PicoScope 2000 Series (A API)

PC Oscilloscopes

Programmer's Guide

ps2000apg.en-3

Contents

1 Introduction ... 1

1 Overview ... 1
2 Minimum PC requirements ... 1
3 Legal information ... 2
4 Company details ... 3

2 Programming the 2000 Series Oscilloscopes ... 4

1 About the ps2000a driver ... 4
2 System requirements ... 4
3 General procedure ... 4
4 Voltage ranges ... 4
5 Digital data ... 4
6 Triggering ... 5
7 Sampling modes ... 5
- 1 Block mode ... 6
- 2 Rapid block mode ... 8
- 3 ETS (Equivalent Time Sampling) ... 13
- 4 Streaming mode ... 15
- 5 Retrieving stored data ... 16

8 Timebases ... 17
9 PicoScope 2205 MSO digital connector diagram ... 17
10 Combining several oscilloscopes ... 18
11 API functions ... 19
- 1 ps2000aBlockReady ... 20
- 2 ps2000aCloseUnit ... 21
- 3 ps2000aDataReady ... 22
- 4 ps2000aEnumerateUnits ... 23
- 5 ps2000aFlashLed ... 24
- 6 ps2000aGetAnalogueOffset ... 25
- 7 ps2000aGetChannelInformation ... 26
- 8 ps2000aGetMaxDownSampleRatio ... 27
- 9 ps2000aGetMaxSegments ... 28
- 10 ps2000aGetNoOfCaptures ... 29
- 11 ps2000aGetNoOfProcessedCaptures ... 30
- 12 ps2000aGetStreamingLatestValues ... 31
- 13 ps2000aGetTimebase ... 32
- 14 ps2000aGetTimebase2 ... 33
- 15 ps2000aGetTriggerTimeOffset ... 34
- 16 ps2000aGetTriggerTimeOffset64 ... 35
- 17 ps2000aGetUnitInfo ... 36
- 18 ps2000aGetValues ... 38
- 19 ps2000aGetValuesAsync ... 40
- 20 ps2000aGetValuesBulk ... 41
- 21 ps2000aGetValuesOverlapped ... 42
- 22 ps2000aGetValuesOverlappedBulk ... 43
- 23 ps2000aGetValuesTriggerTimeOffsetBulk ... 44
- 24 ps2000aGetValuesTriggerTimeOffsetBulk64 ... 45
- 25 ps2000aHoldOff ... 46

26 ps2000aIsReady 47
27 ps2000aIsTriggerOrPulseWidthQualifierEnabled 48
28 ps2000aMaximumValue 49
29 ps2000aMemorySegments 50
30 ps2000aMinimumValue 51
31 ps2000aNoOfStreamingValues 52
32 ps2000aOpenUnit 53
33 ps2000aOpenUnitAsync 54
34 ps2000aOpenUnitProgress 55
35 ps2000aPingUnit 56
36 ps2000aRunBlock 57
37 ps2000aRunStreaming 59
38 ps2000aSetChannel 61
39 ps2000aSetDataBuffer 62
40 ps2000aSetDataBuffers 63
41 ps2000aSetDigitalPort 64
42 ps2000aSetEts 65
43 ps2000aSetEtsTimeBuffer 66
44 ps2000aSetEtsTimeBuffers 67
45 ps2000aSetNoOfCaptures 68
46 ps2000aSetPulseWidthQualifier 69
47 ps2000aSetSigGenArbitrary 72
48 ps2000aSetSigGenBuiltIn 75
49 ps2000aSetSimpleTrigger 78
50 ps2000aSetTriggerChannelConditions 79
51 ps2000aSetTriggerChannelDirections 81
52 ps2000aSetTriggerChannelProperties 82
53 ps2000aSetTriggerDigitalPortProperties 84
54 ps2000aSetTriggerDelay 87
55 ps2000aSigGenSoftwareControl 88
56 ps2000aStop 89
57 ps2000aStreamingReady 90
12 Programming examples 91
1 C 91
2 Excel 91
3 LabView 91
13 Driver status codes 94
14 Enumerated types and constants 98
15 Numeric data types 101
3 Glossary 102
Index 105

1 Introduction

1.1 Overview

The **PicoScope 2000 Series PC Oscilloscopes** from Pico Technology are high-speed real-time measuring instruments. They obtain their power from the USB port, so they do not need an additional power supply. With a built-in external trigger input and arbitrary waveform generator, these scopes contain everything you need in a convenient, portable unit.

This manual explains how to develop your own programs for collecting and analyzing data from the PicoScope 2000 Series oscilloscopes. It applies to devices that use version A of the application programming interface (API), as shown below.

Which manual do I need?

Device	DLL	Manual
PicoScope 2205 MSO PicoScope 2206 PicoScope 2207 PicoScope 2208	ps2000a.dll	***PicoScope 2000 Series (A API) Programmer's Guide -*** This manual.
Other 2000 Series	ps2000.dll	***PicoScope 2000 Series Programmer's Guide*** Available from www.picotech.com.

See the ***PicoScope 2000 Series User's Guide*** for general information on all these devices.

1.2 Minimum PC requirements

To ensure that your **PicoScope 2000 Series PC Oscilloscope** operates correctly, you must have a computer with at least the minimum system requirements to run one of the supported operating systems, as shown in the following table. The performance of the oscilloscope will be better with a more powerful PC, and will benefit from a multi-core processor. Please note the PicoScope software is not installed as part of the SDK.

Item	**Absolute minimum**	**Recommended minimum**	**Recommended full specification**
Operating system	Windows XP SP2 or later Windows Vista Windows 7		
	32 bit and 64* bit versions supported		
Processor	As required by Windows	300 MHz	1 GHz
Memory		256 MB	512 MB
Free disk space**		1.5 GB	2 GB
Ports	USB 1.1 compliant port	USB 2.0 compliant port	

* While the driver will run on a 64 bit operating system, the driver itself is a 32-bit program.

** The PicoScope software does not use all the disk space specified in the table. The free space is required to make Windows run efficiently.

1.3 Legal information

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

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

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

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

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

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

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

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

Support. If you are dissatisfied with the performance of this software, please contact our technical support staff, who will try to fix the problem within a reasonable time. If you are still dissatisfied, please return the product and software to your supplier within 28 days of purchase for a full refund.

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

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

1.4 Company details

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

Address:	Pico Technology James House Colmworth Business Park St Neots Cambridgeshire PE19 8YP United Kingdom
Phone:	+44 (0) 1480 396 395
Fax:	+44 (0) 1480 396 296

Email:

Technical Support:	support@picotech.com
Sales:	sales@picotech.com
Web site:	www.picotech.com

2 Programming the 2000 Series Oscilloscopes

2.1 About the ps2000a driver

Your application will communicate with an API driver called `ps2000a.dll`. The driver exports the ps2000a function definitions in standard C format, but this does not limit you to programming in C. You can use the API with any programming language that supports standard C calls.

The API driver depends on a low-level driver called `WinUsb.sys`. This low-level driver is installed by the SDK when you plug the oscilloscope into the computer for the first time. Your application does not call these drivers directly.

2.2 System requirements

General requirements

See Minimum PC requirements.

USB

The ps2000a driver offers four different methods of recording data, all of which support both USB 1.1 and USB 2.0, although the fastest transfer rates are achieved using USB 2.0.

2.3 General procedure

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

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

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

2.4 Voltage ranges

You can set a device input channel to any voltage range from ±50 mV to ±20 V with the ps2000aSetChannel function. Each sample is scaled to 16 bits, and the minimum and maximum values returned to your application are given by ps2000aMinimumValue and ps2000aMaximumValue respectively.

2.5 Digital data

The data for the digital ports comes back as a 16-bit word. However, only bits 0 to 7 are used in both PORT0 and PORT1:

Data	Bits 0...7	Bits 8...15
PORT0	D0...D7	X
PORT1	D8...D15	X

2.6 Triggering

The **PicoScope 2000 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 2000 trigger function ps2000aSetSimpleTrigger, which in turn calls ps2000aSetTriggerChannelConditions, ps2000aSetTriggerChannelDirections and ps2000aSetTriggerChannelProperties (these can also be called individually, rather than using ps2000aSetSimpleTrigger). 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.

2.7 Sampling modes

PicoScope 2000 Series oscilloscopes can run in various **sampling modes**.

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

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

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

- **Streaming mode.** In this mode, data is passed directly to the PC without being stored in the scope's internal RAM. 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:

    - 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

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

2.7.1 Block mode

In **block mode**, the computer prompts a PicoScope 2000 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 ps2000aMemorySegments).

  *For the PicoScope 2205 MSO, the memory is shared between the digital ports and analogue channels. Therefore if 2 ports and 2 channels are enabled then the memory is divided by four, if either of the 2 ports or 2 channels are enabled and 1 port or 1 channel, the memory is still divided by four.

- **Sampling rate.** A PicoScope 2000 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 2000 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 ps2000aRunBlock, ps2000aStop and ps2000aGetValues.

- **Downsampling.** When the data has been collected, you can set an optional downsampling factor and examine the data. Downsampling is a process that reduces the amount of data by combining adjacent samples. It is useful for zooming in and out of the data without having to repeatedly transfer the entire contents of the scope's buffer to the PC.

- **Memory segmentation.** The scope's internal memory can be divided into segments so that you can capture several waveforms in succession. Configure this using ps2000aMemorySegments.

- **Data retention.** The data is lost when a new run is started in the same segment, the settings are changed, or the scope is powered down.

See Using block mode for programming details.

2.7.1.1 Using block mode

This is the general procedure for reading and displaying data in block mode using a single memory segment:
***Note**: Please use the (*) steps when using the digital ports on the PicoScope 2205 MSO.*

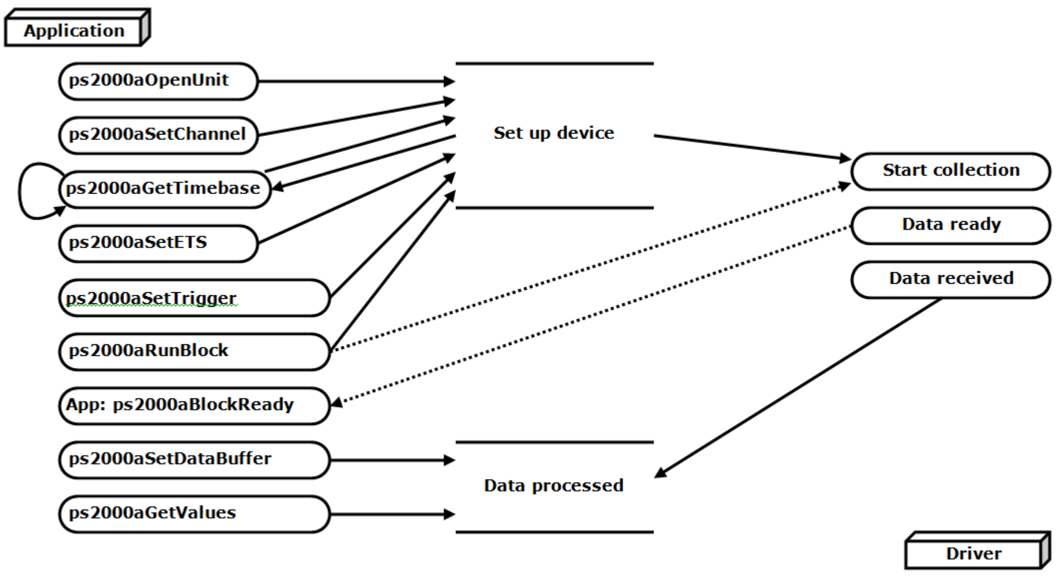
1. Open the oscilloscope using ps2000aOpenUnit.
2. Select channel ranges and AC/DC coupling using ps2000aSetChannel.

*2. Set the digital port using ps2000aSetDigitalPort.

3. Using ps2000aGetTimebase, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions ps2000aSetTriggerChannelConditions, ps2000aSetTriggerChannelDirections and ps2000aSetTriggerChannelProperties to set up the trigger if required.

*4. Use the trigger setup functions ps2000aSetTriggerDigitalPortProperties to set up the digital trigger if required.

5. Start the oscilloscope running using ps2000aRunBlock.
6. Wait until the oscilloscope is ready using the ps2000aBlockReady callback (or poll using ps2000aIsReady).
7. Use ps2000aSetDataBuffer to tell the driver where your memory buffer is.
8. Transfer the block of data from the oscilloscope using ps2000aGetValues.
9. Display the data.
10. Stop the oscilloscope using ps2000aStop.
11. Repeat steps 5 to 9.



12. Request new views of stored data using different downsampling parameters: see Retrieving stored data.

2.7.1.2 Asynchronous calls in block mode

The ps2000aGetValues function may take a long time to complete if a large amount of data is being collected. To avoid hanging the calling thread, it is possible to call ps2000aGetValuesAsync instead. This immediately returns control to the calling thread, which then has the option of waiting for the data or calling ps2000aStop to abort the operation.

2.7.2 Rapid block mode

In normal block mode, the PicoScope 2000 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.

2.7.2.1 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.
***Note**: Please use the * steps when using the digital ports on the PicoScope 2205 MSO.*

Without aggregation

1. Open the oscilloscope using ps2000aOpenUnit.
2. Select channel ranges and AC/DC coupling using ps2000aSetChannel.

*2. Set the digital port using ps2000aSetDigitalPort.

