



PicoScope® 4000 Series

PC Oscilloscopes

Programmer's Guide



Contents

1	Introduction	1
1	Welcome	1
2	Software license conditions	1
3	Trademarks	2
4	System requirements	2
2	Programming with the PicoScope 4000 Series	3
1	Driver	3
2	Voltage ranges	4
3	Channel selection	5
4	Triggering	5
5	Oversampling	6
6	Timebases	7
3	Sampling modes	8
1	Block mode	8
1	Using block mode	9
2	Rapid block mode	10
1	Using rapid block mode	10
2	Rapid block mode example 1: no aggregation	11
3	Rapid block mode example 2: using aggregation	13
3	ETS (Equivalent Time Sampling)	14
1	Using ETS mode	15
4	Streaming mode	15
1	Using streaming mode	16
5	Retrieving stored data	17
4	Combining several oscilloscopes	18
5	API functions	19
1	ps4000BlockReady	21
2	ps4000CloseUnit	22
3	ps4000DataReady	23
4	ps4000EnumerateUnits	24
5	ps4000FlashLed	25
6	ps4000GetChannelInformation	26
7	ps4000GetMaxDownSampleRatio	27
8	ps4000GetNoOfCaptures	28
9	ps4000GetStreamingLatestValues	29
10	ps4000GetTimebase	30
11	ps4000GetTimebase2	31
12	ps4000GetTriggerChannelTimeOffset	32
13	ps4000GetTriggerChannelTimeOffset64	34
14	ps4000GetTriggerTimeOffset	35
15	ps4000GetTriggerTimeOffset64	37

16	ps4000GetUnitInfo	38
17	ps4000GetValues	39
18	ps4000GetValuesAsync	41
19	ps4000GetValuesBulk	42
20	ps4000GetValuesTriggerChannelTimeOffsetBulk	43
21	ps4000GetValuesTriggerChannelTimeOffsetBulk64	44
22	ps4000GetValuesTriggerTimeOffsetBulk	45
23	ps4000GetValuesTriggerTimeOffsetBulk64	47
24	ps4000HoldOff	48
25	ps4000IsLedFlashing	49
26	ps4000IsReady	50
27	ps4000IsTriggerOrPulseWidthQualifierEnabled	51
28	ps4000MemorySegments	52
29	ps4000NoOfStreamingValues	53
30	ps4000OpenUnit	54
31	ps4000OpenUnitAsync	55
32	ps4000OpenUnitAsyncEx	56
33	ps4000OpenUnitEx	57
34	ps4000OpenUnitProgress	58
35	ps4000PingUnit	59
36	ps4000RunBlock	60
37	ps4000RunStreaming	62
38	ps4000RunStreamingEx	64
39	ps4000SetBwFilter	66
40	ps4000SetChannel	67
41	ps4000SetDataBuffer	69
42	ps4000SetDataBufferBulk	70
43	ps4000SetDataBuffers	71
44	ps4000SetDataBuffersWithMode	72
45	ps4000SetDataBufferWithMode	73
46	ps4000SetEts	74
47	ps4000SetEtsTimeBuffer	75
48	ps4000SetEtsTimeBuffers	76
49	ps4000SetExtTriggerRange	77
50	ps4000SetNoOfCaptures	78
51	ps4000SetPulseWidthQualifier	79
	1 PWQ_CONDITIONS structure	80
52	ps4000SetSigGenArbitrary	81
	1 AWG index modes	84
	2 Calculating deltaPhase	84
53	ps4000SetSigGenBuiltIn	86
54	ps4000SetSimpleTrigger	88
55	ps4000SetTriggerChannelConditions	89
	1 TRIGGER_CONDITIONS structure	90
56	ps4000SetTriggerChannelDirections	91

57	ps4000SetTriggerChannelProperties	92
	1 TRIGGER_CHANNEL_PROPERTIES structure	93
58	ps4000SetTriggerDelay	94
59	ps4000SigGenArbitraryMinMaxValues	95
60	ps4000SigGenFrequencyToPhase	96
61	ps4000SigGenSoftwareControl	97
62	ps4000Stop	98
63	ps4000StreamingReady	99
64	Wrapper functions	100
6	Further information	102
	1 Programming examples	102
	2 Driver status codes	102
	3 Enumerated types and constants	102
	4 Numeric data types	102
7	Glossary	103
	Index	105



1 Introduction

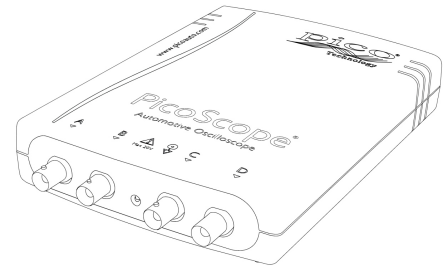
1.1 Welcome

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 manual explains how to use the Application Programming Interface (API) for the PicoScope 4000 Series scopes. The API supports the following models:

- PicoScope 4224 and 4424 12-bit oscilloscopes with optional IEPE interface
- PicoScope 4262 16-bit high-resolution oscilloscope
- PicoScope 4226 and 4227 oscilloscopes (discontinued)

For more information on the hardware, see the *PicoScope 4000 Series User's Guide*, available as a separate manual.



1.2 Software license conditions

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

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

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

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

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

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

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

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

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

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

1.3 Trademarks

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

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

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

1.4 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 7, Windows 8 or Windows 10 32 bit and 64 bit versions supported
Processor	As required by the operating system
Memory	
Free disk space	
Ports	USB 3.0 compliant port USB 2.0 compliant port USB 1.1 compliant port (not recommended)

Using with custom applications

Drivers are available for the operating systems mentioned above.

2 Programming with the PicoScope 4000 Series

The `ps4000.dll` dynamic link library in the `lib` subdirectory of the Pico Technology SDK 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.
- Copy data to a buffer.
- Stop capturing data.
- Close the scope unit.

Numerous [sample programs](#) are installed with the SDK. These show how to use the functions of the driver software in each of the modes available.

2.1 Driver

Your application will communicate with a PicoScope 4000 API driver called `ps4000.dll`, which is supplied in 32-bit and 64-bit versions. The driver exports the `ps4000` [function definitions](#) in standard C format, but this does not limit you to programming in C. You can use the API with any programming language that supports standard C calls.

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

2.2 Voltage ranges

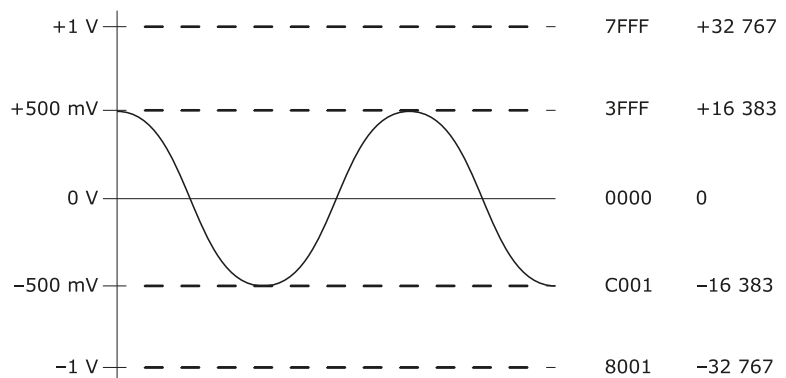
The [ps4000SetChannel](#) function allows you to set the voltage range of each input channel of the scope. Each device in the PicoScope 4000 Series has its own set of voltage ranges described in its data sheet. Each sample is normalized to 16 bits resulting in values returned to your application as follows:

Constant	Voltage	Value returned	
		decimal	hex
PS4000_MIN_VALUE or PS4262_MIN_VALUE	minimum	-32 764	8004
N/A	zero	0	0000
PS4000_MAX_VALUE or PS4262_MAX_VALUE	maximum	32 764	7FFC
PS4000_LOST_DATA	Note 1	-32 768	8000

1. In [streaming mode](#), this special value indicates a buffer overrun.

Example, using the PicoScope 4262

1. Call [ps4000SetChannel](#) with `range` set to `PS4000_1V`.
2. Apply a sine wave input of 500 mV amplitude to the oscilloscope.
3. Capture some data using the desired [sampling mode](#).
4. The data will be encoded as shown opposite.



External trigger input

The PicoScope 4226, 4227 and 4262 models have an external trigger input (marked EXT). The external trigger input is scaled to a 16-bit value as follows. Note that the PicoScope 4262 has two external trigger input voltage ranges.

PicoScope 4262

Voltage	Constant	Digital value
-500 mV	PS4000_EXT_MIN_VALUE	-32 767
0 V		0
+500 mV	PS4000_EXT_MAX_VALUE	+32 767

Voltage	Constant	Digital value
-5 V	PS4000_EXT_MIN_VALUE	-32 767
0 V		0
+5 V	PS4000_EXT_MAX_VALUE	+32 767

PicoScope 4226 and 4227

Voltage	Constant	Digital value
-20 V	PS4000_EXT_MIN_VALUE	-32 767
0 V		0
+20 V	PS4000_EXT_MAX_VALUE	+32 767

2.3 Channel selection

You can switch each channel on and off, and set its coupling mode to either AC or DC, using the [ps4000SetChannel](#) 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.

2.4 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 trigger function [ps4000SetSimpleTrigger](#), which in turn calls:

- [ps4000SetTriggerChannelConditions](#)
- [ps4000SetTriggerChannelDirections](#)
- [ps4000SetTriggerChannelProperties](#)

These can also be called individually, rather than using [ps4000SetSimpleTrigger](#), in order to set up advanced trigger types such as pulse width.

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

The driver supports these triggering methods:

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

The pulse width, delay and drop-out triggering methods additionally require the use of the pulse width qualifier function, [ps4000SetPulseWidthQualifier](#).

2.5 Oversampling

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

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

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

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

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

2.6 Timebases

The API allows you to select one of 2^{30} different timebases related to the maximum sampling rate of the oscilloscope. The timebases allow slow enough sampling in block mode to overlap the streaming mode sampling intervals, so that you can make a smooth transition between block mode and streaming mode.

For all PicoScope 4000 Series scopes except the PicoScope 4262, the range of timebase values is divided into "low" and "high" subranges, with the low subrange specifying a power of 2 and the high subrange specifying a fraction of the clock frequency. The PicoScope 4262 has a single range of timebases specifying a fraction of the clock frequency.

Timebase (n)	Sampling interval (t)		
	PicoScope 4223 PicoScope 4224 PicoScope 4423 PicoScope 4424	PicoScope 4226 PicoScope 4227	PicoScope 4262
Low	$2^n / 80,000,000$ n=0: 12.5 ns n=1: 25 ns n=2: 50 ns	$2^n / 250,000,000$ n=0*: 4 ns n=1: 8 ns n=2: 16 ns n=3: 32 ns	$(n+1) / 10,000,000$ n=0: 100 ns n=1: 200 ns n=2: 300 ns ... n= $2^{30}-1$: ~107 s
High	$(n-1) / 20,000,000$ n=3: 100 ns n=4: 150 ns n=5: 200 ns ... n= $2^{30}-1$: ~54 s	$(n-2) / 31,250,000$ n=4: 64 ns n=5: 96 ns n=6: 128 ns ... n= $2^{30}-1$: ~34 s	

* PicoScope 4227 only

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

Applicability	Use ps4000GetTimebase API call.
----------------------	---

3 Sampling modes

[PicoScope 4000 Series PC Oscilloscopes](#) can run in various **sampling modes**.

- **Block mode.** In this mode, the scope stores data in its buffer memory and then transfers it to the PC. When the data has been collected it is possible to examine the data, with an optional [aggregation](#) 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 [aggregation](#) in this mode if you wish.
- **Streaming mode.** In this mode, data is passed directly to the PC without being stored in the scope's buffer memory. This enables long periods of slow data collection for chart recorder and data-logging applications. Streaming mode provides fast streaming at up to 6.6 MS/s (150 ns per sample). Aggregation 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 and rapid block mode, you can also poll the driver instead of using a callback.

3.1 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 [ps4000MemorySegments](#)).
- **Sampling rate.** The PicoScope 4000 Series PC Oscilloscopes can sample at a number of different rates according to their model, selected [timebase](#) and the combination of channels that are enabled. The maximum sampling rate can be achieved with a single channel enabled, or with these two-channel combinations: AC, AD, BC and BD. All other combinations limit the maximum sampling rate of scope, as specified in its Data Sheet.

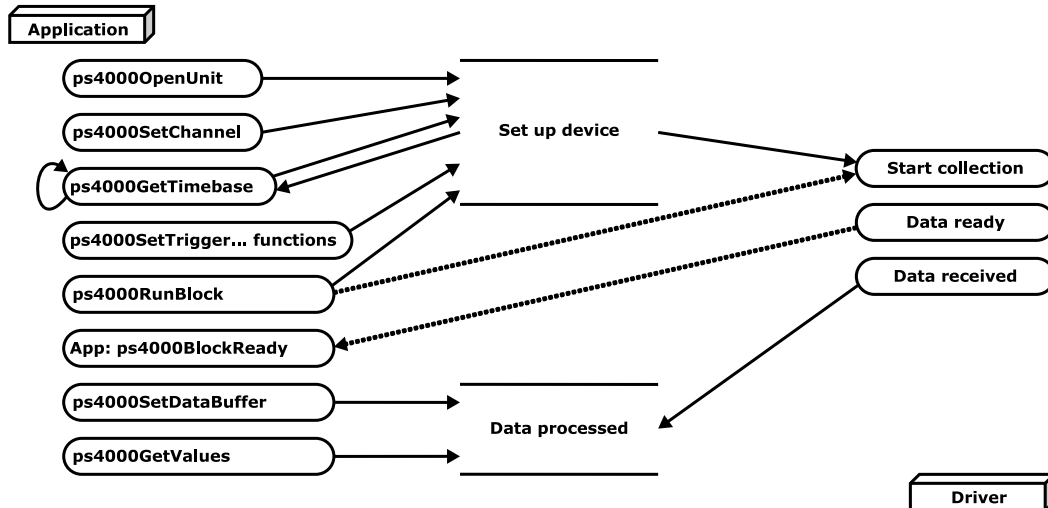
- **Setup time.** The driver normally performs a number of setup operations, which can take up to 50 milliseconds, before collecting each block of data. If you need to collect data with the minimum time interval between blocks, use [rapid block mode](#) and avoid calling setup functions between calls to [ps4000RunBlock](#), [ps4000Stop](#) and [ps4000GetValues](#).
- **Aggregation.** When the data has been collected, you can set an optional [aggregation](#) factor and examine the data. Aggregation is a process that reduces the amount of data by combining adjacent samples using a maximum/minimum algorithm. 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 [ps4000MemorySegments](#).
- **Data retention.** The data is lost when a new run is started in the same segment or the scope is powered down.

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

3.1.1 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 [ps4000OpenUnit](#).
2. Select channel ranges and AC/DC coupling using [ps4000SetChannel](#).
3. Using [ps4000GetTimebase](#), select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions [ps4000SetTriggerChannelConditions](#), [ps4000SetTriggerChannelDirections](#) and [ps4000SetTriggerChannelProperties](#) to set up the trigger if required.
5. Start the oscilloscope running using [ps4000RunBlock](#).
6. Wait until the oscilloscope is ready using the [ps4000BlockReady](#) callback.
7. Use [ps4000SetDataBuffer](#) to tell the driver where your memory buffer is. For greater efficiency with multiple captures, you can call this function outside the loop, after step 4.
8. Transfer the block of data from the oscilloscope using [ps4000GetValues](#).
9. Display the data.
10. Repeat steps 5 to 9.
11. Stop the oscilloscope using [ps4000Stop](#).



