PicoScope® 5000 Series (A API)

Flexible Resolution Oscilloscopes

Programmer's Guide

ps5000apg.en r3

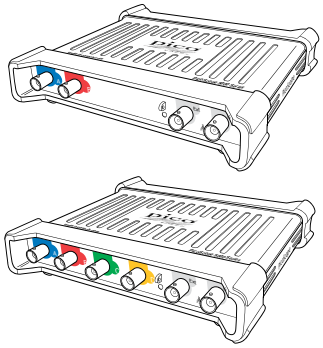

pico
Technology

Contents


pico
Technology

1 Welcome

The PicoScope 5000 A and B Series PC Oscilloscopes from Pico Technology are a range of high-specification, real-time measuring instruments that connect to the USB port of your computer. The series covers various options of portability, deep memory, fast sampling rates and high bandwidth, making it a highly versatile range that suits a wide range of applications. The oscilloscopes are all hi-speed USB 2.0 devices, also compatible with USB 1.1 and USB 3.0.



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

The information in this manual applies to the following oscilloscopes:

- PicoScope 5242A
  PicoScope 5243A
  PicoScope 5244A
  PicoScope 5442A
  PicoScope 5443A
  PicoScope 5444A

  The A models are high speed portable oscilloscopes, with a function generator.

- PicoScope 5242B
  PicoScope 5243B
  PicoScope 5244B
  PicoScope 5442B
  PicoScope 5443B
  PicoScope 5444B

  The B models are as the A models, but feature an arbitrary waveform generator and deeper memory.

For information on any PicoScope 5000 Series oscilloscope, refer to the documentation on our website.

2 Introduction

2.1 License agreement

Grant of license. The material contained in this release is licensed, not sold. Pico Technology Limited ('Pico') 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 and agree to abide by these conditions.

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

Copyright. The software in this release is for use only with Pico products or with data collected using Pico products. 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 and its agents shall not be liable for any loss or damage, howsoever caused, related to the use of Pico equipment or software, unless excluded by statute.

Fitness for purpose. No two applications are the same, so Pico cannot guarantee that its equipment or software is suitable for a given application. It is therefore the user's responsibility to ensure that the product is suitable for the user's application.

Mission-critical applications. Because the software runs on a computer that may be running other software products, and may be subject to interference from these other products, this license specifically excludes usage in 'mission-critical' applications, for example life-support systems.

Viruses. This software was continuously monitored for viruses during production. However, the user is responsible for virus checking the software once it is installed.

Support. No software is ever error-free, but if you are dissatisfied with the performance of this software, please contact our technical support staff.

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.

2.2 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.

2.3 System requirements

Using the Pico Technology SDK

To ensure that your PicoScope 5000 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 multicore processor.

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

USB

The ps5000a driver offers four different methods of recording data, all of which support USB 2.0 and USB 3.0 connections. The 5000 A and B Series oscilloscopes are all hi-speed USB 2.0 devices: the transfer rate will not increase by using USB 3.0.

3 Programming with the PicoScope 5000 Series (A API)

The ps5000a.dll dynamic link library in the lib subdirectory of your SDK installation allows you to program a PicoScope 5000 Series (A API) 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.

3.1 Driver

Your application will communicate with a PicoScope 5000 A API driver called ps5000a.dll, which is supplied in 32-bit and 64-bit versions. This driver is used by all the 5000 A/B Series oscilloscopes (but not the PicoScope 5203 and 5204). The driver exports the ps5000a 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 a low-level driver called WinUsb.sys. These 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.

3.2 Voltage ranges

You can set a device input channel to any voltage range from ±10 mV to ±20 V with the ps5000aSetChannel function. Each sample is scaled to 16 bits, and the minimum and maximum values returned to your application are given by ps5000aMinimumValue and ps5000aMaximumValue as follows:

Function	Voltage	Value returned	
		decimal	hex
8-bit			
ps5000aMaximumValue	maximum	+32 512	7F00
	zero	0	0000
ps5000aMinimumValue	minimum	–32 512	8100
12, 14, 15 and 16-bit			
ps5000aMaximumValue	maximum	+32 767	7FFF
	zero	0	0000
ps5000aMinimumValue	minimum	–32 767	8001

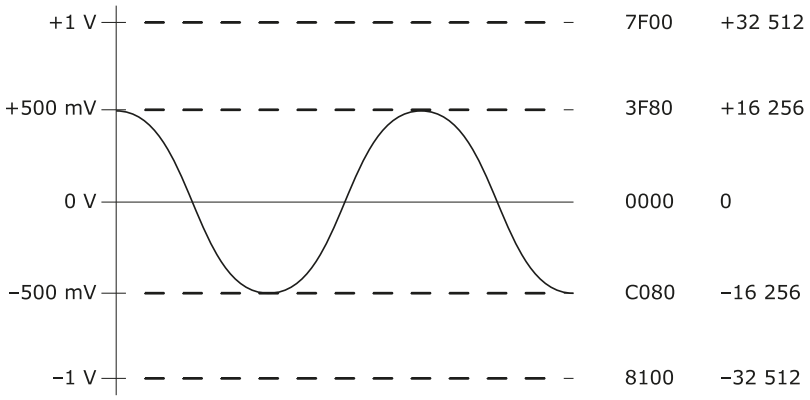
Example at 8-bit resolution

1. Call ps5000aSetChannel with range set to PS5000A_1V.

2. Apply a sinewave 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.



External trigger input

The external trigger input (marked EXT), where available, is scaled to a 16-bit value as follows:

Voltage	Constant	Digital value
–5 V	PS5000A_EXT_MIN_VALUE	–32 767
0 V		0
+5 V	PS5000A_EXT_MAX_VALUE	+32 767

3.3 Triggering

PicoScope 5000 Series 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 PicoScope 5000 trigger function `ps5000aSetSimpleTrigger`, which in turn calls:

- `ps5000aSetTriggerChannelConditions`
- `ps5000aSetTriggerChannelDirections`
- `ps5000aSetTriggerChannelProperties`

These can also be called individually, rather than using `ps5000aSetSimpleTrigger` in order to set up advanced trigger types such as pulse width.

A trigger event can occur when one of the signal or trigger 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, `ps5000aSetPulseWidthQualifier`.

3.4 Sampling modes

PicoScope 5000 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 entire blocks 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 supports downsampling and triggering, while providing fast streaming at up to:

  8-bit mode
  - 7.8125 MS/s (128 ns per sample) when three or four channels are active
  - 15.625 MS/s (64 ns per sample) when two channels are active
  - 31.25 MS/s (32 ns per sample) when one channel is active

  12, 14, 15, and 16-bit modes*
  - 3.906 MS/s (256 ns per sample) when three or four channels are active
  - 7.8125 MS/s (128 ns per sample) when two channels are active
  - 15.625 MS/s (64 ns per sample) when one channel is active

  * 15-bit mode supports a maximum of two channels. 16-bit mode supports only one channel.

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.

For compatibility of programming environments not supporting callback, polling of the driver is available in block mode.

Note: The Oversampling feature has been replaced by `PS5000A_RATIO_MODE_AVERAGE`*.*

3.4.1 Block mode

In **block mode**, the computer prompts a PicoScope 5000 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 `ps5000aMemorySegments`).
- **Sampling rate.** A PicoScope 5000 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 5000 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 `ps5000aRunBlock`, `ps5000aStop` and `ps5000aGetValues`.
- **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.
- **Segmented memory.** The scope's internal memory can be divided into segments so that you can capture several waveforms in succession. Configure this using `ps5000aMemorySegments`.
- **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 or the power source is changed (for flexible power devices).

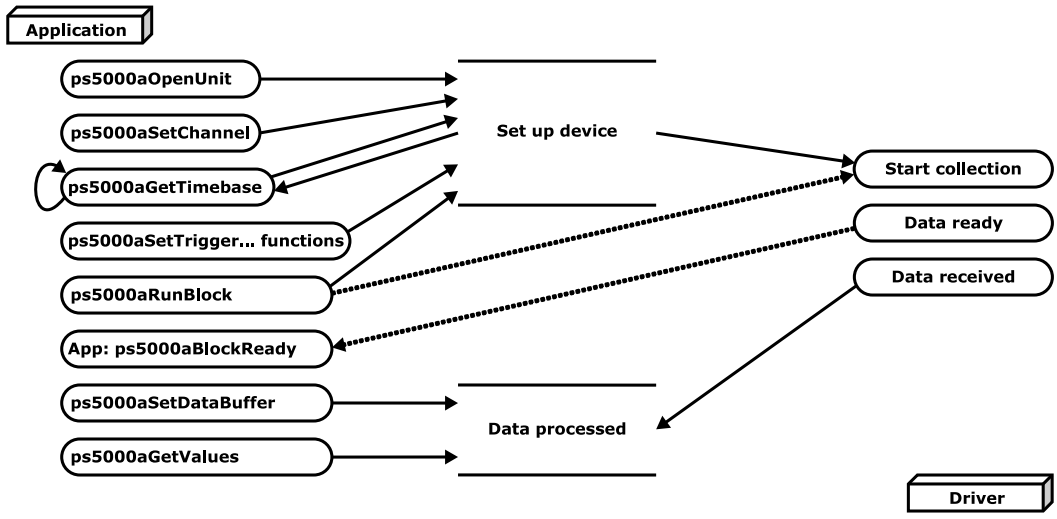
See Using block mode for programming details.

3.4.1.1 Using block mode

You can use **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: see **rapid block mode example 1** for an example of this.

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

1. Open the oscilloscope using `ps5000aOpenUnit`.
2. Select channel ranges and AC/DC coupling using `ps5000aSetChannel`.
3. Using `ps5000aGetTimebase`, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions `ps5000aSetTriggerChannelConditions`, `ps5000aSetTriggerChannelDirections` and `ps5000aSetTriggerChannelProperties` to set up the trigger if required.
5. Start the oscilloscope running using `ps5000aRunBlock`.
6. Wait until the oscilloscope is ready using the `ps5000aBlockReady` callback (or poll using `ps5000aIsReady`).
7. Use `ps5000aSetDataBuffer` to tell the driver where your memory buffer is. For greater efficiency when doing multiple captures, you can call this function outside the loop, after step 4.
8. Transfer the block of data from the oscilloscope using `ps5000aGetValues`.
9. Display the data.
10. Stop the oscilloscope using `ps5000aStop`.
11. Repeat steps 5 to 9.
12. Request new views of stored data using different downsampling parameters: see Retrieving stored data.
13. Close the device using `ps5000aCloseUnit`.



Note that if you use `ps5000aGetValues` or `ps5000aStop` before the oscilloscope is ready, no capture will be available and the driver will return `PICO_NO_SAMPLES_AVAILABLE`.

3.4.1.2 Asynchronous calls in block mode

The ps5000aGetValues function may take a long time to complete if a large amount of data is being collected. For example, it can take 14 seconds (or several minutes on USB 1.1) to retrieve the full 512 megasamples (in 8-bit mode) from a PicoScope 5444B using a USB 2.0 connection. To avoid hanging the calling thread, it is possible to call ps5000aGetValuesAsync instead. This immediately returns control to the calling thread, which then has the option of waiting for the data or calling ps5000aStop to abort the operation.

3.4.2 Rapid block mode

In normal block mode, the PicoScope 5000 Series scopes collect one waveform at a time. You start the 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 2 microseconds (on fastest timebase).

See Using rapid block mode for details.

3.4.2.1 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 `ps5000aOpenUnit`.
2. Select channel ranges and AC/DC coupling using `ps5000aSetChannel`.
3. Set the number of memory segments equal to or greater than the number of captures required using `ps5000aMemorySegments`. Use `ps5000aSetNoOfCaptures` before each run to specify the number of waveforms to capture.
4. Using `ps5000aGetTimebase`, select timebases until the required nanoseconds per sample is located. This will indicate the number of samples per channel available for each segment.
5. Use the trigger setup functions `ps5000aSetTriggerChannelConditions`, `ps5000aSetTriggerChannelDirections` and `ps5000aSetTriggerChannelProperties` to set up the trigger if required.
6. Start the oscilloscope running using `ps5000aRunBlock`.

   THEN EITHER

7a. To obtain data before rapid block capture has finished, call `ps5000aStop` and then `ps5000aGetNoOfCaptures` to find out how many captures were completed.

   OR

7b. Wait until the oscilloscope is ready using `ps5000aIsReady`.

   OR

7c. Wait on the callback function.

8. Use `ps5000aSetDataBuffer` 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 when doing multiple captures, you can call this function outside the loop, after step 5.
9. Transfer the blocks of data from the oscilloscope using `ps5000aGetValuesBulk` (or `ps5000aGetValues` to retrieve one buffer at a time). These functions stop the oscilloscope.
10. Retrieve the time offset for each data segment using `ps5000aGetValuesTriggerTimeOffsetBulk64`.
11. Display the data.
12. Repeat steps 6 to 11 if necessary.
13. Call `ps5000aStop` (usually unnecessary as the scope stops automatically in most cases, but recommended as a precaution).
14. Close the device using `ps5000aCloseUnit`.

With aggregation

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

8a. Call ps5000aSetDataBuffer or (ps5000aSetDataBuffers) to set up one pair of buffers for every waveform segment required.
9a. Call ps5000aGetValuesBulk for each pair of buffers.
10a. Retrieve the time offset for each data segment using ps5000aGetValuesTriggerTimeOffsetBulk64.

Continue from step 11 above.