3. Using ps2000aGetTimebase, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions ps2000aSetTriggerChannelConditions, ps2000aSetTriggerChannelDirections and ps2000aSetTriggerChannelProperties to set up the trigger if required.

*4. Use the trigger setup functions ps2000aSetTriggerDigitalPortProperties to set up the digital trigger if required.

5. Set the number of memory segments equal to or greater than the number of captures required using ps2000aMemorySegments. Use ps2000aSetNoOfCaptures before each run to specify the number of waveforms to capture.
6. Start the oscilloscope running using ps2000aRunBlock.
7. Wait until the oscilloscope is ready using the ps2000aIsReady or wait on the callback function.
8. Use ps2000aSetDataBuffer to tell the driver where your memory buffers are.
9. Transfer the blocks of data from the oscilloscope using ps2000aGetValuesBulk.
10. Retrieve the time offset for each data segment using ps2000aGetValuesTriggerTimeOffsetBulk64.
11. Display the data.
12. Repeat steps 6 to 11 if necessary.
13. Stop the oscilloscope using ps2000aStop.

With aggregation

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

8a. Call ps2000aSetDataBuffer or (ps2000aSetDataBuffers) to set up one pair of buffers for every waveform segment required.
9a. Call ps2000aGetValuesBulk for each pair of buffers.
10a. Retrieve the time offset for each data segment using ps2000aGetValuesTriggerTimeOffsetBulk64.

Continue from step 11.

2.7.2.2 Rapid block mode example 1: no aggregation

```
#define MAX_SAMPLES 40000
```

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 32
ps2000aSetNoOfCaptures (handle,  32);

pParameter = false;
ps2000aRunBlock
(
  handle,
  0,                   // noOfPreTriggerSamples
  10000,               // noOfPostTriggerSamples
  1,                   // timebase to be used
  1,
  &timeIndisposedMs,
  1,                   // segment index
  lpReady,
  &pParameter
);
```

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

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

for (int i = 0; i < 10; i++)
{
  for (int c = PS2000A_CHANNEL_A; c <= PS2000A_CHANNEL_B; c++)
  {
    ps2000aSetDataBuffer
    (
      handle,
      c,
      &buffer[c][i],
      MAX_SAMPLES,
      i
     PS2000A_RATIO_MODE_NONE
    );
  }
}
```

Comments: buffer has been created as a two-dimensional array of pointers to shorts, which will contain 40000 samples as defined by MAX_SAMPLES. There are only 10 buffers set, but it is possible to set up to the number of captures you have requested.

```
ps2000aGetValuesBulk
(
  handle,
  &noOfSamples,           // set to MAX_SAMPLES on entering the
  function
  10,                     // fromSegmentIndex
  19,                     // toSegmentIndex
  1,                      // downsampling ratio
  PS2000A_RATIO_MODE_NONE, // downsampling ratio mode
  overflow                // an array of size 10 shorts
)
```

Comments: the number of samples could be up to `noOfPreTriggerSamples + noOfPostTriggerSamples`, the values set in `ps2000aRunBlock`. The samples are always returned from the first sample taken, unlike the `ps2000aGetValues` 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.

```
ps2000aGetValuesTriggerTimeOffsetBulk64
(
  handle,
  times,
  timeUnits,
  10,
  19
)
```

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

2.7.2.3 Rapid block mode example 2: using aggregation

```
#define MAX_SAMPLES 40000
```

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 32
ps2000aSetNoOfCaptures (handle,  32);

pParameter = false;
ps2000aRunBlock
(
  handle,
  0,                  //noOfPreTriggerSamples,
  1000000,            // noOfPostTriggerSamples,
  1,                  // timebase to be used,
  1,
  &timeIndisposedMs,
  1,                  // oversample
  lpReady,
  &pParameter
);
```

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

```
for (int segment = 10; segment < 20; segment++)
{for (int c = PS2000A_CHANNEL_A; c <= PS2000A_CHANNEL_D; c++)
{
  ps2000aSetDataBuffers
  (
    handle,
    c,
    &bufferMax[c],
    &bufferMin[c]
    MAX_SAMPLES
    Segment,
    PS2000A_RATIO_MODE_AGGREGATE
  );
}
```

Comments: since only one waveform will be retrieved at a time, you only need to set up one pair of buffers; one for the maximum samples and one for the minimum samples. Again, the buffer sizes are 40000 samples.

```
    ps2000aGetValues
    (
      handle,
      0,
      &noOfSamples, // set to MAX_SAMPLES on entering
      40000,
      &downSampleRatioMode, //set to RATIO_MODE_AGGREGATE
      index,
      overflow
    );

    ps2000aGetTriggerTimeOffset64
    (
      handle,
      &time,
      &timeUnits,
      index
    )
  }
```

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

2.7.3 ETS (Equivalent Time Sampling)

Note: *Digital ports are not used in ETS mode.*

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 ps2000a set of trigger functions and the ps2000aSetEts function.

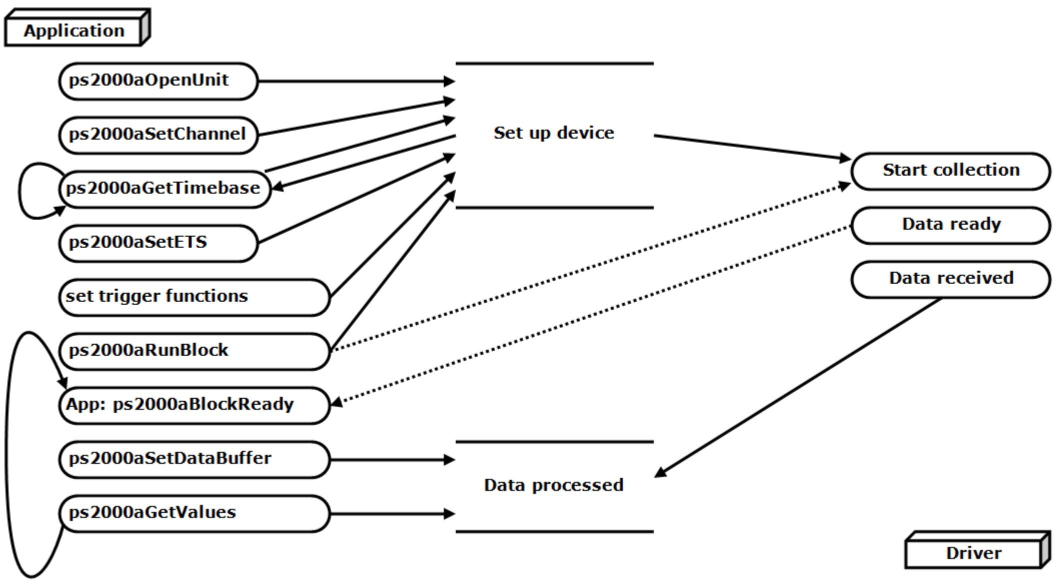
- **Overview.** ETS works by capturing several cycles of a repetitive waveform, then combining them to produce a composite waveform that has a higher effective sampling rate than the individual captures. In some scopes, the hardware adds a short, variable delay, which is a small fraction of a single sampling interval, between each trigger event and the subsequent sample. This shifts each capture slightly in time so that the samples occur at slightly different times relative to those of the previous capture. The result is a larger set of samples spaced by a small fraction of the original sampling interval. Other scopes do not contain special ETS hardware, so the composite waveform is created by software. The maximum effective sampling rates that can be achieved with ETS 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 ps2000aBlockReady callback function when a new waveform is ready for collection. The ps2000aGetValues 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. Digital ports are not used in ETS mode.

2.7.3.1 Using ETS mode

This is the general procedure for reading and displaying data in ETS mode using a single memory segment:
Note: *Digital ports are not used in ETS mode.*

1. Open the oscilloscope using ps2000aOpenUnit.
2. Select channel ranges and AC/DC coupling using ps2000aSetChannel.
3. Using ps2000aGetTimebase, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions ps2000aSetTriggerChannelConditions, ps2000aSetTriggerChannelDirections and ps2000aSetTriggerChannelProperties to set up the trigger if required.
5. Start the oscilloscope running using ps2000aRunBlock.
6. Wait until the oscilloscope is ready using the ps2000aBlockReady callback (or poll using ps2000aIsReady).
7. Use ps2000aSetDataBuffer to tell the driver where your memory buffer is.
8. Transfer the block of data from the oscilloscope using ps2000aGetValues.
9. Display the data.
10. While you want to collect updated captures, repeat steps 6-9.
11. Stop the oscilloscope using ps2000aStop.
12. Repeat steps 5 to 11.



2.7.4 Streaming mode

Streaming mode, unlike block mode, can capture data without gaps between blocks. Streaming mode supports downsampling and triggering, while providing fast streaming at up to 15.625 MS/s (64 ns per sample) when two channels are active, and 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.

2.7.4.1 Using streaming mode

This is the general procedure for reading and displaying data in streaming mode using a single memory segment:
***Note**: Please use the * steps when using the digital ports on the PicoScope 2205 MSO.*

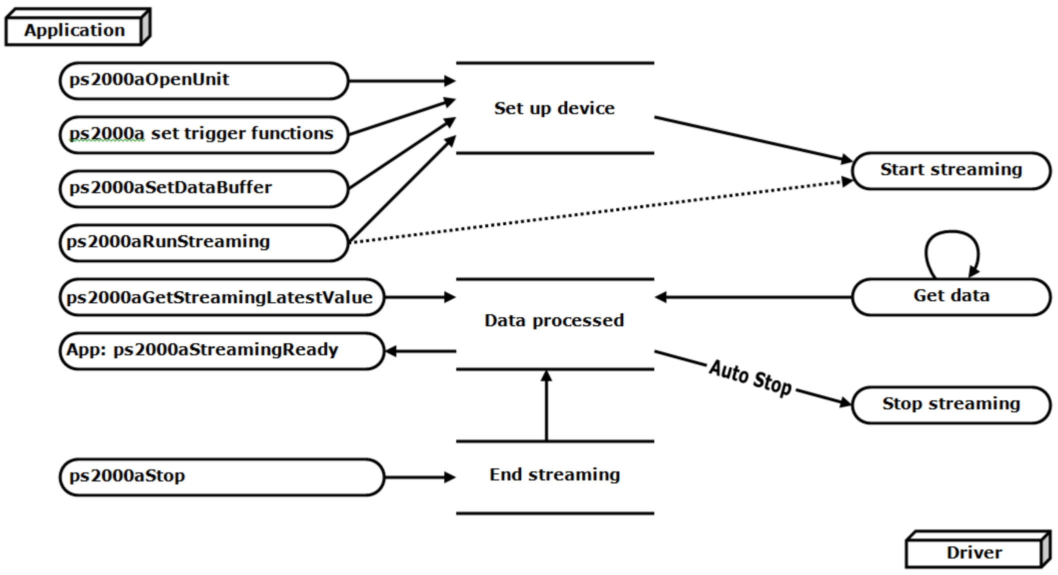
1. Open the oscilloscope using ps2000aOpenUnit.
2. Select channels, ranges and AC/DC coupling using ps2000aSetChannel.

*2. Set the digital port using ps2000aSetDigitalPort.

3. Use the trigger setup functions ps2000aSetTriggerChannelConditions, ps2000aSetTriggerChannelDirections and ps2000aSetTriggerChannelProperties to set up the trigger if required.

*3. Use the trigger setup functions ps2000aSetTriggerDigitalPortProperties to set up the digital trigger if required.

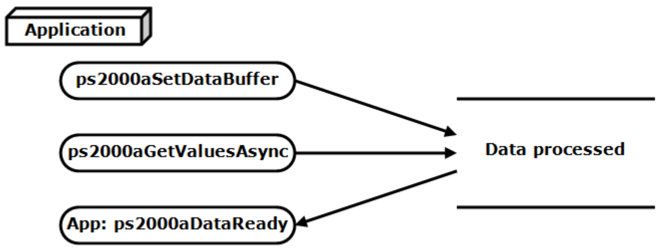
4. Call ps2000aSetDataBuffer to tell the driver where your data buffer is.
5. Set up aggregation and start the oscilloscope running using ps2000aRunStreaming.
6. Call ps2000aGetStreamingLatestValues to get data.
7. Process data returned to your application's function. This example is using Auto Stop, so after the driver has received all the data points requested by the application, it stops the device streaming.
8. Call ps2000aStop, even if Auto Stop is enabled.



9. Request new views of stored data using different downsampling parameters: see Retrieving stored data.

2.7.5 Retrieving stored data

You can collect data from the ps2000a driver with a different downsampling factor when ps2000aRunBlock or ps2000aRunStreaming has already been called and has successfully captured all the data. Use ps2000aGetValuesAsync.



2.8 Timebases

The ps2000a API allows you to select any of 2^{32} different timebases based on a maximum sampling rate of 1 GS/s. The timebases allow slow enough sampling in block mode to overlap the streaming sample intervals, so that you can make a smooth transition between block mode and streaming mode.

timebase	sample interval formula	sample interval examples
0 to 2	$2^{timebase}$ / 1,000,000,000	0 => 1 ns 1 => 2 ns 2 => 4 ns
3 to 2^{32}-1	(timebase - 2) / 125,000,000	3 => 8 ns ... 2^{32}-1 => ~ 34 s

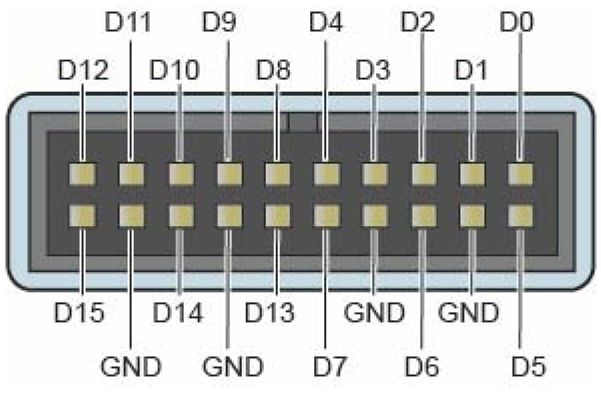
Applicability	Use ps2000aGetTimebase API call.

PicoScope 2205 MSO

timebase	sample interval formula	sample interval examples
0	$2^{timebase}$ / 200,000,000	0 => 5 ns (Timebase 0 is only available when Channel B not active, and when no 3 channels are active)
1 to 2^{32}-1	timebase / 100,000,000	1 => 10 ns 2 => 30 ns 3 => 70 ns ... 2^{32}-1 => ~ 42.94 s

2.9 PicoScope 2205 MSO digital connector diagram

The PicoScope 2205 MSO has a digital input connector. The layout of the 20 pin IDC header plug is detailed below. The diagram is drawn as you look at the front panel of the device.



2.10 Combining several oscilloscopes

It is possible to collect data using up to 64 PicoScope 2000 Series oscilloscopes at the same time, subject to the capabilities of the PC. Each oscilloscope must be connected to a separate USB port. The ps2000aOpenUnit 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 ps2000aBlockReady(...)
// define callback function specific to application

handle1 = ps2000aOpenUnit()
handle2 = ps2000aOpenUnit()

ps2000aSetChannel(handle1)
// set up unit 1
ps2000aSetDigitalPort *(when using PicoScope 2205 MSO only)
ps2000aRunBlock(handle1)

ps2000aSetChannel(handle2)
// set up unit 2
ps2000aSetDigitalPort *(when using PicoScope 2205 MSO only)
ps2000aRunBlock(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
```

