PicoScope 4000 Series (A API)

PC Oscilloscopes

Programmer's Guide

ps4000apg.en r1


pico
Technology

Contents

1 Welcome 1

2 Introduction 2

1 Software licence conditions 2
2 Trademarks 2
3 Company details 3
4 System requirements 3
5 Installation instructions 4

3 Programming with the PicoScope 4000 Series (A API) 5

1 Driver 5
2 Voltage ranges 5
3 Channel selection 6
4 Triggering 6
5 Downsampling 7
6 Sampling modes 8
1 Block mode 8
2 Rapid block mode 10
3 Streaming mode 15
4 Retrieving stored data 16
7 Timebases 17
8 Combining several oscilloscopes 17

4 API functions 18

1 ps4000aBlockReady 20
2 ps4000aChangePowerSource 21
3 ps4000aCurrentPowerSource 22
4 ps4000aCloseUnit 23
5 ps4000aDataReady 24
6 ps4000aEnumerateUnits 25
7 ps4000aFlashLed 26
8 ps4000aGetAnalogueOffset 27
9 ps4000aGetChannelInformation 28
10 ps4000aGetMaxDownSampleRatio 29
11 ps4000aGetMaxSegments 30
12 ps4000aGetNoOfCaptures 31
13 ps4000aGetNoOfProcessedCaptures 32
14 ps4000aGetStreamingLatestValues 33
15 ps4000aGetTimebase 34
16 ps4000aGetTimebase2 35
17 ps4000aGetTriggerTimeOffset 36
18 ps4000aGetTriggerTimeOffset64 37
19 ps4000aGetUnitInfo 38

20 ps4000aGetValues 39
21 ps4000aGetValuesAsync 40
22 ps4000aGetValuesBulk 41
23 ps4000aGetValuesOverlapped 42
24 ps4000aGetValuesOverlappedBulk 43
25 ps4000aIsLedFlashing 44
26 ps4000aIsReady 45
27 ps4000aIsTriggerOrPulseWidthQualifierEnabled 46
28 ps4000aMaximumValue 47
29 ps4000aMinimumValue 48
30 ps4000aMemorySegments 49
31 ps4000aNoOfStreamingValues 50
32 ps4000aOpenUnit 51
33 ps4000aOpenUnitAsync 52
34 ps4000aOpenUnitProgress 53
35 ps4000aRunBlock 54
36 ps4000aRunStreaming 56
37 ps4000aSetBandwidthFilter 58
38 ps4000aSetChannel 59
39 ps4000aSetDataBuffer 60
40 ps4000aSetDataBuffers 61
41 ps4000aSetEts 62
42 ps4000aSetEtsTimeBuffer 63
43 ps4000aSetEtsTimeBuffers 64
44 ps4000aSetNoOfCaptures 65
45 ps4000aSetPulseWidthQualifierConditions 66
46 ps4000aSetPulseWidthQualifierProperties 67
47 ps4000aSetSigGenArbitrary 68
1 AWG index modes 71
48 ps4000aSetSigGenBuiltIn 72
49 ps4000aSetSigGenPropertiesArbitrary 74
50 ps4000aSetSigGenPropertiesBuiltIn 75
51 ps4000aSetSimpleTrigger 76
52 ps4000aSetTriggerChannelConditions 77
1 PS4000A_CONDITION structure 78
53 ps4000aSetTriggerChannelDirections 79
1 PS4000A_DIRECTION structure 80
54 ps4000aSetTriggerChannelProperties 81
1 PS4000A_TRIGGER_CHANNEL_PROPERTIES structure 82
55 ps4000aSetTriggerDelay 83
56 ps4000aSigGenSoftwareControl 84
57 ps4000aStop 85
58 ps4000aStreamingReady 86
5 Enumerated types and constants 87

6 Driver status codes 88

7 Programming examples 89

1 C 89

2 Excel 89

3 LabVIEW 89

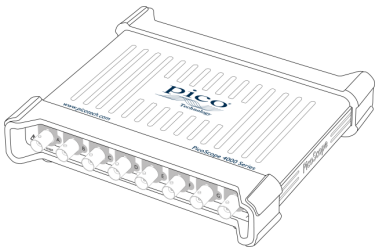
8 Glossary 92

Index 95


pico
Technology

1 Welcome

The **PicoScope 4000 Series** of PC Oscilloscopes from Pico Technology is a range of compact, high-resolution scope units designed to replace traditional bench-top oscilloscopes.



This Programmer's Guide explains how to use the Application Programming Interface (API) for the PicoScope 4000 Series (A API) scopes. The A API supports the following model:

- PicoScope 4824 8 channel oscilloscope (product web page)

Other oscilloscopes in the PicoScope 4000 Series use a different API. This is documented in the original *PicoScope 4000 Series Programmer's Guide*.

2 Introduction

2.1 Software licence conditions

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.

2.2 Trademarks

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

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

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

2.3 Company details

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

2.4 System requirements

Using with PicoScope for Windows

To ensure that your PicoScope 4000 Series PC Oscilloscope operates correctly with the PicoScope software, 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. Please note the PicoScope software is not installed as part of the SDK.

Item	Specification
Operating system	Windows XP SP3, Vista, 7 or 8 32 bit and 64 bit versions supported
Processor	As required by Windows
Memory	
Free disk space	
Ports	USB 3.0 port

Using with custom applications

Drivers are available for the operating systems mentioned above.

USB

The PicoScope 4000 driver offers three different methods of recording data, all of which support USB 1.1, USB 2.0 and USB 3.0, although the fastest transfer rates between the PC and the PicoScope 4000 are achieved using USB 3.0.

2.5 Installation instructions

IMPORTANT
Do not connect your PicoScope 4000 Series scope device to the PC before you have installed the Pico Technology software.
If you do, Windows might not recognise the scope device correctly.

Procedure

- Follow the instructions in the Installation Guide included with your product package.
- Connect your PC Oscilloscope to the PC using the USB cable supplied.

Checking the installation

Once you have installed the software and connected the PC Oscilloscope to the PC, start the PicoScope software. PicoScope should now display any signal connected to the scope inputs. If a probe is connected to your oscilloscope, you should see a small 50 or 60 hertz signal in the oscilloscope window when you touch the probe tip with your finger.

Moving your PicoScope PC Oscilloscope to another USB port

- **Windows XP SP3**

When you first installed the PicoScope 4000 Series PC Oscilloscope by plugging it into a USB port, Windows associated the Pico driver with that port. If you later move the oscilloscope to a different USB port, Windows will display the "New Hardware Found Wizard" again. When this occurs, just click "Next" in the wizard to repeat the installation. If Windows gives a warning about Windows Logo Testing, click "Continue Anyway". As all the software you need is already installed on your computer, there is no need to insert the Pico Software CD again.

- **Windows Vista, 7 and 8**

The process is automatic. When you move the device from one port to another, Windows displays an "Installing device driver software" message and then a "PicoScope 4000 Series PC Oscilloscope" message. The PC Oscilloscope is then ready for use.

3 Programming with the PicoScope 4000 Series (A API)

The ps4000a.dll dynamic link library in your PicoScope installation directory allows you to program a PicoScope 4000 Series oscilloscope using standard C function calls.

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

- Open the scope unit.
- Set up the input channels with the required voltage ranges and coupling mode.
- 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 installed with your PicoScope software. These show how to use the functions of the driver software in each of the modes available.

3.1 Driver

Your application will communicate with a PicoScope 4000 API driver called ps4000a.dll. The driver exports the PicoScope 4000 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 kernel driver, CyUSB3.sys, which has a 64-bit version and a 32-bit version and works with all operating systems except Windows 8. Windows 8 uses WinUsb.sys on both 32-bit and 64-bit versions. Your application does not need to call the kernel driver. Once you have installed the PicoScope 6 software, Windows automatically installs the kernel driver when you plug in the PicoScope 4000 Series PC Oscilloscope for the first time.

3.2 Voltage ranges

The ps4000aSetChannel function allows you to set the voltage range of each input channel of the scope. The allowable voltage ranges are described in the device data sheet. Each sample is normalized to 16 bits between maximum and minimum values that are defined as constants and can also be queried by calling two functions:

Constant	Function	Voltage
PS4000A_MAX_VALUE	ps4000aMaximumValue	maximum
PS4000A_MIN_VALUE	ps4000aMinimumValue	minimum
PS4000A_LOST_DATA[1]		

Note 1. In streaming mode, this special value indicates a buffer overrun.

3.3 Channel selection

You can switch each channel on and off, and set its coupling mode to either AC or DC, using the ps4000aSetChannel function.

- **DC coupling:** The scope accepts all input frequencies from zero (DC) up to its maximum analog bandwidth.
- **AC coupling:** The scope accepts input frequencies from a few hertz up to its maximum analog bandwidth. The lower –3 dB cutoff frequency is about 1 hertz.

3.4 Triggering

PicoScope 4000 Series PC Oscilloscopes can either start collecting data immediately, or be programmed to wait for a **trigger** event to occur. In both cases you need to use the PicoScope 4000 trigger functions:

- ps4000aSetTriggerChannelConditions
- ps4000aSetTriggerChannelDirections
- ps4000aSetTriggerChannelProperties
- ps4000aSetTriggerDelay (optional)

These can be run collectively by calling ps4000aSetSimpleTrigger, or singly.

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.

The driver supports these triggering methods:

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

3.5 Downsampling

The driver can optionally apply a data reduction, or **downsampling**, process before returning data to the application. Downsampling is done by firmware on the device and is generally faster than using the PC's own processor(s). You instruct the driver to downsample by passing a `downSampleRatioMode` argument to one of the data-retrieval functions such as `ps4000aGetValues`. You must also pass in an argument called `downSampleRatio`: how many raw samples are to be combined into each processed sample.

You can optionally retrieve data using more than one downsampling mode with a single call to `ps4000aGetValues`. Set up a buffer for each downsampling mode by calling `ps4000aSetDataBuffer`. Then, when calling `ps4000aGetValues`, set `downSampleRatioMode` to the bitwise OR of the required downsampling modes.

The available downsampling modes are:

`PS4000A_RATIO_MODE_NONE (0)`

No downsampling is performed. The `downSampleRatio` parameter is ignored.

`PS4000A_RATIO_MODE_AGGREGATE (1)`

The *aggregate* method generates two buffers of data for every channel, one containing the minimum sample value for every block of `downSampleRatio` raw samples, and the other containing the maximum value.

`PS4000A_RATIO_MODE_DECIMATE (2)`

The *decimate* method returns the first sample in every block of `downSampleRatio` successive samples and discards all the other samples.

`PS4000A_RATIO_MODE_AVERAGE (4)`

The *average* method returns the sum of all the samples in each block of `downSampleRatio` samples, divided by the size of the block.

`PS4000A_RATIO_MODE_DISTRIBUTION (8)`

This mode is not implemented.

3.6 Sampling modes

PicoScope 4000 Series PC 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.

- **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 provides fast streaming at up to 160 MS/s with a USB 3.0 connection. Downsampling and triggering are supported in this mode.

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

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

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

3.6.1 Block mode

In **block mode**, the computer prompts a PicoScope 4000 Series PC 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 `ps4000aMemorySegments`).

- **Sampling rate.** The maximum sampling rate of 80 MS/s can be achieved with up to four channels enabled. With five or more channels enabled, the sampling rate is reduced to 40 MS/s.

- **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 `ps4000aRunBlock`, `ps4000aStop` and `ps4000aGetValues`.

- **Downsampling.** When the data has been collected, you can set an optional downsampling factor and examine the data. Downsampling is the process of reducing the amount of data by combining adjacent samples using one of several algorithms. 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 `ps4000aMemorySegments`.

