling) and to set the ETS parameters. See ETS overview for an explanation of ETS mode.

Applicability	Block mode
Arguments	`handle`, identifies the device `mode`, the ETS mode. Use one of these values: `PS6000_ETS_OFF` - disables ETS `PS6000_ETS_FAST` - enables ETS and provides `etsCycles` of data, which may contain data from previously returned cycles `PS6000_ETS_SLOW` - enables ETS and provides fresh data every `etsCycles`. This mode takes longer to provide each data set, but the data sets are more stable and are guaranteed to contain only new data. `etscycles`, the number of cycles to store: the computer can then select `etsInterleave` cycles to give the most uniform spread of samples Range: between two and five times the value of `etsInterleave`, and not more than `PS6000_MAX_ETS_CYCLES` `etsInterleave`, the number of waveforms to combine into a single ETS capture Maximum value: `PS6000_MAX_INTERLEAVE` `* sampleTimePicoseconds`, on exit, the minimum possible effective sampling interval of the ETS data. The actual sampling interval depends on the `timebase` argument passed to `ps6000RunBlock`. For example, if the captured sample time is 200 ps and `etsInterleave` is 4, then the effective sample time in ETS mode is 50 ps.

Returns	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`
	`PICO_DRIVER_FUNCTION`

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.40 ps6000SetEtsTimeBuffer

```
PICO_STATUS ps6000SetEtsTimeBuffer
(
  int16_t     handle,
  int64_t   * buffer,
  uint32_t    bufferLth
)
```

This function tells the driver where to find your application's ETS time buffers. These buffers contain the 64-bit timing information for each ETS sample after you run a block-mode ETS capture.

Applicability	ETS mode only. If your programming language does not support 64-bit data, use the 32-bit version `ps6000SetEtsTimeBuffers` instead.
Arguments	`handle,` identifies the device `* buffer,` an array of 64-bit words, each representing the time in femtoseconds (10^{-15} seconds) at which the sample was captured `bufferLth,` the size of the buffer array

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`
	`PICO_DRIVER_FUNCTION`

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.41 ps6000SetEtsTimeBuffers

```
PICO_STATUS ps6000SetEtsTimeBuffers
(
  int16_t      handle,
  uint32_t   * timeUpper,
  uint32_t   * timeLower,
  uint32_t     bufferLth
)
```

This function is a 32-bit equivalent of ps6000SetEtsTimeBuffer for programming environments that do not support 64-bit data. It defines two buffers containing the upper and lower 32-bit parts of the timing information.

Applicability	ETS mode only
Arguments	handle, identifies the device * timeUpper, an array of 32-bit words, each representing the upper 32 bits of the time in femtoseconds (10^{-15} seconds) at which the sample was captured * timeLower, an array of 32-bit words, each representing the lower 32 bits of the time in femtoseconds (10^{-15} seconds) at which the sample was captured bufferLth, the size of the timeUpper and timeLower arrays
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.42 ps6000SetExternalClock

```
PICO_STATUS ps6000SetExternalClock
(
  int16_t                       handle,
  PS6000_EXTERNAL_FREQUENCY     frequency,
  int16_t                       threshold
)
```

This function tells the scope whether or not to use an external clock signal fed into the AUX input. The external clock can be used to synchronize one or more PicoScope 6000 units to an external source.

When the external clock input is enabled, the oscilloscope relies on the clock signal for all of its timing. The driver checks that the clock is running before starting a capture, but if the clock signal stops after the initial check, the oscilloscope will not respond to any further commands until it is powered off and on again.

Note: if the AUX input is set as an external clock input, it cannot also be used as an external trigger input.

Applicability	All modes
Arguments	handle, identifies the device frequency, the external clock frequency. The possible values are: PS6000_FREQUENCY_OFF: the scope generates its own clock PS6000_FREQUENCY_5MHZ: 5 MHz external clock PS6000_FREQUENCY_10MHZ: 10 MHz external clock PS6000_FREQUENCY_20MHZ: 20 MHz external clock PS6000_FREQUENCY_25MHZ: 25 MHz external clock The external clock signal must be within ±5% of the selected frequency, otherwise this function will report an error. threshold, the logic threshold voltage: –32,512 : –1 volt 0 : 0 volts 32,512 : +1 volt
Returns	PICO_OK PICO_USER_CALLBACK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION PICO_EXTERNAL_FREQUENCY_INVALID PICO_FW_FAIL PICO_NOT_RESPONDING PICO_CLOCK_CHANGE_ERROR PICO_WARNING_SIGGEN_AUXIO_TRIGGER_DISABLED (signal genera

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.43 ps6000SetNoOfCaptures

```
PICO_STATUS ps6000SetNoOfCaptures
(
  int16_t     handle,
  uint32_t    nCaptures
)
```

This function sets the number of captures to be collected in one run of rapid block mode. If you do not call this function before a run, the driver will capture only one waveform.