2.11 API functions

The ps2000a 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.

ps2000aBlockReady	indicate when block-mode data ready
ps2000aCloseUnit	close a scope device
ps2000aDataReady	indicate when post-collection data ready
ps2000aEnumerateUnits	find all connected oscilloscopes
ps2000aFlashLed	flash the front-panel LED
ps2000aGetChannelInformation	queries which ranges are available on a scope device
ps2000aGetMaxDownSampleRatio	find out aggregation ratio for data
ps2000aGetNoOfCaptures	find out how many captures are available
ps2000aGetNoOfProcessedCaptures	finds out how many captures have been processed
ps2000aGetStreamingLatestValues	get streaming data while scope is running
ps2000aGetTimebase	find out what timebases are available
ps2000aGetTimebase2	find out what timebases are available
ps2000aGetTriggerTimeOffset	find out when trigger occurred (32-bit)
ps2000aGetTriggerTimeOffset64	find out when trigger occurred (64-bit)
ps2000aGetUnitInfo	read information about scope device
ps2000aGetValues	retrieve block-mode data with callback
ps2000aGetValuesAsync	retrieve streaming data with callback
ps2000aGetValuesBulk	retrieve data in rapid block mode
ps2000aGetValuesOverlapped	set up data collection ahead of capture
ps2000aGetValuesOverlappedBulk	set up data collection in rapid block mode
ps2000aGetValuesTriggerTimeOffsetBulk	retrieve rapid-block waveform times (32-bit)
ps2000aGetValuesTriggerTimeOffsetBulk64	retrieve rapid-block waveform times (64-bit)
ps2000aIsReady	poll driver in block mode
ps2000aIsTriggerOrPulseWidthQualifierEnabled	find out whether trigger is enabled
ps2000aMaximumValue	returns the maximum ADC count in get-values calls
ps2000aMemorySegments	divide scope memory into segments
ps2000aMinimumValue	returns the minimum ADC count in get-values calls
ps2000aNoOfStreamingValues	get number of samples in streaming mode
ps2000aOpenUnit	open a scope device
ps2000aOpenUnitAsync	open a scope device without waiting
ps2000aOpenUnitProgress	check progress of OpenUnit call
ps2000aPingUnit	checks communication with opened device
ps2000aRunBlock	start block mode
ps2000aRunStreaming	start streaming mode
ps2000aSetChannel	set up input channels
ps2000aSetDataBuffer	register data buffer with driver
ps2000aSetDataBuffers	register aggregated data buffers with driver
ps2000aSetDigitalPort	set up digital input
ps2000aSetEts	set up equivalent-time sampling
ps2000aSetEtsTimeBuffer	set up buffer for ETS timings (64-bit)
ps2000aSetEtsTimeBuffers	set up buffer for ETS timings (32-bit)
ps2000aSetNoOfCaptures	set number of captures to collect in one run
ps2000aSetPulseWidthQualifier	set up pulse width triggering
ps2000aSetSigGenArbitrary	set up arbitrary waveform generator
ps2000aSetSigGenBuiltIn	set up standard signal generator
ps2000aSetSimpleTrigger	set up level triggers only
ps2000aSetTriggerChannelConditions	specify which channels to trigger on
ps2000aSetTriggerChannelDirections	set up signal polarities for triggering
ps2000aSetTriggerChannelProperties	set up trigger thresholds
ps2000aSetTriggerDelay	set up post-trigger delay
ps2000aSetTriggerDigitalPortProperties	set up digital channel trigger directions
ps2000aSigGenSoftwareControl	trigger the signal generator
ps2000aStop	stop data capture
ps2000aStreamingReady	indicate when streaming-mode data ready

2.11.1 ps2000aBlockReady

```
typedef void (CALLBACK *ps2000aBlockReady)
(
  short       handle,
  PICO_STATUS status,
  void        * pParameter
)
```

This callback function is part of your application. You register it with the ps2000a driver using ps2000aRunBlock, and the driver calls it back when block-mode data is ready. You can then download the data using the ps2000aGetValues 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 ps2000aRunBlock. Your callback function can write to this location to send any data, such as a status flag, back to your application.
Returns	nothing

2.11.2 ps2000aCloseUnit

```
PICO_STATUS ps2000aCloseUnit
(
  short handle
)
```

This function shuts down an oscilloscope.

Applicability	All modes
Arguments	handle, the handle, returned by ps2000aOpenUnit, of the scope device to be closed.
Returns	PICO_OK PICO_HANDLE_INVALID PICO_USER_CALLBACK PICO_DRIVER_FUNCTION

2.11.3 ps2000aDataReady

```
typedef void (__stdcall *ps2000aDataReady)
(
  short              handle,
  PICO_STATUS        status,
  unsigned long      noOfSamples,
  short              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 ps2000aGetValuesAsync, 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 ps2000aGetValuesAsync. 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

2.11.4 ps2000aEnumerateUnits

```
PICO_STATUS ps2000aEnumerateUnits
(
  short * count,
  char  * serials,
  short * serialLth
)
```

This function counts the number of PicoScope 2000A Series units connected to the computer, and returns a list of serial numbers as a string.

Applicability	All modes
Arguments	`* count,` on exit, the number of PicoScope 2000A 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 char 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

2.11.5 ps2000aFlashLed

```
PICO_STATUS ps2000aFlashLed
(
  short handle,
  short start
)
```

This function flashes the LED on the front of the scope without blocking the calling thread. Calls to ps2000aRunStreaming and ps2000aRunBlock 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`

2.11.6 ps2000aGetAnalogueOffset

```
PICO_STATUS ps2000aGetAnalogueOffset
(
  short                 handle,
  PS2000A_RANGE         range,
  PS2000A_COUPLING      coupling
  float               * maximumVoltage,
  float               * minimumVoltage
)
```

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

Applicability	Applicable to all 2000a units, except the PicoScope 2205 MSO
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`, output: parameter set to the maximum voltage allowed for the range, may be `NULL`. `* minimumVoltage`, output: sets the minimum voltage allowed for the range, may be `NULL`. If both `maximumVoltage` and `minimumVoltage` are `NULL`, the driver will return `PICO_NULL_PARAMETER`.

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

If this function is used with the PicoScope 2205 MSO, it will return 0 V.

2.11.7 ps2000aGetChannelInformation

```
PICO_STATUS ps2000aGetChannelInformation
(
  short handle,
  PS2000A_CHANNEL_INFO   info
  int                    probe
  int                  * ranges
  int                  * length
  int                    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: PS2000A_CI_RANGES probe, not used, must be set to 0. * ranges, an array that will be populated with available PS2000A_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 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

2.11.8 ps2000aGetMaxDownSampleRatio

```
PICO_STATUS ps2000aGetMaxDownSampleRatio
(
  short                   handle,
  unsigned long           noOfUnaggregatedSamples,
  unsigned long         * maxDownSampleRatio,
  PS2000A_RATIO_MODE      downSampleRatioMode,
  unsigned short          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 ps2000aGetValues `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

2.11.9 ps2000aGetMaxSegments

```
PICO_STATUS ps2000aGetMaxSegments
(
  short            handle,
  unsigned short * maxsegments
 )
```

This function returns the maximum number of segments allowed for the opened variant. Refer to ps2000aMemorySegments for specific figures.

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

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

2.11.10 ps2000aGetNoOfCaptures

```
PICO_STATUS ps2000aGetNoOfCaptures
(
  short           handle,
  unsigned long * nCaptures
)
```

This function finds out how many captures are available in rapid block mode after ps2000aRunBlock has been called when either the collection completed or the collection of waveforms was interrupted by calling ps2000aStop. The returned value (nCaptures) can then be used to iterate through the number of segments using ps2000aGetValues, or in a single call to ps2000aGetValuesBulk 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 ps2000aRunBlock.
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

2.11.11 ps2000aGetNoOfProcessedCaptures

```
PICO_STATUS ps2000aGetNoOfProcessedCaptures
(
  short           handle,
  unsigned long * nCaptures
)
```

This function finds out how many captures in rapid block mode have been processed after ps2000aRunBlock has been called when either the collection completed or the collection of waveforms was interrupted by calling ps2000aStop. The returned value (nCaptures) can then be used to iterate through the number of segments using ps2000aGetValues, or in a single call to ps2000aGetValuesBulk where it is used to calculate the toSegmentIndex parameter.

Applicability	in rapid block mode
Arguments	handle: handle of the required device. * nCaptures, output: the number of available captures that has been collected from calling ps2000aRunBlock.
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

2.11.12 ps2000aGetStreamingLatestValues

```
PICO_STATUS ps2000aGetStreamingLatestValues
(
  short                        handle,
  ps2000aStreamingReady        lpPs2000AReady,
  void                       * pParameter
)
```

This function instructs the driver to return the next block of values to your ps2000aStreamingReady callback function. You must have previously called ps2000aRunStreaming beforehand to set up streaming.

Applicability	Streaming mode only
Arguments	handle, the handle of the required device. lpPs2000AReady, a pointer to your ps2000aStreamingReady callback function. * pParameter, a void pointer that will be passed to the ps2000aStreamingReady callback function. The callback function may optionally use this pointer to return information to the application.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_INVALID_CALL PICO_BUSY PICO_NOT_RESPONDING PICO_DRIVER_FUNCTION

2.11.13 ps2000aGetTimebase

```
PICO_STATUS ps2000aGetTimebase
(
  short                   handle,
  unsigned long           timebase,
  long                    noSamples,
  long                  * timeIntervalNanoseconds,
  short                   oversample,
  long                  * maxSamples
  unsigned short          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 ps2000aSetChannel.

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 ps2000aGetTimebase2 instead.

To use ps2000aGetTimebase or ps2000aGetTimebase2, 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. `oversample`, not used. `* maxSamples`, on exit, the maximum number of samples available. The result may vary depending on the number of channels enabled, the timebase chosen and the oversample selected. 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

2.11.14 ps2000aGetTimebase2

```
PICO_STATUS ps2000aGetTimebase2
(
  short              handle,
  unsigned long      timebase,
  long               noSamples,
  float            * timeIntervalNanoseconds,
  short              oversample,
  long             * maxSamples
  unsigned short     segmentIndex
)
```

This function is an upgraded version of ps2000aGetTimebase, and returns the time interval as a `float` rather than a `long`. This allows it to return sub-nanosecond time intervals. See ps2000aGetTimebase 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 ps2000aGetTimebase.
Returns	See ps2000aGetTimebase.

2.11.15 ps2000aGetTriggerTimeOffset

```
PICO_STATUS ps2000aGetTriggerTimeOffset
(
  short                   handle
  unsigned long         * timeUpper
  unsigned long         * timeLower
  PS2000A_TIME_UNITS    * timeUnits
  unsigned short          segmentIndex
)
```

This function gets the time, as two 4-byte values, at which the trigger occurred. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture. A 64-bit version of this function, ps2000aGetTriggerTimeOffset64, is also available.