12. Request new views of stored data using different aggregation parameters: see [Retrieving stored data](#).
13. Close the device using [ps4000CloseUnit](#).

3.2 Rapid block mode

In normal [block mode](#), the PicoScope 4000 Series scopes collect one waveform at a time. You start the device running, wait until all samples are collected by the device, and then download the data to the PC or start another run. There is a time overhead of tens of milliseconds associated with starting a run, causing a gap between waveforms. When you collect data from the device, there is another minimum time overhead which is most noticeable when using a small number of samples.

Rapid block mode allows you to sample several waveforms at a time with the minimum time between waveforms. It reduces the gap from milliseconds to about 2.5 microseconds, at the fastest timebase. See [Using rapid block mode](#) for details.

3.2.1 Using rapid block mode

You can use [rapid block mode](#) with or without [aggregation](#). The following procedure shows you how to use it without aggregation.

Without aggregation

1. Open the oscilloscope using [ps4000OpenUnit](#).
2. Select channel ranges and AC/DC coupling using [ps4000SetChannel](#).
3. Set the number of memory segments equal to or greater than the number of captures required using [ps4000MemorySegments](#). Use [ps4000SetNoOfCaptures](#) before each run to specify the number of waveforms to capture.
4. Using [ps4000GetTimebase](#), select timebases until the required nanoseconds per sample is located. This will indicate the number of samples per channel available for each segment. If you know that the number of samples you will collect will not exceed the limit, you can call this function after step 5, unless you are using a [pulse width qualifier](#).
5. Use the trigger setup functions [ps4000SetTriggerChannelConditions](#), [ps4000SetTriggerChannelDirections](#) and [ps4000SetTriggerChannelProperties](#) to set up the trigger if required.
6. Start the oscilloscope running using [ps4000RunBlock](#).
7. Wait until the oscilloscope is ready using the [ps4000BlockReady](#) callback.

8. Use [ps4000SetDataBufferBulk](#) to tell the driver where your memory buffers are. Call the function once for each channel/[segment](#) combination for which you require data. For greater efficiency of data retrieval or for multiple captures, you can call this function outside the loop, after step 5.
9. Transfer the blocks of data from the oscilloscope using [ps4000GetValuesBulk](#).
10. Retrieve the time offset for each data segment using [ps4000GetValuesTriggerTimeOffsetBulk64](#).
11. Display the data.
12. Repeat steps 6 to 11 if necessary.
13. Stop the oscilloscope using [ps4000Stop](#).
14. Close the device using [ps4000CloseUnit](#).

With aggregation

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

- 10a. Call [ps4000SetDataBuffers](#) to set up one pair of buffers for every waveform segment required.
- 11a. Call [ps4000GetValues](#) for each pair of buffers.
- 12a. Retrieve the time offset for each data segment using [ps4000GetTriggerTimeOffset64](#).

Continue from step 13 above.

3.2.2 Rapid block mode example 1: no aggregation

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

Set up the device up as usual.

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

```
// Set the number of waveforms to MAX_WAVEFORMS
ps4000SetNoOfCaptures (handle, MAX_WAVEFORMS);

pParameter = false;
ps4000RunBlock
(
    handle,
    0, //noOfPreTriggerSamples,
    10000, // noOfPostTriggerSamples,
    1, // timebase to be used,
    1, // oversample
    &timeIndisposedMs,
    0, // 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 (int32_t i = 0; i < 10; i++)
{
    for (int32_t c = PS4000_CHANNEL_A; c <= PS4000_CHANNEL_D; c++)
    {
        ps4000SetDataBufferBulk
        (
            handle,
            c,
            buffer[c][i],
            MAX_SAMPLES,
            i
        );
    }
}

```

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

```

ps4000GetValuesBulk
(
    handle,
    &noOfSamples, // set to MAX_SAMPLES on entering the function
    10, // fromSegmentIndex,
    19, // toSegmentIndex,
    overflow // an array of size 10 int16_t
)

```

Comments: the number of samples could be up to `noOfPreTriggerSamples + noOfPostTriggerSamples`, the values set in `ps4000RunBlock`. The samples are always returned from the first sample taken, unlike the `ps4000GetValues` function which allows the sample index to be set. This function does not support aggregation. The above segments start at 10 and finish at 19 inclusive. It is possible for the `fromSegmentIndex` to wrap around to the `toSegmentIndex`, by setting the `fromSegmentIndex` to 98 and the `toSegmentIndex` to 7.

```

ps4000GetValuesTriggerTimeOffsetBulk64
(
    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.

3.2.3 Rapid block mode example 2: using aggregation

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

Set up the device up as usual.

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

```
// Set the number of waveforms to MAX_WAVEFORMS
ps4000SetNoOfCaptures (handle, MAX_WAVEFORMS);

pParameter = false;
ps4000RunBlock
(
    handle,
    0, //noOfPreTriggerSamples,
    1000000, // noOfPostTriggerSamples,
    1, // timebase to be used,
    1, // oversample
    &timeIndisposedMs,
    0, //segmentIndex
    lpReady,
    &pParameter
);
```

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

```
for (int32_t c = PS4000_CHANNEL_A; c <= PS4000_CHANNEL_D; c++)
{
    ps4000SetDataBuffers
    (
        handle,
        c,
        bufferMax[c],
        bufferMin[c]
        MAX_SAMPLES,
    );
}
```

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

```
for (int32_t segment = 10; segment < 20; segment++)
{
    ps4000GetValues
    (
        handle,
        0,
        &noOfSamples, // set to MAX_SAMPLES on entering
        1000,
        downSampleRatioMode, //set to RATIO_MODE_AGGREGATE
        index,
        overflow
    );
}
```

```

    );

    ps4000GetTriggerTimeOffset64
    (
        handle,
        &time,
        &timeUnits,
        index
    )
}

```

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

3.3 ETS (Equivalent Time Sampling)

Note: only the PicoScope 4226 and 4227 oscilloscopes support ETS mode.

ETS is a way of increasing the effective sampling rate of the scope when capturing repetitive signals. It is a modified form of [block mode](#), and is controlled by the [ps4000SetTrigger](#) and [ps4000SetEts](#) functions.

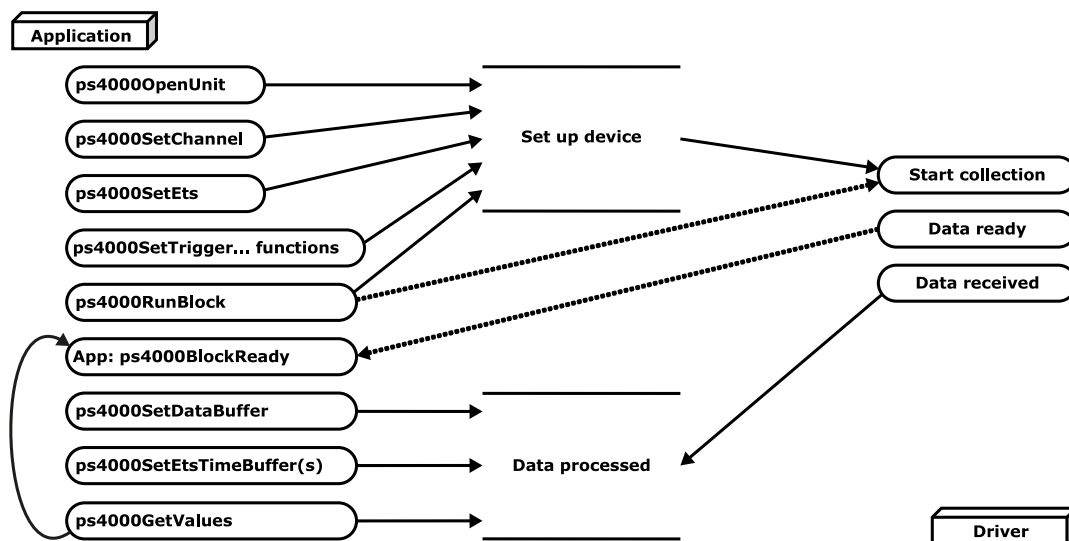
- **Overview.** ETS works by capturing several cycles of a repetitive waveform, then combining them to produce a composite waveform that has a higher effective sampling rate than the individual captures. The maximum effective sampling rates that can be achieved with this method are listed in the User's Guide for the scope device.
- **Trigger stability.** Because of the high sensitivity of ETS mode to small time differences, the trigger must be set up to provide a stable waveform that varies as little as possible from one capture to the next.
- **Callback.** ETS mode returns data to your application using the [ps4000BlockReady](#) callback function.

Applicability	<p>Available in block mode only.</p> <p>Not suitable for one-shot (non-repetitive) signals.</p> <p>Aggregation and oversampling are not supported.</p> <p>Edge-triggering only.</p> <p>Auto trigger delay (<code>autoTriggerMilliseconds</code>) is ignored.</p>
----------------------	--

3.3.1 Using ETS mode

This is the general procedure for reading and displaying data in [ETS mode](#) using a single [memory segment](#):

1. Open the oscilloscope using [ps4000OpenUnit](#).
2. Select channel ranges and AC/DC coupling using [ps4000SetChannel](#).
3. Use [ps4000GetTimebase](#) to verify the number of samples to be collected.
4. Set up ETS using [ps4000SetEts](#).
5. Use the trigger setup functions [ps4000SetTriggerChannelConditions](#), [ps4000SetTriggerChannelDirections](#) and [ps4000SetTriggerChannelProperties](#) to set up the trigger if required.
6. Start the oscilloscope running using [ps4000RunBlock](#).
7. Wait until the oscilloscope is ready using the [ps4000BlockReady](#) callback (or poll using [ps4000IsReady](#)).
8. Use [ps4000SetDataBuffer](#) to tell the driver where to store sampled data.
- 8a. Use [ps4000SetEtsTimeBuffer](#) or [ps4000SetEtsTimeBuffers](#) to tell the driver where to store sample times.
9. Transfer the block of data from the oscilloscope using [ps4000GetValues](#).
10. Display the data.
11. While you want to collect updated captures, repeat steps 7 to 10.
12. Stop the oscilloscope using [ps4000Stop](#).
13. Repeat steps 6 to 12.
14. Close the device driver using [ps4000CloseUnit](#).



3.4 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 6.6 million samples per second (150 nanoseconds per sample), depending on the computer's performance. This makes it suitable for **high-speed data acquisition**, allowing you to capture long data sets limited only by the computer's memory.

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

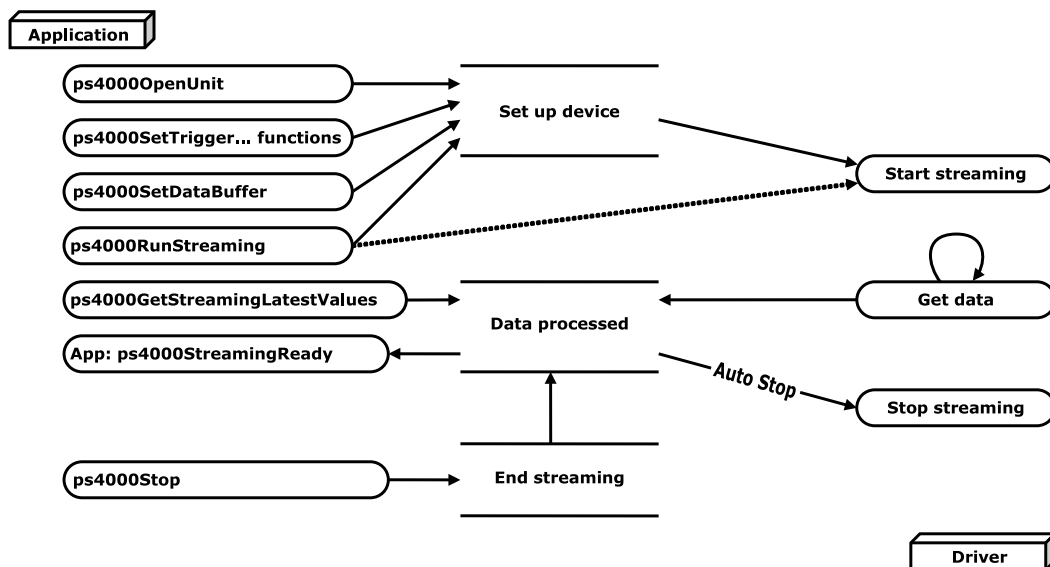
- **Memory segmentation.** The memory can be divided into [segments](#) to reduce the latency of data transfers to the PC. However, this increases the risk of losing data if the PC cannot keep up with the device's sampling rate.

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

3.4.1 Using streaming mode

This is the general procedure for reading and displaying data in [streaming mode](#) using a single [memory segment](#):

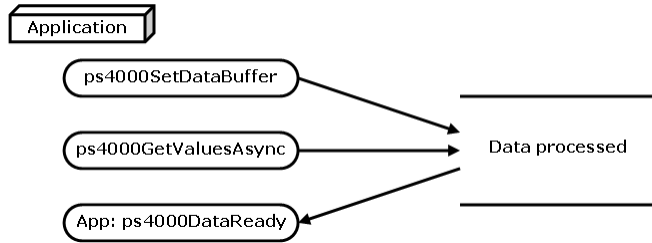
1. Open the oscilloscope using [ps4000OpenUnit](#).
2. Select channels, ranges and AC/DC coupling using [ps4000SetChannel](#).
3. Use the trigger setup functions [\[1\]](#) [\[2\]](#) [\[3\]](#) to set up the trigger if required.
4. Call [ps4000SetDataBuffer](#) to tell the driver where your data buffer is.
5. Set up aggregation and start the oscilloscope running using [ps4000RunStreaming](#).
6. Call [ps4000GetStreamingLatestValues](#) to get data.
7. Process data returned to your application's function. This example is using `autoStop`, so after the driver has received all the data points requested by the application, it stops the device streaming.
8. Call [ps4000Stop](#), even if `autoStop` is enabled.



9. Request new views of stored data using different aggregation parameters: see [Retrieving stored data](#).
10. Close the oscilloscope using [ps4000CloseUnit](#).

3.5 Retrieving stored data

You can collect data from the PicoScope 4000 driver with a different aggregation factor when [ps4000RunBlock](#) or [ps4000RunStreaming](#) has already been called and has successfully captured all the data. Use [ps4000GetValuesAsync](#).



4 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 [ps4000OpenUnit](#) 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 ps4000BlockReady(...)
// Define callback function specific to application

handle1 = ps4000OpenUnit
handle2 = ps4000OpenUnit

ps4000SetChannel(handle1)
// Set up unit 1
ps4000RunBlock(handle1)

ps4000SetChannel(handle2)
// Set up unit 2
ps4000RunBlock(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 synchronize the collection of data between oscilloscopes that are being used in combination.

5 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.

[ps4000BlockReady](#): receive notification when block-mode data ready
[ps4000CloseUnit](#): close a scope device
[ps4000DataReady](#): indicate when post-collection data ready
[ps4000EnumerateUnits](#): find out how many units are connected
[ps4000FlashLed](#): flash the front-panel LED
[ps4000GetChannelInformation](#): find out if extra ranges available
[ps4000GetMaxDownSampleRatio](#): find out aggregation ratio for data
[ps4000GetStreamingLatestValues](#): get streaming data while scope is running
[ps4000GetTimebase](#): find out what timebases are available
[ps4000GetTimebase2](#): find out what timebases are available
[ps4000GetTriggerChannelTimeOffset](#): get trigger times from specified channel
[ps4000GetTriggerChannelTimeOffset64](#): get trigger times from specified channel
[ps4000GetTriggerTimeOffset](#): find out when trigger occurred (32-bit)
[ps4000GetTriggerTimeOffset64](#): find out when trigger occurred (64-bit)
[ps4000GetUnitInfo](#): read information about scope device
[ps4000GetValues](#): retrieve block-mode data with callback
[ps4000GetValuesAsync](#): retrieve streaming data with callback
[ps4000GetValuesBulk](#): retrieve more than one waveform at a time
[ps4000GetValuesTriggerChannelTimeOffsetBulk](#): retrieve time offset from a channel
[ps4000GetValuesTriggerChannelTimeOffsetBulk64](#): retrieve time offset (64-bit)
[ps4000GetValuesTriggerTimeOffsetBulk](#): retrieve time offset for a group of waveforms
[ps4000GetValuesTriggerTimeOffsetBulk64](#): set the buffers for each waveform (64-bit)
[ps4000HoldOff](#): set trigger holdoff
[ps4000IsLedFlashing](#): read status of LED
[ps4000IsReady](#): poll driver in block mode
[ps4000IsTriggerOrPulseWidthQualifierEnabled](#): find out whether trigger is enabled
[ps4000MemorySegments](#): divide scope memory into segments
[ps4000NoOfStreamingValues](#): get number of samples in streaming mode
[ps4000OpenUnit](#): open a scope device
[ps4000OpenUnitAsync](#): open a scope device without waiting
[ps4000OpenUnitAsyncEx](#): open a specified device without waiting
[ps4000OpenUnitEx](#): open a specified device
[ps4000OpenUnitProgress](#): check progress of ps4000OpenUnit call
[ps4000RunBlock](#): start block mode
[ps4000RunStreaming](#): start streaming mode
[ps4000RunStreamingEx](#): start streaming mode with a specified data reduction mode
[ps4000SetChannel](#): set up input channels
[ps4000SetDataBuffer](#): register data buffer with driver
[ps4000SetDataBufferBulk](#): set the buffers for each waveform
[ps4000SetDataBuffers](#): register min/max data buffers with driver
[ps4000SetDataBuffersWithMode](#): register data buffers and specify aggregation mode
[ps4000SetDataBufferWithMode](#): register data buffer and specify aggregation mode
[ps4000SetEts](#): set up equivalent-time sampling (ETS)
[ps4000SetEtsTimeBuffer](#): set up 64-bit buffer for ETS time data
[ps4000SetEtsTimeBuffers](#): set up 32-bit buffers for ETS time data
[ps4000SetExtTriggerRange](#): set EXT trigger input range
[ps4000SetPulseWidthQualifier](#): set up pulse width triggering
[ps4000SetSigGenArbitrary](#): set up arbitrary waveform generator
[ps4000SetSigGenBuiltIn](#): set up function generator
[ps4000SetSimpleTrigger](#): set up level triggers only
[ps4000SetTriggerChannelConditions](#): specify which channels to trigger on
[ps4000SetTriggerChannelDirections](#): set up signal polarities for triggering
[ps4000SetTriggerChannelProperties](#): set up trigger thresholds
[ps4000SetTriggerDelay](#): set up post-trigger delay
[ps4000SigGenArbitraryMinMaxValues](#): get AWG sample value limits
[ps4000SigGenFrequencyToPhase](#): get phase increment for signal generator
[ps4000SigGenSoftwareControl](#): trigger the signal generator

[ps4000Stop](#): stop data capture

[ps4000StreamingReady](#): indicate when streaming-mode data ready

5.1 ps4000BlockReady