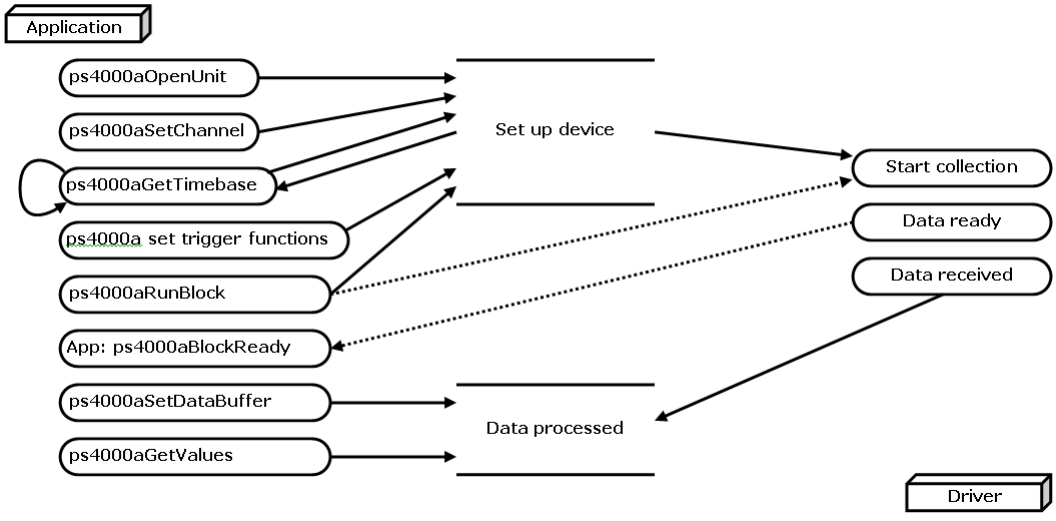
- **Data retention.** The data is lost when a new run is started in the same segment, the number of segments is changed, or the scope is powered down.

See Using block mode for programming details.

3.6.1.1 Using block mode

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

1. Open the oscilloscope using `ps4000aOpenUnit`.
2. Select channel ranges and AC/DC coupling using `ps4000aSetChannel`.
3. Using `ps4000aGetTimebase`, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions [1] [2] [3] [4] to set up the trigger if required.
5. Start the oscilloscope running using `ps4000aRunBlock`.
6. Wait until the oscilloscope is ready using the `ps4000aBlockReady` callback.
7. Use `ps4000aSetDataBuffer` to tell the driver where your memory buffer is.
8. Transfer the block of data from the oscilloscope using `ps4000aGetValues`.
9. Display the data.
10. Repeat steps 5 to 9.
11. Stop the oscilloscope using `ps4000aStop`.



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

3.6.2 Rapid block mode

In normal block mode, the PicoScope 4000 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 about 2.5 microseconds.

See Using rapid block mode for details.

3.6.2.1 Using rapid block mode

You can use **rapid block mode** with or without downsampling. The following procedure shows you how to use it without downsampling.

Without downsampling

1. Open the oscilloscope using `ps4000aOpenUnit`.
2. Select channel ranges and AC/DC coupling using `ps4000aSetChannel`.
3. Using `ps4000aGetTimebase`, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions [1] [2] [3] [4] to set up the trigger if required.
5. Set the number of memory segments equal to or greater than the number of captures required using `ps4000aMemorySegments`. Use `ps4000aSetNoOfCaptures` before each run to specify the number of waveforms to capture.
6. Start the oscilloscope running using `ps4000aRunBlock`.
7. Wait until the oscilloscope is ready using the `ps4000aBlockReady` callback.
8. Use `ps4000aSetDataBuffer` to tell the driver where your memory buffers are.
9. Transfer the blocks of data from the oscilloscope using `ps4000aGetValuesBulk`.
10. Retrieve the time offset for each data segment using `ps4000aGetValuesTriggerTimeOffsetBulk`.
11. Display the data.
12. Repeat steps 6 to 11 if necessary.
13. Stop the oscilloscope using `ps4000aStop`.

With downsampling

To use rapid block mode with downsampling, follow steps 1 to 9 above and then proceed as follows:

10a. Call `ps4000aSetDataBuffers` to set up one pair of buffers for every waveform segment required.
11a. Call `ps4000aGetValues` for each pair of buffers.
12a. Retrieve the time offset for each data segment using `ps4000aGetTriggerTimeOffset64`.

Continue from step 13 above.

3.6.2.2 Rapid block mode example 1: no downsampling

```
#define MAX_SAMPLES 1000
```

Set up the device up as usual.

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

```
// set the number of waveforms to 100
ps4000aSetNoOfCaptures (handle, 100);

pParameter = false;
ps4000aRunBlock
(
  handle,
  0,                  // noOfPreTriggerSamples,
  10000,              // noOfPostTriggerSamples,
  1,                  // timebase to be used,
  &timeIndisposedMs,
  1,                  // segmentIndex
  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 = PS4000A_CHANNEL_A; c <= PS4000A_CHANNEL_H; c++)
  {
    ps4000aSetDataBuffer
    (
      handle,
      c,
      &buffer[c][i],
      MAX_SAMPLES,
      i,
      PS4000A_RATIO_MODE_NONE
    );
  }
}
```

Comments: buffer has been created as a two-dimensional array of pointers to `int16_t` types, which will contain 1000 samples as defined by `MAX_SAMPLES`. There are only 10 buffers set, but it is possible to set up to the number of captures you have requested.


```
ps4000aGetValuesBulk
(
  handle,
  &noOfSamples,             // set to MAX_SAMPLES on entering
  the function
  10,                       // fromSegmentIndex
  19,                       // toSegmentIndex
  1,                        // downSampleRatio
  PS4000A_RATIO_MODE_NONE,..// downSampleRatioMode
  overflow                  // an array of size 10 int16_t
  integers
)
```

Comments: the number of samples could be up to `noOfPreTriggerSamples + noOfPostTriggerSamples`, the values set in `ps4000aRunBlock`. The samples are always returned from the first sample taken, unlike the `ps4000aGetValues` function which allows the sample index to be set. This function does not support downsampling. The above segments start at 10 and finish at 19 inclusive. It is possible for the `fromSegmentIndex` to wrap around to the `toSegementIndex`, by setting the `fromSegmentIndex` to 98 and the `toSegmentIndex` to 7.

```
ps4000aGetValuesTriggerChannelTimeOffsetBulk64
(
  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.

3.6.2.3 Rapid block mode example 2: using downsampling

```
#define MAX_SAMPLES 1000
```

Set up the device up as usual.

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

```
// set the number of waveforms to 100
ps4000aSetNoOfCaptures(handle, 100);

pParameter = false;
ps4000aRunBlock
(
  handle,
  0,                  // noOfPreTriggerSamples
  1000000,            // noOfPostTriggerSamples
  1,                  // timebase to be used
  &timeIndisposedMs,
  1,                  // segmentIndex
  lpReady,
  &pParameter
);
```

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

```
for (int c = PS4000A_CHANNEL_A; c <= PS4000_CHANNEL_H; c++)
{
  ps4000aSetDataBuffers
  (
    handle,
    c,
    &bufferMax[c],
    &bufferMin[c]
    MAX_SAMPLES,
    1,
    downSampleRatioMode, // set to RATIO_MODE_AGGREGATE
  );
}
```

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

```
for (int segment = 10; segment < 20; segment++)
{
  ps4000aGetValues
  (
    handle,
    0,
    &noOfSamples, // set to MAX_SAMPLES on entering
    1000,
    &downSampleRatioMode, //set to RATIO_MODE_AGGREGATE
    index,
```



```
      overflow
    );

    ps4000aGetTriggerTimeOffset64
    (
      handle,
      &time,
      &timeUnits,
      index
    )
  }
```

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

3.6.3 Streaming mode

Streaming mode can capture data without the gaps that occur between blocks when using block mode. It can transfer data to the PC at speeds of up to 160 MS/s, 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.

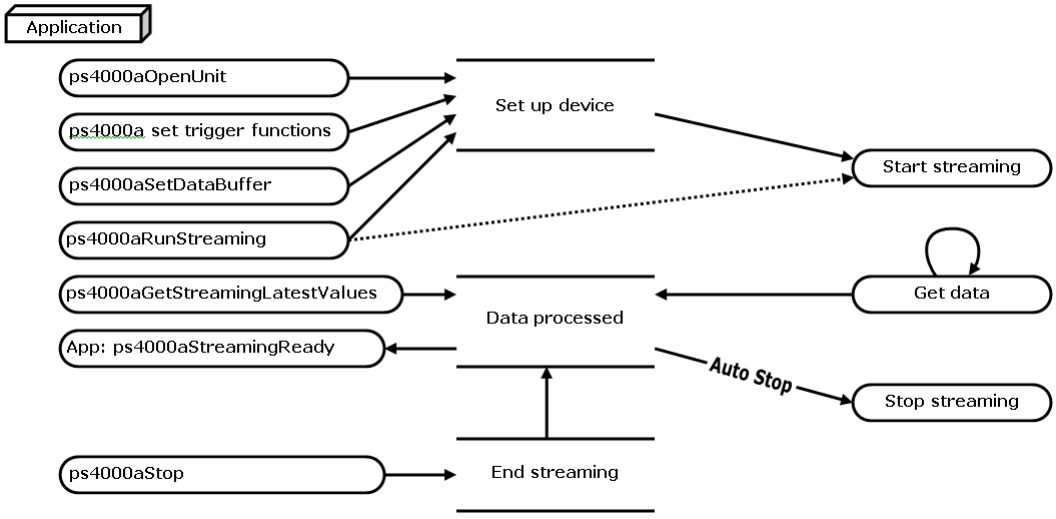
- **Downsampling.** The driver returns downsampled readings while the device is streaming. If the downsampling ratio is set to 1, only one buffer is returned per channel. When the downsampling ratio is greater than 1 and aggregation mode is selected, two buffers (maximum and minimum) per channel are returned.
- **Memory segmentation.** The memory can be divided into segments to reduce the latency of data transfers to the PC. However, this increases the risk of losing data if the PC cannot keep up with the device's sampling rate.

See Using streaming mode for programming details.

3.6.3.1 Using streaming mode

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

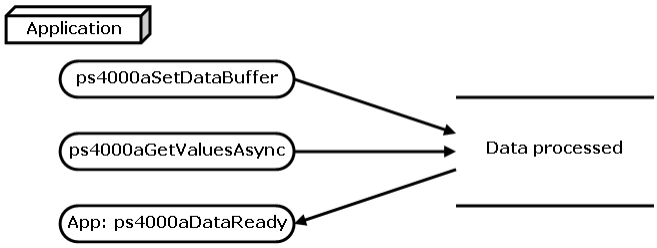
1. Open the oscilloscope using `ps4000aOpenUnit`.
2. Select channels, ranges and AC/DC coupling using `ps4000aSetChannel`.
3. Use the trigger setup functions [1] [2] [3] [4] to set up the trigger if required.
4. Call `ps4000aSetDataBuffer` to tell the driver where your data buffer is.
5. Set up downsampling and start the oscilloscope running using `ps4000aRunStreaming`.
6. Call `ps4000aGetStreamingLatestValues` 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 `ps4000aStop`, even if Auto Stop is enabled.



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

3.6.4 Retrieving stored data

You can collect data from the PicoScope 4000 driver with a different downsampling factor when ps4000aRunBlock or ps4000aRunStreaming has already been called and has successfully captured all the data. Use ps4000aGetValuesAsync.



3.7 Timebases

The API allows you to select one of 2^{32} different timebases created by dividing the oscilloscope's master sampling clock.

Timebase (n)	Sampling interval (t_S)	Sampling frequency (f_S)
n	12.5 ns × (n+1)	80 MHz / (n+1)
0	12.5 ns	80 MHz
1	25 ns	40 MHz
$2^{32}-1$	~54 s	~18.6 mHz

3.8 Combining several oscilloscopes

It is possible to collect data using up to 64 PicoScope 4000 Series PC Oscilloscopes at the same time, depending on the capabilities of the PC. Each oscilloscope must be connected to a separate USB port. The ps4000aOpenUnit 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 ps4000aBlockReady(...)
// define callback function specific to application

handle1 = ps4000aOpenUnit()
handle2 = ps4000aOpenUnit()

ps4000aSetChannel(handle1)
// set up unit 1
ps4000aRunBlock(handle1)

ps4000aSetChannel(handle2)
// set up unit 2
ps4000aRunBlock(handle2)

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

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

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

4 API functions

The PicoScope 4000 Series API exports the following functions for you to use in your own applications. All functions are C functions using the standard call naming convention (`__stdcall`). They are all exported with both decorated and undecorated names.

ps4000aBlockReady
Receive notification when block-mode data ready
ps4000aChangePowerSource
Handle dual-port USB powering
ps4000aCurrentPowerSource
Read current power source
ps4000aCloseUnit
Close a scope device
ps4000aDataReady
Indicate when post-collection data ready
ps4000aEnumerateUnits
Find out how many units are connected
ps4000aFlashLed
Flash the front-panel LED
ps4000aGetAnalogueOffset
Find the allowable analog offset range
ps4000aGetChannelInformation
Find out if extra ranges available
ps4000aGetMaxDownSampleRatio
Find out downsampling ratio for data
ps4000aGetMaxSegments
Get maximum number of memory segments
ps4000aGetNoOfCaptures
Get number of rapid block captures
ps4000aGetNoOfProcessedCaptures
Get number of downsampled rapid block captures
ps4000aGetStreamingLatestValues
Get streaming data while scope is running
ps4000aGetTimebase
Find out what timebases are available
ps4000aGetTimebase2
Find out what timebases are available
ps4000aGetTriggerTimeOffset
Find out when trigger occurred (32-bit)
ps4000aGetTriggerTimeOffset64
Find out when trigger occurred (64-bit)
ps4000aGetUnitInfo
Read information about scope device
ps4000aGetValues
Retrieve block-mode data with callback
ps4000aGetValuesAsync
Retrieve streaming data with callback
ps4000aGetValuesBulk
Retrieve more than one waveform at a time
ps4000aGetValuesOverlapped
Retrieve data in overlapping blocks
ps4000aGetValuesOverlappedBulk
Retrieve overlapping data from multiple segments
ps4000aIsLedFlashing
Read status of LED
ps4000aIsReady
Poll driver in block mode
ps4000aIsTriggerOrPulseWidthQualifierEnabled
Find out whether trigger is enabled
ps4000aMaximumValue
Get maximum allowed sample value

ps4000aMinimumValue
Get minimum allowed sample value
ps4000aMemorySegments
Divide scope memory into segments
ps4000aNoOfStreamingValues
Get number of samples in streaming mode
ps4000aOpenUnit
Open a scope device
ps4000aOpenUnitAsync
Open a scope device without waiting
ps4000aOpenUnitProgress
Check progress of OpenUnit call
ps4000aRunBlock
Start block mode
ps4000aRunStreaming
Start streaming mode
ps4000aSetBandwidthFilter
Enable the bandwidth limiter
ps4000aSetChannel
Set up input channels
ps4000aSetDataBuffer
Register data buffer with driver
ps4000aSetDataBuffers
Register min/max data buffers with driver
ps4000aSetEts
Set up equivalent-time sampling (ETS)
ps4000aSetEtsTimeBuffer
Set up 64-bit buffer for ETS time data
ps4000aSetEtsTimeBuffers
Set up 32-bit buffers for ETS time data
ps4000aSetNoOfCaptures
Set number of rapid block captures
ps4000aSetPulseWidthQualifierConditions
Set up pulse width triggering
ps4000aSetPulseWidthQualifierProperties
Set up pulse width triggering
ps4000aSetSigGenArbitrary
Set up arbitrary waveform generator
ps4000aSetSigGenBuiltIn
Set up function generator
ps4000aSetSigGenPropertiesArbitrary
Set up arbitrary waveform generator
ps4000aSetSigGenPropertiesBuiltIn
Set up function generator
ps4000aSetSimpleTrigger
Set up level triggers only
ps4000aSetTriggerChannelConditions
Specify which channels to trigger on
ps4000aSetTriggerChannelDirections
Set up signal polarities for triggering
ps4000aSetTriggerChannelProperties
Set up trigger thresholds
ps4000aSetTriggerDelay
Set up post-trigger delay
ps4000aSigGenSoftwareControl
Trigger the signal generator
ps4000aStop
Stop data capture
ps4000aStreamingReady
Indicate when streaming-mode data ready

4.1 ps4000aBlockReady