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: - PS2000A_FS PS2000A_PS PS2000A_NS PS2000A_US PS2000A_MS PS2000A_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

2.11.16 ps2000aGetTriggerTimeOffset64

```
PICO_STATUS ps2000aGetTriggerTimeOffset64
(
  short                   handle,
  __int64               * time,
  PS2000A_TIME_UNITS    * timeUnits,
  unsigned short          segmentIndex
)
```

This function gets the time, as a single 64-bit value, at which the trigger occurred. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture. A 32-bit version of this function, ps2000aGetTriggerTimeOffset, is also available.

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: - PS2000A_FS PS2000A_PS PS2000A_NS PS2000A_US PS2000A_MS PS2000A_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

2.11.17 ps2000aGetUnitInfo

```
PICO_STATUS ps2000aGetUnitInfo
(
  short         handle,
  char        * string,
  short         stringLength,
  short       * 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 chars that may be written to `string`. `* requiredSize`, on exit, the required length of the `string` array. `info`, a number specifying what information is required. The possible values are listed in the table below.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_INVALID_INFO PICO_INFO_UNAVAILABLE PICO_DRIVER_FUNCTION

info		**Example**
0	PICO_DRIVER_VERSION Version number of PicoScope 2000A DLL	1,0,0,1
1	PICO_USB_VERSION Type of USB connection to device: 1.1 or 2.0	2.0
2	PICO_HARDWARE_VERSION Hardware version of device	1
3	PICO_VARIANT_INFO Variant number of device	2206
4	PICO_BATCH_AND_SERIAL Batch and serial number of device	KJL87/6
5	PICO_CAL_DATE Calibration date of device	30Sep09
6	PICO_KERNEL_VERSION Version of kernel driver	1,1,2,4
7	PICO_DIGITAL_HARDWARE_VERSION Hardware version of the digital section	1
8	PICO_ANALOGUE_HARDWARE_VERSION Hardware version of the analogue section	1

2.11.18 ps2000aGetValues

```
PICO_STATUS ps2000aGetValues
(
  short                  handle,
  unsigned long          startIndex,
  unsigned long        * noOfSamples,
  unsigned long          downSampleRatio,
  PS2000A_RATIO_MODE     downSampleRatioMode,
  unsigned short         segmentIndex,
  short                * overflow
)
```

This function returns block-mode data, 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.

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: - PS2000A_RATIO_MODE_NONE (downSampleRatio is ignored) PS2000A_RATIO_MODE_AGGREGATE PS2000A_RATIO_MODE_AVERAGE PS2000A_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_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

2.11.18.1 Downsampling modes

Various methods of data reduction, or **downsampling**, are possible with the PicoScope 2000 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 ps2000aGetValues. The following modes are available:

PS2000A_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.
PS2000A_RATIO_MODE_AVERAGE	Reduces every block of *n* values to a single value representing the average (arithmetic mean) of all the values. Equivalent to the 'oversampling' function on older scopes.
PS2000A_RATIO_MODE_DECIMATE	Reduces every block of *n* values to just the first value in the block, discarding all the other values.

2.11.19 ps2000aGetValuesAsync

```
PICO_STATUS ps2000aGetValuesAsync
(
  short                   handle,
  unsigned long           startIndex,
  unsigned long           noOfSamples,
  unsigned long           downSampleRatio,
  PS2000A_RATIO_MODE      downSampleRatioMode,
  unsigned short          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 scope 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:` see ps2000aGetValues `noOfSamples:` see ps2000aGetValues `downSampleRatio:` see ps2000aGetValues `downSampleRatioMode:` see ps2000aGetValues `segmentIndex:` see ps2000aGetValues `* lpDataReady,` a pointer to the user-supplied function that will be called when the data is ready. This will be a ps2000aDataReady function for block-mode data or a ps2000aStreamingReady function for streaming-mode data. `* pParameter,` a void pointer that will be passed to the callback function. The data type is determined by the application.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_DEVICE_SAMPLING PICO_NULL_PARAMETER PICO_STARTINDEX_INVALID PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_PARAMETER PICO_DATA_NOT_AVAILABLE PICO_INVALID_SAMPLERATIO PICO_INVALID_CALL PICO_DRIVER_FUNCTION

2.11.20 ps2000aGetValuesBulk

```
PICO_STATUS ps2000aGetValuesBulk
(
  short                    handle,
  unsigned long          * noOfSamples,
  unsigned short           fromSegmentIndex,
  unsigned short           toSegmentIndex,
  unsigned long            downSampleRatio,
  PS2000A_RATIO_MODE       downSampleRatioMode,
  short                  * 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`: see ps2000aGetValues `downSampleRatioMode`: see ps2000aGetValues `* 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 ps2000aGetValues.
Returns	PICO_OK 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

2.11.21 ps2000aGetValuesOverlapped

```
PICO_STATUS ps2000aGetValuesOverlapped
(
  short                    handle,
  unsigned long            startIndex,
  unsigned long          * noOfSamples,
  unsigned long            downSampleRatio,
  PS2000A_RATIO_MODE       downSampleRatioMode,
  unsigned short           segmentIndex,
  short                  * overflow
)
```

This function allows you to make a deferred data-collection request, which will later be executed, and the arguments validated, when you call ps2000aRunBlock in block mode. The advantage of this function is that the driver makes contact with the scope only once, when you call ps2000aRunBlock, compared with the two contacts that occur when you use the conventional ps2000aRunBlock, ps2000aGetValues calling sequence. This slightly reduces the dead time between successive captures in block mode.

After calling ps2000aRunBlock, you can optionally use ps2000aGetValues to request further copies of the data. This might be required if you wish to display the data with different data reduction settings.

Applicability	Block mode
Arguments	`handle`, the handle of the device `startIndex`: see ps2000aGetValues `* noOfSamples`: see ps2000aGetValues `downSampleRatio`: see ps2000aGetValues `downSampleRatioMode`: see ps2000aGetValues `segmentIndex`: see ps2000aGetValues `* overflow`: see ps2000aGetValuesBulk
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

2.11.22 ps2000aGetValuesOverlappedBulk

```
PICO_STATUS ps2000aGetValuesOverlappedBulk
(
  short                  handle,
  unsigned long          startIndex,
  unsigned long        * noOfSamples,
  unsigned long          downSampleRatio,
  PS2000A_RATIO_MODE     downSampleRatioMode,
  unsigned short         fromSegmentIndex,
  unsigned short         toSegmentIndex,
  short                * overflow
)
```

This function allows you to make a deferred data-collection request, which will later be executed, and the arguments validated, when you call ps2000aRunBlock in rapid block mode. The advantage of this method is that the driver makes contact with the scope only once, when you call ps2000aRunBlock, compared with the two contacts that occur when you use the conventional ps2000aRunBlock, ps2000aGetValuesBulk calling sequence. This slightly reduces the dead time between successive captures in rapid block mode.

After calling ps2000aRunBlock, you can optionally use ps2000aGetValues to request further copies of the data. This might be required if you wish to display the data with different data reduction settings.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the device `startIndex`: see ps2000aGetValues `* noOfSamples`: see ps2000aGetValues `downSampleRatio`: see ps2000aGetValues `downSampleRatioMode`: see ps2000aGetValues `fromSegmentIndex`: see ps2000aGetValuesBulk `toSegmentIndex`: see ps2000aGetValuesBulk `* overflow`, see ps2000aGetValuesBulk
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

2.11.23 ps2000aGetValuesTriggerTimeOffsetBulk

```
PICO_STATUS ps2000aGetValuesTriggerTimeOffsetBulk
(
  short                   handle,
  unsigned long         * timesUpper,
  unsigned long         * timesLower,
  PS2000A_TIME_UNITS    * timeUnits,
  unsigned short          fromSegmentIndex,
  unsigned short          toSegmentIndex
)
```

This function retrieves the time offsets, as lower and upper 32-bit values, for waveforms obtained in rapid block mode.

This function is provided for use in programming environments that do not support 64-bit integers. If your programming environment supports this data type, it is easier to use ps2000aGetValuesTriggerTimeOffsetBulk64.

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 ps2000aGetTriggerTimeOffset 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_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

2.11.24 ps2000aGetValuesTriggerTimeOffsetBulk64

```
PICO_STATUS ps2000aGetValuesTriggerTimeOffsetBulk64
(
  short                     handle,
  __int64                 * times,
  PS2000A_TIME_UNITS      * timeUnits,
  unsigned short            fromSegmentIndex,
  unsigned short            toSegmentIndex
)
```

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

A 32-bit version of this function, ps2000aGetValuesTriggerTimeOffsetBulk, is available for use with programming languages that do not support 64-bit integers.

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 ps2000aGetTriggerTimeOffset64 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_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

2.11.25 ps2000aHoldOff

```
PICO_STATUS ps2000aHoldOff
(
  short                  handle,
  u_int64_t              holdoff,
  PS2000A_HOLDOFF_TYPE type
)
```

This function specifies the minimum time after the end of a capture before the next capture can begin.

Applicability	All
Arguments	handle, the handle of the device holdoff, the holdoff time, qualified by type type, the method used for defining holdoff: PS2000A_TIME: time in sample periods
Returns	PICO_OK PICO_INVALID_HANDLE

2.11.26 ps2000aIsReady

```
PICO_STATUS ps2000aIsReady
(
  short   handle,
  short * ready
)
```

This function may be used instead of a callback function to receive data from `ps2000aRunBlock`. To use this method, pass a NULL pointer as the `lpReady` argument to `ps2000aRunBlock`. 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 `ps2000aGetValues` 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`

2.11.27 ps2000aIsTriggerOrPulseWidthQualifierEnabled

```
PICO_STATUS ps2000aIsTriggerOrPulseWidthQualifierEnabled
(
  short   handle,
  short * triggerEnabled,
  short * 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 ps2000aRunBlock or ps2000aRunStreaming.
Arguments	`handle`, the handle of the required device `* triggerEnabled`, on exit, indicates whether the trigger will successfully be set when ps2000aRunBlock or ps2000aRunStreaming 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 ps2000aRunBlock or ps2000aRunStreaming 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`

2.11.28 ps2000aMaximumValue

```
PICO_STATUS ps2000aMaximumValue
(
  short   handle
  short * value
)
```

This function returns the maximum ADC count returned by calls to get values.

Applicability	All modes
Arguments	`handle,` the handle of the required device `* value,` output: the maximum ADC value.

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

2.11.29 ps2000aMemorySegments

```
PICO_STATUS ps2000aMemorySegments
(
  short            handle
  unsigned short   nSegments,
  long           * 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 32. `* 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`

2.11.30 ps2000aMinimumValue

```
PICO_STATUS ps2000aMinimumValue
(
  short   handle
  short * value
)
```

This function returns the minimum ADC count returned by calls to get values.

Applicability	All modes
Arguments	`handle,` the handle of the required device. `* value,` output: the minimum ADC value.
Returns	PICO_OK PICO_USER_CALLBACK PICO_INVALID_HANDLE PICO_TOO_MANY_SEGMENTS PICO_MEMORY PICO_DRIVER_FUNCTION

2.11.31 ps2000aNoOfStreamingValues

```
PICO_STATUS ps2000aNoOfStreamingValues
(
  short           handle,
  unsigned long * noOfValues
)
```

This function returns the number of samples available after data collection in streaming mode. Call it after calling ps2000aStop.

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

2.11.32 ps2000aOpenUnit

```
PICO_STATUS ps2000aOpenUnit
(
  short * handle,
  char  * serial
)
```

This function opens a PicoScope 2000 Series scope attached to the computer. The maximum number of units that can be opened depends on the operating system, the kernel driver and the computer.

Applicability	All modes
Arguments	`* handle`, on exit, the result of the attempt to open a scope: -1 : if the scope fails to open 0 : if no scope is found > 0 : a number that uniquely identifies the scope If a valid handle is returned, it must be used in all subsequent calls to API functions to identify this scope. `* serial`, on entry, a null-terminated string containing the serial number of the scope to be opened. If `serial` is NULL then the function opens the first scope found; otherwise, it tries to open the scope that matches the string.

Returns	`PICO_OK`
	`PICO_OS_NOT_SUPPORTED`
	`PICO_OPEN_OPERATION_IN_PROGRESS`
	`PICO_EEPROM_CORRUPT`
	`PICO_KERNEL_DRIVER_TOO_OLD`
	`PICO_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`

2.11.33 ps2000aOpenUnitAsync

```
PICO_STATUS ps2000aOpenUnitAsync
(
  short * status
  char  * serial
)
```

This function opens a scope without blocking the calling thread. You can find out when it has finished by periodically calling ps2000aOpenUnitProgress 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 ps2000aOpenUnit
Returns	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_OPERATION_FAILED

2.11.34 ps2000aOpenUnitProgress

```
PICO_STATUS ps2000aOpenUnitProgress
(
  short * handle,
  short * progressPercent,
  short * complete
)
```

This function checks on the progress of a request made to ps2000aOpenUnitAsync to open a scope.

Applicability	Use after ps2000aOpenUnitAsync
Arguments	`* handle:` see ps2000aOpenUnit. 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`

2.11.35 ps2000aPingUnit

```
PICO_STATUS ps2000aPingUnit
(
  short  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_BUSY PICO_NOT_RESPONDING