3.4.2.2 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
ps5000aSetNoOfCaptures (handle, MAX_WAVEFORMS);

pParameter = false;
ps5000aRunBlock
(
  handle,
  0, // noOfPreTriggerSamples
  10000, // noOfPostTriggerSamples
  1, // timebase to be used
  &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);

int16_t buffer[PS5000A_MAX_CHANNELS][MAX_WAVEFORMS][MAX_SAMPLES];

for (int32_t i = 0; i < 20; i++)
{
  for (int32_t c = PS5000A_CHANNEL_A; c <= PS5000A_CHANNEL_B; c++)
  {
    ps5000aSetDataBuffer
    (
      handle,
      c,
      buffer[c][i],
      MAX_SAMPLES,
      i
      PS5000A_RATIO_MODE_NONE
    );
  }
}
```

Comments: buffer has been created as a three-dimensional 16-bit integer array, which will contain 1000 samples as defined by `MAX_SAMPLES`. There are only 20 buffers set, but it is possible to set up to the number of captures you have requested. `PS5000A_RATIO_MODE_NONE` can be substituted for `PS5000A_RATIO_MODE_AGGREGATE`, `PS5000A_RATIO_MODE_DECIMATE`, or `PS5000A_RATIO_MODE_AVERAGE`.


```
int16_t overflow[MAX_WAVEFORMS];

ps5000aGetValuesBulk
(
  handle,
  &noOfSamples, // set to MAX_SAMPLES on entering the function
  10, // fromSegmentIndex
  19, // toSegmentIndex
  1, // downsampling ratio
  PS5000A_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 `ps5000aRunBlock`. The samples are always returned from the first sample taken, unlike the `ps5000aGetValues` function which allows the sample index to be set. The above segments start at 10 and finish at 19 inclusive. It is possible for the `fromSegmentIndex` to wrap around to the `toSegmentIndex`, by setting the `fromSegmentIndex` to 98 and the `toSegmentIndex` to 7.

```
int64_t times[MAX_WAVEFORMS];
PS5000A_TIME_UNITS timeUnits[MAX_WAVEFORMS];

ps5000aGetValuesTriggerTimeOffsetBulk64
(
  handle,
  times, // indices 10 to 19 will be populated
  timeUnits, // indices 10 to 19 will be populated
  10, // fromSegmentIndex, inclusive
  19, // toSegmentIndex, inclusive
)
```

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.

3.4.2.3 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
ps5000aSetNoOfCaptures (handle, MAX_WAVEFORMS);

pParameter = false;
ps5000aRunBlock
(
  handle,
  0, // noOfPreTriggerSamples,
  1000000, // noOfPostTriggerSamples,
  1, // timebase to be used,
  &timeIndisposedMs,
  lpReady,
  &pParameter
);
```

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

```
for (int32_t segment = 10; segment < 20; segment++)
{
  for (int32_t c = PS5000A_CHANNEL_A; c <= PS5000A_CHANNEL_D; c++)
  {
    ps5000aSetDataBuffers
    (
      handle,
      c,
      bufferMax[c],
      bufferMin[c]
      MAX_SAMPLES
      1,
      PS5000A_RATIO_MODE_AGGREGATE
    );
  }
```



```
    ps5000aGetValues
    (
      handle,
      0,
      &noOfSamples, // set to MAX_SAMPLES on entering
      1000,
      downSampleRatioMode, // set to RATIO_MODE_AGGREGATE
      index,
      overflow
    );

    ps5000aGetTriggerTimeOffset64
    (
      handle,
      &time,
      &timeUnits,
      index
    )
  }
```

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

3.4.3 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 ps5000a set of trigger functions and the `ps5000aSetEts` function.

- **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 User's Guide 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 calls the `ps5000aBlockReady` callback function when a new waveform is ready for collection. The `ps5000aGetValues` function needs to be called for the waveform to be retrieved.

Applicability	Available in block mode only. Not suitable for one-shot (non-repetitive) signals. Aggregation is not supported. Edge-triggering only. Auto trigger delay (`autoTriggerMilliseconds`) is ignored.

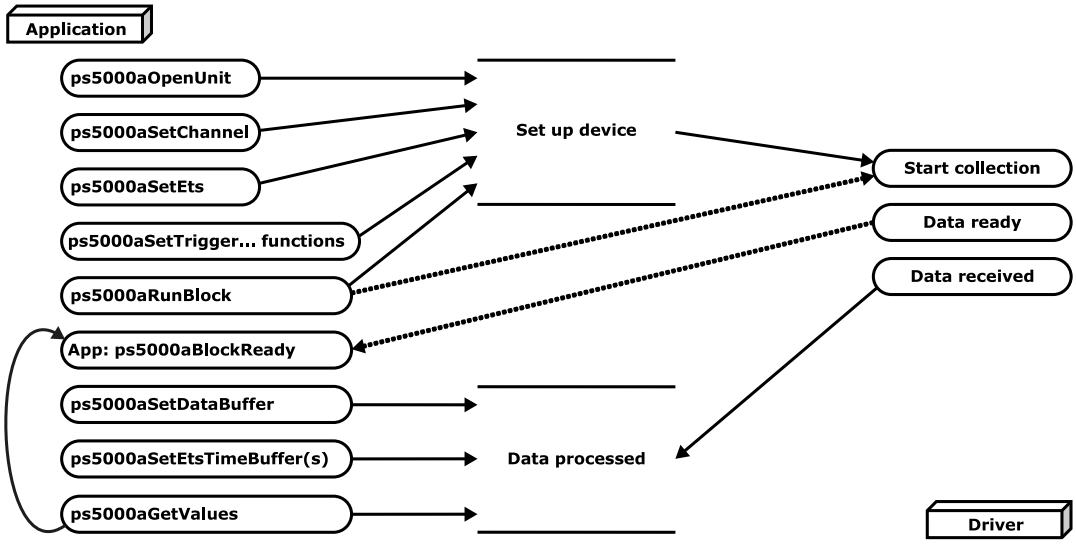
3.4.3.1 Using ETS mode

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

1. Open the oscilloscope using ps5000aOpenUnit.
2. Select channel ranges and AC/DC coupling using ps5000aSetChannel.
3. Use ps5000aGetTimebase to verify the number of samples to be collected.
4. Set up ETS using ps5000aSetEts.
5. Use the trigger setup functions ps5000aSetTriggerChannelDirections and ps5000aSetTriggerChannelProperties to set up the trigger if required.
6. Start the oscilloscope running using ps5000aRunBlock.
7. Wait until the oscilloscope is ready using the ps5000aBlockReady callback (or poll using ps5000aIsReady).
8. Use ps5000aSetDataBuffer to tell the driver where to store sampled data.

8a. Use ps5000aSetEtsTimeBuffer or ps5000aSetEtsTimeBuffers to tell the driver where to store sample times.

9. Transfer the block of data from the oscilloscope using ps5000aGetValues.
10. Display the data.
11. While you want to collect updated captures, repeat steps 7 to 10.
12. Stop the oscilloscope using ps5000aStop.
13. Repeat steps 6 to 12.
14. Close the device using ps5000aCloseUnit.



3.4.4 Streaming mode

Streaming mode can capture data without the gaps that occur between blocks when using block mode. Streaming mode supports downsampling and triggering, while providing fast streaming at up to 31.25 MS/s (32 ns per sample) when one channel is active, depending on the computer's performance. 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 used per channel. When aggregation is set above 1 then two buffers (maximum and minimum) per channel are used.

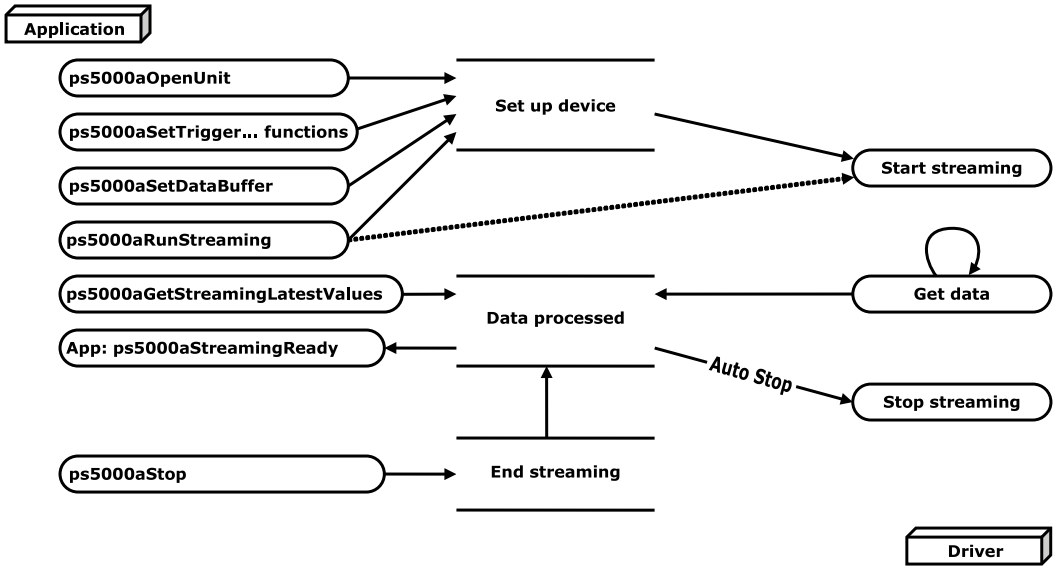
- **Memory segmentation.** The memory can be divided into segments to reduce the latency of data transfers to the PC. However, this increases the risk of losing data if the PC cannot keep up with the device's sampling rate.

See Using streaming mode for programming details.

3.4.4.1 Using streaming mode

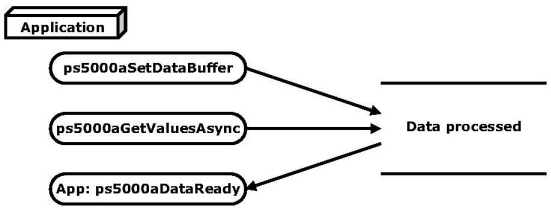
This is the general procedure for reading and displaying data in streaming mode using a single memory segment:

1. Open the oscilloscope using `ps5000aOpenUnit`.
2. Select channels, ranges and AC/DC coupling using `ps5000aSetChannel`.
3. Use the trigger setup functions `ps5000aSetTriggerChannelDirections` and `ps5000aSetTriggerChannelProperties` to set up the trigger if required.
4. Call `ps5000aSetDataBuffer` to tell the driver where your data buffer is.
5. Set up aggregation and start the oscilloscope running using `ps5000aRunStreaming`.
6. Call `ps5000aGetStreamingLatestValues` to get data.
7. Process data returned to your application's function. This example is using Auto Stop, so after the driver has received all the data points requested by the application, it stops the device streaming.
8. Call `ps5000aStop`, even if Auto Stop is enabled.
9. Request new views of stored data using different downsampling parameters: see Retrieving stored data.
10. Close the device using `ps5000aCloseUnit`.



3.4.5 Retrieving stored data

You can collect data from the ps5000a driver with a different downsampling factor when `ps5000aRunBlock` or `ps5000aRunStreaming` has already been called and has successfully captured all the data. Use `ps5000aGetValuesAsync`.



3.5 Timebases

The API allows you to select any of 2^{32} different timebases based on the maximum sampling rate* of your oscilloscope. 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. Calculate the timebase using the ps5000aGetTimebase call. Accepted timebases for each resolution mode are:

8-bit resolution

Timebase	Sample interval formula	Sample interval examples	Notes
0	$2^{timebase}$ / 1,000,000,000	1 ns	Only one channel enabled
1		2 ns	
2		4 ns	
3 to $2^{32}-1$	(timebase–2) / 125,000,000	3 => 8 ns ... $2^{32}-1$ => ~ 34.36 s	

12-bit resolution

Timebase**	Sample interval formula	Sample interval examples	Notes
1	$2^{(timebase-1)}$ / 500,000,000	2 ns	Only one channel enabled
2		4 ns	
3		8 ns	
4 to $2^{32}-2$	(timebase–3) / 62,500,000	4 => 16 ns ... $2^{32}-2$ => ~ 68.72 s	

14, 15-bit resolutions

Timebase†	Sample interval formula	Sample interval examples	Notes
3 to $2^{32}-1$	(timebase–2) / 125,000,000	3 => 8 ns 4 => 16 ns 5 => 24 ns ... $2^{32}-1$ => ~ 34.36 s	Only one channel enabled

16-bit resolution

Timebase‡	Sample interval formula	Sample interval examples	Notes
4 to $2^{32}-2$	(timebase–3) / 62,500,000	4 => 16 ns 5 => 32 ns 6 => 48 ns ... $2^{32}-2$ => ~ 68.72 s	Only one channel enabled

* The fastest available sampling rate may depend on which channels are enabled and on the sampling mode. Please refer to the oscilloscope data sheet for sampling rate specifications. In streaming mode, the speed of the USB port may affect the rate of data transfer.

** Timebase 0 is not available in 12-bit resolution mode.

† Timebases 0, 1 and 2 are not available in 14 and 15-bit resolution modes.

‡ Timebases 0, 1, 2 and 3 are not available in 16-bit resolution mode.

ETS mode

In ETS mode the sample time is not set according to the above tables, but is instead calculated and returned by ps5000aSetEts.

3.6 Power options

The 4-channel 5000 Series oscilloscopes allow you to choose from two different methods of powering your device. Our flexible power feature offers the choice of powering your device using a single-headed USB cable and provided power supply unit, or using our double-headed USB cable to draw power from two powered USB ports for use in 2-channel mode. If the power source is changed (i.e. AC adaptor being connected or disconnected) while the oscilloscope is in operation, the oscilloscope will restart automatically and any unsaved data will be lost.

For further information on these options, refer to the documentation included with your device.

Power options functions

The following functions support the flexible power feature:

- ps5000aChangePowerSource
- ps5000aCurrentPowerSource

If you want the device to run on USB power only, instruct the driver by calling ps5000aChangePowerSource after calling ps5000aOpenUnit. If ps5000aOpenUnit is called without the power supply connected, the driver returns PICO_POWER_SUPPLY_NOT_CONNECTED. If the supply is connected or disconnected during use, the driver will return the relevant status code and you must then call ps5000aChangePowerSource to continue running the scope.

3.7 Combining several oscilloscopes

It is possible to collect data using up to 64 PicoScope 5000 Series oscilloscopes at the same time, depending on the capabilities of the PC. Each oscilloscope must be connected to a separate USB port. The ps5000aOpenUnit 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 ps5000aBlockReady(...)
// define callback function specific to application

handle1 = ps5000aOpenUnit()
handle2 = ps5000aOpenUnit()

ps5000aSetChannel(handle1)
// set up unit 1
ps5000aRunBlock(handle1)

ps5000aSetChannel(handle2)
// set up unit 2
ps5000aRunBlock(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
```