```
typedef void (CALLBACK *ps4000aBlockReady)
(
  int16_t       handle,
  PICO_STATUS   status,
  void        * pParameter
)
```

This callback function is part of your application. You register it with the PicoScope 4000 Series driver using ps4000aRunBlock, and the driver calls it back when block-mode data is ready. You can then download the data using the ps4000aGetValues 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 ps4000aRunBlock. The callback function can write to this location to send any data, such as a status flag, back to your application.
Returns	nothing

4.2 ps4000aChangePowerSource

```
PICO_STATUS ps4000aChangePowerSource
(
  int16_t     handle,
  PICO_STATUS powerstate
)
```

This function controls the two-stage power-up sequence when the device is plugged into a non-USB 3.0 port.

If you receive the PICO_USB3_0_DEVICE_NON_USB3_0_PORT status code from one of the OpenUnit functions (ps4000aOpenUnit, ps4000aOpenUnitAsync or ps4000aOpenUnitProgress), you must then call ps4000aChangePowerSource to switch the device into non-USB 3.0-power mode.

Note. The PicoScope 4824 has two power supply options:
1. *To power it from a USB 3.0 port, use the USB 3.0 cable supplied.*
2. *To power it from a non-USB 3.0 port, use the double-headed USB 2.0 cable supplied and plug it into two USB 1.1 or USB 2.0 ports on the host machine.*

Applicability	All modes
Arguments	handle, the handle of the device. powerstate, the required state of the unit. Must be set to PICO_USB3_0_DEVICE_NON_USB3_0_PORT.
Returns	PICO_OK PICO_POWER_SUPPLY_REQUEST_INVALID PICO_INVALID_PARAMETER PICO_NOT_RESPONDING PICO_INVALID_HANDLE

4.3 ps4000aCurrentPowerSource

```
PICO_STATUS ps4000aCurrentPowerSource
(
  int16_t handle
)
```

This function returns the current power state of the device. There is no need to call this function with the PicoScope 4824 as the device has only one possible state.

Applicability	Reserved for future use
Arguments	`handle`, the handle of the device
Returns	`PICO_OK`

4.4 ps4000aCloseUnit

```
PICO_STATUS ps4000aCloseUnit
(
  int16_t handle
)
```

This function shuts down a PicoScope 4000 scope device.

Applicability	All modes
Arguments	handle, the handle, returned by ps4000aOpenUnit, of the scope device to be closed.
Returns	PICO_OK PICO_HANDLE_INVALID

4.5 ps4000aDataReady

```
typedef void (CALLBACK *ps4000aDataReady)
(
  int16_t       handle,
  PICO_STATUS   status,
  uint32_t      noOfSamples,
  int16_t       overflow,
  void        * pParameter
)
```

This function handles post-collection data returned by the driver after a call to ps4000aGetValuesAsync. It is a callback function that is part of your application. You register it with the PicoScope 4000 Series driver using ps4000aGetValuesAsync, and the driver calls it back when the data is ready.

Applicability	All modes
Arguments	handle, the handle of the device returning the samples. status, indicates success or failure noOfSamples, the number of samples collected. overflow, returns a flag that indicates whether an overvoltage has occurred on any of the channels. It is a bit pattern with bit 0 denoting Channel A. pParameter, a void pointer passed from ps4000aGetValuesAsync. The callback function can write to this location to send any data, such as a status flag, back to the application. The data type is defined by the application programmer.
Returns	nothing

4.6 ps4000aEnumerateUnits

```
PICO_STATUS ps4000aEnumerateUnits
(
  int16_t * count,
  int8_t  * serials,
  int16_t * serialLth
)
```

This function counts the number of PicoScope 4000 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 scopes 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_ANALOG_BOARD`
	`PICO_CONFIG_FAIL_AWG`
	`PICO_INITIALISE_FPGA`

4.7 ps4000aFlashLed

```
PICO_STATUS ps4000aFlashLed
(
  int16_t  handle,
  int16_t  start
)
```

This function flashes the LED on the front of the scope without blocking the calling thread. Calls to ps4000aRunStreaming and ps4000aRunBlock cancel any flashing started by this function.

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`

4.8 ps4000aGetAnalogueOffset

```
PICO_STATUS ps4000aGetAnalogueOffset
(
  int16_t            handle,
  PS4000A_RANGE,     range,
  PS4000A_COUPLING   coupling,
  float            * maximumVoltage,
  float            * minimumVoltage
)
```

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

Applicability	All modes
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`, on exit, the maximum voltage allowed for the range. Pointer may be `NULL` if not required. `* minimumVoltage`, on exit, the minimum voltage allowed for the range. Pointer may be `NULL` if not required. If both `maximumVoltage` and `minimumVoltage` are `NULL`, the driver returns `PICO_NULL_PARAMETER`.

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

4.9 ps4000aGetChannelInformation

```
PICO_STATUS ps4000aGetChannelInformation
(
  int16_t                 handle,
  PS4000A_CHANNEL_INFO    info,
  int32_t                 probe,
  int32_t               * ranges,
  int32_t               * length,
  int32_t                 channel
)
```

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

Applicability	Reserved for future expansion
Arguments	`handle`, the handle of the required device `info`, the type of information required. The only value supported is: `PS4000A_CI_RANGES`, returns the extra ranges available `probe`, not used, must be set to 0 `* ranges`, on exit, an array populated with available ranges for the given value of `info`. May be `NULL`. See `ps4000aSetChannel` for possible values. `* length`, on entry: the length of the `ranges` array; on exit: the number of elements written to `ranges` or, if `ranges` is `NULL`, the number of elements that would have been written. `channel`, the channel for which the information is required. See `ps4000aSetChannel` for possible values.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`

4.10 ps4000aGetMaxDownSampleRatio

```
PICO_STATUS ps4000aGetMaxDownSampleRatio
(
  int16_t              handle,
  uint32_t             noOfUnaggregatedSamples,
  uint32_t           * maxDownSampleRatio,
  PS4000A_RATIO_MODE   downSampleRatioMode,
  uint32_t             segmentIndex
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device `noOfUnaggregatedSamples`, the number of raw samples to be used to calculate the maximum downsampling ratio `* maxDownSampleRatio`, on exit, the maximum possible downsampling ratio `downSampleRatioMode`, see Downsampling `segmentIndex`, the memory segment where the data is stored

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_NULL_PARAMETER`
	`PICO_INVALID_PARAMETER`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_TOO_MANY_SAMPLES`

4.11 ps4000aGetMaxSegments

```
PICO_STATUS ps4000aGetMaxSegments
(
  int16_t     handle,
  uint32_t  * maxSegments
)
```

This function retrieves the maximum number of memory segments allowed by the device.

Applicability	All modes
Arguments	`handle`, the handle of the required device `* maxSegments`, on exit, the maximum number of segments: PicoScope 4824: 125 000
Returns	`PICO_OK`

4.12 ps4000aGetNoOfCaptures

```
PICO_STATUS ps4000aGetNoOfCaptures
(
  int16_t     handle,
  uint32_t  * nCaptures
)
```

This function gets the number of captures collected in one run of rapid block mode.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the device `* nCaptures`, on exit, the number of waveforms captured

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`

4.13 ps4000aGetNoOfProcessedCaptures

```
PICO_STATUS ps4000aGetNoOfProcessedCaptures
(
  int16_t     handle,
  uint32_t  * nCaptures
)
```

This function gets the number of captures collected and processed in one run of rapid block mode.

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

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`

4.14 ps4000aGetStreamingLatestValues

```
PICO_STATUS ps4000aGetStreamingLatestValues
(
  int16_t                 handle,
  ps4000aStreamingReady   lpPs4000Ready,
  void                  * pParameter
)
```

This function is used to collect the next block of values while streaming is running. You must call ps4000aRunStreaming beforehand to set up streaming.