2.11.36 ps2000aRunBlock

```
PICO_STATUS ps2000aRunBlock
(
  short                   handle,
  long                    noOfPreTriggerSamples,
  long                    noOfPostTriggerSamples,
  unsigned long           timebase,
  short                   oversample,
  long                  * timeIndisposedMs,
  unsigned short          segmentIndex,
  ps2000aBlockReady       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 ignored and `noOfPostTriggerSamples` specifies the maximum number of samples to collect. `noOfPostTriggerSamples`, the number of samples to be taken after a trigger event. If no trigger event has been set then this specifies the maximum number of samples to be taken. If a trigger condition has been set, this specifies the number of samples to be taken after a trigger has fired, and the number of samples to be collected is then: - `noOfPreTriggerSamples + noOfPostTriggerSamples` `timebase`, a number in the range 0 to 2^{32}-1. See the guide to calculating timebase values. `oversample`, not used. `* 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 ps2000aBlockReady callback function that the driver will call when the data has been collected. To use the ps2000aIsReady polling method instead of a callback function, set this pointer to NULL.

	* pParameter, a void pointer that is passed to the ps2000aBlockReady callback function. The callback can use this pointer to return arbitrary data to the application.
Returns	PICO_OK 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

2.11.37 ps2000aRunStreaming

```
PICO_STATUS ps2000aRunStreaming
(
  short                   handle,
  unsigned long         * sampleInterval,
  PS2000A_TIME_UNITS      sampleIntervalTimeUnits
  unsigned long           maxPreTriggerSamples,
  unsigned long           maxPostTriggerSamples,
  short                   autoStop,
  unsigned long           downSampleRatio,
  PS2000A_RATIO_MODE      downSampleRatioMode,
  unsigned long           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 ps2000aGetStreamingLatestValues 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: PS2000A_FS PS2000A_PS PS2000A_NS PS2000A_US PS2000A_MS PS2000A_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: see ps2000aGetValues downSampleRatioMode: see ps2000aGetValues

	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 ps2000aSetDataBuffer.
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_TRIGGER_ERROR PICO_INVALID_SAMPLE_INTERVAL PICO_INVALID_BUFFER PICO_DRIVER_FUNCTION PICO_FW_FAIL PICO_MEMORY

2.11.38 ps2000aSetChannel

```
PICO_STATUS ps2000aSetChannel
(
  short                      handle,
  PS2000A_CHANNEL            channel,
  short                      enabled,
  PS2000A_COUPLING           type,
  PS2000A_RANGE              range,
  float                      analogOffset,
)
```

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: PS2000A_CHANNEL_A: Channel A input PS2000A_CHANNEL_B: Channel B 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: PS2000A_AC: 1 megohm impedance, AC coupling. The channel accepts input frequencies from about 1 hertz up to its maximum -3 dB analog bandwidth. PS2000A_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: PS2000A_50MV: ±50 mV PS2000A_1V: ±1 V PS2000A_100MV: ±100 mV PS2000A_2V: ±2 V PS2000A_200MV: ±200 mV PS2000A_5V: ±5 V PS2000A_500MV: ±500 mV PS2000A_10V: ±10 V PS2000A_20V: ±20 V analogOffset, 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 the ps2000aGetAnalogueOffset. **Note:** analogOffset has no effect when using the PicoScope 2205 MSO unit.
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

2.11.39 ps2000aSetDataBuffer

```
PICO_STATUS ps2000aSetDataBuffer
(
  short                    handle,
  PS2000A_CHANNEL          channel,
  short                  * buffer,
  long                     bufferLth,
  unsigned short           segmentIndex
  PS2000A_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 ps2000aSetDataBuffers 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 for analog channels:
	PS2000A_CHANNEL_A
	PS2000A_CHANNEL_B
	
	To set the buffer for a Digital Port then one of these values must be used:
	
	`PS2000A_DIGITAL_PORT0 = 0x80`
	`PS2000A_DIGITAL_PORT1 = 0x81`
	
	`* 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 ps2000aGetValues for the available modes, but note that a single call to ps2000aSetDataBuffer can only associate one buffer with one downsampling mode. If you intend to call ps2000aGetValues with more than one downsampling mode activated, then you must call ps2000aSetDataBuffer 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

2.11.40 ps2000aSetDataBuffers

```
PICO_STATUS ps2000aSetDataBuffers
(
  short                    handle,
  PS2000A_CHANNEL          channel,
  short                  * bufferMax,
  short                  * bufferMin,
  long                     bufferLth,
  unsigned short           segmentIndex
  PS2000A_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 ps2000aSetDataBuffer 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:
	`PS2000A_CHANNEL_A`
	`PS2000A_CHANNEL_B`
	
	To set the buffer for a Digital Port then one of these values must be used:
	
	`PS2000A_DIGITAL_PORT0 = 0x80`
	`PS2000A_DIGITAL_PORT1 = 0x81`
	
	`* 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 ps2000aGetValues

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

2.11.41 ps2000aSetDigitalPort

```
PICO_STATUS ps2000aSetDigitalPort
(
  short                   handle,
  PS2000A_DIGITAL_PORT    port,
  short                   enabled,
  short                   logiclevel
)
```

This function is used to enable the digital port and set the logic level (the voltage point at which the state transitions from 0 to 1).

Applicability	Block and streaming modes with aggregation.	
Arguments	`handle,`	the handle of the required device.

	`port,`	`PS2000A_DIGITAL_PORT0 = 0x80, // digital channel 0 - 7`
		`PS2000A_DIGITAL_PORT1 = 0x81, // digital channel 8 - 15`

	`enabled,`	whether or not to enable the channel. The values are: `TRUE`: enable `FALSE`: do not enable
	`logiclevel,`	the voltage point at which the state transitions from 0 to 1. Accepted values between 32767 (5 V) and -32767 (-5 V)

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`	

2.11.42 ps2000aSetEts

```
PICO_STATUS ps2000aSetEts
(
  short                 handle,
  PS2000A_ETS_MODE      mode,
  short                 etsCycles,
  short                 etsInterleave,
  long                * 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 ETS mode not available when digital port(s) enabled
Arguments	`handle,` the handle of the required device `mode,` the ETS mode. Use one of these values: `PS2000A_ETS_OFF:` disables ETS `PS2000A_ETS_FAST:` enables ETS and provides `etsCycles` of data, which may contain data from previously returned cycles `PS2000A_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 `PS2206_MAX_ETS_CYCLES`, `PS2207_MAX_ETS_CYCLES` or `PS2208_MAX_ETS_CYCLES`. `etsInterleave,` the number of waveforms to combine into a single ETS capture. Maximum value is `PS2206_MAX_INTERLEAVE`, `PS2207_MAX_INTERLEAVE` or `PS2208_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

2.11.43 ps2000aSetEtsTimeBuffer

```
PICO_STATUS ps2000aSetEtsTimeBuffer
(
  short       handle,
  __int64   * buffer,
  long        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 ps2000aSetEtsTimeBuffers instead.
Arguments	`handle`, the handle of the required device `* buffer`, an array of 64-bit words, each representing the time in picoseconds 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`

2.11.44 ps2000aSetEtsTimeBuffers

```
PICO_STATUS ps2000aSetEtsTimeBuffers
(
  short           handle,
  unsigned long * timeUpper,
  unsigned long * timeLower,
  long            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 ps2000aSetEtsTimeBuffer 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 picoseconds at which the sample was captured `* timeLower`, an array of 32-bit words, each representing the lower 32 bits of the time in picoseconds 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

2.11.45 ps2000aSetNoOfCaptures

```
PICO_STATUS ps2000aSetNoOfCaptures
(
  short          handle,
  unsigned short 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`

2.11.46 ps2000aSetPulseWidthQualifier

```
PICO_STATUS ps2000aSetPulseWidthQualifier
(
  short                          handle,
  PS2000A_PWQ_CONDITIONS       * conditions,
  short                          nConditions,
  PS2000A_THRESHOLD_DIRECTION    direction,
  unsigned long                  lower,
  unsigned long                  upper,
  PS2000A_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 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 PS2000A_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 `PS2000A_MAX_PULSE_WIDTH_QUALIFIER_COUNT`. `direction`, the direction of the signal required for the pulse width trigger to fire. See PS2000A_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, `PS2000A_RISING` and `PS2000A_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 `PS2000A_RISING` as the `direction` argument for both ps2000aSetTriggerConditions and ps2000aSetPulseWidthQualifier at the same time. There is no such restriction when using window triggers. `lower`, the lower limit of the pulse-width counter with relation to number of samples captured on the device. `upper`, the upper limit of the pulse-width counter with relation to number of samples captured on the device. This parameter is used only when the type is set to `PS2000A_PW_TYPE_IN_RANGE` or `PS2000A_PW_TYPE_OUT_OF_RANGE`.

Arguments	`type`, the pulse-width type, one of these constants:
	`PS2000A_PW_TYPE_NONE`: do not use the pulse width qualifier
	`PS2000A_PW_TYPE_LESS_THAN`: pulse width less than `lower`
	`PS2000A_PW_TYPE_GREATER_THAN`: pulse width greater than `lower`
	`PS2000A_PW_TYPE_IN_RANGE`: pulse width between `lower` and `upper`
	`PS2000A_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`

2.11.46.1 ps2000A_PWQ_CONDITIONS structure

A structure of this type is passed to ps2000aSetPulseWidthQualifier in the `conditions` argument to specify the trigger conditions. It is defined as follows:

```
typedef struct tPwqConditions
{
  PS2000A_TRIGGER_STATE channelA;
  PS2000A_TRIGGER_STATE channelB;
  PS2000A_TRIGGER_STATE channelC;
  PS2000A_TRIGGER_STATE channelD;
  PS2000A_TRIGGER_STATE external;
  PS2000A_TRIGGER_STATE aux;
  PS2000A_TRIGGER_STATE digital;
} PS2000A_PWQ_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The ps2000aSetPulseWidthQualifier function can OR together a number of these structures to produce the final pulse width qualifier, which can therefore be any possible Boolean function of the scope's inputs.

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

Elements	`channelA, channelB, external:` the type of condition that should be applied to each channel. Use these constants: - PS2000A_CONDITION_DONT_CARE PS2000A_CONDITION_TRUE PS2000A_CONDITION_FALSE The channels that are set to PS2000A_CONDITION_TRUE or PS2000A_CONDITION_FALSE must all meet their conditions simultaneously to produce a trigger. Channels set to PS2000A_CONDITION_DONT_CARE are ignored. `channelC, channelD, aux, digital:` not used

2.11.47 ps2000aSetSigGenArbitrary

```
PICO_STATUS ps2000aSetSigGenArbitrary
(
  short                          handle,
  long                           offsetVoltage,
  unsigned long                  pkToPk
  unsigned long                  startDeltaPhase,
  unsigned long                  stopDeltaPhase,
  unsigned long                  deltaPhaseIncrement,
  unsigned long                  dwellCount,
  short                        * arbitraryWaveform,
  long                           arbitraryWaveformSize,
  PS2000A_SWEEP_TYPE             sweepType,
  PS2000A_EXTRA_OPERATIONS       operation,
  PS2000A_INDEX_MODE             indexMode,
  unsigned long                  shots,
  unsigned long                  sweeps,
  PS2000A_SIGGEN_TRIG_TYPE       triggerType,
  PS2000A_SIGGEN_TRIG_SOURCE     triggerSource,
  short                          extInThreshold
)
```

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

The arbitrary waveform generator uses direct digital synthesis (DDS). It maintains a 32-bit phase accumulator that indicates the present location in the waveform buffer. 13 bits (D30...D18) of the accumulator are used as an index into a buffer containing the arbitrary waveform.

The generator steps through the waveform by adding a "delta phase" between 1 and 2^{32}-1 to the phase accumulator every 50 ns. If the delta phase is constant, then the generator produces a waveform at a constant frequency:

frequency = 20 MHz x ([Delta Phase] / $2^{(32-14)}$) / [Waveform Length]

It is also possible to sweep the frequency by progressively modifying the delta phase. This is done by setting up a "delta phase increment" which is added to the delta phase at specified intervals.

Applicability	All modes
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 `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

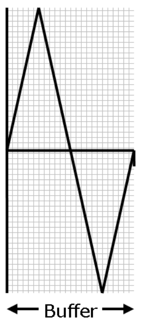
	deltaPhaseIncrement, the amount added to the delta phase value every time the dwellCount period expires. This determines the amount by which the generator sweeps the output frequency in each dwell period. dwellCount, the time, in 50 ns steps, between successive additions of deltaPhaseIncrement to the delta phase accumulator. This determines the rate at which the generator sweeps the output frequency. Minimum value: PS2000A_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 MAX_SIG_GEN_BUFFER_SIZE. sweepType, determines whether the startDeltaPhase is swept up to the stopDeltaPhase, or down to it, or repeatedly swept up and down. Use one of these values: - PS2000A_UP PS2000A_DOWN PS2000A_UPDOWN PS2000A_DOWNUP operation, the type of waveform to be produced, specified by one of the following enumerated types: PS2000A_ES_OFF, normal signal generator operation specified by wavetype. PS2000A_WHITENOISE, the signal generator produces white noise and ignores all settings except pkToPk and offsetVoltage. PS2000A_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: PS2000A_SINGLE PS2000A_DUAL
Arguments	shots, see ps2000aSigGenBuiltIn sweeps, see ps2000aSigGenBuiltIn triggerType, see ps2000aSigGenBuiltIn triggerSource, see ps2000aSigGenBuiltIn extInThreshold, see ps2000aSigGenBuiltIn
Returns	PICO_OK PICO_AWG_NOT_SUPPORTED 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

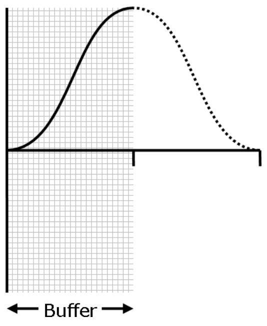
2.11.47.1 AWG index modes

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 and quad modes make 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.