```
typedef void (CALLBACK *ps4000BlockReady)
(
    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 [ps4000RunBlock](#), and the driver calls it back when block-mode data is ready. You can then download the data using the [ps4000GetValues](#) function.

Applicability	Block mode only
Arguments	<p><code>handle</code>, identifies the scope device returning the samples</p> <p><code>status</code>, indicates whether an error occurred during collection of the data</p> <p><code>pParameter</code>, a void pointer passed from ps4000RunBlock. The callback function can write to this location to send any data, such as a status flag, back to your application.</p>
Returns	nothing

5.2 ps4000CloseUnit

```
PICO\_STATUS ps4000CloseUnit  
(  
    int16_t    handle  
)
```

This function shuts down a PicoScope 4000 scope device.

Applicability	All modes
Arguments	handle, returned by ps4000OpenUnit , identifies the scope device to be closed
Returns	PICO_OK PICO_HANDLE_INVALID

5.3 ps4000DataReady

```
typedef void (CALLBACK *ps4000DataReady)
(
    int16_t      handle,
    int32_t      noOfSamples,
    int16_t      overflow,
    uint32_t     triggerAt,
    int16_t      triggered,
    void         * pParameter
)
```

This function handles post-collection data returned by the driver after a call to [ps4000GetValuesAsync](#). It is a [callback](#) function that is part of your application. You register it with the PicoScope 4000 Series driver using [ps4000GetValuesAsync](#), and the driver calls it back when the data is ready.

Applicability	All modes
Arguments	<p><code>handle</code>, identifies the scope device returning the samples</p> <p><code>noOfSamples</code>, the number of samples collected</p> <p><code>overflow</code>, 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 and bit 1 Channel B.</p> <p><code>triggerAt</code>, an index to the buffer indicating the location of the trigger point. This parameter is valid only when <code>triggered</code> is non-zero.</p> <p><code>triggered</code>, a flag indicating whether a trigger occurred. If non-zero, a trigger occurred at the location indicated by <code>triggerAt</code>.</p> <p><code>pParameter</code>, a void pointer passed from ps4000GetValuesAsync. 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.</p>
Returns	nothing

5.4 ps4000EnumerateUnits

```
PICO STATUS ps4000EnumerateUnits
(
    int16_t    * count,
    int8_t     * serials,
    int16_t    * serialLth
)
```

This function counts the number of PicoScope 4000 Series units connected to the computer, and returns a list of serial numbers as a string. Note that this function will only detect devices that are not yet being controlled by an application.

Applicability	All modes
Arguments	<p>* <code>count</code>, on exit, the number of scopes found</p> <p>* <code>serials</code>, 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.</p> <p>* <code>serialLth</code>, on entry, the length of the <code>int8_t</code> buffer pointed to by <code>serials</code>; on exit, the length of the string written to <code>serials</code></p>
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

5.5 ps4000FlashLed

```

PICO\_STATUS ps4000FlashLed
(
    int16_t    handle,
    int16_t    start
)

```

This function flashes the LED on the front of the scope without blocking the calling thread. Calls to [ps4000RunStreaming](#) and [ps4000RunBlock](#) cancel any flashing started by this function.

Applicability	All modes
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>start</code>, the action required:</p> <ul style="list-style-type: none"> < 0 : flash the LED indefinitely. 0 : stop the LED flashing. > 0 : flash the LED <code>start</code> times. If the LED is already flashing on entry to this function, the flash count will be reset to <code>start</code>.
Returns	<p>PICO_OK</p> <p>PICO_HANDLE_INVALID</p> <p>PICO_BUSY</p>

5.6 ps4000GetChannelInformation

```

PICO STATUS ps4000GetChannelInformation
(
    int16_t          handle,
    PS4000\_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	<p><i>handle</i>, identifies the scope device</p> <p><i>info</i>, the type of information required, chosen from the list of PS4000_CHANNEL_INFO values</p> <p><i>probe</i>, not used, must be set to 0</p> <p><i>ranges</i>, an array that will be populated with available ranges for the given value of <i>info</i>. May be NULL. See PS4000_RANGE for possible values.</p> <p><i>length</i>, on entry: the length of the <i>ranges</i> array; on exit: the number of elements written to <i>ranges</i> or, if <i>ranges</i> is NULL, the number of elements that would have been written</p> <p><i>channel</i>, the channel for which the information is required. See PS4000_CHANNEL for possible values.</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER

5.7 ps4000GetMaxDownSampleRatio

```

PICO_STATUS ps4000GetMaxDownSampleRatio
(
    int16_t      handle,
    uint32_t     noOfUnaggregatedSamples,
    uint32_t     * maxDownSampleRatio,
    int16_t      downSampleRatioMode,
    uint16_t     segmentIndex
)

```

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

Applicability	All modes
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>noOfUnaggregatedSamples</code>, the number of unaggregated samples to be used to calculate the maximum downsampling ratio</p> <p><code>maxDownSampleRatio</code>, returns the aggregation ratio</p> <p><code>downSampleRatioMode</code>, see ps4000GetValues</p> <p><code>segmentIndex</code>, the memory segment where the data is stored</p>
Returns	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_NO_SAMPLES_AVAILABLE</p> <p>PICO_NULL_PARAMETER</p> <p>PICO_INVALID_PARAMETER</p> <p>PICO_SEGMENT_OUT_OF_RANGE</p> <p>PICO_TOO_MANY_SAMPLES</p>

5.8 ps4000GetNoOfCaptures

[PICO_STATUS](#) ps4000GetNoOfCaptures

```
(
    int16_t      handle,
    uint16_t    * nCaptures
)
```

This function returns the number of captures the device has made in rapid block mode, since you called [ps4000RunBlock](#). You can call `ps4000GetNoOfCaptures` during device capture, after collection has completed or after interrupting waveform collection by calling [ps4000Stop](#). The returned value (`nCaptures`) can then be used to iterate through the number of segments using [ps4000GetValues](#), or in a single call to [ps4000GetValuesBulk](#), where it is used to calculate the `toSegmentIndex` parameter.

Applicability	Rapid block mode
Arguments	<p><code>handle</code>, handle of the required device.</p> <p>* <code>nCaptures</code>, output: the number of available captures that has been collected from calling ps4000RunBlock.</p>
Returns	<p>PICO_OK PICO_DRIVER_FUNCTION PICO_INVALID_HANDLE PICO_NOT_RESPONDING PICO_NO_SAMPLES_AVAILABLE PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_TOO_MANY_SAMPLES</p>

5.9 ps4000GetStreamingLatestValues

```

PICO\_STATUS ps4000GetStreamingLatestValues
(
    int16_t          handle,
    ps4000StreamingReady lpPs4000Ready,
    void            * pParameter
)

```

This function is used to collect the next block of values while [streaming](#) is running. You must call [ps4000RunStreaming](#) beforehand to set up streaming.

Applicability	Streaming mode only
Arguments	<p>handle, identifies the scope device</p> <p>lpPs4000Ready, a pointer to your ps4000StreamingReady callback function that will return the latest aggregated values</p> <p>pParameter, a void pointer that will be passed to the ps4000StreamingReady callback function</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_INVALID_CALL PICO_BUSY PICO_NOT_RESPONDING

5.10 ps4000GetTimebase

```

PICO_STATUS ps4000GetTimebase
(
    int16_t      handle,
    uint32_t     timebase,
    int32_t      noSamples,
    int32_t      * timeIntervalNanoseconds,
    int16_t      oversample,
    int32_t      * maxSamples
    uint16_t     segmentIndex
)

```

This function discovers which [timebases](#) are available on the oscilloscope. You should set up the channels using [ps4000SetChannel](#) first.

Applicability	All modes
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>timebase</code>, a code between 0 and $2^{30}-1$ that specifies the sampling interval (see timebase guide)</p> <p><code>noSamples</code>, the number of samples required. This value is used to calculate the most suitable time unit to use.</p> <p><code>timeIntervalNanoseconds</code>, a pointer to the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here.</p> <p><code>oversample</code>, the amount of oversample required. An oversample of 4, for example, would quadruple the time interval and quarter the maximum samples, and at the same time would increase the effective resolution by one bit. See the topic on oversampling.</p> <p><code>maxSamples</code>, a pointer to the maximum number of samples available. The scope allocates a certain amount of memory for internal overheads and this may vary depending on the number of segments, number of channels enabled, the timebase chosen and the oversample selected. If this pointer is null, nothing will be written here.</p> <p><code>segmentIndex</code>, the number of the memory segment to use</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_TOO_MANY_SAMPLES PICO_INVALID_CHANNEL PICO_INVALID_TIMEBASE PICO_INVALID_PARAMETER

5.11 ps4000GetTimebase2

```

PICO STATUS ps4000GetTimebase2
(
    int16_t      handle,
    uint32_t    timebase,
    int32_t      noSamples,
    float        * timeIntervalNanoseconds,
    int16_t      oversample,
    int32_t      * maxSamples
    uint16_t     segmentIndex
)

```

This function differs from [ps4000GetTimebase](#) only in the `float *` type of the `timeIntervalNanoseconds` argument.

Applicability	All modes
Arguments	<code>timeIntervalNanoseconds</code> , a pointer to the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. All others as in ps4000GetTimebase .
Returns	See ps4000GetTimebase .

5.12 ps4000GetTriggerChannelTimeOffset

```
PICO STATUS ps4000GetTriggerChannelTimeOffset
(
    int16_t          handle,
    uint32_t         * timeUpper,
    uint32_t         * timeLower,
    PS4000 TIME UNITS * timeUnits,
    uint16_t         segmentIndex,
    PS4000 CHANNEL   channel
)
```

This function is for use in applications that display signal data graphically, and is suitable for use with oscilloscopes that use interleaved sampling: PicoScope 4224, 4424 and 4262.

With a digital signal, the trigger point occurs after the last pre-trigger sample and at or before the first post-trigger sample, so there may be a time offset between the trigger point and the post-trigger sample point. `ps4000GetTriggerChannelTimeOffset` retrieves this time offset, as lower and upper 32-bit values, for a waveform obtained in [block mode](#) or [rapid block mode](#), adjusted for the time skew of the specified channel relative to the trigger source. This allows you to plot a stable trace relative to the trigger point.

A 64-bit version of this function, [ps4000GetTriggerChannelTimeOffset64](#), is available for use with programming languages that support 64-bit integers.

Applicability	<p>Block mode, rapid block mode</p> <p>Recommended for PicoScope 4224, 4424 and 4262 variants only</p> <p>You can also call this function on the PicoScope 4226 and 4227 variants, which use simultaneous sampling, but the results will be the same for all four channels: we recommend users of these models call ps4000GetTriggerTimeOffset instead.</p>
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>timeUpper</code>, a pointer to the most significant 32 bits of the time offset for <code>segmentIndex</code>. The array must be long enough to hold the number of requested times.</p> <p><code>timeLower</code>, a pointer to the least significant 32 bits of the time offset for <code>segmentIndex</code>. The array must be long enough to hold the number of requested times.</p> <p><code>timeUnits</code>, a pointer to <code>PS4000_TIME_UNITS</code>, which returns the units in which <code>timeUpper</code> and <code>timeLower</code> are measured. The allowable values are:</p> <ul style="list-style-type: none"> <code>PS4000_FS</code>: femtoseconds <code>PS4000_PS</code>: picoseconds <code>PS4000_NS</code>: nanoseconds <code>PS4000_US</code>: microseconds <code>PS4000_MS</code>: milliseconds <code>PS4000_S</code>: seconds <p><code>segmentIndex</code>, the number of the memory segment for which the information is required</p> <p><code>channel</code>, the scope channel for which the information is required</p>
Returns	<p><code>PICO_OK</code> <code>PICO_INVALID_HANDLE</code> <code>PICO_DEVICE_SAMPLING</code> <code>PICO_SEGMENT_OUT_OF_RANGE</code> <code>PICO_NULL_PARAMETER</code> <code>PICO_NO_SAMPLES_AVAILABLE</code></p>

5.13 ps4000GetTriggerChannelTimeOffset64