4 API functions

The ps5000a 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.

ps5000aBlockReady	indicate when block-mode data ready
ps5000aChangePowerSource	configures the unit's power source
ps5000aCloseUnit	close a scope device
ps5000aCurrentPowerSource	indicate the current power state of the device
ps5000aDataReady	indicate when post-collection data ready
ps5000aEnumerateUnits	find all connected oscilloscopes
ps5000aFlashLed	flash the front-panel LED
ps5000aGetAnalogueOffset	query the permitted analog offset range
ps5000aGetChannelInformation	queries which ranges are available on a device
ps5000aGetDeviceResolution	retrieves the resolution specified device will run
ps5000aGetMaxDownSampleRatio	query the aggregation ratio for data
ps5000aGetMaxSegments	query the maximum number of segments
ps5000aGetNoOfCaptures	find out how many captures are available
ps5000aGetNoOfProcessedCaptures	query number of captures processed
ps5000aGetStreamingLatestValues	get streaming data while scope is running
ps5000aGetTimebase	find out what timebases are available
ps5000aGetTimebase2	find out what timebases are available
ps5000aGetTriggerTimeOffset	find out when trigger occurred (32-bit)
ps5000aGetTriggerTimeOffset64	find out when trigger occurred (64-bit)
ps5000aGetUnitInfo	read information about scope device
ps5000aGetValues	retrieve block-mode data with callback
ps5000aGetValuesAsync	retrieve streaming data with callback
ps5000aGetValuesBulk	retrieve data in rapid block mode
ps5000aGetValuesOverlapped	set up data collection ahead of capture
ps5000aGetValuesOverlappedBulk	set up data collection in rapid block mode
ps5000aGetValuesTriggerTimeOffsetBulk	get rapid-block waveform timings (32-bit)
ps5000aGetValuesTriggerTimeOffsetBulk64	get rapid-block waveform timings (64-bit)
ps5000aIsReady	poll driver in block mode
ps5000aIsTriggerOrPulseWidthQualifierEnabled	find out whether trigger is enabled
ps5000aMaximumValue	query the max. ADC count in GetValues calls
ps5000aMemorySegments	divide scope memory into segments
ps5000aMinimumValue	query the min. ADC count in GetValues calls
ps5000aNoOfStreamingValues	get number of samples in streaming mode
ps5000aOpenUnit	open a scope device
ps5000aOpenUnitAsync	open a scope device without waiting
ps5000aOpenUnitProgress	check progress of OpenUnit call
ps5000aPingUnit	check communication with device
ps5000aRunBlock	start block mode
ps5000aRunStreaming	start streaming mode
ps5000aSetBandwidthFilter	specifies the bandwidth limit
ps5000aSetChannel	set up input channels
ps5000aSetDataBuffer	register data buffer with driver
ps5000aSetDataBuffers	register aggregated data buffers with driver
ps5000aSetDeviceResolution	sets the resolution a specified device will run
ps5000aSetEts	set up equivalent-time sampling
ps5000aSetEtsTimeBuffer	set up buffer for ETS timings (64-bit)
ps5000aSetEtsTimeBuffers	set up buffer for ETS timings (32-bit)
ps5000aSetNoOfCaptures	set number of captures to collect in one run
ps5000aSetPulseWidthQualifier	set up pulse width triggering
ps5000aSetSigGenArbitrary	set up arbitrary waveform generator
ps5000aSetSigGenBuiltIn	set up standard signal generator
ps5000aSetSigGenPropertiesArbitrary	change AWG settings
ps5000aSetSigGenPropertiesBuiltIn	change function generator settings
ps5000aSetSimpleTrigger	set up level triggers only
ps5000aSetTriggerChannelConditions	specify which channels to trigger on
ps5000aSetTriggerChannelDirections	set up signal polarities for triggering
ps5000aSetTriggerChannelProperties	set up trigger thresholds

ps5000aSetTriggerDelay	set up post-trigger delay
ps5000aSigGenArbitraryMinMaxValues	get AWG parameters
ps5000aSigGenFrequencyToPhase	convert frequency to phase count
ps5000aSigGenSoftwareControl	trigger the signal generator
ps5000aStop	stop data capture
ps5000aStreamingReady	indicate when streaming-mode data ready

4.1 ps5000aBlockReady (callback)

```
typedef void (CALLBACK *ps5000aBlockReady)
(
  int16_t         handle,
  PICO_STATUS     status,
  void            * pParameter
)
```

This callback function is part of your application. You register it with the ps5000a driver using ps5000aRunBlock, and the driver calls it back when block-mode data is ready. You can then download the data using the ps5000aGetValues function.

Applicability	Block mode only
Arguments	handle, the handle of the device returning the samples. status, indicates whether an error occurred during collection of the data. * pParameter, a void pointer passed from ps5000aRunBlock. Your callback function can write to this location to send any data, such as a status flag, back to your application.
Returns	nothing

4.2 ps5000aChangePowerSource

```
PICO_STATUS ps5000aChangePowerSource
(
  int16_t         handle,
  PICO_STATUS     powerstate
)
```

This function selects the power supply mode. If USB power is required, you must explicitly allow it by calling this function. If the AC power adapter is connected or disconnected during use, you must also call this function. If you change power source to PICO_POWER_SUPPLY_NOT_CONNECTED and channels C/D are currently enabled, they will be switched off. If a trigger is set using channels C/D the trigger settings for those channels will also be removed.

Applicability	All modes. 4-Channel 5000 A and B Series oscilloscopes only
Arguments	handle, the handle of the device. powerstate, the required state of the unit. Either PICO_POWER_SUPPLY_CONNECTED or PICO_POWER_SUPPLY_NOT_CONNECTED.
Returns	PICO_OK PICO_POWER_SUPPLY_REQUEST_INVALID PICO_INVALID_PARAMETER PICO_NOT_RESPONDING PICO_INVALID_HANDLE

4.3 ps5000aCloseUnit

```
PICO_STATUS ps5000aCloseUnit
(
  int16_t handle
)
```

This function shuts down the PicoScope 5000 Series oscilloscope.

Applicability	All modes
Arguments	`handle`, the handle, returned by `ps5000aOpenUnit`, of the scope device to be closed.

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

4.4 ps5000aCurrentPowerSource

```
PICO_STATUS ps5000aCurrentPowerSource
(
  int16_t handle
)
```

This function returns the current power state of the device.

Applicability	All modes. 4-Channel 5000 A and B Series oscilloscopes only
Arguments	`handle`, the handle of the device
Returns	`PICO_INVALID_HANDLE` - handle of the device is not recognised. `PICO_POWER_SUPPLY_CONNECTED` - if the device is powered by the AC adapter. `PICO_POWER_SUPPLY_NOT_CONNECTED` - if the device is powered by the USB cable.

4.5 ps5000aDataReady (callback)

```
typedef void (CALLBACK *ps5000aDataReady)
(
  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 ps5000aGetValuesAsync, and the driver calls your function back when the data is ready.

Applicability	All modes
Arguments	`handle,` the handle of the device returning the samples. `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 ps5000aGetValuesAsync. 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

4.6 ps5000aEnumerateUnits

```
PICO_STATUS ps5000aEnumerateUnits
(
  int16_t    * count,
  int8_t     * serials,
  int16_t    * serialLth
)
```

This function counts the number of PicoScope 5000 Series 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 5000 Series 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_CONFIG_FAIL_AWG`
	`PICO_INITIALISE_FPGA`

4.7 ps5000aFlashLed

```
PICO_STATUS ps5000aFlashLed
(
  int16_t handle,
  int16_t start
)
```

This function flashes the LED on the front of the scope without blocking the calling thread. Calls to ps5000aRunStreaming and ps5000aRunBlock 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`, the handle of the scope 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`

4.8 ps5000aGetAnalogueOffset

```
PICO_STATUS ps5000aGetAnalogueOffset
(
  int16_t                handle,
  PS5000A_RANGE          range,
  PS5000A_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	AI models
Arguments	`handle`, the value returned from opening the device. `range`, the voltage range to be used when gathering the min and max information. `coupling`, the type of AC/DC coupling used. `* maximumVoltage`, a pointer to a float, an out parameter set to the maximum voltage allowed for the range, may be `NULL`. `* minimumVoltage`, a pointer to a float, an out parameter set to the minimum voltage allowed for the range, may be `NULL`. If both `maximumVoltage` and `minimumVoltage` are set to `NULL`, the driver will return `PICO_NULL_PARAMETER`.

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

4.9 ps5000aGetChannelInformation

```
PICO_STATUS ps5000aGetChannelInformation
(
  int16_t                    handle,
  PS5000A_CHANNEL_INFO       info,
  int32_t                    probe,
  int32_t                  * ranges,
  int32_t                  * length,
  int32_t                    channels
)
```

This function queries which ranges are available on a scope device.

Applicability	All modes
Arguments	`handle,` the handle of the required device. `info,` the type of information required. The following value is currently supported: `PS5000A_CI_RANGES` `probe,` not used, must be set to 0. `* ranges,` an array that will be populated with available `PS5000A_RANGE` values for the given `info`. If `NULL`, `length` is set to the number of `ranges` available. `* length,` on input: the length of the `ranges` array; on output: the number of elements written to the `ranges` array. `channels,` the channel for which the information is required.

Returns	`PICO_OK`
	`PICO_HANDLE_INVALID`
	`PICO_BUSY`
	`PICO_DRIVER_FUNCTION`
	`PICO_NOT_RESPONDING`
	`PICO_NULL_PARAMETER`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_INFO`

4.10 ps5000aGetDeviceResolution

```
PICO_STATUS ps5000aGetDeviceResolution
(
  int16_t                          handle,
  PS5000A_DEVICE_RESOLUTION     * resolution
)
```

This function retrieves the resolution the specified device will run in.

Applicability	All modes
Arguments	`handle`, the handle of the required device `* resolution`, returns the resolution of the device, values are one of the `PS5000A_DEVICE_RESOLUTION`.

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

4.11 ps5000aGetMaxDownSampleRatio

```
PICO_STATUS ps5000aGetMaxDownSampleRatio
(
  int16_t                 handle,
  uint32_t                noOfUnaggregatedSamples,
  uint32_t              * maxDownSampleRatio,
  PS5000A_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`, the handle of the required device `noOfUnaggregatedSamples`, the number of unprocessed samples to be downsampled `* maxDownSampleRatio`: the maximum possible downsampling ratio output `downSampleRatioMode`: the downsampling mode. See `ps5000aGetValues` `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`

4.12 ps5000aGetMaxSegments

```
PICO_STATUS ps5000aGetMaxSegments
(
  int16_t      handle,
  uint32_t   * maxsegments
)
```

This function returns the maximum number of segments allowed for the opened device. Refer to ps5000aMemorySegments for specific figures.

Applicability	All modes
Arguments	`handle`, the value returned from opening the device. `* maxsegments`, (output) the maximum number of segments allowed.

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

4.13 ps5000aGetNoOfCaptures

```
PICO_STATUS ps5000aGetNoOfCaptures
(
  int16_t    handle,
  uint32_t * nCaptures
)
```

This function returns the number of captures the device has made in rapid block mode, since you called ps5000aRunBlock. You can call ps5000aGetNoOfCaptures during device capture, after collection has completed or after interrupting waveform collection by calling ps5000aStop. The returned value (nCaptures) can then be used to iterate through the number of segments using ps5000aGetValues, or in a single call to ps5000aGetValuesBulk, where it is used to calculate the toSegmentIndex parameter.

Applicability	Rapid block mode
Arguments	handle, handle of the required device. * nCaptures, output: the number of available captures that has been collected from calling ps5000aRunBlock.
Returns	PICO_OK PICO_DRIVER_FUNCTION PICO_INVALID_HANDLE PICO_NOT_RESPONDING PICO_NO_SAMPLES_AVAILABLE PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_TOO_MANY_SAMPLES

4.14 ps5000aGetNoOfProcessedCaptures

```
PICO_STATUS ps5000aGetNoOfProcessedCaptures
(
  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 ps5000aRunBlock. It is for use in rapid block mode, alongside the ps5000aGetValuesOverlappedBulk function, when the driver is set to transfer data from the device automatically as soon as the ps5000aRunBlock function is called. You can call ps5000aGetNoOfProcessedCaptures during device capture, after collection has completed or after interrupting waveform collection by calling ps5000aStop.

The returned value (nProcessedCaptures) can then be used to iterate through the number of segments using ps5000aGetValues, or in a single call to ps5000aGetValuesBulk, where it is used to calculate the toSegmentIndex parameter.

When capture is stopped

If nProcessedCaptures = 0, you will also need to call ps5000aGetNoOfCaptures, in order to determine how many waveform segments were captured, before calling ps5000aGetValues or ps5000aGetValuesBulk.

Applicability	Rapid block mode, using ps5000aGetValuesOverlapped.
Arguments	handle: handle of the required device. * nProcessedCaptures, output: the number of available captures that has been collected from calling ps5000aRunBlock.
Returns	PICO_OK PICO_DRIVER_FUNCTION PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_TOO_MANY_SAMPLES

4.15 ps5000aGetStreamingLatestValues

```
PICO_STATUS ps5000aGetStreamingLatestValues
(
  int16_t                 handle,
  ps5000aStreamingReady   lpPs5000aReady,
  void                  * pParameter
)
```

This function instructs the driver to return the next block of values to your ps5000aStreamingReady callback function. You must have previously called ps5000aRunStreaming beforehand to set up streaming.