2.11.48 ps2000aSetSigGenBuiltIn

```
PICO_STATUS ps2000aSetSigGenBuiltIn
(
  short                        handle,
  long                         offsetVoltage,
  unsigned long                pkToPk
  PS2000A_WAVE_TYPE            waveType
  float                        startFrequency,
  float                        stopFrequency,
  float                        increment,
  float                        dwellTime,
  PS2000A_SWEEP_TYPE           sweepType,
  PS2000A_EXTRA_OPERATIONS     operation,
  unsigned long                shots,
  unsigned long                sweeps,
  PS2000A_SIGGEN_TRIG_TYPE     triggerType,
  PS2000A_SIGGEN_TRIG_SOURCE   triggerSource,
  short                        extInThreshold
)
```

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

Applicability	All modes.
Arguments	`handle`, 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. `PS2000A_SINE` sine wave `PS2000A_SQUARE` square wave `PS2000A_TRIANGLE` triangle wave `PS2000A_DC_VOLTAGE` DC voltage The following `waveTypes` apply to B models only. `PS2000A_RAMP_UP` rising sawtooth `PS2000A_RAMP_DOWN` falling sawtooth `PS2000A_SINC` sin(x)/x `PS2000A_GAUSSIAN` Gaussian `PS2000A_HALF_SINE` half (full-wave rectified) sine `startFrequency`, the frequency that the signal generator will initially produce. For allowable values see `PS2000A_SINE_MAX_FREQUENCY`and related values. `stopFrequency`, the frequency at which the sweep reverses direction or returns to the initial frequency

Arguments

`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,` or in the opposite direction, or repeatedly reverse direction. Use one of these constants:

- PS2000A_UP
- PS2000A_DOWN
- PS2000A_UPDOWN
- PS2000A_DOWNUP

`operation,` the type of waveform to be produced, specified by one of the following enumerated types:

- PS2000A_ES_OFF, normal signal generator operation specified by wavetype.
- PS2000A_WHITENOISE, the signal generator produces white noise and ignores all settings except `pkToPk` and `offsetVoltage.`
- PS2000A_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...`PS2000A_MAX_SWEEPS_SHOTS`:` the number of cycles of the waveform to be produced after a trigger event. `sweeps` must be zero.
- PS2000A_SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN`:` start and run continuously after trigger occurs (PicoScope 2206, 2207 and 2208 only)

`sweeps,`

- `0`: produce number of cycles specified by `shots`
- `1..`PS2000A_MAX_SWEEPS_SHOTS`:` the number of times to sweep the frequency after a trigger event, according to `sweepType.` `shots` must be zero.
- PS2000A_SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN`:` start a sweep and continue after trigger occurs (PicoScope 2206, 2207 and 2208 only)

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

`PS2000A_SIGGEN_RISING`	trigger on rising edge
`PS2000A_SIGGEN_FALLING`	trigger on falling edge
`PS2000A_SIGGEN_GATE_HIGH`	run while trigger is high
`PS2000A_SIGGEN_GATE_LOW`	run while trigger is low

	`triggerSource`, the source that will trigger the signal generator. `PS2000A_SIGGEN_NONE` run without waiting for trigger `PS2000A_SIGGEN_SCOPE_TRIG` use scope trigger `PS2000A_SIGGEN_AUX_IN` use EXT input `PS2000A_SIGGEN_SOFT_TRIG` wait for software trigger provided by ps2000aSigGenSoftwareControl `PS2000A_SIGGEN_TRIGGER_RAW` reserved If a trigger source other than `P2000A_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_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`

2.11.49 ps2000aSetSimpleTrigger

```
PICO_STATUS ps2000aSetSimpleTrigger
(
  short                          handle,
  short                          enable,
  PS2000A_CHANNEL                source,
  short                          threshold,
  PS2000A_THRESHOLD_DIRECTION    direction,
  unsigned long                  delay,
  short                          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 being taken. `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`

2.11.50 ps2000aSetTriggerChannelConditions

```
PICO_STATUS ps2000aSetTriggerChannelConditions
(
  short                          handle,
  PS2000A_TRIGGER_CONDITIONS *   conditions,
  short                          nConditions
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is defined by one or more PS2000A_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 ps2000aSetSimpleTrigger.

Applicability	All modes
Arguments	handle, the handle of the required device. * conditions, an array of PS2000A_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

2.11.50.1 PS2000A_TRIGGER_CONDITIONS structure

A structure of this type is passed to ps2000aSetTriggerChannelConditions in the `conditions` argument to specify the trigger conditions, and is defined as follows: -

```
typedef struct tTriggerConditions
{
  PS2000A_TRIGGER_STATE channelA;
  PS2000A_TRIGGER_STATE channelB;
  PS2000A_TRIGGER_STATE channelC;
  PS2000A_TRIGGER_STATE channelD;
  PS2000A_TRIGGER_STATE external;
  PS2000A_TRIGGER_STATE aux;
  PS2000A_TRIGGER_STATE pulseWidthQualifier;
  PS2000A_TRIGGER_STATE digital;
} PS2000A_TRIGGER_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The ps2000aSetTriggerChannelConditions 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, external, pulseWidthQualifier:` the type of condition that should be applied to each channel. Use these constants: PS2000A_CONDITION_DONT_CARE PS2000A_CONDITION_TRUE PS2000A_CONDITION_FALSE The channels that are set to PS2000A_CONDITION_TRUE or PS2000A_CONDITION_FALSE must all meet their conditions simultaneously to produce a trigger. Channels set to PS2000A_CONDITION_DONT_CARE are ignored. `channelC, channelD, aux, digital:` not used

2.11.51 ps2000aSetTriggerChannelDirections

```
PICO_STATUS ps2000aSetTriggerChannelDirections
(
  short                       handle,
  PS2000A_THRESHOLD_DIRECTION channelA,
  PS2000A_THRESHOLD_DIRECTION channelB,
  PS2000A_THRESHOLD_DIRECTION channelC;
  PS2000A_THRESHOLD_DIRECTION channelD;
  PS2000A_THRESHOLD_DIRECTION ext,
  PS2000A_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, 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 ps2000aSetPulseWidthQualifier for more information. `channelC, channelD and aux:` not used
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_INVALID_PARAMETER

`PS2000A_THRESHOLD_DIRECTION` **constants**

Constant	Trigger type	Direction
PS2000A_ABOVE	gated	above the upper threshold
PS2000A_ABOVE_LOWER	gated	above the lower threshold
PS2000A_BELOW	gated	below the upper threshold
PS2000A_BELOW_LOWER	gated	below the lower threshold
PS2000A_RISING	threshold	rising edge, using upper threshold
PS2000A_RISING_LOWER	threshold	rising edge, using lower threshold
PS2000A_FALLING	threshold	falling edge, using upper threshold
PS2000A_FALLING_LOWER	threshold	falling edge, using lower threshold
PS2000A_RISING_OR_FALLING	threshold	either edge
PS2000A_INSIDE	window-qualified	inside window
PS2000A_OUTSIDE	window-qualified	outside window
PS2000A_ENTER	window	entering the window
PS2000A_EXIT	window	leaving the window
PS2000A_ENTER_OR_EXIT	window	either entering or leaving the window
PS2000A_NONE	none	none

2.11.52 ps2000aSetTriggerChannelProperties

```
PICO_STATUS ps2000aSetTriggerChannelProperties
(
  short                                  handle,
  PS2000A_TRIGGER_CHANNEL_PROPERTIES *   channelProperties
  short                                  nChannelProperties
  short                                  auxOutputEnable,
  long                                   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 PS2000A_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

2.11.52.1 PS2000A_TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to ps2000aSetTriggerChannelProperties in the `channelProperties` argument to specify the trigger mechanism, and is defined as follows: -

```
typedef struct tTriggerChannelProperties
{
  short                      thresholdUpper;
  unsigned short             thresholdUpperHysteresis;
  short                      thresholdLower;
  unsigned short             thresholdLowerHysteresis;
  PS2000A_CHANNEL            channel;
  PS2000A_THRESHOLD_MODE     thresholdMode;
} PS2000A_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 ps2000aSetChannel, or PS2000A_TRIGGER_AUX for the AUX input.
	
	`thresholdMode,` either a level or window trigger. Use one of these constants: -
	`PS2000A_LEVEL`
	`PS2000A_WINDOW`

2.11.53 ps2000aSetTriggerDigitalPortProperties

```
PICO_STATUS ps2000aSetTriggerDigitalPortProperties
(
  short                                handle,
  PS2000A_DIGITAL_CHANNEL_DIRECTIONS * directions
  short                                nDirections
)
```

This function will set the individual Digital channels trigger directions. Each trigger direction consists of a channel name and a direction. If the channel is not included in the array of PS2000A_DIGITAL_CHANNEL_DIRECTIONS the driver assumes the digital channel's trigger direction is PS2000A_DIGITAL_DONT_CARE.

Applicability	All modes
Arguments	handle, the handle of the required device. * directions, a pointer to an array of PS2000A_DIGITAL_CHANNEL_DIRECTIONS 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 digital channels. If directions is null, digital triggering is switched off. A digital channel that is not included in the array will be set to PS2000A_DIGITAL_DONT_CARE. nDirections, the number of digital channel directions being passed to the driver.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_INVALID_DIGITAL_CHANNEL PICO_INVALID_DIGITAL_TRIGGER_DIRECTION

2.11.53.1 PS2000A_DIGITAL_CHANNEL_DIRECTIONS structure

A structure of this type is passed to ps2000aSetTriggerDigitalPortProperties in the `directions` argument to specify the trigger mechanism, and is defined as follows: -

```
pragma pack(1)
typedef struct tPS2000ADigitalChannelDirections
{
  PS2000A_DIGITAL_CHANNEL channel;
  PS2000A_DIGITAL_DIRECTION direction;
} PS2000A_DIGITAL_CHANNEL_DIRECTIONS;
#pragma pack()


typedef enum enPS2000ADigitalChannel
{
  PS2000A_DIGITAL_CHANNEL_0,
  PS2000A_DIGITAL_CHANNEL_1,
  PS2000A_DIGITAL_CHANNEL_2,
  PS2000A_DIGITAL_CHANNEL_3,
  PS2000A_DIGITAL_CHANNEL_4,
  PS2000A_DIGITAL_CHANNEL_5,
  PS2000A_DIGITAL_CHANNEL_6,
  PS2000A_DIGITAL_CHANNEL_7,
  PS2000A_DIGITAL_CHANNEL_8,
  PS2000A_DIGITAL_CHANNEL_9,
  PS2000A_DIGITAL_CHANNEL_10,
  PS2000A_DIGITAL_CHANNEL_11,
  PS2000A_DIGITAL_CHANNEL_12,
  PS2000A_DIGITAL_CHANNEL_13,
  PS2000A_DIGITAL_CHANNEL_14,
  PS2000A_DIGITAL_CHANNEL_15,
  PS2000A_DIGITAL_CHANNEL_16,
  PS2000A_DIGITAL_CHANNEL_17,
  PS2000A_DIGITAL_CHANNEL_18,
  PS2000A_DIGITAL_CHANNEL_19,
  PS2000A_DIGITAL_CHANNEL_20,
  PS2000A_DIGITAL_CHANNEL_21,
  PS2000A_DIGITAL_CHANNEL_22,
  PS2000A_DIGITAL_CHANNEL_23,
  PS2000A_DIGITAL_CHANNEL_24,
  PS2000A_DIGITAL_CHANNEL_25,
  PS2000A_DIGITAL_CHANNEL_26,
  PS2000A_DIGITAL_CHANNEL_27,
  PS2000A_DIGITAL_CHANNEL_28,
  PS2000A_DIGITAL_CHANNEL_29,
  PS2000A_DIGITAL_CHANNEL_30,
  PS2000A_DIGITAL_CHANNEL_31,
  PS2000A_MAX_DIGITAL_CHANNELS
} PS2000A_DIGITAL_CHANNEL;


typedef enum enPS2000ADigitalDirection
{
  PS2000A_DIGITAL_DONT_CARE,
  PS2000A_DIGITAL_DIRECTION_LOW,
  PS2000A_DIGITAL_DIRECTION_HIGH,
  PS2000A_DIGITAL_DIRECTION_RISING,
  PS2000A_DIGITAL_DIRECTION_FALLING,
  PS2000A_DIGITAL_DIRECTION_RISING_OR_FALLING,
  PS2000A_DIGITAL_MAX_DIRECTION
} PS2000A_DIGITAL_DIRECTION;
```

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

2.11.54 ps2000aSetTriggerDelay

```
PICO_STATUS ps2000aSetTriggerDelay
(
  short         handle,
  unsigned long delay
)
```

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

Applicability	All modes
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 1 GS/s, or 1 ns per sample, the total delay would then be 100 x 1 ns = 100 ns. Range: 0 to `MAX_DELAY_COUNT`

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

2.11.55 ps2000aSigGenSoftwareControl

```
PICO_STATUS ps2000aSigGenSoftwareControl
(
  short   handle,
  short   state
)
```

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

Applicability	Use with ps2000aSetSigGenBuiltIn or ps2000aSetSigGenArbitrary.
Arguments	handle, the handle of the required device state, sets the trigger gate high or low when the trigger type is set to either SIGGEN_GATE_HIGH or SIGGEN_GATE_LOW. Ignored for other trigger types.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_TRIGGER_SOURCE PICO_DRIVER_FUNCTION PICO_NOT_RESPONDING

2.11.56 ps2000aStop

```
PICO_STATUS ps2000aStop
(
  short  handle
)
```

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

Always call this function after the end of a capture to ensure that the scope is ready for the next capture.