```

PICO STATUS ps4000GetTriggerChannelTimeOffset64
(
    int16_t          handle,
    int64_t          * time,
    PS4000 TIME UNITS * timeUnits,
    uint16_t         segmentIndex,
    PS4000 CHANNEL   channel
)

```

This function gets the trigger time offset for a waveform. It is equivalent to [ps4000GetTriggerChannelTimeOffset](#) except that the time offset is returned as a single 64-bit value instead of two 32-bit values.

`ps4000GetTriggerChannelTimeOffset64` is for use in applications that display signal data graphically, and is suitable for use with oscilloscopes that use interleaved sampling: PicoScope 4224, 4424 and 4262.

Applicability	<p>Block mode, rapid block mode</p> <p>Recommended for PicoScope 4224, 4424 and 4262 variants only, as these models use interleaved sampling</p> <p>You can also call this function on the PicoScope 4226 and 4227 variants, which use simultaneous sampling, but the results will be the same for all four channels: we recommend users of these models call ps4000GetTriggerTimeOffset64 instead.</p>
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>time</code>, a pointer to the time offset for <code>segmentIndex</code>. The array must be long enough to hold the number of requested times.</p> <p><code>timeUnits</code>, a pointer to <code>PS4000_TIME_UNITS</code>, which returns the units in which <code>timeUpper</code> and <code>timeLower</code> are measured. The allowable values are:</p> <ul style="list-style-type: none"> <code>PS4000_FS</code>: femtoseconds <code>PS4000_PS</code>: picoseconds <code>PS4000_NS</code>: nanoseconds <code>PS4000_US</code>: microseconds <code>PS4000_MS</code>: milliseconds <code>PS4000_S</code>: seconds <p><code>segmentIndex</code>, the number of the memory segment for which the information is required</p> <p><code>channel</code>, the scope channel for which the information is required</p>
Returns	<p><code>PICO_OK</code> <code>PICO_INVALID_HANDLE</code> <code>PICO_DEVICE_SAMPLING</code> <code>PICO_SEGMENT_OUT_OF_RANGE</code> <code>PICO_NULL_PARAMETER</code> <code>PICO_NO_SAMPLES_AVAILABLE</code></p>

5.14 ps4000GetTriggerTimeOffset

```
PICO STATUS ps4000GetTriggerTimeOffset
(
    int16_t          handle
    uint32_t         * timeUpper
    uint32_t         * timeLower
    PS4000 TIME UNITS * timeUnits
    uint16_t         segmentIndex
)
```

This function retrieves the trigger time offset for multiple waveforms obtained in [block mode](#) or [rapid block mode](#), and is suitable for use with oscilloscopes that use interleaved sampling: PicoScope 4224, 4424 and 4262. It is a more efficient alternative to calling [ps4000GetTriggerChannelTimeOffset](#) once for each waveform required. See [ps4000GetTriggerTimeOffset](#) for an explanation of trigger time offsets.

A 64-bit version of this function, [ps4000GetValuesTriggerChannelTimeOffsetBulk64](#), is available for use with programming languages that support 64-bit integers.

Applicability	<p>Block mode, rapid block mode</p> <p>Recommended for PicoScope 4226 and 4227 variants only</p> <p>You can also call this function on the PicoScope 4224, 4424 and 4262 variants, which use interleaved sampling, but the results will only be accurate to one sample period: users of these models can obtain greater accuracy by calling ps4000GetTriggerChannelTimeOffset instead.</p>
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>timeUpper</code>, a pointer to the most significant 32 bits of the time offset for <code>segmentIndex</code>. The array must be long enough to hold the number of requested times.</p> <p><code>timeLower</code>, a pointer to the least significant 32 bits of the time offset for <code>segmentIndex</code>. The array must be long enough to hold the number of requested times.</p> <p><code>timeUnits</code>, a pointer to <code>PS4000_TIME_UNITS</code>, which returns the units in which <code>timeUpper</code> and <code>timeLower</code> are measured. The allowable values are:</p> <ul style="list-style-type: none"> <code>PS4000_FS</code>: femtoseconds <code>PS4000_PS</code>: picoseconds <code>PS4000_NS</code>: nanoseconds <code>PS4000_US</code>: microseconds <code>PS4000_MS</code>: milliseconds <code>PS4000_S</code>: seconds <p><code>segmentIndex</code>, the number of the memory segment for which the information is required.</p>
Returns	<p><code>PICO_OK</code> <code>PICO_INVALID_HANDLE</code> <code>PICO_DEVICE_SAMPLING</code> <code>PICO_SEGMENT_OUT_OF_RANGE</code> <code>PICO_NULL_PARAMETER</code> <code>PICO_NO_SAMPLES_AVAILABLE</code></p>

5.15 ps4000GetTriggerTimeOffset64

```

PICO_STATUS ps4000GetTriggerTimeOffset64
(
    int16_t          handle,
    int64_t          * time,
    PS4000_TIME_UNITS * timeUnits,
    uint16_t         segmentIndex
)

```

This function is for use in applications that display signal data graphically, and is suitable for use with oscilloscopes that use simultaneous sampling: PicoScope 4226 and 4227. It is equivalent to [ps4000GetTriggerTimeOffset](#) except that the time offset is returned as a single 64-bit value instead of two 32-bit values.

Applicability	<p>Block mode, rapid block mode</p> <p>Recommended for PicoScope 4226 and 4227 variants only</p> <p>You can also call this function on the PicoScope 4224, 4424 and 4262 variants, which use interleaved sampling, but the results will only be accurate to one sample period: users of these models can obtain greater accuracy by calling ps4000GetTriggerChannelTimeOffset instead.</p>
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>time</code>, a pointer to the time offset for <code>segmentIndex</code>. The array must be long enough to hold the number of requested times.</p> <p><code>timeUnits</code>, a pointer to <code>PS4000_TIME_UNITS</code>, which returns the units in which <code>timeUpper</code> and <code>timeLower</code> are measured. The allowable values are:</p> <ul style="list-style-type: none"> <code>PS4000_FS</code>: femtoseconds <code>PS4000_PS</code>: picoseconds <code>PS4000_NS</code>: nanoseconds <code>PS4000_US</code>: microseconds <code>PS4000_MS</code>: milliseconds <code>PS4000_S</code>: seconds <p><code>segmentIndex</code>, the number of the memory segment for which the information is required</p>
Returns	<p><code>PICO_OK</code> <code>PICO_INVALID_HANDLE</code> <code>PICO_DEVICE_SAMPLING</code> <code>PICO_SEGMENT_OUT_OF_RANGE</code> <code>PICO_NULL_PARAMETER</code> <code>PICO_NO_SAMPLES_AVAILABLE</code></p>

5.16 ps4000GetUnitInfo

```

PICO STATUS ps4000GetUnitInfo
(
    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 an `int8_t` 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	<p><code>handle</code>, identifies the scope 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.</p> <p><code>string</code>, a pointer to the <code>int8_t</code> string buffer in the calling function where the unit information string (selected with <code>info</code>) will be stored. If a null pointer is passed, only the <code>requiredSize</code>, pointer to an <code>int16_t</code>, of the <code>int8_t</code> string buffer is returned.</p> <p><code>stringLength</code>, used to return the size of the <code>int8_t</code> string buffer</p> <p><code>requiredSize</code>, used to return the required <code>int8_t</code> string buffer size</p> <p><code>info</code>, an enumerated type specifying what information is required from the driver</p>
Returns	<p>PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_INVALID_INFO PICO_INFO_UNAVAILABLE</p>

PICO_INFO constant	Example
0: PICO_DRIVER_VERSION, version number of PicoScope 4000 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	4224
4: PICO_BATCH_AND_SERIAL, batch and serial number of device	KJL87/6
5: PICO_CAL_DATE, calibration date of device	11Nov08
6: PICO_KERNEL_VERSION, version of kernel driver	1,1,2,4

5.17 ps4000GetValues

```
PICO STATUS ps4000GetValues  
(  
    int16_t      handle,  
    uint32_t     startIndex,  
    uint32_t     * noOfSamples,  
    uint32_t     downSampleRatio,  
    int16_t      downSampleRatioMode,  
    uint16_t     segmentIndex,  
    int16_t     * overflow  
)
```

This function returns block-mode data, either with or without [aggregation](#), starting at the specified sample number. It is used to get the stored data from the device after data collection has stopped.

Applicability	Block mode , rapid block mode
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>startIndex</code>, a zero-based index that indicates the start point for data collection. It is measured in sample intervals from the start of the buffer.</p> <p><code>noOfSamples</code>, on entry: the number of samples requested; on exit, the number of samples actually returned</p> <p><code>downSampleRatio</code>, the aggregation factor that will be applied to the raw data</p> <p><code>downSampleRatioMode</code>, whether to use aggregation to reduce the amount of data. The available values are: RATIO_MODE_NONE (downSampleRatio is ignored) RATIO_MODE_AGGREGATE (uses aggregation) RATIO_MODE_AVERAGE</p> <p>RATIO_MODE_AGGREGATE and RATIO_MODE_AVERAGE are single-bit constants that can be ORed to apply multiple downsampling modes to the same data.</p> <p><code>segmentIndex</code>, the zero-based number of the memory segment where the data is stored</p> <p><code>overflow</code>, 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 and bit 1 Channel B.</p>
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

5.18 ps4000GetValuesAsync

```

PICO_STATUS ps4000GetValuesAsync
(
    int16_t      handle,
    uint32_t     startIndex,
    uint32_t     noOfSamples,
    uint32_t     downSampleRatio,
    int16_t      downSampleRatioMode,
    uint16_t     segmentIndex,
    void         * lpDataReady,
    void         * pParameter
)

```

This function returns streaming data, either with or without [aggregation](#), starting at the specified sample number. It is used to get the stored data from the device (in [block mode](#)) or the driver (in [streaming mode](#)) after data collection has stopped. It returns the data using a [callback](#).

Applicability	Streaming mode only
Arguments	<p>handle, identifies the scope device</p> <p>startIndex, noOfSamples, downSampleRatio, downSampleRatioMode, segmentIndex: see ps4000GetValues</p> <p>lpDataReady, a pointer to the ps4000StreamingReady function that is called when the data is ready</p> <p>pParameter, a void pointer that will be passed to the ps4000StreamingReady callback function. The data type depends on the design of the callback function, which is determined by the application programmer.</p>
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

5.19 ps4000GetValuesBulk

```

PICO_STATUS ps4000GetValuesBulk
(
    int16_t      handle,
    uint32_t     * noOfSamples,
    uint16_t     fromSegmentIndex,
    uint16_t     toSegmentIndex,
    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 [aggregation](#).

Applicability	Rapid block mode
Arguments	<p><code>handle</code>, identifies the scope device</p> <p>* <code>noOfSamples</code>, 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.</p> <p><code>fromSegmentIndex</code>, the first segment from which the waveform should be retrieved</p> <p><code>toSegmentIndex</code>, the last segment from which the waveform should be retrieved</p> <p>* <code>overflow</code>, equal to or larger than the number of waveforms to be retrieved. Each segment index has a separate <code>overflow</code> element, with <code>overflow[0]</code> containing the <code>fromSegmentIndex</code>, and the last index the <code>toSegmentIndex</code>. Each element in the array is a bit field as described under ps4000GetValues.</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE PICO_STARTINDEX_INVALID PICO_NOT_RESPONDING

5.20 ps4000GetValuesTriggerChannelTimeOffsetBulk

```

PICO_STATUS ps4000GetValuesTriggerChannelTimeOffsetBulk
(
    int16_t          handle,
    uint32_t         * timesUpper,
    uint32_t         * timesLower,
    PS4000_TIME_UNITS * timeUnits,
    uint16_t         fromSegmentIndex,
    uint16_t         toSegmentIndex,
    PS4000_CHANNEL  channel
)

```

This function retrieves the trigger time offset for multiple waveforms obtained in [rapid block mode](#), and is suitable for use with oscilloscopes that use interleaved sampling: PicoScope 4224, 4424 and 4262. It is a more efficient alternative to calling [ps4000GetTriggerChannelTimeOffset](#) once for each waveform required. See [ps4000GetTriggerTimeOffset](#) for an explanation of trigger time offsets.

A 64-bit version of this function, [ps4000GetValuesTriggerChannelTimeOffsetBulk64](#), is available for use with programming languages that support 64-bit integers.

Applicability	Rapid block mode
Arguments	<p><code>handle</code>, identifies the scope device</p> <p>* <code>timesUpper</code>, a pointer to 32-bit integers. This will hold the most significant 32 bits of the time offset for each requested segment index. <code>times[0]</code> will hold the <code>fromSegmentIndex</code> time offset and the last <code>times</code> index will hold the <code>toSegmentIndex</code> time offset. The array must be long enough to hold the number of requested times.</p> <p>* <code>timesLower</code>, a pointer to 32-bit integers. This will hold the least-significant 32 bits of the time offset for each requested segment index. <code>times[0]</code> will hold the <code>fromSegmentIndex</code> time offset and the last <code>times</code> index will hold the <code>toSegmentIndex</code> time offset. The array must be long enough to hold the number of requested times.</p> <p>* <code>timeUnits</code>, a pointer to a range of <code>PS4000_TIME_UNITS</code>. This must be equal to or larger than the number of requested times. <code>timeUnits[0]</code> will contain the time unit for <code>fromSegmentIndex</code> and the last index will contain the time unit for <code>toSegmentIndex</code>.</p> <p><code>fromSegmentIndex</code>, the first segment for which the time offset is required</p> <p><code>toSegmentIndex</code>, the last segment for which the time offset is required. If <code>toSegmentIndex</code> is less than <code>fromSegmentIndex</code> then the driver will wrap around from the last segment to the first.</p> <p><code>channel</code>, the channel for which the information is required</p>
Returns	<p><code>PICO_OK</code></p> <p><code>PICO_INVALID_HANDLE</code></p> <p><code>PICO_NULL_PARAMETER</code></p> <p><code>PICO_DEVICE_SAMPLING</code></p> <p><code>PICO_SEGMENT_OUT_OF_RANGE</code></p> <p><code>PICO_NO_SAMPLES_AVAILABLE</code></p>

5.21 ps4000GetValuesTriggerChannelTimeOffsetBulk64

```

PICO STATUS ps4000GetValuesTriggerChannelTimeOffsetBulk64
(
    int16_t          handle,
    int64_t          * times,
    PS4000 TIME UNITS * timeUnits,
    uint16_t         fromSegmentIndex,
    uint16_t         toSegmentIndex,
    PS4000 CHANNEL   channel
)

```

This function is equivalent to [ps4000GetValuesTriggerChannelTimeOffsetBulk](#) but retrieves the trigger time offsets as 64-bit values instead of pairs of 32-bit values.

Applicability	Rapid block mode
Arguments	<p>handle, identifies the scope device</p> <p>* times, a pointer to 64-bit integers. This will hold the time offset for each requested segment index. times[0] will hold the time offset for fromSegmentIndex, and the last times index will hold the time offset for toSegmentIndex. The array must be long enough to hold the number of times requested.</p> <p>* timeUnits, a pointer to a range of PS4000_TIME_UNITS. This must be equal or larger than the number of requested times. timeUnits[0] will contain the time unit for fromSegmentIndex, and the last index will contain the time unit for toSegmentIndex.</p> <p>fromSegmentIndex, the first segment for which the time offset is required. The result will be placed in times[0] and timeUnits[0].</p> <p>toSegmentIndex, the last segment for which the time offset is required. The result will be placed in the last elements of the times and timeUnits arrays. If toSegmentIndex is less than fromSegmentIndex then the driver will wrap around from the last segment to the first.</p> <p>channel, the scope channel for which information is required</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE

5.22 ps4000GetValuesTriggerTimeOffsetBulk