Applicability	Streaming mode only
Arguments	handle, the handle of the required device. lpPs5000AReady, a pointer to your ps5000aStreamingReady callback function. * pParameter, a void pointer that will be passed to the ps5000aStreamingReady callback function. The callback function may optionally use this pointer to return information to the application.
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_INVALID_CALL PICO_BUSY PICO_NOT_RESPONDING PICO_DRIVER_FUNCTION

4.16 ps5000aGetTimebase

```
PICO_STATUS ps5000aGetTimebase
(
  int16_t    handle,
  uint32_t   timebase,
  int32_t    noSamples,
  int32_t  * timeIntervalNanoseconds,
  int32_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 ps5000aSetChannel.

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 ps5000aGetTimebase2 instead.

To use ps5000aGetTimebase or ps5000aGetTimebase2, first estimate the timebase number that you require using the information in the timebase guide. Next, call one of these functions with the timebase that you have just chosen and verify that the timeIntervalNanoseconds argument that the function returns is the value that you require. You may need to iterate this process until you obtain the time interval that you need.

Applicability	All modes
Arguments	handle, the handle of the required device. timebase, see timebase guide noSamples, the number of samples required. * timeIntervalNanoseconds, on exit, the time interval between readings at the selected timebase. Use NULL if not required. * maxSamples, on exit, the maximum number of samples available. The scope reserves some 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

4.17 ps5000aGetTimebase2

```
PICO_STATUS ps5000aGetTimebase2
(
  int16_t    handle,
  uint32_t   timebase,
  int32_t    noSamples,
  float    * timeIntervalNanoseconds,
  int32_t  * maxSamples,
  uint32_t   segmentIndex
)
```

This function is an upgraded version of ps5000aGetTimebase, and returns the time interval as a `float` rather than an `int32_t`. This allows it to return sub-nanosecond time intervals. See ps5000aGetTimebase for a full description.

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 ps5000aGetTimebase.
Returns	See ps5000aGetTimebase.

4.18 ps5000aGetTriggerTimeOffset

```
PICO_STATUS ps5000aGetTriggerTimeOffset
(
  int16_t                  handle,
  uint32_t               * timeUpper,
  uint32_t               * timeLower,
  PS5000A_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, `ps5000aGetTriggerTimeOffset64`, is available that returns the time as a single 64-bit value.

Applicability	Block mode, rapid block mode

Arguments	`handle`, the handle of the required 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:
	`PS5000A_FS`
	`PS5000A_PS`
	`PS5000A_NS`
	`PS5000A_US`
	`PS5000A_MS`
	`PS5000A_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_NOT_USED_IN_THIS_CAPTURE_MODE`
	`PICO_NOT_RESPONDING`
	`PICO_NULL_PARAMETER`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_DRIVER_FUNCTION`

4.19 ps5000aGetTriggerTimeOffset64

```
PICO_STATUS ps5000aGetTriggerTimeOffset64
(
  int16_t                 handle,
  int64_t               * time,
  PS5000A_TIME_UNITS    * timeUnits,
  uint32_t                segmentIndex
)
```

This function gets the trigger time offset for a waveform. It is equivalent to ps5000aGetTriggerTimeOffset 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, the handle of the required 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: PS5000A_FS PS5000A_PS PS5000A_NS PS5000A_US PS5000A_MS PS5000A_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_NOT_USED_IN_THIS_CAPTURE_MODE PICO_NOT_RESPONDING PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_DRIVER_FUNCTION

4.20 ps5000aGetUnitInfo

```
PICO_STATUS ps5000aGetUnitInfo
(
  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, or no device is opened only the driver version is available.

Applicability	All modes
Arguments	`handle,` the handle of the device from which information is required. If an invalid handle is passed, only the driver versions can be read. `* 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 8-bit integers (`int8_t`) 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 ps5000a.dll	1,0,0,1
1	PICO_USB_VERSION Type of USB connection to device: 1.1, 2.0 or 3.0	2.0
2	PICO_HARDWARE_VERSION Hardware version of device	1
3	PICO_VARIANT_INFO Variant number of device	5444B
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	1.0
7	PICO_DIGITAL_HARDWARE_VERSION Hardware version of the digital section	1
8	PICO_ANALOGUE_HARDWARE_VERSION Hardware version of the analog section	1

4.21 ps5000aGetValues

```
PICO_STATUS ps5000aGetValues
(
  int16_t                 handle,
  uint32_t                startIndex,
  uint32_t              * noOfSamples,
  uint32_t                downSampleRatio,
  PS5000A_RATIO_MODE      downSampleRatioMode,
  uint32_t                segmentIndex,
  int16_t               * overflow
)
```

This function returns block-mode data from the oscilloscope's buffer memory, with or without downsampling, starting at the specified sample number. It is used to get the stored data after data collection has stopped.

Note that if you are using block mode and call this function before the oscilloscope is ready, no capture will be available and the driver will return PICO_NO_SAMPLES_AVAILABLE.

Applicability	Block mode, rapid block mode
Arguments	handle, the handle of the required 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 starts at startIndex. downSampleRatio, the downsampling factor that will be applied to the raw data. downSampleRatioMode, which downsampling mode to use. The available values are: PS5000A_RATIO_MODE_NONE (downSampleRatio is ignored) PS5000A_RATIO_MODE_AGGREGATE PS5000A_RATIO_MODE_AVERAGE PS5000A_RATIO_MODE_DECIMATE AGGREGATE, AVERAGE, 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.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_NO_SAMPLES_AVAILABLE

	PICO_DEVICE_SAMPLING PICO_NULL_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_STARTINDEX_INVALID PICO_ETS_NOT_RUNNING PICO_BUFFERS_NOT_SET 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

4.21.1 Downsampling modes

Various methods of data reduction, or **downsampling**, are possible with the PicoScope 5000 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 ps5000aGetValues. The following modes are available:

PS5000A_RATIO_MODE_NONE	No downsampling. Returns raw data values.
PS5000A_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.
PS5000A_RATIO_MODE_AVERAGE	Reduces every block of *n* values to a single value representing the average (arithmetic mean) of all the values.
PS5000A_RATIO_MODE_DECIMATE	Reduces every block of *n* values to just the first value in the block, discarding all the other values.

4.22 ps5000aGetValuesAsync

```
PICO_STATUS ps5000aGetValuesAsync
(
  int16_t                 handle,
  uint32_t                startIndex,
  uint32_t                noOfSamples,
  uint32_t                downSampleRatio,
  PS5000A_RATIO_MODE      downSampleRatioMode,
  uint32_t                segmentIndex,
  void                  * lpDataReady,
  void                  * pParameter
)
```

This function returns data either with or without downsampling, starting at the specified sample number. It is used to get the stored data from the driver after data collection has stopped. It returns the data using a callback.

Applicability	Streaming mode and block mode
Arguments	`handle`, the handle of the required device `startIndex`, `noOfSamples`, `downSampleRatio`, `downSampleRatioMode`, `segmentIndex`: see `ps5000aGetValues` `* lpDataReady`, a pointer to the user-supplied function that will be called when the data is ready. This will be a `ps5000aDataReady` function for block-mode data or a `ps5000aStreamingReady` 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_POWER_SUPPLY_CONNECTED`
	`PICO_POWER_SUPPLY_NOT_CONNECTED`
	`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`

4.23 ps5000aGetValuesBulk

```
PICO_STATUS ps5000aGetValuesBulk
(
  int16_t                 handle,
  uint32_t              * noOfSamples,
  uint32_t                fromSegmentIndex,
  uint32_t                toSegmentIndex,
  uint32_t                downSampleRatio,
  PS5000A_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`, the handle of 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 `ps5000aGetValues` `* 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 `ps5000aGetValues`.

Returns	`PICO_OK`
	`PICO_POWER_SUPPLY_CONNECTED`
	`PICO_POWER_SUPPLY_NOT_CONNECTED`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`
	`PICO_INVALID_SAMPLERATIO`
	`PICO_ETS_NOT_RUNNING`
	`PICO_BUFFERS_NOT_SET`
	`PICO_TOO_MANY_SAMPLES`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_NOT_RESPONDING`
	`PICO_DRIVER_FUNCTION`

4.24 ps5000aGetValuesOverlapped

```
PICO_STATUS ps5000aGetValuesOverlapped
(
  int16_t                 handle,
  uint32_t                startIndex,
  uint32_t              * noOfSamples,
  uint32_t                downSampleRatio,
  PS5000A_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 ps5000aRunBlock. The advantage of this function is that the driver makes contact with the scope only once, when you call ps5000aRunBlock, compared with the two contacts that occur when you use the conventional ps5000aRunBlock, ps5000aGetValues calling sequence. This slightly reduces the dead time between successive captures in block mode.

After calling ps5000aRunBlock, you can optionally use ps5000aGetValues 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 ps5000aGetValues * overflow: see ps5000aGetValuesBulk
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

4.24.1 Using the GetValuesOverlapped functions

1. Open the oscilloscope using ps5000aOpenUnit.
2. Select channel ranges and AC/DC coupling using ps5000aSetChannel.
3. Using ps5000aGetTimebase, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions ps5000aSetTriggerChannelConditions, ps5000aSetTriggerChannelDirections and ps5000aSetTriggerChannelProperties to set up the trigger if required.
5. Use ps5000aSetDataBuffer to tell the driver where your memory buffer is.
6. Set up the transfer of the block of data from the oscilloscope using ps5000aGetValuesOverlapped.
7. Start the oscilloscope running using ps5000aRunBlock.
8. Wait until the oscilloscope is ready using the ps5000aBlockReady callback (or poll using ps5000aIsReady).

9. Display the data.
10. Repeat steps 7 to 9 if needed.
11. Stop the oscilloscope by calling ps5000aStop.

A similar procedure can be used with rapid block mode using the ps5000aGetValuesOverlappedBulk function.

4.25 ps5000aGetValuesOverlappedBulk

```
PICO_STATUS ps5000aGetValuesOverlappedBulk
(
  int16_t                 handle,
  uint32_t                startIndex,
  uint32_t              * noOfSamples,
  uint32_t                downSampleRatio,
  PS5000A_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 ps5000aRunBlock. The advantage of this method is that the driver makes contact with the scope only once, when you call ps5000aRunBlock, compared with the two contacts that occur when you use the conventional ps5000aRunBlock, ps5000aGetValuesBulk calling sequence. This slightly reduces the dead time between successive captures in rapid block mode.

After calling ps5000aRunBlock, you can optionally use ps5000aGetValues 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 ps5000aGetValues fromSegmentIndex, toSegmentIndex, * overflow, see ps5000aGetValuesBulk
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

4.26 ps5000aGetValuesTriggerTimeOffsetBulk

```
PICO_STATUS ps5000aGetValuesTriggerTimeOffsetBulk
(
  int16_t                 handle,
  uint32_t              * timesUpper,
  uint32_t              * timesLower,
  PS5000A_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 ps5000aGetTriggerTimeOffset once for each waveform required. See ps5000aGetTriggerTimeOffset for an explanation of trigger time offsets.

There is another version of this function, ps5000aGetValuesTriggerTimeOffsetBulk64, that returns trigger time offsets as 64-bit values instead of pairs of 32-bit values.

Applicability	Rapid block mode
Arguments	handle, the handle of 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. Refer to ps5000aGetTriggerTimeOffset for specific figures 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_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_NOT_USED_IN_THIS_CAPTURE_MODE PICO_NOT_RESPONDING PICO_NULL_PARAMETER PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE PICO_DRIVER_FUNCTION

4.27 ps5000aGetValuesTriggerTimeOffsetBulk64

```
PICO_STATUS ps5000aGetValuesTriggerTimeOffsetBulk64
(
  int16_t                 handle,
  int64_t               * times,
  PS5000A_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, ps5000aGetValuesTriggerTimeOffsetBulk, 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`, the handle of 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`, an array of integers long enough to hold the number of requested times. `timeUnits[0]` will contain the time unit for `fromSegmentIndex`, and the last element will contain the `toSegmentIndex`. Refer to ps5000aGetTriggerTimeOffset64 for specific figures. `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_POWER_SUPPLY_CONNECTED`
	`PICO_POWER_SUPPLY_NOT_CONNECTED`
	`PICO_INVALID_HANDLE`
	`PICO_NOT_USED_IN_THIS_CAPTURE_MODE`
	`PICO_NOT_RESPONDING`
	`PICO_NULL_PARAMETER`
	`PICO_DEVICE_SAMPLING`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_DRIVER_FUNCTION`

4.28 ps5000aIsReady

```
PICO_STATUS ps5000aIsReady
(
  int16_t   handle,
  int16_t * ready
)
```

This function may be used instead of a callback function to receive data from ps5000aRunBlock. To use this method, pass a NULL pointer as the lpReady argument to ps5000aRunBlock. You must then poll the driver to see if it has finished collecting the requested samples.