Applicability	Rapid block mode
Arguments	`handle`, identifies the device `nCaptures`, the number of waveforms to capture in one run

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`
	`PICO_DRIVER_FUNCTION`

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.44 ps6000SetPulseWidthQualifier

```
PICO_STATUS ps6000SetPulseWidthQualifier
(
  int16_t                        handle,
  PS6000_PWQ_CONDITIONS        * conditions,
  int16_t                        nConditions,
  PS6000_THRESHOLD_DIRECTION     direction,
  uint32_t                       lower,
  uint32_t                       upper,
  PS6000_PULSE_WIDTH_TYPE        type
)
```

This function sets up the conditions for pulse width qualification, which is used with either threshold triggering, level triggering or window triggering to produce time-qualified triggers. Each call to this function creates a pulse width qualifier equal to the logical AND of the elements of the conditions array. Calling this function multiple times creates the logical OR of multiple AND operations. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

Applicability	All modes
Arguments	handle, identifies the device * conditions, an array of PS6000_PWQ_CONDITIONS structures specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there are several elements, the overall trigger condition is the logical OR of all the elements. If conditions is NULL, the pulse-width qualifier is not used. nConditions, the number of elements in the conditions array. If nConditions is zero then the pulse-width qualifier is not used. Range: 0 to PS6000_MAX_PULSE_WIDTH_QUALIFIER_COUNT. direction, the direction of the signal required for the trigger to fire. See ps6000SetTriggerChannelDirections for the list of possible values. Each channel of the oscilloscope (except the AUX input) has two thresholds for each direction—for example, PS6000_RISING and PS6000_RISING_LOWER—so that one can be used for the pulse-width qualifier and the other for the level trigger. The driver will not let you use the same threshold for both triggers; so, for example, you cannot use PS6000_RISING as the direction argument for both ps6000SetTriggerConditions and ps6000SetPulseWidthQualifier at the same time. There is no such restriction when using window triggers. lower, the lower limit of the pulse-width counter, in samples. upper, the upper limit of the pulse-width counter, in samples. This parameter is used only when the type is set to PS6000_PW_TYPE_IN_RANGE or PS6000_PW_TYPE_OUT_OF_RANGE.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

	type, the pulse-width type, one of these constants: PS6000_PW_TYPE_NONE: do not use the pulse width qualifier PS6000_PW_TYPE_LESS_THAN: pulse width less than lower PS6000_PW_TYPE_GREATER_THAN: pulse width greater than lower PS6000_PW_TYPE_IN_RANGE: pulse width between lower and upper PS6000_PW_TYPE_OUT_OF_RANGE: pulse width not between lower and upper
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_PULSE_WIDTH_QUALIFIER PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.44.1 PS6000_PWQ_CONDITIONS structure

A structure of this type is passed to ps6000SetPulseWidthQualifier in the conditions argument to specify the trigger conditions. It is defined as follows:

```
typedef struct tPwqConditions
{
  PS6000_TRIGGER_STATE   channelA;
  PS6000_TRIGGER_STATE   channelB;
  PS6000_TRIGGER_STATE   channelC;
  PS6000_TRIGGER_STATE   channelD;
  PS6000_TRIGGER_STATE   external;
  PS6000_TRIGGER_STATE   aux;
} PS6000_PWQ_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The ps6000SetPulseWidthQualifier function can OR together a number of these structures to produce the final pulse width qualifier, which can therefore be any possible Boolean function of the scope's inputs.

The structure is byte-aligned. In C++, for example, you should specify this using the #pragma pack() instruction.

Elements	channelA, channelB, channelC, channelD, aux: the type of condition that should be applied to each channel. Use these constants: PS6000_CONDITION_DONT_CARE PS6000_CONDITION_TRUE PS6000_CONDITION_FALSE The channels that are set to PS6000_CONDITION_TRUE or PS6000_CONDITION_FALSE must all meet their conditions simultaneously to produce a trigger. Channels set to PS6000_CONDITION_DONT_CARE are ignored. external: not used

3.45 ps6000SetSigGenArbitrary