```
PICO STATUS ps4000GetValuesTriggerTimeOffsetBulk
(
    int16_t          handle,
    uint32_t         * timesUpper,
    uint32_t         * timesLower,
    PS4000 TIME UNITS * timeUnits,
    uint16_t         fromSegmentIndex,
    uint16_t         toSegmentIndex
)
```

This function retrieves the trigger time offset for multiple waveforms obtained in [rapid block mode](#). It is a more efficient alternative to calling [ps4000GetTriggerTimeOffset](#) once for each waveform required. See [ps4000GetTriggerTimeOffset](#) for an explanation of trigger time offsets.

`ps4000GetValuesTriggerTimeOffsetBulk` is suitable for use with oscilloscopes that use simultaneous sampling: PicoScope 4226 and 4227. A 64-bit version of this function, [ps4000GetValuesTriggerTimeOffsetBulk64](#), is available for use with programming languages that support 64-bit integers.

Applicability	<p>Rapid block mode</p> <p>Recommended for PicoScope 4226 and 4227 variants only</p> <p>You can also call this function on the PicoScope 4224, 4424 and 4262 variants, which use interleaved sampling, but the results will only be accurate to one sample period: users of these models can obtain greater accuracy by calling ps4000GetValuesTriggerChannelTimeOffsetBulk instead.</p>
Arguments	<p><code>handle</code>, identifies the scope device</p> <p>* <code>timesUpper</code>, a pointer to 32-bit integers. This will hold the most significant 32 bits of the time offset for each requested segment index. <code>times[0]</code> will hold the <code>fromSegmentIndex</code> time offset and the last <code>times</code> index will hold the <code>toSegmentIndex</code> time offset. The array must be long enough to hold the number of requested times.</p> <p>* <code>timesLower</code>, a pointer to 32-bit integers. This will hold the least-significant 32 bits of the time offset for each requested segment index. <code>times[0]</code> will hold the <code>fromSegmentIndex</code> time offset and the last <code>times</code> index will hold the <code>toSegmentIndex</code> time offset. The array must be long enough to hold the number of requested times.</p> <p>* <code>timeUnits</code>, a pointer to a range of <code>PS4000_TIME_UNITS</code>. This must be equal to or larger than the number of requested times. <code>timeUnits[0]</code> will contain the time unit for <code>fromSegmentIndex</code> and the last index will contain the time unit for <code>toSegmentIndex</code>.</p> <p><code>fromSegmentIndex</code>, the first segment for which the time offset is required</p> <p><code>toSegmentIndex</code>, the last segment for which the time offset is required. If <code>toSegmentIndex</code> is less than <code>fromSegmentIndex</code> then the driver will wrap around from the last segment to the first.</p>
Returns	<p><code>PICO_OK</code> <code>PICO_INVALID_HANDLE</code> <code>PICO_NULL_PARAMETER</code> <code>PICO_DEVICE_SAMPLING</code> <code>PICO_SEGMENT_OUT_OF_RANGE</code> <code>PICO_NO_SAMPLES_AVAILABLE</code></p>

5.23 ps4000GetValuesTriggerTimeOffsetBulk64

```

PICO STATUS ps4000GetValuesTriggerTimeOffsetBulk64
(
    int16_t          handle,
    int64_t          * times,
    PS4000 TIME UNITS * timeUnits,
    uint16_t         fromSegmentIndex,
    uint16_t         toSegmentIndex
)

```

This function is equivalent to [ps4000GetValuesTriggerTimeOffsetBulk](#) but retrieves the trigger time offsets as 64-bit values instead of pairs of 32-bit values.

Applicability	<p>Rapid block mode</p> <p>Recommended for PicoScope 4226 and 4227 variants only</p> <p>You can also call this function on the PicoScope 4224, 4424 and 4262 variants, which use interleaved sampling, but the results will only be accurate to one sample period: users of these models can obtain greater accuracy by calling ps4000GetValuesTriggerChannelTimeOffsetBulk64 instead.</p>
Arguments	<p><code>handle</code>, identifies the scope device</p> <p>* <code>times</code>, a pointer to 64-bit integers. This will hold the time offset for each requested segment index. <code>times[0]</code> will hold the time offset for <code>fromSegmentIndex</code>, and the last <code>times</code> index will hold the time offset for <code>toSegmentIndex</code>. The array must be long enough to hold the number of times requested.</p> <p>* <code>timeUnits</code>, a pointer to a range of <code>PS4000_TIME_UNITS</code>. This must be equal or larger than the number of requested times. <code>timeUnits[0]</code> will contain the time unit for <code>fromSegmentIndex</code>, and the last index will contain the time unit for <code>toSegmentIndex</code>.</p> <p><code>fromSegmentIndex</code>, the first segment for which the time offset is required. The result will be placed in <code>times[0]</code> and <code>timeUnits[0]</code>.</p> <p><code>toSegmentIndex</code>, the last segment for which the time offset is required. The result will be placed in the last elements of the <code>times</code> and <code>timeUnits</code> arrays. If <code>toSegmentIndex</code> is less than <code>fromSegmentIndex</code> then the driver will wrap around from the last segment to the first.</p>
Returns	<p>PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE</p>

5.24 ps4000HoldOff

```

PICO_STATUS ps4000HoldOff
(
    int16_t          handle,
    uint64_t         holdoff,
    PS4000_HOLDOFF_TYPE type
)

```

This function sets the holdoff time - the time that the scope waits after each trigger event before allowing the next trigger event.

Applicability	Not currently supported. Reserved for future use.
Arguments	<p>holdoff, the number of samples between trigger events. The time is calculated by multiplying the sample interval by the hold-off.</p> <p>type, the type of hold-off. Only hold-off by time is currently supported: <u>PS4000_TIME</u></p>
Returns	<u>PICO_OK</u> - success <u>PICO_DRIVER_FUNCTION</u> <u>PICO_INVALID_PARAMETER</u>

5.25 ps4000IsLedFlashing

```
PICO STATUS ps4000IsLedFlashing  
(  
    int16_t    handle,  
    int16_t    * status  
)
```

This function reports whether or not the LED is flashing.

Applicability	All modes
Arguments	<code>handle</code> , identifies the scope device <code>status</code> , returns a flag indicating the status of the LED: <> 0 : flashing 0 : not flashing
Returns	PICO_OK PICO_HANDLE_INVALID PICO_NULL_PARAMETER

5.26 ps4000IsReady

```
PICO_STATUS ps4000IsReady
(
    int16_t      handle,
    int16_t      * ready
)
```

This function may be used instead of a callback function to receive data from [ps4000RunBlock](#). To use this method, pass a NULL pointer as the `lpReady` argument to [ps4000RunBlock](#). You must then poll the driver to see if it has finished collecting the requested samples.

Applicability	Block mode
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>ready</code>, on exit, indicates the state of the collection. If zero, the device is still collecting. If non-zero, the device has finished collecting and ps4000GetValues can be used to retrieve the data.</p>
Returns	

5.27 ps4000IsTriggerOrPulseWidthQualifierEnabled

```

PICO\_STATUS ps4000IsTriggerOrPulseWidthQualifierEnabled
(
    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 ps4000RunBlock or ps4000RunStreaming .
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>triggerEnabled</code>, indicates whether the trigger will successfully be set when ps4000RunBlock or ps4000RunStreaming is called. A non-zero value indicates that the trigger is set, otherwise the trigger is not set.</p> <p><code>pulseWidthQualifierEnabled</code>, indicates whether the pulse width qualifier will successfully be set when ps4000RunBlock or ps4000RunStreaming is called. A non-zero value indicates that the pulse width qualifier is set, otherwise the pulse width qualifier is not set.</p>
Returns	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_NULL_PARAMETER</p>

5.28 ps4000MemorySegments

```

PICO_STATUS ps4000MemorySegments
(
    int16_t      handle
    uint16_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	<p><code>handle</code>, identifies the scope device</p> <p><code>nSegments</code>, the number of segments to be used, from 1 to 8192</p> <p><code>nMaxSamples</code>, returns 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 <code>nMaxSamples</code> divided by the number of channels.</p>
Returns	<p>PICO_OK</p> <p>PICO_USER_CALLBACK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_TOO_MANY_SEGMENTS</p> <p>PICO_MEMORY</p>

5.29 ps4000NoOfStreamingValues

```

PICO_STATUS ps4000NoOfStreamingValues
(
    int16_t      handle,
    uint32_t    * noOfValues
)

```

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

Applicability	Streaming mode . Call after ps4000Stop .
Arguments	<code>handle</code> , identifies the scope device <code>noOfValues</code> , returns the number of samples
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_NOT_USED PICO_BUSY

5.30 ps4000OpenUnit

```
PICO STATUS ps4000OpenUnit
(
    int16_t    * handle
)
```

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	<p>handle, pointer to an <code>int16_t</code> that receives the handle number:</p> <ul style="list-style-type: none"> -1 : if the unit fails to open, 0 : if no unit is found > 0 : if successful (value is handle of the device opened) <p>The handle number must be used in all subsequent calls to API functions to identify this scope device.</p>
Returns	<p>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</p>

5.31 ps4000OpenUnitAsync

```
PICO\_STATUS ps4000OpenUnitAsync  
(  
    int16_t * status  
)
```

This function opens a scope device without blocking the calling thread. You can find out when it has finished by periodically calling [ps4000OpenUnitProgress](#) until that function returns a non-zero value.

Applicability	All modes
Arguments	<code>status</code> , pointer to an <code>int16_t</code> that indicates: 0 if there is already an open operation in progress 1 if the open operation is initiated
Returns	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_OPERATION_FAILED

5.32 ps4000OpenUnitAsyncEx

```

PICO_STATUS ps4000OpenUnitAsyncEx
(
    int16_t    * status,
    int8_t     * serial
)

```

This function opens a scope device selected by serial number without blocking the calling thread. You can find out when it has finished by periodically calling [ps4000OpenUnitProgress](#) until that function returns a non-zero value.

Applicability	All modes
Arguments	<p><code>status</code>, pointer to a <code>int16_t</code> that indicates: 0 if there is already an open operation in progress 1 if the open operation is initiated</p> <p><code>serial</code>, the serial number of the device to be opened. A null-terminated string.</p>
Returns	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_OPERATION_FAILED

5.33 ps4000OpenUnitEx

```

PICO_STATUS ps4000OpenUnitEx
(
    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	<p><code>handle</code>, pointer to an <code>int16_t</code> that receives the handle number: -1 : if the unit fails to open, 0 : if no unit is found or > 0 : if successful (value is handle to the device opened) The handle number must be used in all subsequent calls to API functions to identify this scope device.</p> <p><code>serial</code>, the serial number of the device to be opened. A null-terminated string.</p>
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

5.34 ps4000OpenUnitProgress

```

PICO STATUS ps4000OpenUnitProgress
(
    int16_t    * handle,
    int16_t    * progressPercent,
    int16_t    * complete
)

```

This function checks on the progress of [ps4000OpenUnitAsync](#).

Applicability	Use after ps4000OpenUnitAsync
Arguments	<p><code>handle</code>, pointer to an <code>int16_t</code> where the unit handle is to be written. -1 if the unit fails to open, 0 if no unit is found or a non-zero handle to the device.</p> <p>Note: This handle is not valid unless the function returns <code>PICO_OK</code>.</p> <p><code>progressPercent</code>, pointer to an <code>int16_t</code> to which the percentage progress is to be written. 100% implies that the open operation is complete.</p> <p><code>complete</code>, pointer to an <code>int16_t</code> that is set to 1 when the open operation has finished</p>
Returns	<p><code>PICO_OK</code> <code>PICO_NULL_PARAMETER</code> <code>PICO_OPERATION_FAILED</code></p>

5.35 ps4000PingUnit

```
PICO\_STATUS ps4000PingUnit  
(  
    int16_t    handle  
)
```

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

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

5.36 ps4000RunBlock

```

PICO STATUS ps4000RunBlock
(
    int16_t          handle,
    int32_t          noOfPreTriggerSamples,
    int32_t          noOfPostTriggerSamples,
    uint32_t         timebase,
    int16_t          oversample,
    int32_t          * timeIndisposedMs,
    uint16_t         segmentIndex,
    ps4000BlockReady 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	<p><code>handle</code>, identifies the scope device</p> <p><code>noOfPreTriggerSamples</code>, the number of samples to return before the trigger event. If no trigger has been set, then this argument is added to <code>noOfPostTriggerSamples</code> to give the maximum number of data points (samples) to collect.</p> <p><code>noOfPostTriggerSamples</code>, the number of samples to return after the trigger event. If no trigger event has been set, then this argument is added to <code>noOfPreTriggerSamples</code> to give the maximum number of data points to collect. If a trigger condition has been set, this specifies the number of data points to collect after a trigger has fired, and the number of data points to be collected is:</p> $\text{noOfPreTriggerSamples} + \text{noOfPostTriggerSamples}$ <p><code>timebase</code>, a number in the range 0 to $2^{30}-1$. See the guide to calculating timebase values.</p> <p><code>oversample</code>, the oversampling factor, a number in the range 1 to 16</p> <p><code>timeIndisposedMs</code>, returns the time, in milliseconds, that the device will spend collecting samples. This does not include any auto trigger timeout. If this pointer is null, nothing will be written here.</p> <p><code>segmentIndex</code>, zero-based, specifies which memory segment to use</p> <p><code>lpReady</code>, a pointer to the ps4000BlockReady callback that the driver will call when the data has been collected. To use the ps4000IsReady polling method instead of a callback function, set this pointer to NULL.</p> <p><code>pParameter</code>, a void pointer that is passed to the ps4000BlockReady callback function. The callback can use the pointer to return arbitrary data to your application.</p>
Returns	<pre> 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 </pre>

5.37 ps4000RunStreaming

```

PICO STATUS ps4000RunStreaming
(
    int16_t          handle,
    uint32_t         * sampleInterval,
    PS4000 TIME UNITS sampleIntervalTimeUnits
    uint32_t         maxPreTriggerSamples,
    uint32_t         maxPostTriggerSamples,
    int16_t          autoStop
    uint32_t         downSampleRatio,
    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 [aggregated](#) and the values returned to the application. Call [ps4000GetStreamingLatestValues](#) 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 then this will become the maximum number of unaggregated samples.

Applicability	Streaming mode only
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>sampleInterval</code>, a pointer to the requested time interval between data points on entry and the actual time interval assigned on exit</p> <p><code>sampleIntervalTimeUnits</code>, the unit of time that the <code>sampleInterval</code> is set to. Use one of these values:</p> <pre> PS4000_FS PS4000_PS PS4000_NS PS4000_US PS4000_MS PS4000_S </pre> <p><code>maxPreTriggerSamples</code>, the maximum number of raw samples before a trigger condition for each enabled channel. If no trigger condition is set this argument is ignored.</p> <p><code>maxPostTriggerSamples</code>, 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.</p> <p><code>autoStop</code>, a flag to specify if the streaming should stop when all of <code>maxSamples</code> have been taken</p> <p><code>downSampleRatio</code>, the number of raw values to each aggregated value</p> <p><code>overviewBufferSize</code>, 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 <code>bufferLth</code> value passed to ps4000SetDataBuffer.</p>
Returns	<pre> 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 </pre>

5.38 ps4000RunStreamingEx

```

PICO STATUS ps4000RunStreamingEx
(
    int16_t          handle,
    uint32_t         * sampleInterval,
    PS4000 TIME UNITS sampleIntervalTimeUnits
    uint32_t         maxPreTriggerSamples,
    uint32_t         maxPostTriggerSamples,
    int16_t          autoStop
    uint32_t         downSampleRatio,
    int16_t          downSampleRatioMode,
    uint32_t         overviewBufferSize
)

```

This function tells the oscilloscope to start collecting data in [streaming mode](#) and with a specified data reduction mode. When data has been collected from the device it is [aggregated](#) and the values returned to the application. Call [ps4000GetStreamingLatestValues](#) 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 then this will become the maximum number of unaggregated samples.