Applicability	Block mode
Arguments	handle, the handle of the required 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 ps5000aGetValues can be used to retrieve the data.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_CANCELLED PICO_NOT_RESPONDING

4.29 ps5000aIsTriggerOrPulseWidthQualifierEnabled

```
PICO_STATUS ps5000aIsTriggerOrPulseWidthQualifierEnabled
(
  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, and just before calling either ps5000aRunBlock or ps5000aRunStreaming.
Arguments	handle, the handle of the required device * triggerEnabled, on exit, indicates whether the trigger will successfully be set when ps5000aRunBlock or ps5000aRunStreaming 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 ps5000aRunBlock or ps5000aRunStreaming 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

4.30 ps5000aMaximumValue

```
PICO_STATUS ps5000aMaximumValue
(
  int16_t   handle,
  int16_t * value
)
```

This function returns a status code and outputs the maximum ADC count value to a parameter. The output value depends on the currently selected resolution.

Applicability	All modes
Arguments	`handle`, the handle of the required device `* value`, pointer to an `int16_t` (output), set to the maximum ADC value.

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

4.31 ps5000aMemorySegments

```
PICO_STATUS ps5000aMemorySegments
(
  int16_t    handle,
  uint32_t   nSegments,
  int32_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`, the handle of the required device `nSegments`, the number of segments required, from: 1 to 65,535: PicoScope 5242A/B, 5243A/B, 5442A/B, 5443A/B 1 to 125,000: PicoScope 5244A, 5444A 1 to 250,000: PicoScope 5244B, 5444B Note that, at 12-bit resolution or higher, the maximum number of segments is 16,384 for the PicoScope 5242A and 5442A and 32,768 for the PicoScope 5242B and 5442B. `* 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`

4.32 ps5000aMinimumValue

```
PICO_STATUS ps5000aMinimumValue
(
  int16_t   handle,
  int16_t * value
)
```

This function returns a status code and outputs the minimum ADC count value to a parameter. The output value depends on the currently selected resolution.

Applicability	All modes
Arguments	`handle`, the handle of the required device `* value`, pointer to an `int16_t`, (output) set to the minimum ADC value.

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

4.33 ps5000aNoOfStreamingValues

```
PICO_STATUS ps5000aNoOfStreamingValues
(
  int16_t    handle,
  uint32_t * noOfValues
)
```

This function returns the number of samples available after data collection in streaming mode. Call it after calling ps5000aStop.

Applicability	Streaming mode
Arguments	handle, the handle of the required 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

4.34 ps5000aOpenUnit

```
PICO_STATUS ps5000aOpenUnit
(
  int16_t                   * handle,
  int8_t                    * serial
  PS5000A_DEVICE_RESOLUTION   resolution
)
```

This function opens a PicoScope 5000 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. If `ps5000aOpenUnit` is called without the power supply connected, the driver returns `PICO_POWER_SUPPLY_NOT_CONNECTED`.

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. `resolution,` determines the resolution of the device when opened, the available values are one of the `PS5000A_DEVICE_RESOLUTION`. If resolution is out of range the device will return `PICO_INVALID_DEVICE_RESOLUTION`.

Returns	`PICO_OK`
	`PICO_OS_NOT_SUPPORTED`
	`PICO_INVALID_DEVICE_RESOLUTION.`
	`PICO_OPEN_OPERATION_IN_PROGRESS`
	`PICO_EEPROM_CORRUPT`
	`PICO_KERNEL_DRIVER_TOO_OLD`
	`PICO_FPGA_FAIL`
	`PICO_MEMORY_CLOCK_FREQUENCY`
	`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`
	`PICO_POWER_SUPPLY_NOT_CONNECTED`

4.35 ps5000aOpenUnitAsync

```
PICO_STATUS ps5000aOpenUnitAsync
(
  int16_t                   * status,
  int8_t                    * serial
  PS5000A_DEVICE_RESOLUTION   resolution
)
```

This function opens a scope without blocking the calling thread. You can find out when it has finished by periodically calling ps5000aOpenUnitProgress 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 ps5000aOpenUnit resolution, determines the resolution of the device when opened, the available values are one of the PS5000A_DEVICE_RESOLUTION. If resolution is out of range the device will return PICO_INVALID_DEVICE_RESOLUTION.
Returns	PICO_OK PICO_INVALID_DEVICE_RESOLUTION PICO_OPEN_OPERATION_IN_PROGRESS PICO_OPERATION_FAILED

4.36 ps5000aOpenUnitProgress

```
PICO_STATUS ps5000aOpenUnitProgress
(
  int16_t * handle,
  int16_t * progressPercent,
  int16_t * complete
)
```

This function checks on the progress of a request made to ps5000aOpenUnitAsync to open a scope.

Applicability	Use after ps5000aOpenUnitAsync
Arguments	* handle: see ps5000aOpenUnit. 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

4.37 ps5000aPingUnit

```
PICO_STATUS ps5000aPingUnit
(
  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`

4.38 ps5000aRunBlock

```
PICO_STATUS ps5000aRunBlock
(
  int16_t             handle,
  int32_t             noOfPreTriggerSamples,
  int32_t             noOfPostTriggerSamples,
  uint32_t            timebase,
  int32_t           * timeIndisposedMs,
  uint32_t            segmentIndex,
  ps5000aBlockReady   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`.

Applicability	Block mode, rapid block mode
Arguments	`handle`, the handle of the required 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. `* 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. `lpReady`, a pointer to the `ps5000aBlockReady` callback function that the driver will call when the data has been collected. To use the `ps5000aIsReady` polling method instead of a callback function, set this pointer to NULL. `* pParameter`, a void pointer that is passed to the `ps5000aBlockReady` callback function. The callback can use this pointer to return arbitrary data to the application.

Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_BUFFERS_NOT_SET (in Overlapped mode) 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_FW_FAIL PICO_NOT_ENOUGH_SEGMENTS (in Bulk mode) 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

4.39 ps5000aRunStreaming

```
PICO_STATUS ps5000aRunStreaming
(
  int16_t                 handle,
  uint32_t              * sampleInterval,
  PS5000A_TIME_UNITS      sampleIntervalTimeUnits,
  uint32_t                maxPreTriggerSamples,
  uint32_t                maxPostTriggerSamples,
  int16_t                 autoStop,
  uint32_t                downSampleRatio,
  PS5000A_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 ps5000aGetStreamingLatestValues 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, the handle of the required 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: PS5000A_FS PS5000A_PS PS5000A_NS PS5000A_US PS5000A_MS PS5000A_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 ps5000aGetValues 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 ps5000aSetDataBuffer.
Returns	PICO_OK PICO_INVALID_HANDLE

	PICO_ETS_MODE_SET PICO_USER_CALLBACK PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_STREAMING_FAILED PICO_NOT_RESPONDING PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_TRIGGER_ERROR PICO_INVALID_SAMPLE_INTERVAL PICO_INVALID_BUFFER PICO_DRIVER_FUNCTION PICO_FW_FAIL PICO_MEMORY

4.40 ps5000aSetBandwidthFilter

```
PICO_STATUS ps5000aSetBandwidthFilter
(
  int16_t                   handle,
  PS5000A_CHANNEL           channel,
  PS5000A_BANDWIDTH_LIMITER bandwidth
)
```

This function controls the hardware bandwidth limiter.

Applicability	All modes. All models.
Arguments	`handle`, the handle of the required device `channel`, the channel to be configured. The values are: `PS5000A_CHANNEL_A`: Channel A input `PS5000A_CHANNEL_B`: Channel B input `PS5000A_CHANNEL_C`: Channel C input `PS5000A_CHANNEL_D`: Channel D input `bandwidth`, the bandwidth is either `PS5000A_BW_FULL` or `PS5000A_BW_20MHZ`

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_BANDWIDTH`

4.41 ps5000aSetChannel

```
PICO_STATUS ps5000aSetChannel
(
  int16_t           handle,
  PS5000A_CHANNEL   channel,
  int16_t           enabled,
  PS5000A_COUPLING  type,
  PS5000A_RANGE     range,
  float             analogueOffset
)
```

This function specifies whether an input channel is to be enabled, its input coupling type, voltage range, analog offset and bandwidth limit.

Applicability	All modes
Arguments	`handle`, the handle of the required device `channel`, the channel to be configured. The values are: `PS5000A_CHANNEL_A`: Channel A input `PS5000A_CHANNEL_B`: Channel B input `PS5000A_CHANNEL_C`: Channel C input `PS5000A_CHANNEL_D`: Channel D 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 are: `PS5000A_AC`: 1 megohm impedance, AC coupling. The channel accepts input frequencies from about 1 hertz up to its maximum -3 dB analog bandwidth. `PS5000A_DC`: 1 megohm impedance, DC coupling. The scope accepts all input frequencies from zero (DC) up to its maximum -3 dB analog bandwidth. `range`, the input voltage range: `PS5000A_10MV`: ±10 mV `PS5000A_20MV`: ±20 mV `PS5000A_50MV`: ±50 mV `PS5000A_100MV`: ±100 mV `PS5000A_200MV`: ±200 mV `PS5000A_500MV`: ±500 mV `PS5000A_1V`: ±1 V `PS5000A_2V`: ±2 V `PS5000A_5V`: ±5 V `PS5000A_10V`: ±10 V `PS5000A_20V`: ±20 V `analogueOffset`, a voltage to add to the input channel before digitization. The allowable range of offsets depends on the input range selected for the channel, as obtained from `ps5000aGetAnalogueOffset`.

Returns	PICO_OK PICO_USER_CALLBACK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_INVALID_VOLTAGE_RANGE PICO_INVALID_COUPLING PICO_INVALID_ANALOGUE_OFFSET PICO_DRIVER_FUNCTION

4.42 ps5000aSetDataBuffer

```
PICO_STATUS ps5000aSetDataBuffer
(
  int16_t                 handle,
  PS5000A_CHANNEL         channel,
  int16_t                 * buffer,
  int32_t                 bufferLth,
  uint32_t                segmentIndex,
  PS5000A_RATIO_MODE      mode
)
```

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 need to call ps5000aSetDataBuffers instead.

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`, the handle of the required device `channel`, the channel you want to use with the buffer. Use one of these values: PS5000A_CHANNEL_A PS5000A_CHANNEL_B PS5000A_CHANNEL_C PS5000A_CHANNEL_D `* buffer`, the location of the buffer `bufferLth`, the size of the `buffer` array `segmentIndex`, the number of the memory segment to be used `mode`, the downsampling mode. See ps5000aGetValues for the available modes, but note that a single call to ps5000aSetDataBuffer can only associate one buffer with one downsampling mode. If you intend to call ps5000aGetValues with more than one downsampling mode activated, then you must call ps5000aSetDataBuffer 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_SEGMENT_OUT_OF_RANGE PICO_DRIVER_FUNCTION PICO_INVALID_PARAMETER

4.43 ps5000aSetDataBuffers

```
PICO_STATUS ps5000aSetDataBuffers
(
  int16_t                 handle,
  PS5000A_CHANNEL         channel,
  int16_t                 * bufferMax,
  int16_t                 * bufferMin,
  int32_t                 bufferLth,
  uint32_t                segmentIndex,
  PS5000A_RATIO_MODE      mode
)
```

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 ps5000aSetDataBuffer instead.

Applicability	Block and streaming modes with aggregation.
Arguments	handle, the handle of the required device. channel, the channel for which you want to set the buffers. Use one of these constants: PS5000A_CHANNEL_A PS5000A_CHANNEL_B PS5000A_CHANNEL_C PS5000A_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. segmentIndex, the number of the memory segment to be used mode: see ps5000aGetValues
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_RATIO_MODE_NOT_SUPPORTED PICO_SEGMENT_OUT_OF_RANGE PICO_DRIVER_FUNCTION PICO_INVALID_PARAMETER

4.44 ps5000aSetDeviceResolution

```
PICO_STATUS ps5000aSetDeviceResolution
(
  int16_t                   handle,
  PS5000A_DEVICE_RESOLUTION resolution
)
```

This function sets the new resolution. When using 12 bits or more the memory is halved. When using 15-bit resolution only 2 channels can be enabled to capture data, and when using 16-bit resolution only one channel is available. If resolution is changed, any data captured that has not been saved will be lost. If ps5000aSetChannel is not called, ps5000aRunBlock and ps5000aRunStreaming may fail.

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. resolution, determines the resolution of the device when opened, the available values are one of the PS5000A_DEVICE_RESOLUTION. If resolution is out of range the device will return PICO_INVALID_DEVICE_RESOLUTION.
Returns	PICO_OK PICO_INVALID_DEVICE_RESOLUTION PICO_OS_NOT_SUPPORTED PICO_OPEN_OPERATION_IN_PROGRESS PICO_EEPROM_CORRUPT PICO_KERNEL_DRIVER_TOO_OLD PICO_FPGA_FAIL PICO_MEMORY_CLOCK_FREQUENCY 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 PICO_POWER_SUPPLY_NOT_CONNECTED

4.45 ps5000aSetEts

```
PICO_STATUS ps5000aSetEts
(
  int16_t              handle,
  PS5000A_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`, the handle of the required device
	
	`mode`, the ETS mode. Use one of these values:
	`PS5000A_ETS_OFF`: disables ETS
	`PS5000A_ETS_FAST`: enables ETS and provides `etsCycles` of data, which may contain data from previously returned cycles
	`PS5000A_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 either:
	`PS5242A_MAX_ETS_CYCLES`
	`PS5243A_MAX_ETS_CYCLES`
	`PS5244A_MAX_ETS_CYCLES`
	
	`etsInterleave`, the number of waveforms to combine into a single ETS capture.
	Maximum value is either:
	`PS5242A_MAX_INTERLEAVE`
	`PS5243A_MAX_INTERLEAVE`
	`PS5244A_MAX_INTERLEAVE`
	
	`* sampleTimePicoseconds`, on exit, the effective sampling interval of the ETS data. For example, if the captured sample time is 4 ns and `etsInterleave` is 10, then the effective sample time in ETS mode is 400 ps.