```
PICO_STATUS ps6000SetSigGenArbitrary
(
  int16_t                      handle,
  int32_t                      offsetVoltage,
  uint32_t                     pkToPk
  uint32_t                     startDeltaPhase,
  uint32_t                     stopDeltaPhase,
  uint32_t                     deltaPhaseIncrement,
  uint32_t                     dwellCount,
  int16_t                    * arbitraryWaveform,
  int32_t                      arbitraryWaveformSize,
  PS6000_SWEEP_TYPE            sweepType,
  PS6000_EXTRA_OPERATIONS      operation,
  PS6000_INDEX_MODE            indexMode,
  uint32_t                     shots,
  uint32_t                     sweeps,
  PS6000_SIGGEN_TRIG_TYPE      triggerType,
  PS6000_SIGGEN_TRIG_SOURCE    triggerSource,
  int16_t                      extInThreshold
)
```

This function programs the arbitrary waveform generator (AWG).

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

The output frequency is controlled by the *startDeltaPhase* and *stopDeltaPhase* arguments. Only *startDeltaPhase* is required to generate a fixed frequency, *stopDeltaPhase* being additionally required when generating a frequency sweep. Each *deltaPhase* argument can be calculated by calling ps6000SigGenFrequencyToPhase. For information on how this works, see Calculating deltaPhase.

Applicability	PicoScope 6402/3/4, 6402B/3B/4B, 6402D/3D/4D
Arguments	

`handle`, identifies the device

`offsetVoltage`, the voltage offset, in microvolts, to be applied to the waveform

`pkToPk`, the peak-to-peak voltage, in microvolts, of the waveform signal

`startDeltaPhase`, the initial value of *deltaPhase* added to the phase counter as the generator begins to step through the waveform buffer. This argument defines the output frequency when a fixed frequency is desired, or the initial output frequency when a frequency sweep is desired. Call ps6000SigGenFrequencyToPhase to calculate a suitable value.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

	`stopDeltaPhase`, the final value of *deltaPhase* added to the phase counter before the generator restarts or reverses the sweep. This argument defines the final output frequency when a frequency sweep is desired. Call ps6000SigGenFrequencyToPhase to calculate a suitable value. This argument is ignored if `deltaPhaseIncrement` is zero. `deltaPhaseIncrement`, the amount added to the delta phase value after every `dwellCount` period. This determines the amount by which the generator increments or decrements the output frequency in each `dwellCount` period. If no frequency sweep is required, `deltaPhaseIncrement` must be zero. `dwellCount`, the time, in units of *dacPeriod*, between successive additions of `deltaPhaseIncrement` to the delta phase counter. This determines the rate at which the generator sweeps the output frequency. If `deltaPhaseIncrement` is zero, this argument is ignored. Minimum value: PS6000_MIN_DWELL_COUNT `* arbitraryWaveform`, a buffer that holds the waveform pattern as a set of samples equally spaced in time. If `pkToPk` is set to its maximum (4 V) and `offsetVoltage` is set to 0: a sample of `minArbitraryWaveformValue` corresponds to −2 V a sample of `maxArbitraryWaveformValue` corresponds to +2 V where `minArbitraryWaveformValue` and `maxArbitraryWaveformValue` are the values returned by ps6000SigGenArbitraryMinMaxValues. `arbitraryWaveformSize`, the size of the arbitrary waveform buffer, in samples. The minimum and maximum allowable values are returned by ps6000SigGenArbitraryMinMaxValues. `sweepType`, determines whether the `startDeltaPhase` is swept up to the `stopDeltaPhase`, or down to it, or repeatedly swept up and down. Use one of these values: PS6000_UP PS6000_DOWN PS6000_UPDOWN PS6000_DOWNUP `operation`, see ps6000SigGenBuiltIn `indexMode`, specifies how the signal will be formed from the arbitrary waveform data. Single, dual and quad index modes are possible. Use one of these constants: PS6000_SINGLE PS6000_DUAL PS6000_QUAD `shots`, `sweeps`, `triggerType`, `triggerSource`, `extInThreshold`, see ps6000SigGenBuiltIn
Returns	PICO_OK PICO_INVALID_HANDLE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING PICO_NOT_RESPONDING PICO_WARNING_AUX_OUTPUT_CONFLICT

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

	PICO_WARNING_EXT_THRESHOLD_CONFLICT PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_OFFSET_VOLTAGE PICO_SIGGEN_PK_TO_PK PICO_SIGGEN_OUTPUT_OVER_VOLTAGE PICO_DRIVER_FUNCTION PICO_SIGGEN_WAVEFORM_SETUP_FAILED PICO_AWG_NOT_SUPPORTED (e.g. if device is a 6402/3/4 A/C)

3.45.1 Calculating deltaPhase

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

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

where:

outputFrequency	= repetition rate of the complete arbitrary waveform
dacFrequency	= update rate of AWG DAC (see table below)
deltaPhase	= calculated from *startDeltaPhase* and *deltaPhaseIncrement*
phaseAccumulatorSize	= maximum count of phase accumulator (see table below)
awgBufferSize	= maximum AWG buffer size (see table below)
arbitraryWaveformSize	= length in samples of the user-defined waveform

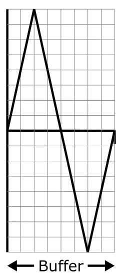
Parameter	Original/A/B models	C/D models
dacFrequency	200 MHz	
dacPeriod (= 1/*dacFrequency*)	5 ns	
phaseAccumulatorSize	4 294 967 296 (2^{32})	
awgBufferSize	16 384	65 536

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

3.45.2 Index modes

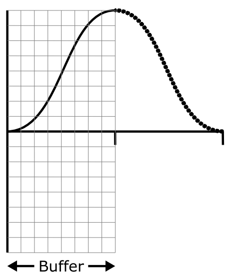
The arbitrary waveform generator supports **single, dual** and **quad** index modes to help you make the best use of the waveform buffer.

Single mode. The generator outputs the raw contents of the buffer repeatedly. This mode is the only one that can generate asymmetrical waveforms. You can also use this mode for symmetrical waveforms, but the dual and quad modes make more efficient use of the buffer memory.

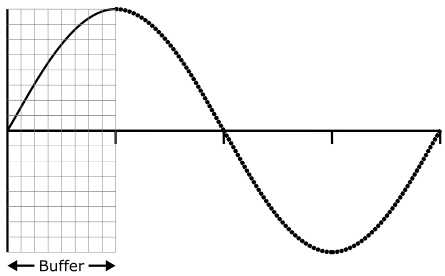


 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

Dual mode. The generator outputs the contents of the buffer from beginning to end, and then does a second pass in the reverse direction through the buffer. This allows you to specify only the first half of a waveform with twofold symmetry, such as a Gaussian function, and let the generator fill in the other half.



Quad mode. The generator outputs the contents of the buffer, then on its second pass through the buffer outputs the same data in reverse order. On the third and fourth passes it does the same but with a negative version of the data. This allows you to specify only the first quarter of a waveform with fourfold symmetry, such as a sine wave, and let the generator fill in the other three quarters.



Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.46 ps6000SetSigGenBuiltIn