Applicability	All modes
Arguments	`handle`, the handle of the required device.

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

2.11.57 ps2000aStreamingReady

```
typedef void (CALLBACK *ps2000aStreamingReady)
(
  short           handle,
  long            noOfSamples,
  unsigned long   startIndex,
  short           overflow,
  unsigned long   triggerAt,
  short           triggered,
  short           autoStop,
  void          * pParameter
)
```

This callback function is part of your application. You register it with the driver using ps2000aGetStreamingLatestValues, and the driver calls it back when streaming-mode data is ready. You can then download the data using the ps2000aGetValuesAsync function.

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 ps2000aSetDataBuffer. `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. 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 ps2000aRunStreaming. `* pParameter`, a void pointer passed from ps2000aGetStreamingLatestValues. The callback function can write to this location to send any data, such as a status flag, back to the application.
Returns	nothing

2.12 Programming examples

Your PicoScope installation includes programming examples in the following languages and development environments:

- C
- Excel
- LabView

2.12.1 C

The **C** example program is a comprehensive console mode program that demonstrates all of the facilities of the driver.

To compile the program, create a new project for an Application containing the following files: -

- `ps2000acon.c`

and:

- `ps2000abc.lib` (Borland 32-bit applications) or
- `ps2000a.lib` (Microsoft Visual C 32-bit applications)

The following files must be in the compilation directory:

- `ps2000aApi.h`
- `picoStatus.h`

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

- `ps2000a.dll`

2.12.2 Excel

1. Load the spreadsheet `ps2000a.xls`
2. Select **Tools | Macro**
3. Select **GetData**
4. Select **Run**

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

2.12.3 LabView

The SDK contains a library of VIs that can be used to control the PicoScope 2000 Series scopes and some simple examples of using these VIs in streaming mode, block mode and rapid block mode.

The LabVIEW library (`PicoScope2000A.llb`) can be placed in the `user.lib` sub-directory to make the VIs available on the 'User Libraries' palette. You must also copy `ps2000a.dll` and `ps2000awrap.dll` to the folder containing your LabView project.

The library contains the following VIs:

- `PicoErrorHandler.vi` - takes an error cluster and, if an error has occurred, displays a message box indicating the source of the error and the status code returned by the driver

- `PicoScope2000AAdvancedTriggerSettings.vi` - an interface for the advanced trigger features of the oscilloscope

  This VI is not required for setting up simple triggers, which are configured using `PicoScope2000ASettings.vi`.

  For further information on these trigger settings, see descriptions of the trigger functions:

  ```
  ps2000aSetTriggerChannelConditions
  ps2000aSetTriggerChannelDirections
  ps2000aSetTriggerChannelProperties
  ps2000aSetPulseWidthQualifier
  ps2000aSetTriggerDelay
  ```

- `PicoScope2000AAWG.vi` - controls the arbitrary waveform generator

  Standard waveforms or an arbitrary waveform can be selected under ‘Wave Type’. There are three settings clusters: general settings that apply to both arbitrary and standard waveforms, settings that apply only to standard waveforms and settings that apply only to arbitrary waveforms. It is not necessary to connect all of these clusters if only using arbitrary waveforms or only using standard waveforms.

  When selecting an arbitrary waveform, it is necessary to specify a text file containing the waveform. This text file should have a single value on each line in the range -1 to 1. For further information on the settings, see descriptions of `ps2000aSetSigGenBuiltIn` and `ps2000aSetSigGenArbitrary`.

- `PicoScope2000AClose.vi` - closes the oscilloscope

  Should be called before exiting an application.

- `PicoScope2000AGetBlock.vi` - collects a block of data from the oscilloscope

  This can be called in a loop in order to continually collect blocks of data. The oscilloscope should first be set up by using `PicoScope2000ASettings.vi`. The VI outputs data arrays in two clusters (max and min). If not using aggregation, ‘Min Buffers’ is not used.

- `PicoScope2000AGetRapidBlock.vi` - collects a set of data blocks or captures from the oscilloscope in rapid block mode

  This VI is similar to `PicoScope2000AGetBlock.vi`. It outputs two-dimensional arrays for each channel that contain data from all the requested number of captures.

- `PicoScope2000AGetStreamingValues.vi` - used in streaming mode to get the latest values from the driver

  This VI should be called in a loop after the oscilloscope has been set up using `PicoScope2000ASettings.vi` and streaming has been started by calling `PicoScope2000AStartStreaming.vi`. The VI outputs the number of samples available and the start index of these samples in the array output by `PicoScope2000AStartStreaming.vi`.

- `PicoScope2000AOpen.vi` - opens a PicoScope 2000A and returns a handle to the device
- `PicoScope2000ASettings.vi` - sets up the oscilloscope

  The inputs are clusters for setting up channels and simple triggers. Advanced triggers can be set up using `PicoScope2000AAdvancedTriggerSettings.vi`.
- `PicoScope2000AStartStreaming.vi` - starts the oscilloscope streaming

  It outputs arrays that will contain samples once `PicoScope2000AGetStreamingValues.vi` has returned.
- `PicoStatus.vi` - checks the status value returned by calls to the driver

  If the driver returns an error, the status member of the error cluster is set to ‘true’ and the error code and source are set.

2.13 Driver status codes

Every function in the ps2000a 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 PicoScope 2000 Series SDK.

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 PS2000A_MAX_UNITS devices.
02	PICO_MEMORY_FAIL. Not enough memory could be allocated on the host machine.
03	PICO_NOT_FOUND. No PicoScope 2000 Series device 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 PicoScope is not responding to commands from the PC.
08	PICO_CONFIG_FAIL. The configuration information in the oscilloscope is corrupt or 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 is 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 specified handle.
0D	PICO_INVALID_PARAMETER. A parameter 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.
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 ps2000a...Ready 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 ps2000aStop, or use ps2000aGetStreamingLatestValues.
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.
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.
45	PICO_INVALID_COUPLING. An invalid coupling type was specified in ps2000aSetChannel.
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 ps2000aSetTriggerChannelProperties.
4A	PICO_INTERFACE_NOT_CONNECTED. The driver was unable to contact the oscilloscope.
4D	PICO_SIGGEN_WAVEFORM_SETUP_FAILED. A problem occurred in ps2000aSetSigGenBuiltIn or ps2000aSetSigGenArbitrary.
4E	PICO_FPGA_FAIL
4F	PICO_POWER_MANAGER
50	PICO_INVALID_ANALOGUE_OFFSET. An impossible analogue offset value was specified in ps2000aSetChannel.
51	PICO_PLL_LOCK_FAILED. Unable to configure the oscilloscope.
52	PICO_ANALOG_BOARD. The oscilloscope's analog board is not detected.
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 cannot configure the AUX input as both a trigger and a reference clock.
59	PICO_PWQ_AND_EXTERNAL_CLOCK_CLASH. You cannot 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 cannot be opened.
5B	PICO_MEMORY_CLOCK_FREQUENCY. The frequency of the memory is reporting incorrectly.
5C	PICO_I2C_NOT_RESPONDING. The I^{2}C bus 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 incorrect IP address has been passed to the driver.
106	PICO_IPSOCKET_FAILED
107	PICO_IPSOCKET_TIMEDOUT. The IP socket has timed out.
108	PICO_SETTINGS_FAILED. The requested settings could not 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 specified 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.
113	PICO_INVALID_DIGITAL_PORT. A port number that does not evaluate to either PS2000A_DIGITAL_PORT0 or PS2000A_DIGITAL_PORT1, the ports that are supported.
114	PICO_INVALID_DIGITAL_CHANNEL. The digital channel is not in the range PS2000A_DIGITAL_CHANNEL0 to PS2000_DIGITAL_CHANNEL15, the digital channels that are supported.
115	PICO_INVALID_DIGITAL_TRIGGER_DIRECTION. The digital trigger direction is not a valid trigger direction and should be equal in value to one of the PS2000A_DIGITAL_DIRECTION enumerations.
116	PICO_SIG_GEN_PRBS_NOT_SUPPORTED. The pseudo random bit stream option on the AWG is not supported.
117	PICO_ETS_NOT_AVAILABLE_WITH_LOGIC_CHANNELS. When a digital port is enabled, ETS sample mode is not available for use.