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

4.46 ps5000aSetEtsTimeBuffer

```
PICO_STATUS ps5000aSetEtsTimeBuffer
(
  int16_t   handle,
  int64_t * buffer,
  int32_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 ps5000aSetEtsTimeBuffers instead.
Arguments	handle, the handle of the required device * buffer, an array of 64-bit words, each representing the time in femtoseconds (10^{-15} s) 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

4.47 ps5000aSetEtsTimeBuffers

```
PICO_STATUS ps5000aSetEtsTimeBuffers
(
  int16_t    handle,
  uint32_t * timeUpper,
  uint32_t * timeLower,
  int32_t    bufferLth
)
```

This function tells the driver where to find your application's ETS time buffers. These buffers contain the timing information for each ETS sample after you run a block-mode ETS capture. There are two buffers containing the upper and lower 32-bit parts of the timing information, to allow programming languages that do not support 64-bit data to retrieve the timings.

Applicability	ETS mode only. If your programming language supports 64-bit data then you can use ps5000aSetEtsTimeBuffer instead.
Arguments	handle, the handle of the required device * timeUpper, an array of 32-bit words, each representing the upper 32 bits of the time in femtoseconds (10^{-15} s) at which the sample was captured * timeLower, an array of 32-bit words, each representing the lower 32 bits of the time in femtoseconds 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

4.48 ps5000aSetNoOfCaptures

```
PICO_STATUS ps5000aSetNoOfCaptures
(
  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. Once a value has been set, the value remains constant unless changed.

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

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

4.49 ps5000aSetPulseWidthQualifier

```
PICO_STATUS ps5000aSetPulseWidthQualifier
(
  int16_t                         handle,
  PS5000A_PWQ_CONDITIONS      * conditions,
  int16_t                         nConditions,
  PS5000A_THRESHOLD_DIRECTION   direction,
  uint32_t                        lower,
  uint32_t                        upper,
  PS5000A_PULSE_WIDTH_TYPE      type
)
```

This function sets up pulse-width qualification, which can be used on its own for pulse-width triggering or combined with threshold triggering, level triggering or window triggering to produce more complex triggers. The pulse-width qualifier is set by defining one or more 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.

Applicability	All modes
Arguments	`handle`, the handle of the required device `* conditions`, an array of PS5000A_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` then 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 PS5000A_MAX_PULSE_WIDTH_QUALIFIER_COUNT. `direction`, the direction of the signal required for the pulse width trigger to fire. See PS5000A_THRESHOLD_DIRECTION constants for the list of possible values. Each channel of the oscilloscope (except the EXT input) has two thresholds for each direction—for example, PS5000A_RISING and PS5000A_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 PS5000A_RISING as the `direction` argument for both ps5000aSetTriggerConditions and ps5000aSetPulseWidthQualifier 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 PS5000A_PW_TYPE_IN_RANGE or PS5000A_PW_TYPE_OUT_OF_RANGE.

	type, the pulse-width type, one of these constants: PS5000A_PW_TYPE_NONE: do not use the pulse width qualifier PS5000A_PW_TYPE_LESS_THAN: pulse width less than lower PS5000A_PW_TYPE_GREATER_THAN: pulse width greater than lower PS5000A_PW_TYPE_IN_RANGE: pulse width between lower and upper PS5000A_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

4.49.1 ps5000a_PWQ_CONDITIONS structure

A structure of this type is passed to ps5000aSetPulseWidthQualifier in the conditions argument to specify the trigger conditions. It is defined as follows:

```
typedef struct tPS5000APwqConditions
{
  PS5000A_TRIGGER_STATE channelA;
  PS5000A_TRIGGER_STATE channelB;
  PS5000A_TRIGGER_STATE channelC;
  PS5000A_TRIGGER_STATE channelD;
  PS5000A_TRIGGER_STATE external;
  PS5000A_TRIGGER_STATE aux;
} PS5000A_PWQ_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The ps5000aSetPulseWidthQualifier 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.

Applicability	All models
Elements	channelA, channelB, channelC*, channelD*, external: the type of condition that should be applied to each channel. Use these constants: - PS5000A_CONDITION_DONT_CARE PS5000A_CONDITION_TRUE PS5000A_CONDITION_FALSE The channels that are set to PS5000A_CONDITION_TRUE or PS5000A_CONDITION_FALSE must all meet their conditions simultaneously to produce a trigger. Channels set to PS5000A_CONDITION_DONT_CARE are ignored. aux: not used

*Note: applicable to 4-channel analog devices only.

4.50 ps5000aSetSigGenArbitrary

```
PICO_STATUS ps5000aSetSigGenArbitrary
(
  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,
  PS5000A_SWEEP_TYPE               sweepType,
  PS5000A_EXTRA_OPERATIONS         operation,
  PS5000A_INDEX_MODE               indexMode,
  uint32_t                         shots,
  uint32_t                         sweeps,
  PS5000A_SIGGEN_TRIG_TYPE         triggerType,
  PS5000A_SIGGEN_TRIG_SOURCE       triggerSource,
  int16_t                          extInThreshold
)
```

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

The arbitrary waveform generator (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 phase accumulator initially increments by `startDeltaPhase`. If the AWG is set to sweep mode, the phase increment is increased or decreased at specified intervals until it reaches `stopDeltaPhase`. The easiest way to obtain the values of `startDeltaPhase` and `stopDeltaPhase` necessary to generate the desired frequency is to call `ps5000aSigGenFrequencyToPhase`. Alternatively, see Calculating deltaPhase below for more information on how to calculate these values.

Applicability	All modes. B models only.
Arguments	`handle`, the handle of the required 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. Note that if the signal voltages defined by the combination of `offsetVoltage` and `pkToPk` extend outside the voltage range of the signal generator, the output waveform will be clipped. `startDeltaPhase`, the initial value added to the phase accumulator as the generator begins to step through the waveform buffer.

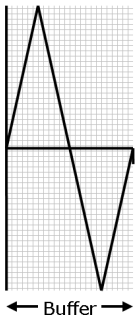
	`stopDeltaPhase`, the final value added to the phase accumulator before the generator restarts or reverses the sweep.
	
	`deltaPhaseIncrement`, the amount added to the delta phase value every time the `dwellCount` period expires. This determines the amount by which the generator sweeps the output frequency in each dwell period.
	
	`dwellCount`, the time, in 50 ns steps, between successive additions of `deltaPhaseIncrement` to the delta phase accumulator. This determines the rate at which the generator sweeps the output frequency.
	Minimum value: `PS5000A_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, then a sample of −32768 corresponds to −2 V, and +32767 to +2 V.
	
	`arbitraryWaveformSize`, the size of the arbitrary waveform buffer, in samples, from `MIN_SIG_GEN_BUFFER_SIZE` to `PS5X42A_MAX_SIG_GEN_BUFFER_SIZE`, `PS5X43A_MAX_SIG_GEN_BUFFER_SIZE` or `PS5X44A_MAX_SIG_GEN_BUFFER_SIZE`, depending on the oscilloscope model.
	
	`sweepType`, determines whether the `startDeltaPhase` is swept up to the `stopDeltaPhase`, down to it, or repeatedly up and down. Use one of these values:
	`PS5000A_UP`
	`PS5000A_DOWN`
	`PS5000A_UPDOWN`
	`PS5000A_DOWNUP`
	
	`operation`, the type of waveform to be produced, specified by one of the following enumerated types:
	`PS5000A_ES_OFF`, normal signal generator operation specified by wavetype.
	`PS5000A_WHITENOISE`, the signal generator produces white noise and ignores all settings except `pkToPk` and `offsetVoltage`.
	`PS5000A_PRBS`, produces a random bitstream with a bit rate specified by the start and stop frequency.
	
	`indexMode`, specifies how the signal will be formed from the arbitrary waveform data. Single and dual index modes are possible. Use one of these constants:
	`PS5000A_SINGLE`
	`PS5000A_DUAL`
	
	`shots`,
	`sweeps`,
	`triggerType`,
	`triggerSource`,
	`extInThreshold`: see `ps5000aSigGenBuiltIn`

Returns	PICO_OK PICO_AWG_NOT_SUPPORTED PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_BUSY PICO_INVALID_HANDLE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING PICO_NOT_RESPONDING 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

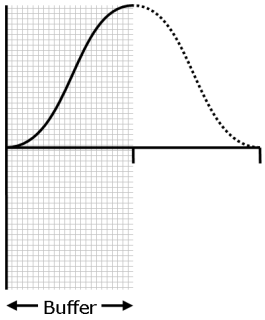
4.50.1 AWG index modes

The arbitrary waveform generator supports **single** and **dual** 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 mode makes more efficient use of the buffer memory.



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.



4.50.2 Calculating deltaPhase

The arbitrary waveform generator steps through the waveform by adding a *deltaPhase* value between 1 and *phaseAccumulatorSize-1* to the phase accumulator every *dacPeriod* (*1/dacFrequency*). If the *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

You can call `ps5000aSigGenFrequencyToPhase` to calculate the value for *deltaPhase* for the desired frequency.

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.

Parameter	PicoScope 5242B PicoScope 5442B	PicoScope 5243B PicoScope 5443B	PicoScope 5244B PicoScope 5444B
dacFrequency	200 MHz		
dacPeriod (= 1/*dacFrequency*)	5 ns		
phaseAccumulatorSize	4 294 967 296 (2^{32})		
awgBufferSize	16 384 (2^{14})	32 768 (2^{15})	49 152 (3×2^{14})

4.51 ps5000aSetSigGenBuiltIn

```
PICO_STATUS ps5000aSetSigGenBuiltIn
(
  int16_t                          handle,
  int32_t                          offsetVoltage,
  uint32_t                         pkToPk,
  PS5000A_WAVE_TYPE                waveType,
  float                            startFrequency,
  float                            stopFrequency,
  float                            increment,
  float                            dwellTime,
  PS5000A_SWEEP_TYPE               sweepType,
  PS5000A_EXTRA_OPERATIONS         operation,
  uint32_t                         shots,
  uint32_t                         sweeps,
  PS5000A_SIGGEN_TRIG_TYPE         triggerType,
  PS5000A_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 models
Arguments	`handle`, the handle of the required 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. Note that if the signal voltages described by the combination of `offsetVoltage` and `pkToPk` extend outside the voltage range of the signal generator, the output waveform will be clipped. `waveType`, the type of waveform to be generated. `PS5000A_SINE` sine wave `PS5000A_SQUARE` square wave `PS5000A_TRIANGLE` triangle wave `PS5000A_DC_VOLTAGE` DC voltage The following `waveTypes` apply to B models only: `PS5000A_RAMP_UP` rising sawtooth `PS5000A_RAMP_DOWN` falling sawtooth `PS5000A_SINC` sin (x)/x `PS5000A_GAUSSIAN` Gaussian `PS5000A_HALF_SINE` half (full-wave rectified) sine `startFrequency`, the frequency that the signal generator will initially produce. For allowable values see `PS5000A_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

`sweepType`, whether the frequency will sweep from `startFrequency` to `stopFrequency`, in the opposite direction, or repeatedly reverse direction. Use one of these constants:
- PS5000A_UP
- PS5000A_DOWN
- PS5000A_UPDOWN
- PS5000A_DOWNUP

`operation`, the type of waveform to be produced, specified by one of the following enumerated types (B models only):
- PS5000A_ES_OFF, normal signal generator operation specified by wavetype.
- PS5000A_WHITENOISE, the signal generator produces white noise and ignores all settings except `pkToPk` and `offsetVoltage`.
- PS5000A_PRBS, produces a random bitstream with a bit rate specified by the start and stop frequency.

`shots`,
- `0`: sweep the frequency as specified by `sweeps`
- `1...`PS5000A_MAX_SWEEPS_SHOTS: the number of cycles of the waveform to be produced after a trigger event. `sweeps` must be zero.
- PS5000A_SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN: start and run continuously after trigger occurs

`sweeps`,
- `0`: produce number of cycles specified by `shots`
- `1..`PS5000A_MAX_SWEEPS_SHOTS: the number of times to sweep the frequency after a trigger event, according to `sweepType`. `shots` must be zero.
- PS5000A_SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN: start a sweep and continue after trigger occurs

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

PS5000A_SIGGEN_RISING	trigger on rising edge
PS5000A_SIGGEN_FALLING	trigger on falling edge
PS5000A_SIGGEN_GATE_HIGH	run while trigger is high
PS5000A_SIGGEN_GATE_LOW	run while trigger is low

	triggerSource, the source that will trigger the signal generator. PS5000A_SIGGEN_NONE — run without waiting for trigger PS5000A_SIGGEN_SCOPE_TRIG — use scope trigger PS5000A_SIGGEN_EXT_IN — use EXT input PS5000A_SIGGEN_SOFT_TRIG — wait for software trigger provided by ps5000aSigGenSoftwareControl PS5000A_SIGGEN_TRIGGER_RAW — reserved If a trigger source other than P5000A_SIGGEN_NONE is specified, then either shots or sweeps, but not both, must be non-zero. extInThreshold, used to set trigger level for external trigger.
Returns	PICO_OK PICO_BUSY PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING PICO_NOT_RESPONDING PICO_WARNING_AUX_OUTPUT_CONFLICT 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

4.52 ps5000aSetSigGenBuiltInV2

```
PICO_STATUS ps5000aSetSigGenBuiltIn
(
  int16_t                         handle,
  int32_t                         offsetVoltage,
  uint32_t                        pkToPk,
  PS5000A_WAVE_TYPE               waveType,
  float                           startFrequency,
  float                           stopFrequency,
  float                           increment,
  float                           dwellTime,
  PS5000A_SWEEP_TYPE              sweepType,
  PS5000A_EXTRA_OPERATIONS        operation,
  uint32_t                        shots,
  uint32_t                        sweeps,
  PS5000A_SIGGEN_TRIG_TYPE        triggerType,
  PS5000A_SIGGEN_TRIG_SOURCE      triggerSource,
  int16_t                         extInThreshold
)
```

This function is the same as ps5000aSetSigGenBuiltIn, except that it allows you to set the frequency arguments with greater precision. It 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 models
Arguments	handle, the handle of the required device offsetVoltage, pkToPk, waveType, startFrequency, stopFrequency, increment, dwellTime, sweepType, operation, shots, sweeps, triggerType, triggerSource, extInThreshold: see ps5000aSetSigGenBuiltIn
Returns	PICO_OK PICO_BUSY PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING PICO_NOT_RESPONDING PICO_WARNING_AUX_OUTPUT_CONFLICT 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

4.53 ps5000aSetSigGenPropertiesArbitrary

```
PICO_STATUS ps5000aSetSigGenPropertiesArbitrary
(
  int16_t                     handle,
  uint32_t                    startDeltaPhase,
  uint32_t                    stopDeltaPhase,
  uint32_t                    deltaPhaseIncrement,
  uint32_t                    dwellCount,
  PS5000A_SWEEP_TYPE          sweepType,
  uint32_t                    shots,
  uint32_t                    sweeps,
  PS5000A_SIGGEN_TRIG_TYPE    triggerType,
  PS5000A_SIGGEN_TRIG_SOURCE  triggerSource,
  int16_t                     extInThreshold
)
```

This function reprograms the arbitrary waveform generator. All values can be reprogrammed while the oscilloscope is waiting for a trigger.