Applicability	Streaming mode only
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>sampleInterval</code>, a pointer to the requested time interval between data points on entry and the actual time interval assigned on exit</p> <p><code>sampleIntervalTimeUnits</code>, the unit of time that the <code>sampleInterval</code> is set to. Use one of these values:</p> <pre>PS4000_FS PS4000_PS PS4000_NS PS4000_US PS4000_MS PS4000_S</pre> <p><code>maxPreTriggerSamples</code>, the maximum number of raw samples before a trigger condition for each enabled channel. If no trigger condition is set this argument is ignored.</p> <p><code>maxPostTriggerSamples</code>, 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.</p> <p><code>autoStop</code>, a flag to specify if the streaming should stop when all of <code>maxSamples</code> have been taken</p> <p><code>downSampleRatio</code>, the number of raw values to each aggregated value</p> <p><code>downSampleRatioMode</code>, the data reduction mode to use</p> <p><code>overviewBufferSize</code>, 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 <code>bufferLth</code> value passed to ps4000SetDataBuffer.</p>
Returns	<pre>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</pre>

5.39 ps4000SetBwFilter

```

PICO_STATUS ps4000SetBwFilter
(
    int16_t          handle,
    PS4000_CHANNEL channel,
    int16_t          enable
)

```

This function enables or disables the bandwidth-limiting filter on the specified channel.

Applicability	PicoScope 4262 only
Arguments	<p>handle, identifies the scope device</p> <p>channel, an enumerated type. The values are: PS4000_CHANNEL_A PS4000_CHANNEL_B</p> <p>enable, whether to enable or disable the filter: TRUE = enable FALSE = disable</p>
Returns	PICO_OK PICO_USER_CALLBACK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL

5.40 ps4000SetChannel

```

PICO STATUS ps4000SetChannel
(
    int16_t          handle,
    PS4000_CHANNEL channel,
    int16_t          enabled,
    int16_t          dc,
    PS4000_RANGE   range
)

```

This function specifies whether an input channel is to be enabled, the [AC/DC coupling](#) mode and the voltage range.

Applicability	All modes
Arguments	<p>handle, identifies the scope device</p> <p>channel, an enumerated type. The values are: PS4000_CHANNEL_A PS4000_CHANNEL_B PS4000_CHANNEL_C (4-channel scopes only) PS4000_CHANNEL_D (4-channel scopes only)</p> <p>enabled, specifies if the channel is active. The values are: TRUE = active FALSE = inactive</p> <p>dc, specifies the AC/DC coupling mode. The values are: TRUE = DC FALSE = AC</p> <p>range, specifies the measuring range. Measuring ranges 0 to 12, for standard scopes, are shown in the table below. Additional ranges for special-purpose scopes are listed under PS4000_RANGE. For example, to enable IEPE input mode on an IEPE-enabled scope, select one of the ranges beginning with PS4000_ACCELEROMETER_.</p>
Returns	PICO_OK PICO_USER_CALLBACK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_INVALID_VOLTAGE_RANGE

range		Voltage range
0	PS4000_10MV	±10 mV
1	PS4000_20MV	±20 mV
2	PS4000_50MV	±50 mV
3	PS4000_100MV	±100 mV
4	PS4000_200MV	±200 mV
5	PS4000_500MV	±500 mV
6	PS4000_1V	±1 V
7	PS4000_2V	±2 V
8	PS4000_5V	±5 V
9	PS4000_10V	±10 V
10	PS4000_20V	±20 V
11	PS4000_50V	±50 V
12	PS4000_100V	±100 V

5.41 ps4000SetDataBuffer

```

PICO\_STATUS ps4000SetDataBuffer
(
    int16_t          handle,
    PS4000\_CHANNEL channel,
    int16_t          * buffer,
    int32_t          bufferLth
)

```

This function registers your data buffer, for non-[aggregated](#) data, with the PicoScope 4000 driver. You need to allocate the buffer before calling this function.

Applicability	All modes. For aggregated data, use ps4000SetDataBuffers instead.
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>channel</code>, the channel for which you want to set the buffers. Use one of these values:</p> <ul style="list-style-type: none"> PS4000_CHANNEL_A PS4000_CHANNEL_B PS4000_CHANNEL_C (4-channel scopes only) PS4000_CHANNEL_D (4-channel scopes only) <p><code>buffer</code>, a buffer to receive the data values</p> <p><code>bufferLth</code>, the size of the <code>buffer</code> array</p>
Returns	<ul style="list-style-type: none"> PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL

5.42 ps4000SetDataBufferBulk

```

PICO_STATUS ps4000SetDataBufferBulk
(
    int16_t          handle,
    PS4000_CHANNEL channel,
    int16_t          * buffer,
    int32_t          bufferLth,
    uint16_t         waveform
)

```

This function allows the buffers to be set for each waveform in [rapid block mode](#). The number of waveforms captured is determined by the `nCaptures` argument sent to [ps4000SetNoOfCaptures](#). There is only one buffer for each waveform, because bulk collection does not support [aggregation](#).

Applicability	Rapid block mode
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>channel</code>, the scope channel with which the buffer is to be associated. The data should be retrieved from this channel by calling one of the GetValues functions.</p> <p>* <code>buffer</code>, an array in which the captured data is stored</p> <p><code>bufferLth</code>, the size of the buffer</p> <p><code>waveform</code>, an index to the waveform number, between 0 and <code>nCaptures-1</code></p>
Returns	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_INVALID_CHANNEL</p> <p>PICO_INVALID_PARAMETER</p>

5.43 ps4000SetDataBuffers

```

PICO\_STATUS ps4000SetDataBuffers
(
    int16_t          handle,
    PS4000\_CHANNEL channel,
    int16_t          * bufferMax,
    int16_t          * bufferMin,
    int32_t          bufferLth
)

```

This function registers your data buffers, for receiving [aggregated](#) data, with the PicoScope 4000 driver. You need to allocate memory for the buffers before calling this function.

Applicability	All sampling modes. For non-aggregated data, use ps4000SetDataBuffer instead.
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>channel</code>, the channel for which you want to set the buffers. Use one of these constants: PS4000_CHANNEL_A PS4000_CHANNEL_B PS4000_CHANNEL_C (4-channel scopes only) PS4000_CHANNEL_D (4-channel scopes only)</p> <p><code>bufferMax</code>, a buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise</p> <p><code>bufferMin</code>, a buffer to receive the minimum data values when <code>downSampleRatio > 1</code>. Not used when <code>downSampleRatio</code> is 1.</p> <p><code>bufferLth</code>, specifies the size of the <code>bufferMax</code> and <code>bufferMin</code> arrays</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL

5.44 ps4000SetDataBuffersWithMode

```

PICO STATUS ps4000SetDataBuffersWithMode
(
    int16_t          handle,
    PS4000 CHANNEL channel,
    int16_t          * bufferMax,
    int16_t          * bufferMin,
    int32_t          bufferLth,
    RATIO MODE      mode
)

```

This function registers your data buffers, for receiving [aggregated](#) data, with the PicoScope 4000 driver. You need to allocate memory for the buffers before calling this function.

Applicability	All sampling modes. For non-aggregated data, use ps4000SetDataBuffer instead.
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>channel</code>, the channel for which you want to set the buffers. Use one of these constants: PS4000_CHANNEL_A PS4000_CHANNEL_B PS4000_CHANNEL_C (4-channel scopes only) PS4000_CHANNEL_D (4-channel scopes only)</p> <p><code>bufferMax</code>, a buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise</p> <p><code>bufferMin</code>, a buffer to receive the minimum data values when <code>downSampleRatio > 1</code>. Not used when <code>downSampleRatio</code> is 1.</p> <p><code>bufferLth</code>, specifies the size of the <code>bufferMax</code> and <code>bufferMin</code> arrays</p> <p><code>mode</code>, the data reduction mode to use</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL

5.45 ps4000SetDataBufferWithMode

```

PICO\_STATUS ps4000SetDataBufferWithMode
(
    int16_t          handle,
    PS4000\_CHANNEL channel,
    int16_t          * buffer,
    int32_t          bufferLth,
    RATIO\_MODE      mode
)

```

This function registers your data buffer, for non-[aggregated](#) data, with the PicoScope 4000 driver. You need to allocate the buffer before calling this function.

Applicability	All modes. For aggregated data, use ps4000SetDataBuffers instead.
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>channel</code>, the channel for which you want to set the buffers. Use one of these values: PS4000_CHANNEL_A PS4000_CHANNEL_B PS4000_CHANNEL_C (4-channel scopes only) PS4000_CHANNEL_D (4-channel scopes only)</p> <p><code>buffer</code>, a buffer to receive the data values</p> <p><code>bufferLth</code>, the size of the <code>buffer</code> array</p> <p><code>mode</code>, the type of data reduction to use</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL

5.46 ps4000SetEts

```

PICO STATUS ps4000SetEts
(
    int16_t          handle,
    PS4000 ETS MODE mode,
    int16_t          etsCycles,
    int16_t          etsInterleave,
    int32_t          * sampleTimePicoseconds
)

```

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

Applicability	Block mode only. ETS is only supported by PicoScope 4226 and 4227.						
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>mode</code>, the ETS mode. Use one of these values:</p> <table> <tr> <td><code>PS4000_ETS_OFF</code></td> <td>disables ETS</td> </tr> <tr> <td><code>PS4000_ETS_FAST</code></td> <td>enables ETS and provides <code>etsCycles</code> cycles of data, which may contain data from previously returned cycles</td> </tr> <tr> <td><code>PS4000_ETS_SLOW</code></td> <td>enables ETS and provides fresh data every <code>etsCycles</code> cycles. This mode takes longer to provide each data set, but the data sets are more stable and are guaranteed to contain only new data.</td> </tr> </table> <p><code>etsCycles</code>, the number of cycles to store: the computer can then select <code>etsInterleave</code> cycles to give the most uniform spread of samples. <code>etsCycles</code> should be between two and five times the value of <code>etsInterleave</code>.</p> <p><code>etsInterleave</code>, the number of ETS interleaves to use. If the sample time is 20 ns and the interleave is 10, the approximate time per sample will be 2 ns.</p> <p><code>sampleTimePicoseconds</code>, returns the effective sample time used by the function</p>	<code>PS4000_ETS_OFF</code>	disables ETS	<code>PS4000_ETS_FAST</code>	enables ETS and provides <code>etsCycles</code> cycles of data, which may contain data from previously returned cycles	<code>PS4000_ETS_SLOW</code>	enables ETS and provides fresh data every <code>etsCycles</code> cycles. This mode takes longer to provide each data set, but the data sets are more stable and are guaranteed to contain only new data.
<code>PS4000_ETS_OFF</code>	disables ETS						
<code>PS4000_ETS_FAST</code>	enables ETS and provides <code>etsCycles</code> cycles of data, which may contain data from previously returned cycles						
<code>PS4000_ETS_SLOW</code>	enables ETS and provides fresh data every <code>etsCycles</code> cycles. This mode takes longer to provide each data set, but the data sets are more stable and are guaranteed to contain only new data.						
Returns	<p><code>PICO_OK</code></p> <p><code>PICO_USER_CALLBACK</code></p> <p><code>PICO_INVALID_HANDLE</code></p> <p><code>PICO_INVALID_PARAMETER</code></p>						

5.47 ps4000SetEtsTimeBuffer

```

PICO\_STATUS ps4000SetEtsTimeBuffer
(
    int16_t      handle,
    int64_t      * buffer,
    int32_t      bufferLth
)

```

This function tells the PicoScope 4000 driver where to find your application's ETS time buffers. These buffers contain the 64-bit timing information for each ETS sample after you run a block-mode ETS capture.

Applicability	ETS mode only. ETS is only supported by PicoScope 4226 and 4227. If your programming language does not support 64-bit data, use the 32-bit version ps4000SetEtsTimeBuffers instead.
Arguments	<code>handle</code> , identifies the scope device <code>buffer</code> , a pointer to a set of 64-bit words, each representing the time in femtoseconds (10^{-15} s) at which the sample was captured <code>bufferLth</code> , the size of the buffer array
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER

5.48 ps4000SetEtsTimeBuffers

```

PICO_STATUS ps4000SetEtsTimeBuffers
(
    int16_t      handle,
    uint32_t     * timeUpper,
    uint32_t     * timeLower,
    int32_t      bufferLth
)

```

This function tells the PicoScope 4000 driver where to find your application's ETS time buffers. These buffers contain the timing information for each ETS sample after you run a block-mode ETS capture. There are two buffers containing the upper and lower 32-bit parts of the timing information, to allow programming languages that do not support 64-bit data to retrieve the timings correctly.

Applicability	<p>ETS mode only. ETS is only supported by PicoScope 4226 and 4227.</p> <p>If your programming language supports 64-bit data, then you can use ps4000SetEtsTimeBuffer instead.</p>
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>timeUpper</code>, a pointer to a set of 32-bit words, each representing the upper 32 bits of the time in femtoseconds (10^{-15} s) at which the sample was captured</p> <p><code>timeLower</code>, a pointer to a set of 32-bit words, each representing the lower 32 bits of the time in femtoseconds at which the sample was captured</p> <p><code>bufferLth</code>, the size of the <code>timeUpper</code> and <code>timeLower</code> arrays</p>
Returns	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_NULL_PARAMETER</p>

5.49 ps4000SetExtTriggerRange

```

PICO\_STATUS ps4000SetExtTriggerRange
(
    int16_t      handle,
    PS4000\_RANGE extRange
)

```

This function sets the range of the external trigger.

Applicability	PicoScope 4262 only
Arguments	<p>handle, identifies the scope device</p> <p>extRange, specifies the range for the external trigger (± 500 mV or ± 5 V)</p>
Returns	<p>PICO_OK</p> <p>PICO_INVALID_PARAMETER</p>

	extRange	Voltage range
5	PS4000_500MV	± 500 mV
8	PS4000_5V	± 5 V

5.50 ps4000SetNoOfCaptures

```

PICO_STATUS ps4000SetNoOfCaptures
(
    int16_t      handle,
    uint16_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, identifies the scope device nCaptures, the number of waveforms to be captured in one run
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER

5.51 ps4000SetPulseWidthQualifier

```

PICO_STATUS ps4000SetPulseWidthQualifier
(
    int16_t          handle,
    PWQ_CONDITIONS * conditions,
    int16_t          nConditions,
    THRESHOLD_DIRECTION direction,
    uint32_t         lower,
    uint32_t         upper,
    PULSE_WIDTH_TYPE type
)

```

This function sets up pulse width qualification, which can be used on its own for pulse width triggering or combined with threshold triggering, level triggering or window triggering to produce more complex triggers. The pulse width qualifier is set by defining one or more conditions structures that are then ORed together. Each structure is itself the AND of the states of one or more of the inputs. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

Applicability	All modes
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>conditions</code>, a pointer to an array of PWQ_CONDITIONS structures specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there are several elements, the overall trigger condition is the logical OR of all the elements. If <code>conditions</code> is set to <code>null</code> then the pulse width qualifier is not used.</p> <p><code>nConditions</code>, the number of elements in the <code>conditions</code> array. If <code>nConditions</code> is zero then the pulse width qualifier is not used.</p> <p><code>direction</code>, the direction of the signal required for the trigger to fire</p> <p><code>lower</code>, the lower limit of the pulse width counter</p> <p><code>upper</code>, the upper limit of the pulse width counter. This parameter is used only when the type is set to <code>PW_TYPE_IN_RANGE</code> or <code>PW_TYPE_OUT_OF_RANGE</code>.</p> <p><code>type</code>, the pulse width type, one of these constants: <code>PW_TYPE_NONE</code> (do not use the pulse width qualifier) <code>PW_TYPE_LESS_THAN</code> (pulse width less than <code>lower</code>) <code>PW_TYPE_GREATER_THAN</code> (pulse width greater than <code>lower</code>) <code>PW_TYPE_IN_RANGE</code> (pulse width between <code>lower</code> and <code>upper</code>) <code>PW_TYPE_OUT_OF_RANGE</code> (pulse width not between <code>lower</code> and <code>upper</code>)</p>
Returns	<p><code>PICO_OK</code> <code>PICO_INVALID_HANDLE</code> <code>PICO_USER_CALLBACK</code> <code>PICO_CONDITIONS</code> <code>PICO_PULSE_WIDTH_QUALIFIER</code></p>

5.51.1 PWQ_CONDITIONS structure

A structure of this type is passed to [ps4000SetPulseWidthQualifier](#) in the `conditions` argument to specify the trigger conditions, and is defined as follows:

```
typedef struct tPwqConditions
{
    TRIGGER_STATE channelA;
    TRIGGER_STATE channelB;
    TRIGGER_STATE channelC;
    TRIGGER_STATE channelD;
    TRIGGER_STATE external;
    TRIGGER_STATE aux;
} PWQ_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The [ps4000SetPulseWidthQualifier](#) function can OR together a number of these structures to produce the final pulse width qualifier, which can be any possible Boolean function of the scope's inputs.