Applicability	Streaming mode only
Arguments	handle, the handle of the required device. lpPs4000Ready, a pointer to your ps4000aStreamingReady callback function that will return the latest downsampled values. pParameter, a void pointer that will be passed to the ps4000aStreamingReady callback function.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_INVALID_CALL PICO_BUSY PICO_NOT_RESPONDING

4.15 ps4000aGetTimebase

```
PICO_STATUS ps4000aGetTimebase
(
  int16_t    handle,
  uint32_t   timebase,
  int32_t    noSamples,
  int32_t  * timeIntervalNanoseconds,
  int32_t  * maxSamples
  uint32_t   segmentIndex
)
```

This function discovers which timebases are available on the oscilloscope. You should set up the channels using `ps4000aSetChannel` first.

Applicability	All modes
Arguments	`handle`, the handle of the required device. `timebase`, a code between 0 and $2^{32}-1$ that specifies the sampling interval (see Timebases). `noSamples`, the number of samples required. This value is used to calculate the most suitable time unit to use. `* timeIntervalNanoseconds`, on exit, the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. `* maxSamples`, on exit, the maximum number of samples available. This number may vary depending on the number of channels enabled, the timebase chosen and the oversample selected. If this pointer is null, nothing will be written here. `segmentIndex`, the number 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`

4.16 ps4000aGetTimebase2

```
PICO_STATUS ps4000aGetTimebase2
(
  int16_t    handle,
  uint32_t   timebase,
  int32_t    noSamples,
  float    * timeIntervalNanoseconds,
  int32_t  * maxSamples,
  uint32_t   segmentIndex
)
```

This function differs from ps4000aGetTimebase only in the type of the timeIntervalNanoseconds argument.

Applicability	All modes
Arguments	* timeIntervalNanoseconds, on exit, the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. All others as in ps4000aGetTimebase.
Returns	See ps4000aGetTimebase.

4.17 ps4000aGetTriggerTimeOffset

```
PICO_STATUS ps4000aGetTriggerTimeOffset
(
  int16_t              handle,
  uint32_t           * timeUpper,
  uint32_t           * timeLower,
  PS4000A_TIME_UNITS * timeUnits,
  uint32_t             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.

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`, on exit, the time units in which * `timeUpper` and * `timeLower` are measured. The allowable values are:
	`PS4000A_FS`
	`PS4000A_PS`
	`PS4000A_NS`
	`PS4000A_US`
	`PS4000A_MS`
	`PS4000A_S`
	
	`segmentIndex`, the number of the memory segment for which the information is required.

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

4.18 ps4000aGetTriggerTimeOffset64

```
PICO_STATUS ps4000aGetTriggerTimeOffset64
(
  int16_t             handle,
  int64_t           * time,
  PS4000A_TIME_UNITS * timeUnits,
  uint32_t            segmentIndex
)
```

This function gets the time, as a single 8-byte 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.

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. See `ps4000aGetTriggerTimeOffset`. `segmentIndex`, the number of the memory segment for which the information is required

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

4.19 ps4000aGetUnitInfo

```
PICO_STATUS ps4000aGetUnitInfo
(
  int16_t     handle,
  int8_t    * string,
  int16_t     stringLength,
  int16_t   * requiredSize,
  PICO_INFO   info
)
```

This function writes information about the specified scope device to a character string. If the device fails to open, only the driver version and error code are available to explain why the last open unit call failed.

Applicability	All modes
Arguments	`handle`, the handle of the device from which information is required. If an invalid handle is passed, the error code from the last unit that failed to open is returned. `string`, the character string buffer in the calling function where the unit information string (selected with `info`) will be stored. If a null pointer is passed, only `requiredSize` is returned. `stringLength`, the size of the character string buffer. `* requiredSize`, on exit, the required character string buffer size. `info`, an enumerated type specifying what information is required from the driver. Values are listed below.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`
	`PICO_INVALID_INFO`
	`PICO_INFO_UNAVAILABLE`

PICO_INFO constant	**Example**
`0: PICO_DRIVER_VERSION`, version number of PicoScope 4000A 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	`4824`
`4: PICO_BATCH_AND_SERIAL`, batch and serial number of device	`KJL87/6`
`5: PICO_CAL_DATE`, calibration date of device	`11Nov13`
`6: PICO_KERNEL_VERSION`, version of kernel driver	`1,1,2,4`

4.20 ps4000aGetValues

```
PICO_STATUS ps4000aGetValues
(
  int16_t              handle,
  uint32_t             startIndex,
  uint32_t           * noOfSamples,
  uint32_t             downSampleRatio,
  PS4000A_RATIO_MODE   downSampleRatioMode,
  uint32_t             segmentIndex,
  int16_t            * overflow
)
```

This function returns block-mode 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.

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 requested; on exit, the number of samples actually returned. `downSampleRatio`, the downsampling factor that will be applied to the raw data. Multiple downsampling modes can be bitwise-ORed together, but the `downSampleRatio` must be the same for all modes. `downSampleRatioMode`, whether to use downsampling to reduce the amount of data. See Downsampling. `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 pattern with bit 0 corresponding to Channel A.

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

4.21 ps4000aGetValuesAsync

```
PICO_STATUS ps4000aGetValuesAsync
(
  int16_t              handle,
  uint32_t             startIndex,
  uint32_t             noOfSamples,
  uint32_t             downSampleRatio,
  PS4000A_RATIO_MODE   downSampleRatioMode,
  uint32_t             segmentIndex,
  void               * lpDataReady,
  void               * pParameter
)
```

This function returns streaming 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 only
Arguments	handle, the handle of the required device startIndex, see ps4000aGetValues noOfSamples, see ps4000aGetValues downSampleRatio, see ps4000aGetValues downSampleRatioMode, see ps4000aGetValues segmentIndex, see ps4000aGetValues * lpDataReady, the ps4000aStreamingReady function that is called when the data is ready pParameter, a void pointer that will be passed to the ps4000aStreamingReady callback function. The data type depends on the design of the callback function, which is determined by the application programmer.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_DEVICE_SAMPLING – streaming only PICO_NULL_PARAMETER PICO_STARTINDEX_INVALID PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_PARAMETER PICO_DATA_NOT_AVAILABLE PICO_INVALID_SAMPLERATIO PICO_INVALID_CALL

4.22 ps4000aGetValuesBulk

```
PICO_STATUS ps4000aGetValuesBulk
(
  int16_t              handle,
  uint32_t           * noOfSamples,
  uint32_t             fromSegmentIndex,
  uint32_t             toSegmentIndex,
  unit32_t             downSampleRatio,
  PS4000A_RATIO_MODE   downSampleRatioMode,
  int16_t            * overflow
)
```

This function allows more than one waveform to be retrieved at a time in rapid block mode. The waveforms must have been collected sequentially and in the same run. This method of collection does not support downsampling.

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 Downsampling `downSampleRatioMode`, see Downsampling `* overflow`, an array of at least as many integers as the number of waveforms to be retrieved. Each segment index has a separate `overflow` element, with `overflow[0]` containing the `fromSegmentIndex` and the last index the `toSegmentIndex`.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_STARTINDEX_INVALID`
	`PICO_NOT_RESPONDING`

4.23 ps4000aGetValuesOverlapped

```
PICO_STATUS ps4000aGetValuesOverlapped
(
  int16_t              handle,
  uint32_t             startIndex,
  uint32_t           * noOfSamples,
  uint32_t             downSampleRatio,
  PS4000A_RATIO_MODE   downSampleRatioMode,
  uint32_t             segmentIndex,
  int16_t            * overflow
)
```

This function allows you to make a deferred data-collection request, which will later be executed, and the arguments validated, when you call ps4000aRunBlock in block mode. The advantage of this function is that the driver makes contact with the scope only once, when you call ps4000aRunBlock, compared with the two contacts that occur when you use the conventional ps4000aRunBlock, ps4000aGetValues calling sequence. This slightly reduces the dead time between successive captures in block mode.

After calling ps4000aRunBlock, you can optionally use ps4000aGetValues 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 ps4000aGetValues * noOfSamples: see ps4000aGetValues downSampleRatio: see ps4000aGetValues downSampleRatioMode: see ps4000aGetValues segmentIndex: see ps4000aGetValues * overflow: see ps4000aGetValuesBulk
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

4.24 ps4000aGetValuesOverlappedBulk

```
PICO_STATUS ps4000aGetValuesOverlappedBulk
(
  int16_t              handle,
  uint32_t             startIndex,
  uint32_t           * noOfSamples,
  uint32_t             downSampleRatio,
  PS4000A_RATIO_MODE   downSampleRatioMode,
  uint32_t             fromSegmentIndex,
  uint32_t             toSegmentIndex,
  int16_t            * overflow
)
```

This function allows you to make a deferred data-collection request, which will later be executed, and the arguments validated, when you call ps4000aRunBlock in rapid block mode. The advantage of this method is that the driver makes contact with the scope only once, when you call ps4000aRunBlock, compared with the two contacts that occur when you use the conventional ps4000aRunBlock, ps4000aGetValuesBulk calling sequence. This slightly reduces the dead time between successive captures in rapid block mode.

After calling ps4000aRunBlock, you can optionally use ps4000aGetValues 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 ps4000aGetValues * noOfSamples: see ps4000aGetValues downSampleRatio: see ps4000aGetValues downSampleRatioMode: see ps4000aGetValues fromSegmentIndex: see ps4000aGetValuesBulk toSegmentIndex: see ps4000aGetValuesBulk * overflow, see ps4000aGetValuesBulk
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

4.25 ps4000aIsLedFlashing

```
PICO_STATUS ps4000aIsLedFlashing
(
  int16_t   handle,
  int16_t * status
)
```

This function reports whether or not the LED is flashing.

Applicability	All modes
Arguments	`handle`, the handle of the scope device `status`, returns a flag indicating the status of the LED: <> 0 : flashing 0 : not flashing

Returns	`PICO_OK`
	`PICO_HANDLE_INVALID`
	`PICO_NULL_PARAMETER`

4.26 ps4000aIsReady

```
PICO_STATUS ps4000aIsReady
(
  int16_t   handle,
  int16_t * ready
)
```

This function may be used instead of a callback function to receive data from ps4000aRunBlock. To use this method, pass a NULL pointer as the lpReady argument to ps4000aRunBlock. 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, on exit, indicates the state of the collection. If zero, the device is still collecting. If non-zero, the device has finished collecting and ps4000aGetValues can be used to retrieve the data.
Returns	

4.27 ps4000aIsTriggerOrPulseWidthQualifierEnabled

```
PICO_STATUS ps4000aIsTriggerOrPulseWidthQualifierEnabled
(
  int16_t   handle,
  int16_t * triggerEnabled,
  int16_t * pulseWidthQualifierEnabled
)
```

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

Applicability	Call after setting up the trigger, and just before calling either ps4000aRunBlock or ps4000aRunStreaming.
Arguments	handle, the handle of the required device * triggerEnabled, on exit, indicates whether the trigger will successfully be set when ps4000aRunBlock or ps4000aRunStreaming is called. A non-zero value indicates that the trigger is set, otherwise the trigger is not set. * pulseWidthQualifierEnabled, on exit, indicates whether the pulse width qualifier will successfully be set when ps4000aRunBlock or ps4000aRunStreaming is called. A non-zero value indicates that the pulse width qualifier is set, otherwise the pulse width qualifier is not set.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER

4.28 ps4000aMaximumValue

```
PICO_STATUS ps4000aMaximumValue
(
  int16_t    handle,
  int16_t  * value
)
```

This function returns the maximum possible sample value in the current operating mode.

Applicability	All modes
Arguments	`handle`, the handle of the required device `* value`, on exit, the maximum value
Returns	`PICO_OK`

4.29 ps4000aMinimumValue

```
PICO_STATUS ps4000aMinimumValue
(
  int16_t    handle,
  int16_t  * value
)
```

This function returns the minimum possible sample value in the current operating mode.

Applicability	All modes
Arguments	`handle`, the handle of the required device `* value`, on exit, the minimum value
Returns	`PICO_OK`

4.30 ps4000aMemorySegments