```
PICO_STATUS ps6000SetSigGenBuiltIn
(
  int16_t                          handle,
  int32_t                          offsetVoltage,
  uint32_t                         pkToPk
  int16_t                          waveType
  float                            startFrequency,
  float                            stopFrequency,
  float                            increment,
  float                            dwellTime,
  PS6000_SWEEP_TYPE                sweepType,
  PS6000_EXTRA_OPERATIONS          operation,
  uint32_t                         shots,
  uint32_t                         sweeps,
  PS6000_SIGGEN_TRIG_TYPE          triggerType,
  PS6000_SIGGEN_TRIG_SOURCE        triggerSource,
  int16_t                          extInThreshold
)
```

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

Applicability	All modes
Arguments	

handle, identifies the device

offsetVoltage, the voltage offset, in microvolts, to be applied to the waveform

pkToPk, the peak-to-peak voltage, in microvolts, of the waveform signal

waveType, the type of waveform to be generated:

PS6000_SINE	sine wave
PS6000_SQUARE	square wave
PS6000_TRIANGLE	triangle wave
PS6000_RAMP_UP	rising sawtooth
PS6000_RAMP_DOWN	falling sawtooth
PS6000_SINC	sin (x)/x
PS6000_GAUSSIAN	Gaussian
PS6000_HALF_SINE	half (full-wave rectified) sine
PS6000_DC_VOLTAGE	DC voltage
PS6000_WHITE_NOISE	white noise

startFrequency, the frequency that the signal generator will initially produce. For allowable values see PS6000_SINE_MAX_FREQUENCY and related values.

stopFrequency, the frequency at which the sweep reverses direction or returns to the initial frequency

increment, the amount of frequency increase or decrease in sweep mode

dwellTime, the time for which the sweep stays at each frequency, in seconds

 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

`sweepType`, whether the frequency will sweep from `startFrequency` to `stopFrequency`, or in the opposite direction, or repeatedly reverse direction. Use one of these constants:
PS6000_UP
PS6000_DOWN
PS6000_UPDOWN
PS6000_DOWNUP

`operation`, selects periodic signal, white noise or PRBS:

PS6000_ES_OFF (0)	produces the waveform specified by `waveType`
PS6000_WHITENOISE (1)	produces white noise and ignores all settings except `offsetVoltage` and `pkTopk`
PS6000_PRBS (2)	produces a pseudo-random binary sequence (PRBS) and ignores all settings except `offsetVoltage` and `pkTopk`

`shots`, the number of cycles of the waveform to be produced after a trigger event. If non-zero (from 1 to MAX_SWEEPS_SHOTS), `sweeps` must be zero.

`sweeps`, the number of times to sweep the frequency after a trigger event, according to `sweepType`. If non-zero (from 1 to MAX_SWEEPS_SHOTS), `shots` must be zero.

`triggerType`, the type of trigger that will be applied to the signal generator:

PS6000_SIGGEN_RISING	trigger on rising edge
PS6000_SIGGEN_FALLING	trigger on falling edge
PS6000_SIGGEN_GATE_HIGH	run while trigger is high (not available if `triggerSource` is AUX)
PS6000_SIGGEN_GATE_LOW	run while trigger is low (not available if `triggerSource` is AUX)

`triggerSource`, the source that will trigger the signal generator:

PS6000_SIGGEN_NONE	run without waiting for trigger
PS6000_SIGGEN_SCOPE_TRIG	use scope trigger
PS6000_SIGGEN_AUX_IN	use AUX input
PS6000_SIGGEN_SOFT_TRIG	wait for software trigger provided by `ps6000SigGenSoftwareControl`
PS6000_SIGGEN_TRIGGER_RAW	reserved

If a trigger source other than P6000_SIGGEN_NONE is specified, either `shots` or `sweeps`, but not both, must be non-zero.

`extInThreshold`, the threshold voltage on the AUX input when used as a trigger source. If a different AUX threshold has previously been set up by `ps6000SetTriggerChannelProperties`, `ps6000SetPulseWidthQualifier` or `ps6000SetSimpleTrigger`, this function will override it and return PICO_WARNING_AUX_OUTPUT_CONFLICT.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

Returns	PICO_OK PICO_INVALID_HANDLE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING PICO_NOT_RESPONDING PICO_WARNING_AUX_OUTPUT_CONFLICT (see extInThreshold above) PICO_WARNING_EXT_THRESHOLD_CONFLICT PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_OFFSET_VOLTAGE PICO_SIGGEN_PK_TO_PK PICO_SIGGEN_OUTPUT_OVER_VOLTAGE PICO_DRIVER_FUNCTION PICO_SIGGEN_WAVEFORM_SETUP_FAILED PICO_NOT_RESPONDING PICO_SIGGEN_GATING_AUXIO_NOT_AVAILABLE (AUX input cannot be used with requested triggerType) PICO_SIGGEN_TRIGGER_AND_EXTERNAL_CLOCK_CLASH (cannot use AUX as trigger input because it is being used a clock input)

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.47 ps6000SetSigGenBuiltInV2