Applicability	All modes
Arguments	See ps5000aSetSigGenArbitrary
Returns	PICO_OK if successful. PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_NO_SIGNAL_GENERATOR PICO_WARNING_AUX_OUTPUT_CONFLICT PICO_WARNING_EXT_THRESHOLD_CONFLICT PICO_SIGGEN_OFFSET_VOLTAGE PICO_SIGGEN_OUTPUT_OVER_VOLTAGE PICO_SIGGEN_PK_TO_PK PICO_SIGGEN_OFFSET_VOLTAGE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING PICO_AWG_NOT_SUPPORTED PICO_BUSY PICO_SIGGEN_WAVEFORM_SETUP_FAILED PICO_NOT_RESPONDING PICO_POWER_SUPPLY_UNDERVOLTAGE PICO_POWER_SUPPLY_NOT_CONNECTED PICO_POWER_SUPPLY_CONNECTED

4.54 ps5000aSetSigGenPropertiesBuiltIn

```
PICO_STATUS ps5000aSetSigGenPropertiesBuiltIn
(
  int16_t                    handle,
  double                     startFrequency,
  double                     stopFrequency,
  double                     increment,
  double                     dwellTime,
  PS5000A_SWEEP_TYPE         sweepType,
  uint32_t                   shots,
  uint32_t                   sweeps,
  PS5000A_SIGGEN_TRIG_TYPE   triggerType,
  PS5000A_SIGGEN_TRIG_SOURCE triggerSource,
  int16_t                    extInThreshold
)
```

This function reprograms the signal generator. Values can be changed while the oscilloscope is waiting for a trigger.

Applicability	All modes
Arguments	See ps5000aSetSigGenBuiltIn
Returns	PICO_OK if successful. PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_NO_SIGNAL_GENERATOR PICO_SIG_GEN_PARAM PICO_WARNING_AUX_OUTPUT_CONFLICT PICO_WARNING_EXT_THRESHOLD_CONFLICT PICO_SIGGEN_DC_VOLTAGE_NOT_CONFIGURABLE PICO_BUSY PICO_SIGGEN_WAVEFORM_SETUP_FAILED PICO_NOT_RESPONDING PICO_POWER_SUPPLY_UNDERVOLTAGE PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_POWER_SUPPLY_NOT_CONNECTED PICO_POWER_SUPPLY_CONNECTED

4.55 ps5000aSetSimpleTrigger

```
PICO_STATUS ps5000aSetSimpleTrigger
(
  int16_t                          handle,
  int16_t                          enable,
  PS5000A_CHANNEL                  source,
  int16_t                          threshold,
  PS5000A_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 cancelled.

Applicability	All modes
Arguments	`handle`, the handle of the required device. `enable`, zero to disable the trigger, any non-zero value to set the trigger. `source`, the channel on which to trigger. `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. For example, if delay=100 then the scope would wait 100 sample periods before sampling. At a timebase of 500 MS/s, or 2 ns per sample, the total delay would then be 100 x 2 ns = 200 ns. Range: 0 to `MAX_DELAY_COUNT`. `autoTrigger_ms`, the number of milliseconds the device will wait if no trigger occurs. If this is set to zero, the scope device will wait indefinitely for a trigger.

Returns	`PICO_OK`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_PARAMETER`
	`PICO_MEMORY`
	`PICO_CONDITIONS`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_DRIVER_FUNCTION`

4.56 ps5000aSetTriggerChannelConditions

```
PICO_STATUS ps5000aSetTriggerChannelConditions
(
  int16_t                        handle,
  PS5000A_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 PS5000A_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 ps5000aSetSimpleTrigger.

Applicability	All modes
Arguments	handle, the handle of the required device. * conditions, an array of PS5000A_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 PICO_DRIVER_FUNCTION

4.56.1 PS5000A_TRIGGER_CONDITIONS structure

A structure of this type is passed to ps5000aSetTriggerChannelConditions in the conditions argument to specify the trigger conditions, and is defined as follows:
-

```
typedef struct tPS5000ATriggerConditions
{
  PS5000A_TRIGGER_STATE channelA;
  PS5000A_TRIGGER_STATE channelB;
  PS5000A_TRIGGER_STATE channelC;
  PS5000A_TRIGGER_STATE channelD;
  PS5000A_TRIGGER_STATE external;
  PS5000A_TRIGGER_STATE aux;
  PS5000A_TRIGGER_STATE pulseWidthQualifier;
} PS5000A_TRIGGER_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The ps5000aSetTriggerChannelConditions 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, external, pulseWidthQualifier: the type of condition that should be applied to each channel. Use these constants: PS5000A_CONDITION_DONT_CARE PS5000A_CONDITION_TRUE PS5000A_CONDITION_FALSE The channels that are set to PS5000A_CONDITION_TRUE or PS5000A_CONDITION_FALSE must all meet their conditions simultaneously to produce a trigger. Channels set to PS5000A_CONDITION_DONT_CARE are ignored. aux: not used

4.57 ps5000aSetTriggerChannelDirections

```
PICO_STATUS ps5000aSetTriggerChannelDirections
(
  int16_t                    handle,
  PS5000A_THRESHOLD_DIRECTION channelA,
  PS5000A_THRESHOLD_DIRECTION channelB,
  PS5000A_THRESHOLD_DIRECTION channelC;
  PS5000A_THRESHOLD_DIRECTION channelD;
  PS5000A_THRESHOLD_DIRECTION ext,
  PS5000A_THRESHOLD_DIRECTION aux
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device `channelA`, `channelB`, `channelC`, `channelD`, `ext`, 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 `ps5000aSetPulseWidthQualifier` for more information. `aux`: not used

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

`PS5000A_THRESHOLD_DIRECTION` constants

`PS5000A_ABOVE`	for gated triggers: above the upper threshold
`PS5000A_ABOVE_LOWER`	for gated triggers: above the lower threshold
`PS5000A_BELOW`	for gated triggers: below the upper threshold
`PS5000A_BELOW_LOWER`	for gated triggers: below the lower threshold
`PS5000A_RISING`	for threshold triggers: rising edge, using upper threshold
`PS5000A_RISING_LOWER`	for threshold triggers: rising edge, using lower threshold
`PS5000A_FALLING`	for threshold triggers: falling edge, using upper threshold
`PS5000A_FALLING_LOWER`	for threshold triggers: falling edge, using lower threshold
`PS5000A_RISING_OR_FALLING`	for threshold triggers: either edge
`PS5000A_INSIDE`	for window-qualified triggers: inside window
`PS5000A_OUTSIDE`	for window-qualified triggers: outside window
`PS5000A_ENTER`	for window triggers: entering the window
`PS5000A_EXIT`	for window triggers: leaving the window
`PS5000A_ENTER_OR_EXIT`	for window triggers: either entering or leaving the window
`PS5000A_POSITIVE_RUNT`	for window-qualified triggers
`PS5000A_NEGATIVE_RUNT`	for window-qualified triggers
`PS5000A_NONE`	no trigger

4.58 ps5000aSetTriggerChannelProperties

```
PICO_STATUS ps5000aSetTriggerChannelProperties
(
  int16_t                                 handle,
  PS5000A_TRIGGER_CHANNEL_PROPERTIES * channelProperties,
  int16_t                                 nChannelProperties,
  int16_t                                 auxOutputEnable,
  int32_t                                 autoTriggerMilliseconds
)
```

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

Applicability	All modes
Arguments	`handle,` the handle of the required device. `* channelProperties,` a pointer to an array of PS5000A_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 PICO_INVALID_TRIGGER_PROPERTY PICO_DRIVER_FUNCTION PICO_INVALID_PARAMETER

4.58.1 PS5000A_TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to ps5000aSetTriggerChannelProperties in the channelProperties argument to specify the trigger mechanism, and is defined as follows: -

```
typedef struct tPS5000ATriggerChannelProperties
{
  int16_t                 thresholdUpper;
  uint16_t                thresholdUpperHysteresis;
  int16_t                 thresholdLower;
  uint16_t                thresholdLowerHysteresis;
  PS5000A_CHANNEL         channel;
  PS5000A_THRESHOLD_MODE  thresholdMode;
} PS5000A_TRIGGER_CHANNEL_PROPERTIES
```

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

Elements	thresholdUpper, the upper threshold at which the trigger must fire. This is scaled in 16-bit ADC counts at the currently selected range for that channel. thresholdUpperHysteresis, the hysteresis by which the trigger must exceed the upper threshold before it will fire. It is scaled in 16-bit counts. thresholdLower, the lower threshold at which the trigger must fire. This is scaled in 16-bit ADC counts at the currently selected range for that channel. thresholdLowerHysteresis, the hysteresis by which the trigger must exceed the lower threshold before it will fire. It is scaled in 16-bit counts. channel, the channel to which the properties apply. This can be one of the four input channels listed under ps5000aSetChannel, or PS5000A_TRIGGER_AUX for the AUX input. thresholdMode, either a level or window trigger. Use one of these constants: PS5000A_LEVEL PS5000A_WINDOW

4.59 ps5000aSetTriggerDelay

```
PICO_STATUS ps5000aSetTriggerDelay
(
  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	All modes (but `delay` is ignored in streaming mode)
Arguments	`handle`, the handle of the required 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 500 MS/s, or 2 ns per sample, the total delay would then be: 100 x 2 ns = 200 ns Range: 0 to `MAX_DELAY_COUNT`

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

4.60 ps5000aSigGenArbitraryMinMaxValues

```
PICO_STATUS ps5000aSigGenArbitraryMinMaxValues
(
  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 ps5000aSetSignGenArbitrary for setting up the arbitrary waveform generator (AWG). These values vary between different models in the PicoScope 5000 Series.

Applicability	All models with AWG
Arguments	`handle`, the handle of the required device. `minArbitraryWaveformValue`, on exit, the lowest sample value allowed in the `arbitraryWaveform` buffer supplied to ps5000aSetSignGenArbitrary. `maxArbitraryWaveformValue`, on exit, the highest sample value allowed in the `arbitraryWaveform` buffer supplied to ps5000aSetSignGenArbitrary. `minArbitraryWaveformSize`, on exit, the minimum value allowed for the `arbitraryWaveformSize` argument supplied to ps5000aSetSignGenArbitrary. `maxArbitraryWaveformSize`, on exit, the maximum value allowed for the `arbitraryWaveformSize` argument supplied to ps5000aSetSignGenArbitrary.

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`

4.61 ps5000aSigGenFrequencyToPhase

```
PICO_STATUS ps5000aSigGenFrequencyToPhase
(
  int16_t                 handle,
  double                  frequency,
  PS5000A_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 ps5000aSetSigGenArbitrary or ps5000aSetSigGenPropertiesArbitrary.

Applicability	All models with AWG
Arguments	`handle`, the handle of the required 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`

4.62 ps5000aSigGenSoftwareControl

```
PICO_STATUS ps5000aSigGenSoftwareControl
(
  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.

Gating occurs when the trigger type is set to either PS5000A_SIGGEN_GATE_HIGH or PS5000A_SIGGEN_GATE_LOW. With other trigger types, calling this function causes the signal generator to trigger immediately.

Applicability	Use with ps5000aSetSigGenBuiltIn or ps5000aSetSigGenArbitrary.
Arguments	handle, the handle of the required device state, sets the trigger gate high or low: 0: gate low condition <> 0: gate high condition Ignored if trigger type is not set to either PS5000A_SIGGEN_GATE_HIGH or PS5000A_SIGGEN_GATE_LOW.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_TRIGGER_SOURCE PICO_DRIVER_FUNCTION PICO_NOT_RESPONDING

4.63 ps5000aStop

```
PICO_STATUS ps5000aStop
(
  int16_t       handle
)
```

This function stops the scope device from sampling data.

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

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

Note that if you are using block mode and call this function before the oscilloscope is ready, no capture will be available and the driver will return PICO_NO_SAMPLES_AVAILABLE.