```
PICO_STATUS ps4000aMemorySegments
(
  int16_t    handle,
  uint32_t   nSegments,
  int32_t  * nMaxSamples
)
```

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

By default, each capture fills the scope device's available memory. This function allows you to divide the memory into a number of segments so that the scope can store several captures sequentially. The number of segments defaults to 1 when the scope device is opened.

Applicability	All modes
Arguments	`handle`, the handle of the required device `nSegments`, the number of segments to be used, from 1 to the number returned by `ps4000aGetMaxSegments`. `* nMaxSamples`, on exit, the number of samples that are available in each segment. This is independent of the number of 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`

4.31 ps4000aNoOfStreamingValues

```
PICO_STATUS ps4000aNoOfStreamingValues
(
  int16_t    handle,
  uint32_t * noOfValues
)
```

This function returns the available number of samples from a streaming run.

Applicability	Streaming mode. Call after ps4000aStop.
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

4.32 ps4000aOpenUnit

```
PICO_STATUS ps4000aOpenUnit
(
  int16_t * handle,
  int8_t  * serial
)
```

This function opens a scope device. The maximum number of units that can be opened is determined by the operating system, the kernel driver and the PC's hardware.

Applicability	All modes
Arguments	`handle`, on exit, the handle of the device: -1 : if the unit fails to open, 0 : if no unit is found or > 0 : if successful (value is handle of the device opened) The handle must be used in all subsequent calls to API functions to identify this scope device. `* serial`, on exit, a null-terminated string containing the device's serial number.
Returns	PICO_OK PICO_OS_NOT_SUPPORTED PICO_OPEN_OPERATION_IN_PROGRESS PICO_EEPROM_CORRUPT PICO_KERNEL_DRIVER_TOO_OLD PICO_FW_FAIL PICO_MAX_UNITS_OPENED PICO_NOT_FOUND PICO_NOT_RESPONDING PICO_USB3_0_DEVICE_NON_USB3_0_PORT

4.33 ps4000aOpenUnitAsync

```
PICO_STATUS ps4000aOpenUnitAsync
(
  int16_t * status,
  int8_t  * serial
)
```

This function opens a scope device without blocking the calling thread. You can find out when it has finished by periodically calling ps4000aOpenUnitProgress until that function returns a non-zero value.

Applicability	All modes
Arguments	* status, on exit, indicates: 0 if there is already an open operation in progress 1 if the open operation is initiated * serial, on exit, a null-terminated string containing the device's serial number.
Returns	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_OPERATION_FAILED

4.34 ps4000aOpenUnitProgress

```
PICO_STATUS ps4000aOpenUnitProgress
(
  int16_t * handle,
  int16_t * progressPercent,
  int16_t * complete
)
```

This function checks on the progress of ps4000aOpenUnitAsync.

Applicability	Use after ps4000aOpenUnitAsync
Arguments	* handle, on exit, the unit handle. -1 if the unit fails to open, 0 if no unit is found or a non-zero handle to the device. **This handle is valid only if the function returns PICO_OK.** * progressPercent, on exit, the percentage progress. 100% implies that the open operation is complete. * complete, on exit, set to 1 when the open operation has finished
Returns	PICO_OK PICO_NULL_PARAMETER PICO_OPERATION_FAILED PICO_USB3_0_DEVICE_NON_USB3_0_PORT

4.35 ps4000aRunBlock

```
PICO_STATUS ps4000aRunBlock
(
  int16_t                handle,
  int32_t                noOfPreTriggerSamples,
  int32_t                noOfPostTriggerSamples,
  uint32_t               timebase,
  int32_t              * timeIndisposedMs,
  uint32_t               segmentIndex,
  ps4000aBlockReady      lpReady,
  void                 * pParameter
)
```

This function starts a collection of data points (samples) in 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 memory depth 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 data points (samples) to collect. `noOfPostTriggerSamples`, the number of samples to be taken after a trigger event. If no trigger event is set then this specifies the maximum number of samples to be taken. If a trigger condition has been set, this specifies the number of data points (samples) to be taken after a trigger has fired, and the number of data points to be collected is: - `noOfPreTriggerSamples + noOfPostTriggerSamples` `timebase`, a number in the range 0 to 2^{32}-1. See the guide to calculating timebase values. `* timeIndisposedMs`, on exit, the time, in milliseconds, that the scope will spend collecting samples. This does not include any auto trigger timeout. If this pointer is null, nothing will be written here. `segmentIndex`, zero-based, specifies which memory segment to use. `lpReady`, a pointer to the `ps4000aBlockReady` callback that the driver will call when the data has been collected. To use the `ps4000aIsReady` polling method instead of a callback function, set this pointer to NULL. `* pParameter`, a void pointer that is passed to the `ps4000aBlockReady` callback function. The callback can use the pointer to return arbitrary data to your application.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_TRIGGER_CHANNEL`
	`PICO_INVALID_CONDITION_CHANNEL`
	`PICO_TOO_MANY_SAMPLES`
	`PICO_INVALID_TIMEBASE`
	`PICO_NOT_RESPONDING`
	`PICO_CONFIG_FAIL`
	`PICO_INVALID_PARAMETER`
	`PICO_NOT_RESPONDING`
	`PICO_TRIGGER_ERROR`

4.36 ps4000aRunStreaming

```
PICO_STATUS ps4000aRunStreaming
(
  int16_t              handle,
  uint32_t           * sampleInterval,
  PS4000A_TIME_UNITS   sampleIntervalTimeUnits,
  uint32_t             maxPreTriggerSamples,
  uint32_t             maxPostTriggerSamples,
  int16_t              autoStop,
  uint32_t             downSampleRatio,
  PS4000A_RATIO_MODE   downSampleRatioMode,
  uint32_t             overviewBufferSize
)
```

This function tells the oscilloscope to start collecting data in streaming mode. When data has been collected from the device it is downsampled and the values returned to the application. Call `ps4000aGetStreamingLatestValues` to retrieve the data.

When a trigger is set, the sum of `maxPreTriggerSamples` and `maxPostTriggerSamples` is the total number of samples stored in the driver. If `autoStop` is false, this will become the maximum number of unaggregated samples.

Applicability	Streaming mode only
Arguments	`handle`, the handle of the required device. `* sampleInterval`, on entry, the requested time interval between data points on entry; on exit, the actual time interval assigned. `sampleIntervalTimeUnits`, the unit of time that the `sampleInterval` is set to. See `ps4000aGetTriggerTimeOffset` for values. `maxPreTriggerSamples`, the maximum number of raw samples before a trigger condition for each enabled channel. If no trigger condition is set this argument is ignored. `maxPostTriggerSamples`, the maximum number of raw samples after a trigger condition for each enabled channel. If no trigger condition is set this argument states the maximum number of samples to be stored. `autoStop`, a flag to specify if the streaming should stop when all of `maxSamples` have been taken. `downSampleRatio`, the number of raw values to each downsampled value. `downSampleRatioMode`, the type of data reduction to use. See `ps4000aGetValues` for details. `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 `ps4000aSetDataBuffer`.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_NULL_PARAMETER`
	`PICO_INVALID_PARAMETER`
	`PICO_STREAMING_FAILED`
	`PICO_NOT_RESPONDING`
	`PICO_TRIGGER_ERROR`
	`PICO_INVALID_SAMPLE_INTERVAL`
	`PICO_INVALID_BUFFER`

4.37 ps4000aSetBandwidthFilter

```
PICO_STATUS ps4000aSetBandwidthFilter
(
  int16_t                   handle,
  PS4000A_CHANNEL           channel,
  PS4000A_BANDWIDTH_LIMITER bandwidth
)
```

This function is reserved for future use.

Applicability	Not implemented
Arguments	`handle`, the handle of the required device `channel`, an enumerated type. The values are: `PS4000A_CHANNEL_A ... PS4000A_CHANNEL_H` `bandwidth`, the required cutoff frequency of the filter. Allowable values are: `PS4000A_BW_FULL`, the full bandwidth of the scope `PS4000A_BW_20MHZ`, 20 MHz

Returns	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`

4.38 ps4000aSetChannel

```
PICO_STATUS ps4000aSetChannel
(
  int16_t           handle,
  PS4000A_CHANNEL   channel,
  int16_t           enabled,
  PS4000A_COUPLING  type,
  PS4000A_RANGE     range,
  float             analogOffset
)
```

This function sets up the characteristics of the specified input channel.

Applicability	All modes
Arguments	`handle`, a unique identifier for the device. `channel`, the channel to be configured. The allowable values are: `PS4000A_CHANNEL_A ... PS4000A_CHANNEL_H` `enabled`, specifies if the channel is active (`TRUE`) or inactive (`FALSE`). type, specifies the coupling mode: DC (`TRUE`) or AC (`FALSE`). `range`, specifies the measuring range. Measuring ranges 0 to 13 are shown in the table below. `analogOffset`, a voltage, in volts, to be added to the input signal before it reaches the input amplifier and digitizer. See the device data sheet for the allowable range.

Returns	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_VOLTAGE_RANGE`

range		Voltage range
0	PS4000A_10MV	±10 mV
1	PS4000A_20MV	±20 mV
2	PS4000A_50MV	±50 mV
3	PS4000A_100MV	±100 mV
4	PS4000A_200MV	±200 mV
5	PS4000A_500MV	±500 mV
6	PS4000A_1V	±1 V
7	PS4000A_2V	±2 V
8	PS4000A_5V	±5 V
9	PS4000A_10V	±10 V
10	PS4000A_20V	±20 V
11	PS4000A_50V	±50 V
12	PS4000A_100V	±100 V
13	PS4000A_200V	±200 V

4.39 ps4000aSetDataBuffer

```
PICO_STATUS ps4000aSetDataBuffer
(
  int16_t             handle,
  PS4000A_CHANNEL     channel,
  int16_t           * buffer,
  int32_t             bufferLth,
  uint32_t            segmentIndex,
  PS4000A_RATIO_MODE  mode
)
```

This function registers your data buffer, for non-downsampled data, with the PicoScope 4000 driver. You need to allocate the buffer before calling this function.

Applicability	All modes. For downsampled data, use ps4000aSetDataBuffers instead.
Arguments	handle, the handle of the required device channel, the channel for which you want to set the buffers. Use one of these values: PS4000A_CHANNEL_A ... PS4000A_CHANNEL_H * buffer, a buffer to receive the data values bufferLth, the size of the buffer array segmentIndex, the serial number of the segment to be retrieved mode, the type of data reduction to use. See Downsampling for options.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL

4.40 ps4000aSetDataBuffers

```
PICO_STATUS ps4000aSetDataBuffers
(
  int16_t             handle,
  PS4000A_CHANNEL     channel,
  int16_t           * bufferMax,
  int16_t           * bufferMin,
  int32_t             bufferLth,
  uint32_t            segmentIndex,
  PS4000A_RATIO_MODE  mode
)
```

This function registers your data buffers, for receiving downsampled data, with the PicoScope 4000 driver. You need to allocate memory for the buffers before calling this function.

Applicability	All sampling modes. For non-downsampled data, use `ps4000aSetDataBuffer` instead.
Arguments	`handle`, the handle of the required device. `channel`, the channel for which you want to set the buffers. Use one of these constants: - `PS4000A_CHANNEL_A ... PS4000A_CHANNEL_H` `* bufferMax`, a buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise. `* bufferMin`, a buffer to receive the minimum data values when `downSampleRatio` > 1 (downsampling modes other than aggregation). Not used when `downSampleRatio` is 1 (aggregation mode). `bufferLth`, specifies the size of the `bufferMax` and `bufferMin` arrays. `segmentIndex`, the serial number of the segment to be retrieved. `mode`, the type of downsampling to use. See Downsampling.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`

4.41 ps4000aSetEts

```
PICO_STATUS ps4000aSetEts
(
  int16_t            handle,
  PS4000A_ETS_MODE   mode,
  int16_t            etsCycles,
  int16_t            etsInterleave,
  int32_t          * sampleTimePicoseconds
)
```

This function is reserved for future use.

Applicability	Not implemented
Arguments	`handle`, the handle of the required device `mode`, not used `ets_cycles`, not used `ets_interleave`, not used `* sampleTimePicoseconds`, not used
Returns	`PICO_ETS_NOT_SUPPORTED`

4.42 ps4000aSetEtsTimeBuffer