```
PICO_STATUS ps6000SetSigGenBuiltInV2
(
  int16_t                        handle,
  int32_t                        offsetVoltage,
  uint32_t                       pkToPk
  int16_t                        waveType
  double                         startFrequency,
  double                         stopFrequency,
  double                         increment,
  double                         dwellTime,
  PS6000_SWEEP_TYPE              sweepType,
  PS6000_EXTRA_OPERATIONS        operation,
  uint32_t                       shots,
  uint32_t                       sweeps,
  PS6000_SIGGEN_TRIG_TYPE        triggerType,
  PS6000_SIGGEN_TRIG_SOURCE      triggerSource,
  int16_t                        extInThreshold
)
```

This function sets up the signal generator. It differs from ps6000SetSigGenBuiltIn in having double-precision arguments instead of floats, giving greater resolution when setting the output frequency.

Applicability	All modes
Arguments	See ps6000SetSigGenBuiltIn
Returns	See ps6000SetSigGenBuiltIn

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.48 ps6000SetSimpleTrigger

```
PICO_STATUS ps6000SetSimpleTrigger
(
  int16_t                         handle,
  int16_t                         enable,
  PS6000_CHANNEL                  source,
  int16_t                         threshold,
  PS6000_THRESHOLD_DIRECTION      direction,
  uint32_t                        delay,
  int16_t                         autoTrigger_ms
)
```

This function simplifies arming the trigger. It supports only the LEVEL trigger types and does not allow more than one channel to have a trigger applied to it. Any previous pulse width qualifier is canceled.

Applicability	All modes
Arguments	handle, identifies the device enabled: zero to disable the trigger, any non-zero value to set the trigger. source: the channel on which to trigger. This can be one of the four input channels listed under ps6000SetChannel, or PS6000_TRIGGER_AUX for the AUX input. threshold: the ADC count at which the trigger will fire. direction: the direction in which the signal must move to cause a trigger. The following directions are supported: ABOVE, BELOW, RISING, FALLING and RISING_OR_FALLING. delay: the time between the trigger occurring and the first sample being taken. autoTrigger_ms: the number of milliseconds the device will wait if no trigger occurs.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.49 ps6000SetTriggerChannelConditions

```
PICO_STATUS ps6000SetTriggerChannelConditions
(
  int16_t                        handle,
  PS6000_TRIGGER_CONDITIONS    * conditions,
  int16_t                        nConditions
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is defined by one or more PS6000_TRIGGER_CONDITIONS structures that are then ORed together. Each structure is itself the AND of the states of one or more of the inputs. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

If complex triggering is not required, use ps6000SetSimpleTrigger.

Applicability	All modes
Arguments	handle, identifies the device conditions, an array of PS6000_TRIGGER_CONDITIONS structures specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there is more than one element, the overall trigger condition is the logical OR of all the elements. nConditions, the number of elements in the conditions array. If nConditions is zero then triggering is switched off.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_MEMORY_FAIL PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.49.1 PS6000_TRIGGER_CONDITIONS structure

A structure of this type is passed to ps6000SetTriggerChannelConditions in the conditions argument to specify the trigger conditions, and is defined as follows:

```
typedef struct tTriggerConditions
{
  PS6000_TRIGGER_STATE   channelA;
  PS6000_TRIGGER_STATE   channelB;
  PS6000_TRIGGER_STATE   channelC;
  PS6000_TRIGGER_STATE   channelD;
  PS6000_TRIGGER_STATE   external;
  PS6000_TRIGGER_STATE   aux;
  PS6000_TRIGGER_STATE   pulseWidthQualifier;
} PS6000_TRIGGER_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The ps6000SetTriggerChannelConditions function can OR together a number of these structures to produce the final trigger condition, which can be any possible Boolean function of the scope's inputs.

The structure is byte-aligned. In C++, for example, you should specify this using the #pragma pack() instruction.

Elements	channelA, channelB, channelC, channelD, aux, pulseWidthQualifier: the type of condition that should be applied to each channel. Use these constants: PS6000_CONDITION_DONT_CARE PS6000_CONDITION_TRUE PS6000_CONDITION_FALSE The channels that are set to PS6000_CONDITION_TRUE or PS6000_CONDITION_FALSE must all meet their conditions simultaneously to produce a trigger. Channels set to PS6000_CONDITION_DONT_CARE are ignored. external: not used

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.50 ps6000SetTriggerChannelDirections