2.14 Enumerated types and constants

Here are the enumerated types used in the PicoScope 2000 Series (A API) SDK, as defined in the file `ps2000aApi.h` . We recommend that you refer to these constants by name unless your programming language allows only numerical values.

```
#define PS2208_MAX_ETS_CYCLES  500
#define PS2208_MAX_INTERLEAVE  20

#define PS2207_MAX_ETS_CYCLES      500
#define PS2207_MAX_INTERLEAVE      20

#define PS2206_MAX_ETS_CYCLES      250
#define PS2206_MAX_INTERLEAVE      10

#define PS2000A_EXT_MAX_VALUE      32767
#define PS2000A_EXT_MIN_VALUE      -32767

#define PS2000A_MAX_LOGIC_LEVEL    32767
#define PS2000A_MIN_LOGIC_LEVEL    -32767

#define MIN_SIG_GEN_FREQ           0.0f
#define MAX_SIG_GEN_FREQ           20000000.0f

#define MAX_SIG_GEN_BUFFER_SIZE 8192
#define MIN_SIG_GEN_BUFFER_SIZE 1
#define MIN_DWELL_COUNT         10
#define MAX_SWEEPS_SHOTS        ((1 << 30) - 1)

#define PS2000A_MAX_ANALOGUE_OFFSET_50MV_200MV    0.250f
#define PS2000A_MIN_ANALOGUE_OFFSET_50MV_200MV    -0.250f
#define PS2000A_MAX_ANALOGUE_OFFSET_500MV_2V      2.500f
#define PS2000A_MIN_ANALOGUE_OFFSET_500MV_2V      -2.500f
#define PS2000A_MAX_ANALOGUE_OFFSET_5V_20V        20.f
#define PS2000A_MIN_ANALOGUE_OFFSET_5V_20V        -20.f
#define PS2000A_SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN 0xFFFFFFFF

typedef enum enPS2000AChannel
{
   PS2000A_CHANNEL_A,
   PS2000A_CHANNEL_B,
   PS2000A_CHANNEL_C,
   PS2000A_CHANNEL_D,
   PS2000A_EXTERNAL,
   PS2000A_MAX_CHANNELS = PS2000A_EXTERNAL,
   PS2000A_TRIGGER_AUX,
   PS2000A_MAX_TRIGGER_SOURCES
}   PS2000A_CHANNEL;

typedef enum enPS2000AChannelBufferIndex
{
   PS2000A_CHANNEL_A_MAX,
   PS2000A_CHANNEL_A_MIN,
   PS2000A_CHANNEL_B_MAX,
   PS2000A_CHANNEL_B_MIN,
   PS2000A_CHANNEL_C_MAX,
   PS2000A_CHANNEL_C_MIN,
   PS2000A_CHANNEL_D_MAX,
   PS2000A_CHANNEL_D_MIN,
   PS2000A_MAX_CHANNEL_BUFFERS
} PS2000A_CHANNEL_BUFFER_INDEX;

typedef enum enPS2000ATriggerOperand
{
   PS2000A_OPERAND_NONE,
   PS2000A_OPERAND_OR,
   PS2000A_OPERAND_AND,
   PS2000A_OPERAND_THEN
} PS2000A_TRIGGER_OPERAND;

typedef enum enPS2000ARange
```

```
{
   PS2000A_10MV,
   PS2000A_20MV,
   PS2000A_50MV,
   PS2000A_100MV,
   PS2000A_200MV,
   PS2000A_500MV,
   PS2000A_1V,
   PS2000A_2V,
   PS2000A_5V,
   PS2000A_10V,
   PS2000A_20V,
   PS2000A_50V,
   PS2000A_MAX_RANGES
}   PS2000A_RANGE;


typedef enum enPS2000ACoupling
{
   PS2000A_AC,
   PS2000A_DC,
} PS2000A_COUPLING;


typedef enum enPS2000AChannelInfo
{
   PS2000A_CI_RANGES,
} PS2000A_CHANNEL_INFO;


typedef enum enPS2000AEtsMode
  {
  PS2000A_ETS_OFF,
  PS2000A_ETS_FAST,
  PS2000A_ETS_SLOW,
  PS2000A_ETS_MODES_MAX
  }   PS2000A_ETS_MODE;


typedef enum enPS2000ATimeUnits
  {
  PS2000A_FS,
  PS2000A_PS,
  PS2000A_NS,
  PS2000A_US,
  PS2000A_MS,
  PS2000A_S,
  PS2000A_MAX_TIME_UNITS,
  }   PS2000A_TIME_UNITS;


typedef enum enPS2000ASweepType
{
   PS2000A_UP,
   PS2000A_DOWN,
   PS2000A_UPDOWN,
   PS2000A_DOWNUP,
   PS2000A_MAX_SWEEP_TYPES
} PS2000A_SWEEP_TYPE;


typedef enum enPS2000AWaveType
{
   PS2000A_SINE,
   PS2000A_SQUARE,
   PS2000A_TRIANGLE,
   PS2000A_RAMP_UP,
   PS2000A_RAMP_DOWN,
   PS2000A_SINC,
   PS2000A_GAUSSIAN,
   PS2000A_HALF_SINE,
   PS2000A_DC_VOLTAGE,
   PS2000A_MAX_WAVE_TYPES
} PS2000A_WAVE_TYPE;


typedef enum enPS2000AExtraOperations
{
   PS2000A_ES_OFF,

   PS2000A_WHITENOISE,

   PS2000A_PRBS
} PS2000A_EXTRA_OPERATIONS;
```

```
#define PS2000A_SINE_MAX_FREQUENCY    1000000.f
#define PS2000A_SQUARE_MAX_FREQUENCY      1000000.f
#define PS2000A_TRIANGLE_MAX_FREQUENCY    1000000.f
#define PS2000A_SINC_MAX_FREQUENCY        1000000.f
#define PS2000A_RAMP_MAX_FREQUENCY        1000000.f
#define PS2000A_HALF_SINE_MAX_FREQUENCY   1000000.f
#define PS2000A_GAUSSIAN_MAX_FREQUENCY    1000000.f
#define PS2000A_PRBS_MAX_FREQUENCY        1000000.f
#define PS2000A_PRBS_MIN_FREQUENCY        0.03f
#define PS2000A_MIN_FREQUENCY             0.03f


typedef enum enPS2000ASigGenTrigType
{
   PS2000A_SIGGEN_RISING,
   PS2000A_SIGGEN_FALLING,
   PS2000A_SIGGEN_GATE_HIGH,
   PS2000A_SIGGEN_GATE_LOW
} PS2000A_SIGGEN_TRIG_TYPE;


typedef enum enPS2000ASigGenTrigSource
{
   PS2000A_SIGGEN_NONE,
   PS2000A_SIGGEN_SCOPE_TRIG,
   PS2000A_SIGGEN_AUX_IN,
   PS2000A_SIGGEN_EXT_IN,
   PS2000A_SIGGEN_SOFT_TRIG
} PS2000A_SIGGEN_TRIG_SOURCE;


typedef enum enPS2000AIndexMode
{
   PS2000A_SINGLE,
   PS2000A_DUAL,
   PS2000A_QUAD,
   PS2000A_MAX_INDEX_MODES
} PS2000A_INDEX_MODE;


typedef enum enPS2000AThresholdMode
{
   PS2000A_LEVEL,
   PS2000A_WINDOW
} PS2000A_THRESHOLD_MODE;


typedef enum enPS2000AThresholdDirection
{
   PS2000A_ABOVE,
   PS2000A_BELOW,
   PS2000A_RISING,
   PS2000A_FALLING,
   PS2000A_RISING_OR_FALLING,
   PS2000A_ABOVE_LOWER,
   PS2000A_BELOW_LOWER,
   PS2000A_RISING_LOWER,
   PS2000A_FALLING_LOWER,

   // Windowing using both thresholds
   PS2000A_INSIDE        = PS2000A_ABOVE,
   PS2000A_OUTSIDE       = PS2000A_BELOW,
   PS2000A_ENTER         = PS2000A_RISING,
   PS2000A_EXIT          = PS2000A_FALLING,
   PS2000A_ENTER_OR_EXIT = PS2000A_RISING_OR_FALLING,
   PS2000A_POSITIVE_RUNT = 9,
  PS2000A_NEGATIVE_RUNT,

   // no trigger set
   PS2000A_NONE = PS2000A_RISING
} PS2000A_THRESHOLD_DIRECTION;


typedef enum enPS2000ATriggerState
{
  PS2000A_CONDITION_DONT_CARE,
  PS2000A_CONDITION_TRUE,
  PS2000A_CONDITION_FALSE,
   PS2000A_CONDITION_MAX
} PS2000A_TRIGGER_STATE;


typedef enum enPS2000ARatioMode
{
   PS2000A_RATIO_MODE_NONE,
   PS2000A_RATIO_MODE_AGGREGATE = 1,
   PS2000A_RATIO_MODE_DECIMATE = 2,
   PS2000A_RATIO_MODE_AVERAGE = 4,
} PS2000A_RATIO_MODE;

typedef enum enPS2000APulseWidthType
{
   PS2000A_PW_TYPE_NONE,
   PS2000A_PW_TYPE_LESS_THAN,
   PS2000A_PW_TYPE_GREATER_THAN,
   PS2000A_PW_TYPE_IN_RANGE,
   PS2000A_PW_TYPE_OUT_OF_RANGE
} PS2000A_PULSE_WIDTH_TYPE;

typedef enum enPS2000AHoldOffType
{
  PS2000A_TIME,
  PS2000A_MAX_HOLDOFF_TYPE
} PS2000A_HOLDOFF_TYPE;
```

2.15 Numeric data types

Here is a list of the sizes and ranges of the numeric data types used in the PicoScope 2000 Series A API.

Type	Bits	Signed or unsigned?
short	16	signed
enum	32	enumerated
int	32	signed
long	32	signed
unsigned long	32	unsigned
float	32	signed (IEEE 754)
__int64	64	signed

3 Glossary

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

Aggregation. The PicoScope 2000A 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 ps2000aRunStreaming for real-time capture, and when you call ps2000aGetStreamingLatestValues 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.

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** on the oscilloscope. It can be used to start a data collection run but cannot be used to record data.

IDC. Insulation-displacement connector. An electrical connector designed to be connected to the conductors of an insulated cable by a connection process which forces sharpened blades through the insulation.

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.

MSO (Mixed signal oscilloscope). An oscilloscope that has both analog and digital inputs.

Oversampling. Oversampling is taking more than one measurement during a time interval and returning an average. If the signal contains a small amount of noise, this technique can increase the effective vertical resolution of the oscilloscope.

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

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

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

Signal generator. This is a feature of some oscilloscopes which allows a signal to be generated without an external input device being present. The signal generator output is the BNC socket marked **AWG** 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.

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

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

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

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

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

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

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

pico®
Technology

Index

A

AC coupling 61
AC/DC control 102
Access 2
ADC count 49, 51
Address 3
Aggregation 15
Aliasing 102
Analog offset 61
Analogue bandwidth 102
Analogue offset 25
API function calls 19
Arbitrary waveform generator 72, 74

B

Bandwidth limiter 61
Block mode 5, 6, 7, 102
  asynchronous call 7
  callback 20
  polling status 47
  running 57
Buffer size 102

C

C programming 91
Callback 5, 13
  block mode 20
  for data 22
  streaming mode 90
Channels
  enabling 61
  settings 61
Closing units 21
Common-mode voltage 102
Communication 56
Connection 56
Constants 98
Contact details 3
Copyright 2
Coupling type, setting 61

D

Data acquisition 15
Data buffers
  declaring 62
  declaring, aggregation mode 63
Data retention 6
DC coupling 61
Digital connector 17
Digital data 4
Digital port 4
Downsampling 6, 38
  maximum ratio 27
  modes 39
Driver 4
  status codes 94

E

Email address 3
Enabling channels 61
Enumerated types 98
Enumerating oscilloscopes 23
ETS
  mode 5
  overview 13
  setting time buffers 66, 67
  setting up 65
  using 14
Excel macros 91

F

Fax number 3
Fitness for purpose 2
Function calls 19
Functions
  ps2000aBlockReady 20
  ps2000aCloseUnit 21
  ps2000aDataReady 22
  ps2000aEnumerateUnits 23
  ps2000aFlashLed 24
  ps2000aGetAnalogueOffset 25
  ps2000aGetChannelInformation 26
  ps2000aGetMaxDownSampleRatio 27
  ps2000aGetMaxSegments 28
  ps2000aGetNoOfCaptures 29, 30
  ps2000aGetStreamingLatestValues 31
  ps2000aGetTimebase 17, 32
  ps2000aGetTimebase2 33
  ps2000aGetTriggerTimeOffset 34
  ps2000aGetTriggerTimeOffset64 35
  ps2000aGetUnitInfo 36
  ps2000aGetValues 7, 38
  ps2000aGetValuesAsync 7, 40
  ps2000aGetValuesBulk 41
  ps2000aGetValuesOverlapped 42

Functions
- ps2000aGetValuesOverlappedBulk 43
- ps2000aGetValuesTriggerTimeOffsetBulk 44
- ps2000aGetValuesTriggerTimeOffsetBulk64 45, 46
- ps2000aIsReady 47
- ps2000aIsTriggerOrPulseWidthQualifierEnabled 48
- ps2000aMaximumValue 4, 49
- ps2000aMemorySegments 50
- ps2000aMinimumValue 4, 51
- ps2000aNoOfStreamingValues 52
- ps2000aOpenUnit 53
- ps2000aOpenUnitAsync 54
- ps2000aOpenUnitProgress 55
- ps2000aPingUnit 56
- ps2000aRunBlock 57
- ps2000aRunStreaming 59
- ps2000aSetChannel 4, 61
- ps2000aSetDataBuffer 62
- ps2000aSetDataBuffers 63
- ps2000aSetEts 13, 65
- ps2000aSetEtsTimeBuffer 66
- ps2000aSetEtsTimeBuffers 67
- ps2000aSetNoOfCaptures 68
- ps2000aSetPulseWidthQualifier 69
- ps2000aSetSigGenArbitrary 72
- ps2000aSetSigGenBuiltIn 75
- ps2000aSetSimpleTrigger 5, 78
- ps2000aSetTriggerChannelConditions 5, 79
- ps2000aSetTriggerChannelDirections 5, 81
- ps2000aSetTriggerChannelProperties 5, 82
- ps2000aSetTriggerDelay 87
- ps2000aSetTriggerDigitalPortProperties 84
- ps2000aSigGenSoftwareControl 88
- ps2000aStop 7, 89
- ps2000aStreamingReady 90

H

Hysteresis 83, 85

I

Index modes
- dual 74
- single 74

Information, reading from units 36
Input range, selecting 61
Intended use 1

L

LabVIEW 91
LED
- flashing 24

Legal information 2
Liability 2

M

Macros in Excel 91
Memory in scope 6
Memory segmentation 6, 7, 15, 50
Mission-critical applications 2
Multi-unit operation 18

N

Numeric data types 101

O

One-shot signals 13
Opening a unit 53
- checking progress 55
- without blocking 54

Oversampling 39

P

PC Oscilloscope 1, 102
PC requirements 1
PICO_STATUS enum type 94
PicoScope 2000 Series 1
PicoScope software 1, 4, 94, 102
PORT0, PORT1 4
Programming
- C 91
- Excel 91
- general procedure 4
- LabVIEW 91

ps2000a.dll 4
PS2000A_CONDITION_ constants 71, 80
PS2000A_LEVEL constant 83, 85
PS2000A_PWQ_CONDITIONS structure 71
PS2000A_RATIO_MODE_AGGREGATE 39
PS2000A_RATIO_MODE_AVERAGE 39
PS2000A_RATIO_MODE_DECIMATE 39
PS2000A_TIME_UNITS constant 34, 35
PS2000A_TRIGGER_CHANNEL_PROPERTIES structure 83, 85
PS2000A_TRIGGER_CONDITIONS 79
PS2000A_TRIGGER_CONDITIONS structure 80

PS2000A_WINDOW constant 83, 85
Pulse-width qualifier 69
  conditions 71
  requesting status 48

R

Ranges 26
Rapid block mode 5, 8, 29, 30
  aggregation 11
  no aggregation 9
  setting number of captures 68
Resolution, vertical 102
Retrieving data 38, 40
  block mode, deferred 42
  rapid block mode 41
  rapid block mode, deferred 43
  stored 16
  streaming mode 31
Retrieving times
  rapid block mode 44, 45, 46

S

Sampling rate 102
  maximum 6
Scaling 4
Serial numbers 23
Setup time 6
Signal generator
  arbitrary waveforms 72
  built-in waveforms 75
  software trigger 88
Spectrum analyzer 1, 102
Status codes 94
Stopping sampling 89
Streaming mode 5, 15, 102
  callback 90
  getting number of samples 52
  retrieving data 31
  running 59
  using 16
Support 2

T

Technical assistance 3
Telephone number 3
Threshold voltage 5
Time buffers
  setting for ETS 66, 67
Timebase 17, 102
  calculating 32, 33
Trademarks 2
Trigger 5
  channel properties 82, 84
  conditions 79, 80
  delay 87
  digital ports 84
  directions 81
  pulse-width qualifier 69
  pulse-width qualifier conditions 71
  requesting status 48
  setting up 78
  time offset 34, 35
Trigger stability 13

U

Upgrades 2
Usage 2
USB 1, 4, 102
  hub 18

V

Viruses 2
Voltage range 4, 102
  selecting 61

W

Website address 3
WinUsb.sys 4

pico®
Technology

Pico Technology

James House
Colmworth Business Park
ST. NEOTS
Cambridgeshire
PE19 8YP
United Kingdom
Tel: +44 (0) 1480 396 395
Fax: +44 (0) 1480 396 296
www.picotech.com

ps2000apg.en-3

13.8.12