```
PICO_STATUS ps4000aSetEtsTimeBuffer
(
  int16_t   handle,
  int64_t * buffer,
  int32_t   bufferLth
)
```

This is reserved for future use.

Applicability	Not implemented
Arguments	`handle`, the handle of the required device `* buffer`, not used `bufferLth`, not used
Returns	`PICO_ETS_NOT_SUPPORTED`

4.43 ps4000aSetEtsTimeBuffers

```
PICO_STATUS ps4000aSetEtsTimeBuffers
(
  int16_t     handle,
  uint32_t  * timeUpper,
  uint32_t  * timeLower,
  int32_t     bufferLth
)
```

This function is reserved for future use.

Applicability	Not implemented
Arguments	`handle`, the handle of the required device `* timeUpper`, not used `* timeLower`, not used `bufferLth`, not used
Returns	`PICO_ETS_NOT_SUPPORTED`

4.44 ps4000aSetNoOfCaptures

```
PICO_STATUS ps4000aSetNoOfCaptures
(
  int16_t   handle,
  uint32_t  nCaptures
)
```

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

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

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`

4.45 ps4000aSetPulseWidthQualifierConditions

```
PICO_STATUS ps4000aSetPulseWidthQualifierConditions
(
  int16_t                    handle,
  PS4000A_CONDITION       * conditions,
  int16_t                    nConditions,
  PS4000A_CONDITIONS_INFO    info
)
```

This function sets up the conditions for pulse width qualification, which can be used on its own for pulse width triggering or combined with window triggering to produce more complex triggers. Each call to this function creates a pulse width qualifier equal to the logical AND of the elements of the `conditions` array. Calling this function multiple times creates the logical OR of multiple AND operations. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

Other settings of the pulse width qualifier are configured by calling `ps4000SetPulseWidthQualifierProperties`.

Applicability	All modes

Arguments	`handle`, the handle of the required device
	
	`* conditions`, see `ps4000aTriggerConditions`
	`nConditions`, see `ps4000aTriggerConditions`
	`info`, see `ps4000aTriggerConditions`

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_CONDITIONS`
	`PICO_PULSE_WIDTH_QUALIFIER`

4.46 ps4000aSetPulseWidthQualifierProperties

```
PICO_STATUS ps4000aSetPulseWidthQualifierProperties
(
  int16_t                         handle,
  PS4000A_THRESHOLD_DIRECTION     direction,
  uint32_t                        lower,
  uint32_t                        upper,
  PS4000A_PULSE_WIDTH_TYPE        type
)
```

This function configures the general properties of the pulse width qualifier.

Applicability	All modes
Arguments	`handle`, the handle of the required device `direction`, the direction of the signal required for the trigger to fire. See `PS4000A_DIRECTION` for allowable values. `lower`, the lower limit of the pulse width counter `upper`, the upper limit of the pulse width counter. This parameter is used only when the type is set to `PW_TYPE_IN_RANGE` or `PW_TYPE_OUT_OF_RANGE`. `type`, the pulse width type, one of these constants: `PW_TYPE_NONE` (do not use the pulse width qualifier) `PW_TYPE_LESS_THAN` (pulse width less than `lower`) `PW_TYPE_GREATER_THAN` (pulse width greater than `lower`) `PW_TYPE_IN_RANGE` (pulse width between `lower` and `upper`) `PW_TYPE_OUT_OF_RANGE` (pulse width not between `lower` and `upper`)

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_CONDITIONS`
	`PICO_PULSE_WIDTH_QUALIFIER`

4.47 ps4000aSetSigGenArbitrary

```
PICO_STATUS ps4000aSetSigGenArbitrary (
  int16_t                    handle,
  int32_t                    offsetVoltage, // see note 1
  uint32_t                   pkToPk, // see note 1
  uint32_t                   startDeltaPhase,
  uint32_t                   stopDeltaPhase,
  uint32_t                   deltaPhaseIncrement,
  uint32_t                   dwellCount,
  int16_t                  * arbitraryWaveform, // see note 1
  int32_t                    arbitraryWaveformSize, // see
                          note 1
  PS4000A_SWEEP_TYPE         sweepType,
  PS4000A_EXTRA_OPERATIONS   operation, // see note 1
  PS4000A_INDEX_MODE         indexMode,
  uint32_t                   shots,
  uint32_t                   sweeps,
  PS4000A_SIGGEN_TRIG_TYPE   triggerType,
  PS4000A_SIGGEN_TRIG_SOURCE triggerSource,
  int16_t                    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. The top bits of the phase accumulator are used as an index into a buffer containing the arbitrary waveform. The remaining bits act as the fractional part of the index, enabling high-resolution control of output frequency and allowing the generation of lower frequencies.

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

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

where:

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

Parameter	Value
dacFrequency	80 MHz
dacPeriod (= 1/*dacFrequency*)	12.5 ns
phaseAccumulatorSize	4 294 967 296 (2^{32})
awgBufferSize	16 384 (2^{14})

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

Note 1: in general, this function can be called with new arguments while waiting for a trigger; the exceptions are the arguments `offsetVoltage, pkToPk, arbitraryWaveform, arbitraryWaveformSize` and `operation`, which must unchanged on subsequent calls, otherwise the function will return a `PICO_BUSY` status code.

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.

`startDeltaPhase,` the initial value added to the phase counter as the generator begins to step through the waveform buffer.

`stopDeltaPhase,` the final value added to the phase counter before the generator restarts or reverses the sweep. When frequency sweeping is not required, set equal to `startDeltaPhase.`

`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. When frequency sweeping is not required, set to zero.

`dwellCount,` the time, in multiples of *dacPeriod,* between successive additions of `deltaPhaseIncrement` to the delta phase counter. This determines the rate at which the generator sweeps the output frequency. Minimum allowable values are as follows:

PicoScope 4824: `MIN_DWELL_COUNT`

`* arbitraryWaveform,` a buffer that holds the waveform pattern as a set of samples equally spaced in time. The sample value range is [0, 4095].

`arbitraryWaveformSize,` the size of the arbitrary waveform buffer, in samples:
Min: `MIN_SIG_GEN_BUFFER_SIZE` (1)
Max: `MAX_SIG_GEN_BUFFER_SIZE` (16 384)

`sweepType,` determines whether the `startDeltaPhase` is swept up to the `stopDeltaPhase,` or down to it, or repeatedly up and down. Use one of the following values: `UP, DOWN, UPDOWN, DOWNUP.`

`operation,` configures the white noise/PRBS (pseudo-random binary sequence) generator:

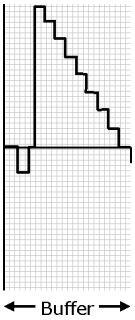
`PS4000A_ES_OFF:`	White noise/PRBS output disabled. The waveform is defined by the other arguments.
`PS4000A_WHITENOISE:`	The signal generator produces white noise and ignores all settings except `offsetVoltage` and `pkTopk.`
`PS4000A_PRBS:`	The signal generator produces a PRBS.

`indexMode`, specifies how the signal will be formed from the arbitrary waveform data. `SINGLE`, `DUAL` and `QUAD` index modes are possible (see AWG index modes). `shots`, the number of cycles of the waveform to be produced after a trigger event. If this is set to a non-zero value [1, `MAX_SWEEPS_SHOTS`], then `sweeps` must be set to zero. `sweeps`, the number of times to sweep the frequency after a trigger event, according to `sweepType`. If this is set to a non-zero value [1, `MAX_SWEEPS_SHOTS`], then `shots` must be set to zero. `triggerType`, the type of trigger that will be applied to the signal generator: `SIGGEN_RISING`: rising edge `SIGGEN_FALLING`: falling edge `SIGGEN_GATE_HIGH`: high level `SIGGEN_GATE_LOW`: low level `triggerSource`, the source that will trigger the signal generator: `SIGGEN_NONE`: no trigger (free-running) `SIGGEN_SCOPE_TRIG`: the selected oscilloscope channel (see ps4000aSetSimpleTrigger) `SIGGEN_SOFT_TRIG`: a software trigger (see ps4000aSigGenSoftwareControl) If a trigger source other than `SIGGEN_NONE` is specified, then either `shots` or `sweeps`, but not both, must be set to a non-zero value. `extInThreshold`, not used	
Returns	0: if successful. Error code: if failed

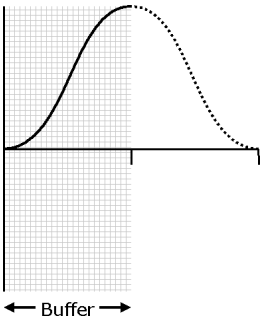
4.47.1 AWG index modes

The arbitrary waveform generator supports SINGLE, DUAL and QUAD index modes to make the best use of the waveform buffer.

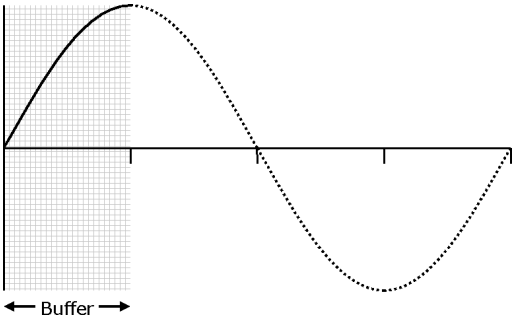
SINGLE mode. The generator outputs the raw contents of the buffer repeatedly. This mode is the only one that can generate asymmetrical waveforms. You can also use this mode for symmetrical waveforms, but the dual and quad modes make more efficient use of the buffer memory.



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.



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



4.48 ps4000aSetSigGenBuiltIn

```
PICO_STATUS ps4000aSetSigGenBuiltIn (
  int16_t                    handle,
  int32_t                    offsetVoltage, // see note 1
  uint32_t                   pkToPk, // see note 1
  PS4000A_WAVE_TYPE          waveType, // see note 1
  double                     startFrequency,
  double                     stopFrequency,
  double                     increment,
  double                     dwellTime,
  PS4000A_SWEEP_TYPE         sweepType,
  PS4000A_EXTRA_OPERATIONS   operation, // see note 1
  uint32_t                   shots,
  uint32_t                   sweeps,
  PS4000A_SIGGEN_TRIG_TYPE   triggerType,
  PS4000A_SIGGEN_TRIG_SOURCE triggerSource,
  int16_t                    extInThreshold
)
```

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

Note 1: in general, this function can be called with new arguments while waiting for a trigger; the exceptions are the arguments `offsetVoltage`, `pkToPk`, `arbitraryWaveform`, `arbitraryWaveformSize` and `operation`, which must unchanged on subsequent calls, otherwise the function will return a `PICO_BUSY` status code.

Applicability	All modes
Arguments	`handle`, the handle of the required oscilloscope. `offsetVoltage`, the voltage offset, in microvolts, to be applied to the waveform. `pkToPk`, the peak-to-peak voltage, in microvolts, of the waveform signal. `waveType`, the type of waveform to be generated by the oscilloscope: `PS4000A_SINE` sine wave `PS4000A_SQUARE` square wave `PS4000A_TRIANGLE` triangle wave `PS4000A_RAMP_UP` rising sawtooth `PS4000A_RAMP_DOWN` falling sawtooth `PS4000A_SINC` sin(x)/x `PS4000A_GAUSSIAN` normal distribution `PS4000A_HALF_SINE` full-wave rectified sinusoid `PS4000A_DC_VOLTAGE` DC voltage `PS4000A_WHITE_NOISE` random values `startFrequency`, the frequency in hertz at which the signal generator should begin. Range: `MIN_SIG_GEN_FREQ` to `MAX_SIG_GEN_FREQ`.

	stopFrequency, the frequency in hertz at which the sweep should reverse direction or return to the start frequency. Range: MIN_SIG_GEN_FREQ to MAX_SIG_GEN_FREQ. increment, the amount in hertz by which the frequency rises or falls every dwellTime seconds in sweep mode. dwellTime, the time in seconds between frequency changes in sweep mode. sweepType, see ps4000aSetSigGenArbitrary operation, see ps4000aSetSigGenArbitrary shots, see ps4000aSigGenArbitrary sweeps, see ps4000aSigGenArbitrary triggerType, see ps4000aSigGenArbitrary triggerSource, see ps4000aSigGenArbitrary extInThreshold, see ps4000aSigGenArbitrary
Returns	0: if successful. Error code: if failed.