```
PICO_STATUS ps6000SetTriggerChannelDirections
(
  int16_t                       handle,
  PS6000_THRESHOLD_DIRECTION    channelA,
  PS6000_THRESHOLD_DIRECTION    channelB,
  PS6000_THRESHOLD_DIRECTION    channelC,
  PS6000_THRESHOLD_DIRECTION    channelD,
  PS6000_THRESHOLD_DIRECTION    ext,
  PS6000_THRESHOLD_DIRECTION    aux
)
```

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

Applicability	All modes
Arguments	handle, identifies the device channelA, channelB, channelC, channelD, aux, the direction in which the signal must pass through the threshold to activate the trigger. See the table below for allowable values. If using a level trigger in conjunction with a pulse-width trigger, see the description of the direction argument to ps6000SetPulseWidthQualifier for more information. ext: not used
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_INVALID_PARAMETER

PS6000_THRESHOLD_DIRECTION constants

Constant	Trigger type	Threshold	Polarity
PS6000_ABOVE	Gated	Upper	Above
PS6000_ABOVE_LOWER	Gated	Lower	Above
PS6000_BELOW	Gated	Upper	Below
PS6000_BELOW_LOWER	Gated	Lower	Below
PS6000_RISING	Threshold	Upper	Rising
PS6000_RISING_LOWER	Threshold	Lower	Rising
PS6000_FALLING	Threshold	Upper	Falling
PS6000_FALLING_LOWER	Threshold	Lower	Falling
PS6000_RISING_OR_FALLING	Threshold	Lower (for rising edge) Upper (for falling edge)	
PS6000_INSIDE	Window-qualified	Both	Inside
PS6000_OUTSIDE	Window-qualified	Both	Outside
PS6000_ENTER	Window	Both	Entering
PS6000_EXIT	Window	Both	Leaving
PS6000_ENTER_OR_EXIT	Window	Both	Either entering or leaving
PS6000_POSITIVE_RUNT	Window-qualified	Both	Entering from below
PS6000_NEGATIVE_RUNT	Window-qualified	Both	Entering from above
PS6000_NONE	None	None	None

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

3.51 ps6000SetTriggerChannelProperties

```
PICO_STATUS ps6000SetTriggerChannelProperties
(
  int16_t                              handle,
  PS6000_TRIGGER_CHANNEL_PROPERTIES  * channelProperties
  int16_t                              nChannelProperties
  int16_t                              auxOutputEnable,
  uint32_t                             autoTriggerMilliseconds
)
```

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

Applicability	All modes
Arguments	handle, identifies the device channelProperties, a pointer to an array of TRIGGER_CHANNEL_PROPERTIES structures describing the requested properties. The array can contain a single element describing the properties of one channel, or a number of elements describing several channels. If NULL is passed, triggering is switched off. nChannelProperties, the size of the channelProperties array. If zero, triggering is switched off. auxOutputEnable: not used autoTriggerMilliseconds, the time in milliseconds for which the scope device will wait before collecting data if no trigger event occurs. If this is set to zero, the scope device will wait indefinitely for a trigger.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_TRIGGER_ERROR PICO_MEMORY_FAIL PICO_INVALID_TRIGGER_PROPERTY PICO_DRIVER_FUNCTION PICO_INVALID_PARAMETER

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.51.1 TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to ps6000SetTriggerChannelProperties in the channelProperties argument to specify the trigger mechanism, and is defined as follows:

```
typedef struct tTriggerChannelProperties
{
  int16_t                thresholdUpper;
  uint16_t               hysteresisUpper;
  int16_t                thresholdLower;
  uint16_t               hysteresisLower;
  PS6000_CHANNEL         channel;
  PS6000_THRESHOLD_MODE  thresholdMode;
} PS6000_TRIGGER_CHANNEL_PROPERTIES
```

The structure is byte-aligned. In C++, for example, you should specify this using the #pragma pack() instruction.

There are two trigger thresholds called Upper and Lower. Each trigger type uses one or other of these thresholds, or both, as specified in ps6000SetTriggerChannelDirections. Each trigger threshold has its own hysteresis setting.

Elements	thresholdUpper, the upper threshold at which the trigger fires. It is scaled in 16-bit ADC counts at the currently selected range for that channel. Use when "Upper" or "Both" is specified in ps6000SetTriggerChannelDirections. hysteresisUpper, the distance by which the signal must fall below the upper threshold (for rising edge triggers) or rise above the upper threshold (for falling edge triggers) in order to rearm the trigger for the next event. It is scaled in 16-bit counts. thresholdLower, lower threshold (see thresholdUpper). Use when "Lower" or "Both" is specified in ps6000SetTriggerChannelDirections. hysteresisLower, lower threshold hysteresis (see hysteresisUpper) channel, the channel to which the properties apply. This can be one of the four input channels listed under ps6000SetChannel, or PS6000_TRIGGER_AUX for the AUX input. thresholdMode, either a level or window trigger. Use one of these constants: PS6000_LEVEL PS6000_WINDOW

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

3.52 ps6000SetTriggerDelay

```
PICO_STATUS ps6000SetTriggerDelay
(
  int16_t     handle,
  uint32_t    delay
)
```

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

Applicability	Block and rapid block modes
Arguments	handle, identifies the device delay, the time between the trigger occurring and the first sample. For example, if delay=100 then the scope would wait 100 sample periods before sampling. At a timebase of 5 GS/s, or 200 ps per sample (timebase = 0), the total delay would then be 100 x 200 ps = 20 ns. Range: 0 to MAX_DELAY_COUNT
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.53 ps6000SigGenArbitraryMinMaxValues