Applicability	All modes
Arguments	handle, the handle of the required device.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_DRIVER_FUNCTION

4.64 ps5000aStreamingReady (callback)

```
typedef void (CALLBACK *ps5000aStreamingReady)
(
  int16_t    handle,
  int32_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 ps5000aGetStreamingLatestValues, and the driver calls it back when streaming-mode data is ready. You can then download the data using the ps5000aGetValuesAsync function.

Your callback function should do nothing more than copy the data to another buffer within your application. To maintain the best application performance, the function should return as quickly as possible without attempting to process or display the data.

Applicability	Streaming mode only
Arguments	`handle`, the handle of the device returning the samples. `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 ps5000aSetDataBuffer. `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 ps5000aRunStreaming. `* pParameter`, a void pointer passed from ps5000aGetStreamingLatestValues. The callback function can write to this location to send any data, such as a status flag, back to the application.
Returns	nothing

4.65 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 projects 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 `ps5000aWrap.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 `ps5000aOpenUnit`.

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

2. Select channels, ranges and AC/DC coupling using `ps5000aSetChannel`.

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 `ps5000aSetDataBuffer` (or for aggregated data collection `ps5000aSetDataBuffers`) 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 `ps5000aRunStreaming`.

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 `ps5000aRunStreaming`.

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

10. Call ps5000aStop, even if the autoStop parameter was set to TRUE.

11. To disconnect a device, call ps5000aCloseUnit.

5 Programming examples

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

6 Driver status codes

Every function in the ps5000a driver returns a **driver status code** from the following list of PICO_STATUS values. These definitions can also be found in the file picoStatus.h, which is included in the Pico Technology SDK. Not all codes apply to the ps5000a driver.

Code (hex)	Symbol and meaning
00	PICO_OK The oscilloscope is functioning correctly
01	PICO_MAX_UNITS_OPENED An attempt has been made to open more than PS5000A_MAX_UNITS
02	PICO_MEMORY_FAIL Not enough memory could be allocated on the host machine
03	PICO_NOT_FOUND No oscilloscope could be found
04	PICO_FW_FAIL Unable to download firmware
05	PICO_OPEN_OPERATION_IN_PROGRESS
06	PICO_OPERATION_FAILED
07	PICO_NOT_RESPONDING The oscilloscope is not responding to commands from the PC
08	PICO_CONFIG_FAIL The configuration information in the oscilloscope has become corrupt or is missing
09	PICO_KERNEL_DRIVER_TOO_OLD The picopp.sys file is too old to be used with the device driver
0A	PICO_EEPROM_CORRUPT The EEPROM has become corrupt, so the device will use a default setting
0B	PICO_OS_NOT_SUPPORTED The operating system on the PC is not supported by this driver
0C	PICO_INVALID_HANDLE There is no device with the handle value passed
0D	PICO_INVALID_PARAMETER A parameter value is not valid
0E	PICO_INVALID_TIMEBASE The timebase is not supported or is invalid
0F	PICO_INVALID_VOLTAGE_RANGE The voltage range is not supported or is invalid
10	PICO_INVALID_CHANNEL The channel number is not valid on this device or no channels have been set
11	PICO_INVALID_TRIGGER_CHANNEL The channel set for a trigger is not available on this device
12	PICO_INVALID_CONDITION_CHANNEL The channel set for a condition is not available on this device
13	PICO_NO_SIGNAL_GENERATOR The device does not have a signal generator
14	PICO_STREAMING_FAILED Streaming has failed to start or has stopped without user request
15	PICO_BLOCK_MODE_FAILED Block failed to start - a parameter may have been set wrongly
16	PICO_NULL_PARAMETER A parameter that was required is NULL
18	PICO_DATA_NOT_AVAILABLE No data is available from a run block call

19	PICO_STRING_BUFFER_TOO_SMALL The buffer passed for the information was too small
1A	PICO_ETS_NOT_SUPPORTED ETS is not supported on this device
1B	PICO_AUTO_TRIGGER_TIME_TOO_SHORT The auto trigger time is less than the time it will take to collect the pre-trigger data
1C	PICO_BUFFER_STALL The collection of data has stalled as unread data would be overwritten
1D	PICO_TOO_MANY_SAMPLES Number of samples requested is more than available in the current memory segment
1E	PICO_TOO_MANY_SEGMENTS Not possible to create number of segments requested
1F	PICO_PULSE_WIDTH_QUALIFIER A null pointer has been passed in the trigger function or one of the parameters is out of range
20	PICO_DELAY One or more of the hold-off parameters are out of range
21	PICO_SOURCE_DETAILS One or more of the source details are incorrect
22	PICO_CONDITIONS One or more of the conditions are incorrect
23	PICO_USER_CALLBACK The driver's thread is currently in the ps5000aBlockReady or ps5000aStreamingReady callback function and therefore the action cannot be carried out
24	PICO_DEVICE_SAMPLING An attempt is being made to get stored data while streaming. Either stop streaming by calling ps5000aStop, or use ps5000aGetStreamingLatestValues.
25	PICO_NO_SAMPLES_AVAILABLE ...because a run has not been completed
26	PICO_SEGMENT_OUT_OF_RANGE The memory index is out of range
27	PICO_BUSY Data cannot be returned yet
28	PICO_STARTINDEX_INVALID The start time to get stored data is out of range
29	PICO_INVALID_INFO The information number requested is not a valid number
2A	PICO_INFO_UNAVAILABLE The handle is invalid so no information is available about the device. Only PICO_DRIVER_VERSION is available.
2B	PICO_INVALID_SAMPLE_INTERVAL The sample interval selected for streaming is out of range
2C	PICO_TRIGGER_ERROR
2D	PICO_MEMORY Driver cannot allocate memory
35	PICO_SIGGEN_OUTPUT_OVER_VOLTAGE The combined peak to peak voltage and the analog offset voltage exceed the allowable voltage the signal generator can produce
36	PICO_DELAY_NULL NULL pointer passed as delay parameter
37	PICO_INVALID_BUFFER The buffers for overview data have not been set while streaming
38	PICO_SIGGEN_OFFSET_VOLTAGE The analog offset voltage is out of range
39	PICO_SIGGEN_PK_TO_PK

	The analog peak to peak voltage is out of range
3A	PICO_CANCELLED A block collection has been cancelled
3B	PICO_SEGMENT_NOT_USED The segment index is not currently being used
3C	PICO_INVALID_CALL The wrong GetValues function has been called for the collection mode in use
3F	PICO_NOT_USED The function is not available
40	PICO_INVALID_SAMPLERATIO The aggregation ratio requested is out of range
41	PICO_INVALID_STATE Device is in an invalid state
42	PICO_NOT_ENOUGH_SEGMENTS The number of segments allocated is fewer than the number of captures requested
43	PICO_DRIVER_FUNCTION You called a driver function while another driver function was still being processed
	PICO_RESERVED
45	PICO_INVALID_COUPLING An invalid coupling type was specified in ps5000aSetChannel
46	PICO_BUFFERS_NOT_SET An attempt was made to get data before a data buffer was defined
47	PICO_RATIO_MODE_NOT_SUPPORTED The selected downsampling mode (used for data reduction) is not allowed
49	PICO_INVALID_TRIGGER_PROPERTY An invalid parameter was passed to ps5000aSetTriggerChannelProperties
4A	PICO_INTERFACE_NOT_CONNECTED The driver was unable to contact the oscilloscope
4D	PICO_SIGGEN_WAVEFORM_SETUP_FAILED A problem occurred in ps5000aSetSigGenBuiltIn or ps5000aSetSigGenArbitrary
4E	PICO_FPGA_FAIL FPGA not successfully set up
4F	PICO_POWER_MANAGER
50	PICO_INVALID_ANALOGUE_OFFSET An impossible analog offset value was specified in ps5000aSetChannel
51	PICO_PLL_LOCK_FAILED Unable to configure the oscilloscope
52	PICO_ANALOG_BOARD The oscilloscope's analog board is not detected, or is not connected to the digital board
53	PICO_CONFIG_FAIL_AWG Unable to configure the signal generator
54	PICO_INITIALISE_FPGA The FPGA cannot be initialized, so unit cannot be opened
56	PICO_EXTERNAL_FREQUENCY_INVALID The frequency for the external clock is not within ±5% of the stated value
57	PICO_CLOCK_CHANGE_ERROR The FPGA could not lock the clock signal
58	PICO_TRIGGER_AND_EXTERNAL_CLOCK_CLASH You are trying to configure the AUX input as both a trigger and a reference clock
59	PICO_PWQ_AND_EXTERNAL_CLOCK_CLASH You are trying to configure the AUX input as both a pulse width qualifier and a reference clock
5A	PICO_UNABLE_TO_OPEN_SCALING_FILE The scaling file set can not be opened

5B	PICO_MEMORY_CLOCK_FREQUENCY The frequency of the memory is reporting incorrectly
5C	PICO_I2C_NOT_RESPONDING The I2C that is being actioned is not responding to requests
5D	PICO_NO_CAPTURES_AVAILABLE There are no captures available and therefore no data can be returned
5E	PICO_NOT_USED_IN_THIS_CAPTURE_MODE The capture mode the device is currently running in does not support the current request
103	PICO_GET_DATA_ACTIVE Reserved
104	PICO_IP_NETWORKED The device is currently connected via the IP Network socket and thus the call made is not supported
105	PICO_INVALID_IP_ADDRESS An IP address that is not correct has been passed to the driver
106	PICO_IPSOCKET_FAILED The IP socket has failed
107	PICO_IPSOCKET_TIMEDOUT The IP socket has timed out
108	PICO_SETTINGS_FAILED The settings requested have failed to be set
109	PICO_NETWORK_FAILED The network connection has failed
10A	PICO_WS2_32_DLL_NOT_LOADED Unable to load the WS2 DLL
10B	PICO_INVALID_IP_PORT The IP port is invalid
10C	PICO_COUPLING_NOT_SUPPORTED The type of coupling requested is not supported on the opened device
10D	PICO_BANDWIDTH_NOT_SUPPORTED Bandwidth limit is not supported on the opened device
10E	PICO_INVALID_BANDWIDTH The value requested for the bandwidth limit is out of range
10F	PICO_AWG_NOT_SUPPORTED The arbitrary waveform generator is not supported by the opened device
110	PICO_ETS_NOT_RUNNING Data has been requested with ETS mode set but run block has not been called, or stop has been called
111	PICO_SIG_GEN_WHITENOISE_NOT_SUPPORTED White noise is not supported on the opened device
112	PICO_SIG_GEN_WAVETYPE_NOT_SUPPORTED The wave type requested is not supported by the opened device
116	PICO_SIG_GEN_PRBS_NOT_SUPPORTED Siggen does not generate pseudo-random binary sequence
117	PICO_ETS_NOT_AVAILABLE_WITH_LOGIC_CHANNELS When a digital port is enabled, ETS sample mode is not available for use
118	PICO_WARNING_REPEAT_VALUE Not applicable to this device
119	PICO_POWER_SUPPLY_CONNECTED 4-channel only - the DC power supply is connected
11A	PICO_POWER_SUPPLY_NOT_CONNECTED 4-channel only - the DC power supply isn't connected
11B	PICO_POWER_SUPPLY_REQUEST_INVALID Incorrect power mode passed for current power source
11C	PICO_POWER_SUPPLY_UNDERVOLTAGE

	The supply voltage from the USB source is too low
11D	PICO_CAPTURING_DATA The device is currently busy capturing data
11E	PICO_USB3_0_DEVICE_NON_USB3_0_PORT A Pico USB 3.0 device has been connected to a non-USB 3.0 port
11F	PICO_NOT_SUPPORTED_BY_THIS_DEVICE A function has been called that is not supported by the current device variant
120	PICO_INVALID_DEVICE_RESOLUTION The device resolution is invalid (out of range)
121	PICO_INVALID_NO_CHANNELS_FOR_RESOLUTION The number of channels which can be enabled is limited in 15 and 16-bit modes
122	PICO_CHANNEL_DISABLED_DUE_TO_USB_POWERED USB power not sufficient to power all channels

7 Enumerated types and constants

The enumerated types used used by the ps5000a driver are defined in the file `ps5000aApi.h`. We recommend that you refer to these constants by name unless your programming language allows only numerical values.

8 Numeric data types

Here is a list of the sizes and ranges of the numeric data types used in the ps5000a 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
double	64	signed (IEEE 754)

9 Glossary

Aggregation. The ps5000a driver can use a method called aggregation to reduce the amount of data your application needs to process. This means that for every block of consecutive samples, it stores only the minimum and maximum values. You can set the number of samples in each block, called the aggregation parameter, when you call `ps5000aRunStreaming` for real-time capture, and when you call `ps5000aGetStreamingLatestValues` to obtain post-processed data.

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

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

Block mode. A sampling mode in which the computer prompts the oscilloscope to collect a block of data into its internal memory before stopping the oscilloscope and transferring the whole block into computer memory. This mode of operation is effective when the input signal being sampled is high frequency. Note: To avoid **aliasing** effects, the maximum input frequency must be less than half the sampling rate.

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

Callback. A mechanism that the ps5000a 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.

Coupling mode. This mode selects either AC or DC coupling in the oscilloscope's input path. Use AC mode for small signals that may be superimposed on a DC level. Use DC mode for measuring absolute voltage levels. Set the coupling mode using `ps5000aSetChannel`.

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

External trigger. This is the BNC socket marked **EXT** or **Ext**. It can be used to start a data collection run but cannot be used to record data.

Flexible power. The 5000 Series oscilloscopes can be powered by either the two-headed USB cable supplied for obtaining power from two USB ports, or a single USB port and the AC adapter (included with 4-channel models only).

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

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

Signal generator. The signal generator output is the BNC socket marked **GEN** or **Gen** on the oscilloscope. If you connect a BNC cable between this and one of the channel inputs, you can send a signal into one of the channels. It can generate a sine, square or triangle wave that can be swept back and forth.

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

USB 1.1. An early version of the Universal Serial Bus standard found on older PCs. Although your PicoScope 5000 Series device will work with a USB 1.1 port, it will operate much more slowly than with a USB 2.0 or 3.0 port.

USB 2.0. A typical USB 2.0 port supports a data transfer rate that is 40 times faster than USB 1.1. USB 2.0 is backwards-compatible with USB 1.1.

USB 3.0. A typical USB 3.0 port supports a data transfer rate that is 10 times faster than USB 2.0. USB 3.0 is backwards-compatible with USB 2.0 and USB 1.1.

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

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


pico
Technology

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

UK headquarters

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

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

sales@picotech.com
support@picotech.com

www.picotech.com

USA headquarters

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

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

ps5000apg.en r3 2016-06-29