4.49 ps4000aSetSigGenPropertiesArbitrary

```
PICO_STATUS ps4000aSetSigGenPropertiesArbitrary (
  int16_t                     handle,
  uint32_t                    startDeltaPhase,
  uint32_t                    stopDeltaPhase,
  uint32_t                    deltaPhaseIncrement,
  uint32_t                    dwellCount,
  PS4000A_SWEEP_TYPE          sweepType,
  uint32_t                    shots,
  uint32_t                    sweeps,
  PS4000A_SIGGEN_TRIG_TYPE    triggerType,
  PS4000A_SIGGEN_TRIG_SOURCE  triggerSource,
  int16_t                     extInThreshold
)
```

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

Applicability	All modes
Arguments	See ps4000SetSigGenArbitrary
Returns	0: if successful. Error code: if failed

4.50 ps4000aSetSigGenPropertiesBuiltIn

```
PICO_STATUS ps4000aSetSigGenPropertiesBuiltIn (
  int16_t                    handle,
  double                     startFrequency,
  double                     stopFrequency,
  double                     increment,
  double                     dwellTime,
  PS4000A_SWEEP_TYPE         sweepType,
  uint32_t                   shots,
  uint32_t                   sweeps,
  PS4000A_SIGGEN_TRIG_TYPE   triggerType,
  PS4000A_SIGGEN_TRIG_SOURCE triggerSource,
  int16_t                    extInThreshold
)
```

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

Applicability	All modes
Arguments	See ps4000SetSigGenBuiltIn
Returns	0: if successful. Error code: if failed

4.51 ps4000aSetSimpleTrigger

```
PICO_STATUS ps4000aSetSimpleTrigger
(
  int16_t                         handle,
  int16_t                         enable,
  PS4000A_CHANNEL                 source,
  int16_t                         threshold,
  PS4000A_THRESHOLD_DIRECTION     direction,
  uint32_t                        delay,
  int16_t                         autoTrigger_ms
)
```

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

Applicability	All modes
Arguments	handle, the handle of the required device. enabled, zero to disable the trigger, any non-zero value to set the trigger. source, the channel on which to trigger. See ps4000aSetChannel. 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, in sample periods, between the trigger occurring and the first sample being taken. autoTrigger_ms, the number of milliseconds the device will wait if no trigger occurs.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_DRIVER_FUNCTION

4.52 ps4000aSetTriggerChannelConditions

```
PICO_STATUS ps4000aSetTriggerChannelConditions
(
  int16_t                    handle,
  PS4000A_CONDITION       * conditions,
  int16_t                    nConditions,
  PS4000A_CONDITIONS_INFO    info
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is set up by defining an array of one or more PS4000A_CONDITION structures that are then ANDed together. The function can be called multiple times, in which case the trigger logic is ORed with that defined by previous calls. 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 PS4000A_CONDITION 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 AND of all the elements. nConditions, the number of elements in the conditions array, or zero to switch off triggering. info, determines whether the function clears previous conditions: PS4000A_CLEAR, clears previous conditions PS4000A_ADD, adds the specified conditions (ORing them with previously set conditions, if any) You can combine both actions by passing (PS4000A_CONDITIONS_INFO)(PS4000A_CLEAR | PS4000A_ADD).
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_MEMORY_FAIL

4.52.1 PS4000A_CONDITION structure

A structure of this type is passed to ps4000aSetPulseWidthQualifier and ps4000SetTriggerChannelConditions in the conditions argument to specify the trigger conditions, and is defined as follows: -

```
typedef struct tPS4000ACondition
{
  PS4000A_CHANNEL       source;
  PS4000A_TRIGGER_STATE condition;
} PS4000A_CONDITION;
```

The above-mentioned functions can OR together a number of these structures to produce the final trigger condition or pulse width qualifier, which can be any possible Boolean function of the scope's inputs.

Elements	source, the input to the trigger or pulse width qualifier. See ps4000aSetChannel for values. condition, the type of condition that should be applied to each channel. Use any these constants: CONDITION_DONT_CARE CONDITION_TRUE CONDITION_FALSE The channels that are set to CONDITION_TRUE or CONDITION_FALSE must all meet their conditions simultaneously to produce a trigger. Channels set to CONDITION_DONT_CARE are ignored.

4.53 ps4000aSetTriggerChannelDirections

```
PICO_STATUS ps4000aSetTriggerChannelDirections
(
  int16_t             handle,
  PS4000A_DIRECTION * directions,
  int16_t             nDirections
)
```

This function sets the direction of the trigger for the specified channels.

Applicability	All modes.
Arguments	`handle`, the handle of the required device. `* directions`, on entry, an array of structures containing trigger directions. See `PS4000A_DIRECTION` for allowable values. `nDirections`, the length of the `directions` array.

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

4.53.1 PS4000A_DIRECTION structure

A structure of this type is passed to ps4000aSetTriggerChannelDirections in the directions argument to specify the trigger direction for a specified source, and is defined as follows: -

```
typedef struct tPS4000ADirection
{
  PS4000A_CHANNEL             channel;
  PS4000A_THRESHOLD_DIRECTION direction;
} PS4000A_DIRECTION;
```

Each structure is the logical AND of the states of the scope's inputs. The ps4000aSetTriggerChannelDirections 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.

Elements

channel, the channel being configured. See ps4000aSetChannel for allowable values.

direction, the trigger direction that should be applied to each channel. Use one of these constants:

Constant	Type	Direction
PS4000A_ABOVE	gated	above the upper threshold
PS4000A_ABOVE_LOWER	gated	above the lower threshold
PS4000A_BELOW	gated	below the upper threshold
PS4000A_BELOW_LOWER	gated	below the lower threshold
PS4000A_RISING	threshold	rising edge, using upper threshold
PS4000A_RISING_LOWER	threshold	rising edge, using lower threshold
PS4000A_FALLING	threshold	falling edge, using upper threshold
PS4000A_FALLING_LOWER	threshold	falling edge, using lower threshold
PS4000A_RISING_OR_FALLING	threshold	either edge
PS4000A_INSIDE	window-qualified	inside window
PS4000A_OUTSIDE	window-qualified	outside window
PS4000A_ENTER	window	entering the window
PS4000A_EXIT	window	leaving the window
PS4000A_ENTER_OR_EXIT	window	either entering or leaving the window
PS4000A_POSITIVE_RUNT	window-qualified	entering and leaving from below
PS4000A_NEGATIVE_RUNT	window-qualified	entering and leaving from above
PS4000A_NONE	none	none

4.54 ps4000aSetTriggerChannelProperties

```
PICO_STATUS ps4000aSetTriggerChannelProperties
(
  int16_t                                  handle,
  PS4000A_TRIGGER_CHANNEL_PROPERTIES * channelProperties,
  int16_t                                  nChannelProperties,
  int16_t                                  auxOutputEnable,
  int32_t                                  autoTriggerMilliseconds
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device. `* channelProperties`, an array of `PS4000A_TRIGGER_CHANNEL_PROPERTIES` structures describing the requested properties. The array can contain a single element describing the properties of one channel or a number of elements describing several channels. If `NULL` is passed, triggering is switched off. `nChannelProperties`, the size of the `channelProperties` array. If zero, triggering is switched off. `auxOutputEnable`, not used `autoTriggerMilliseconds`, the time in milliseconds for which the scope device will wait before collecting data if no trigger event occurs. If this is set to zero, the scope device will wait indefinitely for a trigger.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_TRIGGER_ERROR`
	`PICO_MEMORY_FAIL`
	`PICO_INVALID_TRIGGER_PROPERTY`

4.54.1 PS4000A_TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to ps4000aSetTriggerChannelProperties in the channelProperties argument to specify the trigger mechanism, and is defined as follows:

```
typedef struct tPS4000ATriggerChannelProperties
{
  int16_t                 thresholdUpper;
  uint16_t                thresholdUpperHysteresis;
  int16_t                 thresholdLower;
  uint16_t                thresholdLowerHysteresis;
  PS4000A_CHANNEL         channel;
  PS4000A_THRESHOLD_MODE  thresholdMode;
} PS4000A_TRIGGER_CHANNEL_PROPERTIES
```

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. See ps4000aSetChannel for possible values. thresholdMode, either a level or window trigger. Use one of these constants: PS4000A_LEVEL PS4000A_WINDOW

4.55 ps4000aSetTriggerDelay

```
PICO_STATUS ps4000aSetTriggerDelay
(
  int16_t  handle,
  uint32_t delay
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device `delay`, the time between the trigger occurring and the first sample, in sample periods. For example, if `delay = 100` then the scope would wait 100 sample periods before sampling. Example: with the PicoScope 4824, at a timebase of 80 MS/s, or 12.5 ns per sample `(timebase = 0)` the total delay would then be 100 x 12.5 ns = 1.25 µs.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`

4.56 ps4000aSigGenSoftwareControl

```
PICO_STATUS ps4000aSigGenSoftwareControl
(
  int16_t handle,
  int16_t state
)
```

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

Applicability	Use with ps4000aSetSigGenBuiltIn or ps4000aSetSigGenArbitrary.
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

4.57 ps4000aStop

```
PICO_STATUS ps4000aStop
(
  int16_t  handle
)
```

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

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`

4.58 ps4000aStreamingReady