```
PICO_STATUS ps6000SigGenArbitraryMinMaxValues
(
  int16_t     handle,
  int16_t   * minArbitraryWaveformValue,
  int16_t   * maxArbitraryWaveformValue,
  uint32_t  * minArbitraryWaveformSize,
  uint32_t  * maxArbitraryWaveformSize
)
```

This function returns the range of possible sample values and waveform buffer sizes that can be supplied to ps6000SetSigGenArbitrary for setting up the arbitrary waveform generator (AWG). These values vary between different models in the PicoScope 6000 Series.

Applicability	All models with AWG
Arguments	handle, identifies the device minArbitraryWaveformValue, on exit, the lowest sample value allowed in the arbitraryWaveform buffer supplied to ps6000SetSigGenArbitrary. maxArbitraryWaveformValue, on exit, the highest sample value allowed in the arbitraryWaveform buffer supplied to ps6000SetSigGenArbitrary. minArbitraryWaveformSize, on exit, the minimum value allowed for the arbitraryWaveformSize argument supplied to ps6000SetSigGenArbitrary. maxArbitraryWaveformSize, on exit, the maximum value allowed for the arbitraryWaveformSize argument supplied to ps6000SetSigGenArbitrary.
Returns	PICO_OK PICO_NOT_SUPPORTED_BY_THIS_DEVICE, if the device does not have an arbitrary waveform generator. PICO_NULL_PARAMETER, if all the parameter pointers are NULL. PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

3.54 ps6000SigGenFrequencyToPhase

```
PICO_STATUS ps6000SigGenFrequencyToPhase
(
  int16_t               handle,
  double                frequency,
  PS6000_INDEX_MODE     indexMode,
  uint32_t              bufferLength,
  uint32_t            * phase
)
```

This function converts a frequency to a phase count for use with the arbitrary waveform generator (AWG). The value returned depends on the length of the buffer, the index mode passed and the device model. The phase count can then be sent to the driver through `ps6000SetSigGenArbitrary`.

Applicability	All models with AWG
Arguments	`handle`, identifies the device `frequency`, the required AWG output frequency `indexMode`, see AWG index modes `bufferLength`, the number of samples in the AWG buffer `phase`, on exit, the `deltaPhase` argument to be sent to the AWG setup function

Returns	`PICO_OK`
	`PICO_NOT_SUPPORTED_BY_THIS_DEVICE`, if the device does not have an AWG.
	`PICO_SIGGEN_FREQUENCY_OUT_OF_RANGE`, if the frequency is out of range.
	`PICO_NULL_PARAMETER`, if `phase` is a NULL pointer.
	`PICO_SIG_GEN_PARAM`, if `indexMode` or `bufferLength` is out of range.
	`PICO_INVALID_HANDLE`
	`PICO_DRIVER_FUNCTION`

 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.55 ps6000SigGenSoftwareControl

```
PICO_STATUS ps6000SigGenSoftwareControl
(
  int16_t    handle,
  int16_t    state
)
```

This function causes a trigger event, or starts and stops gating. It is used when the signal generator is set to SIGGEN_SOFT_TRIG.

Applicability	Use with ps6000SetSigGenBuiltIn or ps6000SetSigGenArbitrary.
Arguments	handle, identifies the device state, sets the trigger gate high or low when the trigger type is set to either SIGGEN_GATE_HIGH or SIGGEN_GATE_LOW. Ignored for other trigger types.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_TRIGGER_SOURCE PICO_DRIVER_FUNCTION PICO_NOT_RESPONDING

Copyright © 2009-2016 Pico Technology Ltd. All rights reserved.

3.56 ps6000Stop

```
PICO_STATUS ps6000Stop
(
  int16_t    handle
)
```

This function stops the scope device from sampling data. If this function is called before a trigger event occurs, the oscilloscope may not contain valid data.

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

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

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

Applicability	All modes
Arguments	handle, identifies the device
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_DRIVER_FUNCTION

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.57 ps6000StreamingReady

```
typedef void (CALLBACK *ps6000StreamingReady)
(
  int16_t     handle,
  uint32_t    noOfSamples,
  uint32_t    startIndex,
  int16_t     overflow,
  uint32_t    triggerAt,
  int16_t     triggered,
  int16_t     autoStop,
  void      * pParameter
)
```

This callback function is part of your application. You register it with the driver using `ps6000GetStreamingLatestValues`, and the driver calls it back when streaming-mode data is ready. You can then download the data using the `ps6000GetValuesAsync` function.

The 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.

Applicability	Streaming mode only
Arguments	`handle`, identifies the device `noOfSamples`, the number of samples to collect `startIndex`, an index to the first valid sample in the buffer. This is the buffer that was previously passed to `ps6000SetDataBuffer`. `overflow`, returns a set of flags that indicate whether an overvoltage has occurred on any of the channels. It is a bit pattern with bit 0 denoting Channel A. `triggerAt`, an index to the buffer indicating the location of the trigger point relative to `startIndex`. This parameter is valid only when `triggered` is non-zero. `triggered`, a flag indicating whether a trigger occurred. If non-zero, a trigger occurred at the location indicated by `triggerAt`. `autoStop`, the flag that was set in the call to `ps6000RunStreaming`. `pParameter`, a void pointer passed from `ps6000GetStreamingLatestValues`. The callback function can write to this location to send any data, such as a status flag, back to the application.
Returns	nothing

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

3.58 Wrapper functions

The software development kits (SDKs) for PicoScope devices contain wrapper dynamic link library (DLL) files in the `lib` subdirectory of your SDK installation for 32-bit and 64-bit systems. The wrapper functions provided by the wrapper DLLs are for use with programming languages such as MathWorks MATLAB, National Instruments LabVIEW and Microsoft Excel VBA that do not support features of the C programming language such as callback functions.

The source code contained in the wrapper project contains a description of the functions and the input and output parameters.

Below we explain the sequence of calls required to capture data in streaming mode using the wrapper API functions.

The `ps6000Wrap.dll` wrapper DLL has a callback function for streaming data collection that copies data from the driver buffer specified to a temporary application buffer of the same size. To do this, the driver and application buffers must be registered with the wrapper and the corresponding channel(s) must be specified as being enabled. You should process the data in the temporary application buffer accordingly, for example by copying the data into a large array.

Procedure:

1. Open the oscilloscope using `ps6000OpenUnit`.

1a. Inform the wrapper of the number of channels on the device by calling `setChannelCount`.

2. Select channels, ranges and AC/DC coupling using `ps6000SetChannel`.

2a. Inform the wrapper which channels have been enabled by calling `setEnabledChannels`.

3. Use the appropriate trigger setup functions. For programming languages that do not support structures, use the wrapper's advanced trigger setup functions.

4. Call `ps6000SetDataBuffer` (or for aggregated data collection `ps6000SetDataBuffers`) to tell the driver where your data buffer(s) is(are).

4a. Register the data buffer(s) with the wrapper and set the application buffer(s) into which the data will be copied. Call `setAppAndDriverBuffers` (or `setMaxMinAppAndDriverBuffers` for aggregated data collection).

5. Start the oscilloscope running using `ps6000RunStreaming`.

6. Loop and call `GetStreamingLatestValues` and `IsReady` to get data and flag when the wrapper is ready for data to be retrieved.

6a. Call the wrapper's `AvailableData` function to obtain information on the number of samples collected and the start index in the buffer.

6b. Call the wrapper's `IsTriggerReady` function for information on whether a trigger has occurred and the trigger index relative to the start index in the buffer.

7. Process data returned to your application data buffers.

8. Call `AutoStopped` if the `autoStop` parameter has been set to `TRUE` in the call to `ps6000RunStreaming`.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

9. Repeat steps 6 to 8 until AutoStopped returns true or you wish to stop data collection.

10. Call ps6000Stop, even if the autoStop parameter was set to TRUE.

11. To disconnect a device, call ps6000CloseUnit.

4 Programming support and examples

Your Pico Technology SDK installation includes programming examples in various languages and development environments.

5 Numeric data types

Here is a list of the sizes and ranges of the numeric data types used in the PicoScope 6000 Series API.

Type	Bits	Signed or unsigned?
int16_t	16	signed
enum	32	enumerated
int32_t	32	signed
uint32_t	32	unsigned
float	32	signed (IEEE 754)
int64_t	64	signed

6 Enumerated types and constants

The enumerated types and constants used in the PicoScope 6000 Series API driver are defined in the file ps6000Api.h, which is included in the SDK. We recommend that you refer to these constants by name unless your programming language allows only numerical values.

7 Driver status codes

Every function in the ps6000 driver returns a **driver status code** from the list of PICO_STATUS values in the file PicoStatus.h, which is included in the Pico Technology SDK. Not all codes in PicoStatus.h apply to the PicoScope 6000 Series.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

8 Glossary

Callback. A mechanism that the PicoScope 6000 driver uses to communicate asynchronously with your application. At design time, you add a function (a *callback* function) to your application to deal with captured data. At run time, when you request captured data from the driver, you also pass it a pointer to your function. The driver then returns control to your application, allowing it to perform other tasks until the data is ready. When this happens, the driver calls your function in a new thread to signal that the data is ready. It is then up to your function to communicate this fact to the rest of your application.

Driver. A program that controls a piece of hardware. The driver for the PicoScope 6000 Series oscilloscopes is supplied in the form of a 32-bit Windows DLL, `ps6000.dll`. This is used by the PicoScope software, and by user-designed applications, to control the oscilloscopes.

PC Oscilloscope. A virtual instrument formed by connecting a PicoScope 6000 Series oscilloscope to a computer running the PicoScope software.

PicoScope 6000 Series. A range of PC Oscilloscopes from Pico Technology. The common features include 5 GS/s maximum sampling rate and 8-bit resolution. The scopes are available with a range of buffer sizes up to 2 GS.

PicoScope software. A software product that accompanies all Pico PC Oscilloscopes. It turns your PC into an oscilloscope, spectrum analyzer.

PRBS (pseudo-random binary sequence). A fixed, repeating sequence of binary digits that appears random when analyzed over a time shorter than the repeat period. The waveform swings between two values: logic high (binary 1) and logic low (binary 0).

USB 1.1. Universal Serial Bus (USB) is a standard port that enables you to connect external devices to PCs. A USB 1.1 port uses signaling speeds of up to 12 megabits per second, much faster than an RS-232 port.

USB 2.0. The second generation of USB interface. The port supports a data transfer rate of up to 480 megabits per second.

USB 3.0. A USB 3.0 port uses signaling speeds of up to 5 gigabits per second and is backwards-compatible with USB 2.0 and USB 1.1.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

Index

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

 Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.

Copyright © 2009–2016 Pico Technology Ltd. All rights reserved.