```
typedef void (CALLBACK *ps4000aStreamingReady)
(
  int16_t    handle,
  int32_t    noOfSamples,
  uint32_t   startIndex,
  int16_t    overflow,
  uint32_t   triggerAt,
  int16_t    triggered,
  int16_t    autoStop,
  void     * pParameter
)
```

This callback function is part of your application. You register it with the PicoScope 4000 Series driver using ps4000aGetStreamingLatestValues, and the driver calls it back when streaming-mode data is ready. You can then download the data using the ps4000aGetValuesAsync 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 ps4000aSetDataBuffer. `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 ps4000aRunStreaming. `pParameter`, a void pointer passed from ps4000aGetStreamingLatestValues. The callback function can write to this location to send any data, such as a status flag, back to the application.
Returns	nothing

5 Enumerated types and constants

Enumerated types and constants are defined in the file `ps4000aApi.h`, which is included in the SDK. We recommend that you refer to these constants by name unless your programming environment understands only numeric values.

6 Driver status codes

Every function in the ps4000a.dll driver returns a status code from the list of PICO_STATUS values defined in the picoStatus.h header file supplied with the SDK. See the header file for more information.

7 Programming examples

The SDK includes programming examples for a number of languages and development environments.

7.1 C

The SDK includes a console-mode program (ps4000acon.c) that demonstrates how to use the PicoScope 4000 driver in Windows. The program demonstrates the following procedures:

- Open a PicoScope 4000 Series oscilloscope
- Collect a block of samples immediately
- Collect a block of samples when a trigger event occurs
- Collect a stream of data immediately
- Collect a stream of data when a trigger event occurs

To build this application:

- Set up a project for a 32-bit or 64-bit console mode application
- Add ps4000acon.c to the project
- Add ps4000a.lib to the project (Microsoft C only)
- Add ps4000aApi.h and picoStatus.h to the project
- Build the project

7.2 Excel

The Excel example demonstrates how to capture data in Excel from a PicoScope 4000 Series scope.

1. Copy the following files (for a 32-bit Windows system) from the SDK to a location that is on your Windows execution path (for example, C:\windows\system32):
   - ps4000aWrap.dll
   - ps4000a.dll
   - PicoIpp.dll
2. Load the spreadsheet ps4000a.xls
3. Select **Tools | Macro**
4. Select **GetData**
5. 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.

7.3 LabVIEW

The SDK contains a library of VIs that can be used to control the PicoScope 4000 and some simple examples of using these VIs in streaming mode, block mode and rapid block mode.

The LabVIEW library (PicoScope4000a.llb) can be placed in the user.lib subdirectory to make the VIs available on the 'User Libraries' palette. You must also copy ps4000a.dll and ps4000aWrap.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

- PicoScope4000aAdvancedTriggerSettings.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 PicoScope4000aSettings.vi.

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

  ps4000aSetTriggerChannelConditions
  ps4000aSetTriggerChannelDirections
  ps4000aSetTriggerChannelProperties
  ps4000aSetPulseWidthQualifierConditions
  ps4000aSetPulseWidthQualifierProperties
  ps4000aSetTriggerDelay

- PicoScope4000aAWG.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 ps4000aSetSigGenBuiltIn and ps4000aSetSigGenArbitrary.

- PicoScope4000aClose.vi - closes the oscilloscope

  Should be called before exiting an application.

- PicoScope4000aGetBlock.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 PicoScope4000aSettings.vi. The VI outputs data arrays in two clusters (max and min). If not using aggregation, 'Min Buffers' is not used.

- PicoScope4000aGetRapidBlock.vi - collects a set of data blocks or captures from the oscilloscope in rapid block mode

  This VI is similar to PicoScope4000aGetBlock.vi. It outputs two-dimensional arrays for each channel that contain data from all the requested number of captures.

- `PicoScope4000aGetStreamingValues.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 `PicoScope4000aSettings.vi` and streaming has been started by calling `PicoScope4000aStartStreaming.vi`. The VI outputs the number of samples available and the start index of these samples in the array output by `PicoScope4000aStartStreaming.vi`.

- `PicoScope4000aOpen.vi` - opens a PicoScope 4000 and returns a handle to the device

- `PicoScope4000aSettings.vi` - sets up the oscilloscope

  The inputs are clusters for setting up channels and simple triggers. Advanced triggers can be set up using `PicoScope4000aAdvancedTriggerSettings.vi`.

- `PicoScope4000aStartStreaming.vi` - starts the oscilloscope streaming

  It outputs arrays that will contain samples once `PicoScope4000aGetStreamingValues.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.

8 Glossary

AC/DC switch. To switch from AC coupling to DC coupling, or vice versa, select AC or DC from the control on the PicoScope toolbar. The AC setting filters out very low-frequency components of the input signal, including DC, and is suitable for viewing small AC signals superimposed on a DC or slowly changing offset. In this mode you can measure the peak-to-peak amplitude of an AC signal but not its absolute value. Use the DC setting for measuring the absolute value of a signal.

ADC. Analog-to-digital converter. The electronic component in a PC oscilloscope that converts analog signals from the inputs into digital data suitable for transmission to the PC.

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. Choose this mode of operation when the input signal being sampled contains high frequencies. Note: To avoid sampling errors, the maximum input frequency must be less than half the sampling rate.

Buffer size. The size of the oscilloscope buffer memory, measured in samples. The buffer allows the oscilloscope to sample data faster than it can transfer it to the computer.

Callback. A mechanism that the PicoScope 4000 driver uses to communicate asynchronously with your application. At design time, you add a function (a *callback* function) to your application to deal with captured data. At run time, when you request captured data from the driver, you also pass it a pointer to your function. The driver then returns control to your application, allowing it to perform other tasks until the data is ready. When this happens, the driver calls your function in a new thread to signal that the data is ready. It is then up to your function to communicate this fact to the rest of your application.

Device Manager. Device Manager is a Windows program that displays the current hardware configuration of your computer. On Windows XP or Vista, right-click 'My Computer,' choose 'Properties', then click the 'Hardware' tab and the 'Device Manager' button.

Driver. A program that controls a piece of hardware. The driver for the PicoScope 4000 Series PC Oscilloscopes is supplied in the form of a 32-bit Windows DLL, `ps4000.dll`. This is used by the PicoScope software, and by user-designed applications, to control the oscilloscopes.

ETS. Equivalent-time sampling. A technique for increasing the effective sampling rate of an oscilloscope beyond the maximum sampling rate of its ADC. The scope triggers on successive cycles of a repetitive waveform and collects one sample from each cycle. Each sample is delayed relative to the trigger by a time that increases with each cycle, so that after a number of cycles a complete period of the waveform has been sampled. The waveform must be stable and repetitive for this method to work.

GS/s. Gigasample (billion samples) per second.

Maximum sampling rate. A figure indicating the maximum number of samples the oscilloscope can acquire per second. The higher the sampling rate of the oscilloscope, the more accurate the representation of the high-frequency details in a fast signal.

MS/s. Megasample (million samples) per second.

PC Oscilloscope. A virtual instrument formed by connecting a PicoScope 4000 Series scope unit to a computer running the PicoScope software.

PicoScope 4000 Series. A range of high-resolution PC Oscilloscopes from Pico Technology. The range includes two-channel and four-channel models, with or without a built-in function generator and arbitrary waveform generator.

PicoScope software. A software product that accompanies all Pico PC Oscilloscopes. It turns your PC into an oscilloscope, spectrum analyser, and meter display.

Streaming mode. A sampling mode in which the oscilloscope samples data and returns it to the computer in an unbroken stream. This mode allows the capture of data sets whose size is not limited by the size of the scope's memory buffer, at sampling rates up to 160 million samples per second.

Timebase. The timebase controls the time interval that each horizontal division of a scope view represents. There are ten divisions across the scope view, so the total time across the view is ten times the timebase per division.

Trigger bandwidth. The external trigger input is less sensitive to very high-frequency input signals than to low-frequency signals. The trigger bandwidth is the frequency at which a trigger signal will be attenuated by 3 decibels.

USB 1.1. Universal Serial Bus (Full Speed). This is a standard port used to connect external devices to PCs. The maximum signalling rate is 12 megabits per second, so is much faster than an RS-232 (COM) port.

USB 2.0. Universal Serial Bus (Hi-Speed). The maximum signalling rate is 480 megabits per second.

USB 3.0. Universal Serial Bus (SuperSpeed). The maximum signalling rate is 5 gigabits per second.

Vertical resolution. A value, in bits, indicating the precision with which the oscilloscope converts input voltages to digital values.

Voltage range. The range of input voltages that the oscilloscope can measure. For example, a voltage range of ±100 mV means that the oscilloscope can measure voltages between –100 mV and +100 mV. Input voltages outside this range will not damage the instrument as long as they remain within the protection limits of ±200 V.


pico
Technology

Index

A

AC/DC coupling 6
  setting 59
Aggregation 7, 15
  getting ratio 29
Analog offset 27
API function calls 18
Arbitrary waveform generator 68
  index modes 71
AWG 68
AWG index modes 71

B

Bandwidth-limiting filter 58
Block mode 6, 8, 9, 20
  polling status 45
  starting 54
Buffers
  overrun 5

C

C programming 89
Callback function
  block mode 20
  streaming mode 24, 86
Channel information, reading 28
Channel selection 6
  settings 59
Closing a scope device 23
Company information 3
Constants 87
Contact details 3

D

Data acquisition 15
Data buffers, setting 60, 61
Decimation 7
Disk space 3
Downsampling 7
Driver 5
  status codes 88

E

Enumerated types 87
Enumerating oscilloscopes 25
Excel 89

F

Filter, bandwidth-limiting 58
Function calls 18
Functions
  ps4000aBlockReady 20
  ps4000aChangePowerSource 21
  ps4000aCloseUnit 23
  ps4000aCurrentPowerSource 22
  ps4000aDataReady 24
  ps4000aEnumerateUnits 25
  ps4000aFlashLed 26
  ps4000aGetAnalogueOffset 27
  ps4000aGetChannelInformation 28
  ps4000aGetMaxDownSampleRatio 29
  ps4000aGetMaxSegments 30
  ps4000aGetNoOfCaptures 31
  ps4000aGetNoOfProcessedCaptures 32
  ps4000aGetStreamingLatestValues 33
  ps4000aGetTimebase 34
  ps4000aGetTimebase2 35
  ps4000aGetTriggerTimeOffset 36
  ps4000aGetTriggerTimeOffset64 37
  ps4000aGetUnitInfo 38
  ps4000aGetValues 39
  ps4000aGetValuesAsync 40
  ps4000aGetValuesBulk 41
  ps4000aGetValuesOverlapped 42
  ps4000aGetValuesOverlappedBulk 43
  ps4000aIsLedFlashing 44
  ps4000aIsReady 45
  ps4000aIsTriggerOrPulseWidthQualifierEnabled 46
  ps4000aMaximumValue 47
  ps4000aMemorySegments 49
  ps4000aMinimumValue 48
  ps4000aNoOfStreamingValues 50
  ps4000aOpenUnit 51
  ps4000aOpenUnitAsync 52
  ps4000aOpenUnitProgress 53
  ps4000aRunBlock 54
  ps4000aRunStreaming 56
  ps4000aSetBandwidthFilter 58
  ps4000aSetChannel 59
  ps4000aSetDataBuffer 60
  ps4000aSetDataBuffers 61
  ps4000aSetEts 62
  ps4000aSetEtsTimeBuffer 63
  ps4000aSetEtsTimeBuffers 64
  ps4000aSetNoOfCaptures 65

Functions
ps4000aSetPulseWidthQualifierConditions 66
ps4000aSetPulseWidthQualifierProperties 67
ps4000aSetSigGenArbitrary 68
ps4000aSetSigGenBuiltIn 72
ps4000aSetSigGenPropertiesArbitrary 74
ps4000aSetSigGenPropertiesBuiltIn 75
ps4000aSetSimpleTrigger 76
ps4000aSetTriggerChannelConditions 77
ps4000aSetTriggerChannelDirections 79
ps4000aSetTriggerChannelProperties 81
ps4000aSetTriggerDelay 83
ps4000aSigGenSoftwareControl 84
ps4000aStop 85
ps4000aStreamingReady 86

H

Hysteresis 82

I

IEPE mode 59
Installation 4

L

LabVIEW 89
LED
programming 26, 44

M

Memory in scope 8
Memory segments 49
Multi-unit operation 17

O

Opening a unit 51, 52, 53
Operating system 3

P

Pico Technical Support 3
PICO_STATUS enum type 88
picopp.inf 5
picopp.sys 5
PicoScope 4000 Series 1
PicoScope software 4, 5
Power source 21, 22
Processor 3
Programming
C 89
Excel 89
LabVIEW 89
PS4000A_CHANNEL constants 59
PS4000A_CONDITION structure 78
PS4000A_DIRECTION structure 80
PS4000A_LEVEL 82
PS4000A_LOST_DATA 5
PS4000A_MAX_VALUE_16BIT 5
PS4000A_MAX_VALUE_8BIT 5
PS4000A_MIN_VALUE_16BIT 5
PS4000A_MIN_VALUE_8BIT 5
PS4000A_THRESHOLD_DIRECTION constants 80
PS4000A_THRESHOLD_MODE constants 82
PS4000A_TRIGGER_CHANNEL_PROPERTIES structure 82
PS4000A_TRIGGER_STATE constants 78
PS4000A_WINDOW 82
Pulse width trigger 66, 67

R

Rapid block mode 10
Retrieving data 39, 40
block mode, deferred 42
rapid block mode, deferred 43
stored (API) 16
streaming mode 33

S

Sampling rate
maximum 8
Scaling 5
Segments
maximum number 30
Serial numbers 25
Signal generator 9
arbitrary waveforms 68
built-in waveforms 72
software trigger 84
Software licence conditions 2
Status codes 88
Stopping sampling 85
Streaming mode 8, 15
getting number of values 50
retrieving data 33
starting 56
using (API) 15
Summation (downsampling mode) 7
Synchronising units 17
System memory 3
System requirements 3

T

Technical support 3
Threshold voltage 6
Timebase 17
  setting 34, 35
Trademarks 2
Trigger 6
  conditions 77
  delay 83
  directions 79, 80
  pulse width qualifier 46, 66, 67
  pulse width qualifier conditions 78
  setting up 76
  time offset 36, 37

U

USB 3
  changing ports 4
  hub 17

V

Voltage ranges 5

W

Windows, Microsoft 3


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

ps4000apg.en r1 2014-02-17