Elements	<p><code>channelA</code>, <code>channelB</code>, <code>channelC</code>, <code>channelD</code>: the type of condition that should be applied to each channel. Use these constants:</p> <pre>CONDITION_DONT_CARE CONDITION_TRUE CONDITION_FALSE</pre> <p>The channels that are set to <code>CONDITION_TRUE</code> or <code>CONDITION_FALSE</code> must all meet their conditions simultaneously to produce a trigger. Channels set to <code>CONDITION_DONT_CARE</code> are ignored.</p> <p><code>external</code>, <code>aux</code>: not used</p>
-----------------	--

5.52 ps4000SetSigGenArbitrary

```

PICO STATUS ps4000SetSigGenArbitrary
(
    int16_t          handle,
    int32_t          offsetVoltage,
    uint32_t         pkToPk,
    uint32_t         startDeltaPhase,
    uint32_t         stopDeltaPhase,
    uint32_t         deltaPhaseIncrement,
    uint32_t         dwellCount,
    int16_t          * arbitraryWaveform,
    int32_t          arbitraryWaveformSize,
    SWEEP TYPE     sweepType,
    int16_t          operationType,
    INDEX MODE    indexMode,
    uint32_t         shots,
    uint32_t         sweeps,
    SIGGEN TRIG TYPE triggerType,
    SIGGEN TRIG SOURCE triggerSource,
    int16_t         extInThreshold
)

```

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

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

Applicability	PicoScope 4226, 4227 and 4262 only
Arguments	
handle, identifies the scope 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. Call ps4000SigGenFrequencyToPhase to calculate this.	
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. Call ps4000SigGenFrequencyToPhase to calculate this.	
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 4226 and 4227: `MIN_DWELL_COUNT` (10)
 PicoScope 4262: `PS4262_MIN_DWELL_COUNT` (3)

`arbitraryWaveform`, a pointer to a buffer that holds the waveform pattern as a set of samples equally spaced in time. You can obtain the range of allowable values by calling [ps4000SigGenArbitraryMinMaxValues](#). Sample value ranges are as follows:

PicoScope 4226 and 4227: [0, 4095]
 PicoScope 4262: [-32768, 32767]

`arbitraryWaveformSize`, the size of the arbitrary waveform buffer, in samples:

All models: Min: `MIN_SIG_GEN_BUFFER_SIZE` (1)
 PicoScope 4226 and 4227: Max: `MAX_SIG_GEN_BUFFER_SIZE` (8192)
 PicoScope 4262: Max: `PS4262_MAX_WAVEFORM_BUFFER_SIZE` (4096)

`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.

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

`PS4000_OP_NONE`: White noise/PRBS output disabled. The waveform is defined by the other arguments.
`PS4000_WHITENOISE`: The signal generator produces white noise and ignores all settings except `offsetVoltage` and `pkToPk`
`PS4000_PRBS`: The signal generator produces a PRBS (PicoScope 4262 only)

`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 [ps4000SetSimpleTrigger](#))
`SIGGEN_EXT_IN`: the EXT input

SIGGEN_SOFT_TRIG: a software trigger (see [ps4000SigGenSoftwareControl](#))

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`, an [ADC](#) count for use when the trigger source is SIGGEN_EXT_IN. If the EXT input is also being used as the scope trigger then the same ADC count must be specified in both places, otherwise a warning will be issued. Minimum and maximum 16-bit values correspond to the following voltages:

PicoScope 4226 & 4227: ± 20 V

PicoScope 4262: ± 500 mV or ± 5 V depending on range selected by [ps4000SetExtTriggerRange](#)

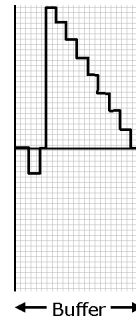
Returns

0: if successful.
Error code: if failed

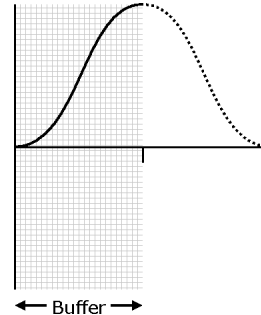
5.52.1 AWG index modes

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

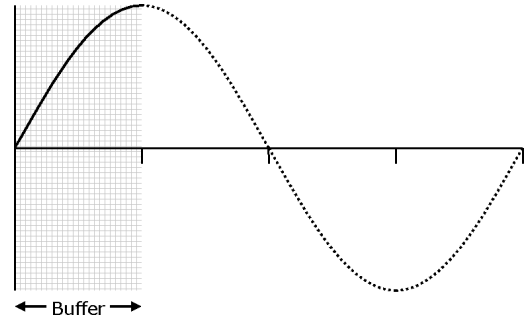
SINGLE mode. The generator outputs the raw contents of the buffer repeatedly. This mode is the only one that can generate asymmetrical waveforms. You can also use this mode for symmetrical waveforms, but the dual and quad modes make more efficient use of the buffer memory.



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.



5.52.2 Calculating deltaPhase

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

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

where:

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

You can call [ps4000SigGenFrequencyToPhase](#) to calculate *deltaPhase*.

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

Parameter	PicoScope 4226/4227	PicoScope 4262
<i>phaseAccumulatorSize</i>	32 bits	32 bits
<i>bufferAddressWidth</i>	13 bits	12 bits
<i>dacFrequency</i>	20 MHz	192 kHz
<i>dacPeriod</i> (= $1/dacFrequency$)	50 ns	$\approx 5.208 \mu s$

5.53 ps4000SetSigGenBuiltIn

```

PICO STATUS ps4000SetSigGenBuiltIn
(
    int16_t          handle,
    int32_t          offsetVoltage,
    uint32_t         pkToPk,
    int16_t          waveType,
    float            startFrequency,
    float            stopFrequency,
    float            increment,
    float            dwellTime,
    SWEEP TYPE     sweepType,
    int16_t          operationType,
    uint32_t         shots,
    uint32_t         sweeps,
    SIGGEN TRIG TYPE triggerType,
    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.

Applicability	PicoScope 4226, 4227 and 4262 only																				
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>offsetVoltage</code>, the voltage offset, in microvolts, to be applied to the waveform</p> <p><code>pkToPk</code>, the peak-to-peak voltage, in microvolts, of the waveform signal</p> <p><code>waveType</code>, the type of waveform to be generated by the oscilloscope:</p> <table> <tr><td>PS4000_SINE</td><td>sine wave</td></tr> <tr><td>PS4000_SQUARE</td><td>square wave</td></tr> <tr><td>PS4000_TRIANGLE</td><td>triangle wave</td></tr> <tr><td>PS4000_RAMP_UP</td><td>rising sawtooth</td></tr> <tr><td>PS4000_RAMP_DOWN</td><td>falling sawtooth</td></tr> <tr><td>PS4000_SINC</td><td>sin(x)/x</td></tr> <tr><td>PS4000_GAUSSIAN</td><td>normal distribution</td></tr> <tr><td>PS4000_HALF_SINE</td><td>full-wave rectified sinusoid</td></tr> <tr><td>PS4000_DC_VOLTAGE</td><td>DC voltage</td></tr> <tr><td>PS4000_WHITE_NOISE</td><td>random values</td></tr> </table> <p><code>startFrequency</code>, the frequency at which the signal generator should begin. Range: MIN SIG GEN FREQ to MAX SIG GEN FREQ.</p> <p><code>stopFrequency</code>, the frequency at which the sweep should reverse direction or return to the start frequency. Range: MIN SIG GEN FREQ to MAX SIG GEN FREQ.</p> <p><code>increment</code>, the amount by which the frequency rises or falls every <code>dwellTime</code> seconds in sweep mode</p>	PS4000_SINE	sine wave	PS4000_SQUARE	square wave	PS4000_TRIANGLE	triangle wave	PS4000_RAMP_UP	rising sawtooth	PS4000_RAMP_DOWN	falling sawtooth	PS4000_SINC	sin(x)/x	PS4000_GAUSSIAN	normal distribution	PS4000_HALF_SINE	full-wave rectified sinusoid	PS4000_DC_VOLTAGE	DC voltage	PS4000_WHITE_NOISE	random values
PS4000_SINE	sine wave																				
PS4000_SQUARE	square wave																				
PS4000_TRIANGLE	triangle wave																				
PS4000_RAMP_UP	rising sawtooth																				
PS4000_RAMP_DOWN	falling sawtooth																				
PS4000_SINC	sin(x)/x																				
PS4000_GAUSSIAN	normal distribution																				
PS4000_HALF_SINE	full-wave rectified sinusoid																				
PS4000_DC_VOLTAGE	DC voltage																				
PS4000_WHITE_NOISE	random values																				

	<code> dwellTime</code> , the time in seconds between frequency changes in sweep mode <code> sweepType</code> , <code> operationType</code> , <code> shots</code> , <code> sweeps</code> , <code> triggerType</code> , <code> triggerSource</code> , <code> extInThreshold</code> : see ps4000SigGenArbitrary
Returns	0: if successful. Error code: if failed.

5.54 ps4000SetSimpleTrigger

```

PICO_STATUS ps4000SetSimpleTrigger
(
    int16_t          handle,
    int16_t          enable,
    PS4000\_CHANNEL   source,
    int16_t          threshold,
    THRESHOLD\_DIRECTION direction,
    uint32_t         delay,
    int16_t          autoTrigger_ms
)

```

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

Applicability	All modes
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>enabled</code>, zero to disable the trigger, any non-zero value to set the trigger</p> <p><code>source</code>, the channel on which to trigger</p> <p><code>threshold</code>, the ADC count at which the trigger will fire</p> <p><code>direction</code>, 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.</p> <p><code>delay</code>, the time, in sample periods, between the trigger occurring and the first sample being taken</p> <p><code>autoTrigger_ms</code>, the number of milliseconds the device will wait if no trigger occurs</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_DRIVER_FUNCTION

5.55 ps4000SetTriggerChannelConditions

```

PICO\_STATUS ps4000SetTriggerChannelConditions
(
    int16_t          handle,
    TRIGGER\_CONDITIONS * conditions,
    int16_t          nConditions
)

```

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

Applicability	All modes
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>conditions</code>, a pointer to an array of TRIGGER_CONDITIONS structures specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there are several elements, the overall trigger condition is the logical OR of all the elements.</p> <p><code>nConditions</code>, the number of elements in the <code>conditions</code> array. If <code>nConditions</code> is zero then triggering is switched off.</p>
Returns	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_USER_CALLBACK</p> <p>PICO_CONDITIONS</p> <p>PICO_MEMORY_FAIL</p>

5.55.1 TRIGGER_CONDITIONS structure

A structure of this type is passed to [ps4000SetTriggerChannelConditions](#) in the `conditions` argument to specify the trigger conditions, and is defined as follows:

```
typedef struct tTriggerConditions
{
    TRIGGER\_STATE channelA;
    TRIGGER\_STATE channelB;
    TRIGGER\_STATE channelC;
    TRIGGER\_STATE channelD;
    TRIGGER\_STATE external;
    TRIGGER\_STATE aux;
    TRIGGER\_STATE pulseWidthQualifier;
} TRIGGER_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The [ps4000SetTriggerChannelConditions](#) 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	<p><code>channelA</code>, <code>channelB</code>, <code>channelC</code>, <code>channelD</code>, <code>external</code>, <code>pulseWidthQualifier</code>: the type of condition that should be applied to each channel. Use these constants:</p> <pre>CONDITION_DONT_CARE CONDITION_TRUE CONDITION_FALSE</pre> <p>The channels that are set to <code>CONDITION_TRUE</code> or <code>CONDITION_FALSE</code> must all meet their conditions simultaneously to produce a trigger. Channels set to <code>CONDITION_DONT_CARE</code> are ignored.</p> <p><code>aux</code>, not used</p>
-----------------	--

5.56 ps4000SetTriggerChannelDirections

```

PICO STATUS ps4000SetTriggerChannelDirections
(
    int16_t          handle,
    THRESHOLD DIRECTION channelA,
    THRESHOLD DIRECTION channelB,
    THRESHOLD DIRECTION channelC,
    THRESHOLD DIRECTION channelD,
    THRESHOLD DIRECTION ext,
    THRESHOLD DIRECTION aux
)

```

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

Applicability	All modes.
Arguments	<p>handle, identifies the scope device</p> <p>channelA, channelB, channelC, channelD, ext all specify the direction in which the signal must pass through the threshold to activate the trigger. See the table below.</p> <p>aux, not used</p>
Returns	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_USER_CALLBACK</p> <p>PICO_INVALID_PARAMETER</p>

Trigger direction constants

Constant	Type	Direction
ABOVE	gated	above the upper threshold
ABOVE_LOWER	gated	above the lower threshold
BELOW	gated	below the upper threshold
BELOW_LOWER	gated	below the lower threshold
RISING	threshold	rising edge, using upper threshold
RISING_LOWER	threshold	rising edge, using lower threshold
FALLING	threshold	falling edge, using upper threshold
FALLING_LOWER	threshold	falling edge, using lower threshold
RISING_OR_FALLING	threshold	either edge
INSIDE	window-qualified	inside window
OUTSIDE	window-qualified	outside window
ENTER	window	entering the window
EXIT	window	leaving the window
ENTER_OR_EXIT	window	either entering or leaving the window
POSITIVE_RUNT	window-qualified	entering and leaving from below
NEGATIVE_RUNT	window-qualified	entering and leaving from above
NONE	none	none

5.57 ps4000SetTriggerChannelProperties

```

PICO STATUS ps4000SetTriggerChannelProperties
(
    int16_t          handle,
    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	<p><code>handle</code>, identifies the scope device</p> <p><code>channelProperties</code>, a pointer to an array of TRIGGER CHANNEL PROPERTIES structures describing the requested properties. The array can contain a single element describing the properties of one channel, or a number of elements describing several channels. If <code>null</code> is passed, triggering is switched off.</p> <p><code>nChannelProperties</code>, the size of the <code>channelProperties</code> array. If zero, triggering is switched off.</p> <p><code>auxOutputEnable</code>, not used</p> <p><code>autoTriggerMilliseconds</code>, the time in milliseconds for which the scope device will wait before collecting data if no trigger event occurs. If this is set to zero, the scope device will wait indefinitely for a trigger.</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_TRIGGER_ERROR PICO_MEMORY_FAIL PICO_INVALID_TRIGGER_PROPERTY

5.57.1 TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to [ps4000SetTriggerChannelProperties](#) in the `channelProperties` argument to specify the trigger mechanism, and is defined as follows:

```
typedef struct tTriggerChannelProperties
{
    int16_t          thresholdUpper;
    uint16_t         thresholdUpperHysteresis;
    int16_t          thresholdLower;
    uint16_t         thresholdLowerHysteresis;
    PS4000\_CHANNEL channel;
    THRESHOLD\_MODE thresholdMode;
} TRIGGER_CHANNEL_PROPERTIES
```

Elements	
	<p><code>thresholdUpper</code>, 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.</p> <p><code>thresholdUpperHysteresis</code>, the hysteresis by which the trigger must exceed the upper threshold before it will fire. It is scaled in 16-bit counts.</p> <p><code>thresholdLower</code>, 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.</p> <p><code>thresholdLowerHysteresis</code>, the hysteresis by which the trigger must exceed the lower threshold before it will fire. It is scaled in 16-bit counts.</p> <p><code>channel</code>, the channel to which the properties apply. See ps4000SetChannel for possible values.</p> <p><code>thresholdMode</code>, either a level or window trigger. Use one of these constants: LEVEL WINDOW</p>

5.58 ps4000SetTriggerDelay

```

PICO_STATUS ps4000SetTriggerDelay
(
    int16_t      handle,
    uint32_t     delay
)

```

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

Applicability	Block mode and rapid block mode only
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>delay</code>, the time between the trigger occurring and the first sample, in sample periods. For example, if <code>delay = 100</code> then the scope would wait 100 sample periods before sampling. Example: with the PicoScope 4224, at a timebase of 80 MS/s, or 12.5 ns per sample (<code>timebase = 0</code>) the total delay would then be 100 x 12.5 ns = 1.25 μs.</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK

5.59 ps4000SigGenArbitraryMinMaxValues

```

PICO_STATUS ps4000SigGenArbitraryMinMaxValues
(
    int16_t      handle,
    int16_t      * minArbitraryWaveformValue,
    int16_t      * maxArbitraryWaveformValue,
    uint32_t     * minArbitraryWaveformSize,
    uint32_t     * maxArbitraryWaveformSize
)

```

This function returns the range of possible sample values and waveform buffer sizes that can be supplied to [ps4000SetSigGenArbitrary](#) for setting up the arbitrary waveform generator (AWG). These values may vary between different models.

Applicability	All models with AWG
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>minArbitraryWaveformValue</code>, on exit, the lowest sample value allowed in the <code>arbitraryWaveform</code> buffer supplied to ps4000SetSigGenArbitrary</p> <p><code>maxArbitraryWaveformValue</code>, on exit, the highest sample value allowed in the <code>arbitraryWaveform</code> buffer supplied to ps4000SetSigGenArbitrary</p> <p><code>minArbitraryWaveformSize</code>, on exit, the minimum value allowed for the <code>arbitraryWaveformSize</code> argument supplied to ps4000SetSigGenArbitrary</p> <p><code>maxArbitraryWaveformSize</code>, on exit, the maximum value allowed for the <code>arbitraryWaveformSize</code> argument supplied to ps4000SetSigGenArbitrary</p>
Returns	<p>PICO_OK</p> <p>PICO_NOT_SUPPORTED_BY_THIS_DEVICE, if the device does not have an arbitrary waveform generator.</p> <p>PICO_NULL_PARAMETER, if all the parameter pointers are NULL.</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_DRIVER_FUNCTION</p>

5.60 ps4000SigGenFrequencyToPhase

```

PICO_STATUS ps4000SigGenFrequencyToPhase
(
    int16_t          handle,
    double           frequency,
    PS4000_INDEX_MODE indexMode,
    uint32_t         bufferLength,
    uint32_t         * phase
)

```

This function converts a frequency to a phase count for use with the arbitrary waveform generator (AWG). The value returned depends on the length of the buffer, the index mode passed and the device model. The phase count can then be sent to the driver through [ps4000SetSigGenArbitrary](#).

Applicability	All models with AWG
Arguments	<p><code>handle</code>, identifies the scope device</p> <p><code>frequency</code>, the required AWG output frequency</p> <p><code>indexMode</code>, see AWG index modes</p> <p><code>bufferLength</code>, the number of samples in the AWG buffer</p> <p><code>phase</code>, on exit, the <code>deltaPhase</code> argument to be sent to the AWG setup function</p>
Returns	<p>PICO_OK</p> <p>PICO_NOT_SUPPORTED_BY_THIS_DEVICE, if the device does not have an AWG.</p> <p>PICO_SIGGEN_FREQUENCY_OUT_OF_RANGE, if the frequency is out of range.</p> <p>PICO_NULL_PARAMETER, if <code>phase</code> is a NULL pointer.</p> <p>PICO_SIG_GEN_PARAM, if <code>indexMode</code> or <code>bufferLength</code> is out of range.</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_DRIVER_FUNCTION</p>

5.61 ps4000SigGenSoftwareControl

```

PICO\_STATUS ps4000SigGenSoftwareControl
(
    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 ps4000SetSigGenBuiltIn or ps4000SetSigGenArbitrary .
Arguments	<p>handle, identifies the scope device</p> <p>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.</p>
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_TRIGGER_SOURCE

5.62 ps4000Stop

```
PICO\_STATUS ps4000Stop  
(  
    int16_t    handle  
)
```

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

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

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

5.63 ps4000StreamingReady

```
typedef void (CALLBACK *ps4000StreamingReady)
(
    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 [ps4000GetStreamingLatestValues](#), and the driver calls it back when streaming-mode data is ready. You can then download the data using the [ps4000GetValuesAsync](#) function. Your callback function should do nothing more than copy the data to another buffer within your application. To maintain the best application performance, the function should return as quickly as possible without attempting to process or display the data.

Applicability	Streaming mode only
Arguments	<p><code>handle</code>, identifies the scope device returning the samples</p> <p><code>noOfSamples</code>, the number of samples to collect</p> <p><code>startIndex</code>, an index to the first valid sample in the buffer. This is the buffer that was previously passed to ps4000SetDataBuffer or ps4000SetDataBuffers.</p> <p><code>overflow</code>, 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 corresponding to Channel A and so on.</p> <p><code>triggerAt</code>, an index to the buffer indicating the location of the trigger point relative to <code>startIndex</code>. This parameter is valid only when <code>triggered</code> is non-zero.</p> <p><code>triggered</code>, a flag indicating whether a trigger occurred. If non-zero, a trigger occurred at the location indicated by <code>triggerAt</code>.</p> <p><code>autoStop</code>, the flag that was set in the call to ps4000RunStreaming</p> <p><code>pParameter</code>, a void pointer passed from ps4000GetStreamingLatestValues. The callback function can write to this location to send any data, such as a status flag, back to the application.</p>
Returns	nothing

5.64 Wrapper functions

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

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

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

The `ps4000Wrap.dll` wrapper DLL has a callback function for streaming data collection that copies data from the driver buffer specified to a temporary application buffer of the same size. To do this, the driver and application buffers must be registered with the wrapper and the corresponding channel(s) must be specified as being enabled. You should process the data in the temporary application buffer accordingly, for example by copying the data into a large array.

Procedure:

1. Open the oscilloscope using [ps4000OpenUnit](#).
 - 1a. Inform the wrapper of the number of channels on the device by calling `setChannelCount`.
2. Select channels, ranges and AC/DC coupling using [ps4000SetChannel](#).
 - 2a. Inform the wrapper which channels have been enabled by calling `setEnabledChannels`.
3. Use the appropriate trigger setup functions. For programming languages that do not support structures, use the wrapper's advanced trigger setup functions.
4. Call [ps4000SetDataBuffer](#) (or for aggregated data collection [ps4000SetDataBuffers](#)) to tell the driver where your data buffer(s) is(are).
 - 4a. Register the data buffer(s) with the wrapper and set the application buffer(s) into which the data will be copied. Call `setAppAndDriverBuffers` (or `setMaxMinAppAndDriverBuffers` for aggregated data collection).
5. Start the oscilloscope running using [ps4000RunStreaming](#).
6. Loop and call `GetStreamingLatestValues` and `IsReady` to get data and flag when the wrapper is ready for data to be retrieved.
 - 6a. Call the wrapper's `AvailableData` function to obtain information on the number of samples collected and the start index in the buffer.
 - 6b. Call the wrapper's `IsTriggerReady` function for information on whether a trigger has occurred and the trigger index relative to the start index in the buffer.
7. Process data returned to your application data buffers.
8. Call `AutoStopped` if the `autoStop` parameter has been set to `TRUE` in the call to [ps4000RunStreaming](#).

9. Repeat steps 6 to 8 until `AutoStopped` returns true or you wish to stop data collection.
10. Call [ps4000Stop](#), even if the `autoStop` parameter has been set to `TRUE`.
11. To disconnect a device, call [ps4000CloseUnit](#).

6 Further information

6.1 Programming examples

Your SDK installation includes programming examples in several languages and development environments. Please refer to the SDK for details.

6.2 Driver status codes

Every function in the `ps4000.dll` driver returns a **driver status code** from the list of `PICO_STATUS` values in the `PicoStatus.h` header file, which is included in the `inc` subdirectory of the Pico Technology SDK.

6.3 Enumerated types and constants

Enumerated types and constants are defined in the file `ps4000Api.h`, which is included in the `inc` subdirectory of your Pico Technology SDK installation. We recommend that you refer to these constants by name unless your programming language allows only numerical values.

6.4 Numeric data types

Here is a list of the sizes and ranges of the numeric data types used in the PicoScope 4000 Series API.

Type	Bits	Signed or unsigned?
<code>int16_t</code>	16	signed
<code>enum</code>	32	enumerated
<code>int32_t</code>	32	signed
<code>uint32_t</code>	32	unsigned
<code>float</code>	32	signed (IEEE 754)
<code>int64_t</code>	64	signed
<code>uint64_t</code>	64	unsigned

7 Glossary

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.

Aggregation. The [PicoScope 4000](#) driver can use this method to reduce the amount of data your application needs to process. This means that for every block of consecutive samples, it stores only the minimum and maximum values. You can set the number of samples in each block, called the aggregation parameter, when you call [PS4000RunStreaming](#) for real-time capture, and when you call [ps4000GetStreamingLatestValues](#) to obtain post-processed data.

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.

Coupling mode. This mode selects either AC or DC coupling in the oscilloscope's input path. Use AC mode for small signals that may be superimposed on a DC level. Use DC mode for measuring absolute voltage levels. Set the coupling mode using [ps4000SetChannel](#).

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 or 64-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. Gigasamples per second. A gigasample is equal to one billion samples.

IEPE. Integrated Electronics Piezoelectric. A standard for accelerometers and other piezoelectric sensors that require an external power supply. Special IEPE-enabled PicoScope 4000 Series scopes have a phantom-powered input that supports these sensors.

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. Megasamples per second. A megasample is equal to one million samples.

Oversampling. Oversampling is taking measurements more frequently than the requested sample rate, and then combining them to produce the required number of samples. If, as is usually the case, the signal contains a small amount of noise, this technique can increase the effective [vertical resolution](#) of the oscilloscope.

PC Oscilloscope. A virtual instrument consisting of a PicoScope oscilloscope unit and a software application.

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.

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 6.6 million samples per second.

Timebase. The sampling rate that the scope uses to acquire data. The timebase can be set to any value returned by the [ps4000GetTimebase](#) or [ps4000GetTimebase2](#) functions.

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. A typical USB 1.1 port supports a data transfer rate of 12 megabits per second, so is much faster than an RS232 COM port.

USB 2.0. Universal Serial Bus (High Speed). This is a standard port used to connect external devices to PCs. A typical USB 2.0 port supports a data transfer rate 40 times faster than USB 1.1 when used with a USB 2.0 device, but can also be used with USB 1.1 devices.

USB 3.0. A faster version of the Universal Serial Bus standard. Your PicoScope is fully compatible with USB 3.0 ports and will operate with the same performance as on a USB 2.0 port.

Vertical resolution. A value, in bits, indicating the precision with which the oscilloscope converts input voltages to digital values. [Oversampling](#) (see above) can improve the effective vertical resolution.

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.

Index

A

- AC/DC control 103
- AC/DC coupling 5
 - setting 67
- ADC 103
- Aggregation 15, 103
 - getting ratio 27
- Analog-to-digital converter 103
- API function calls 19
- Arbitrary waveform generator 81
 - index modes 84
- AWG 81
- AWG index modes 84

B

- Bandwidth-limiting filter 66
- Block mode 5, 8, 8, 9, 21, 103
 - polling status 50
 - starting 60
- Buffer size 103
- Buffers
 - overrun 4

C

- Callback 103
- Callback function
 - block mode 21
 - streaming mode 23, 99
- Channel information, reading 26
- Channel selection 5
 - settings 67
- Closing a scope device 22
- CONDITION_ constants 80, 90
- Constants 102
- Coupling mode 103

D

- Data acquisition 15
- Data buffers, setting 69, 70, 71, 72, 73
- Disk space 2
- Driver 3, 103
 - status codes 102

E

- Enumerated types 102

- Enumerating oscilloscopes 24
- ETS 103
 - overview (API) 14
 - setting time buffers 75, 76
 - setting up 74
 - using (API) 15

F

- Filter, bandwidth-limiting 66
- Function calls 19
- Functions
 - ps4000BlockReady 21
 - ps4000CloseUnit 22
 - ps4000DataReady 23
 - ps4000EnumerateUnits 24
 - ps4000FlashLed 25
 - ps4000GetChannelInformation 26
 - ps4000GetMaxDownSampleRatio 27
 - ps4000GetNoOfCaptures 28
 - ps4000GetStreamingLatestValues 29
 - ps4000GetTimebase 30
 - ps4000GetTimebase2 31
 - ps4000GetTriggerChannelTimeOffset 32
 - ps4000GetTriggerChannelTimeOffset64 34
 - ps4000GetTriggerTimeOffset 35
 - ps4000GetTriggerTimeOffset64 37
 - ps4000GetUnitInfo 38
 - ps4000GetValues 39
 - ps4000GetValuesAsync 41
 - ps4000GetValuesBulk 42
 - ps4000GetValuesTriggerChannelTimeOffsetBulk 43
 - ps4000GetValuesTriggerTimeOffsetBulk 45
 - ps4000GetValuesTriggerTimeOffsetBulk64 44, 47
 - ps4000HoldOff 48
 - ps4000IsLedFlashing 49
 - ps4000IsReady 50
 - ps4000IsTriggerOrPulseWidthQualifierEnabled 51
 - ps4000MemorySegments 52
 - ps4000NoOfStreamingValues 53
 - ps4000OpenUnit 54
 - ps4000OpenUnitAsync 55
 - ps4000OpenUnitAsyncEx 56
 - ps4000OpenUnitEx 57
 - ps4000OpenUnitProgress 58
 - ps4000PingUnit 59
 - ps4000RunBlock 60
 - ps4000RunStreaming 62
 - ps4000RunStreamingEx 64
 - ps4000SetBwFilter 66

Functions

- ps4000SetChannel 67
- ps4000SetDataBuffer 69
- ps4000SetDataBufferBulk 70
- ps4000SetDataBuffers 71
- ps4000SetDataBuffersWithMode 72
- ps4000SetDataBufferWithMode 73
- ps4000SetEts 74
- ps4000SetEtsTimeBuffer 75
- ps4000SetEtsTimeBuffers 76
- ps4000SetExtTriggerRange 77
- ps4000SetNoOfCaptures 78
- ps4000SetPulseWidthQualifier 79
- ps4000SetSigGenArbitrary 81
- ps4000SetSigGenBuiltIn 86
- ps4000SetSimpleTrigger 88
- ps4000SetTriggerChannelConditions 89
- ps4000SetTriggerChannelDirections 91
- ps4000SetTriggerChannelProperties 92
- ps4000SetTriggerDelay 94
- ps4000SigGenArbitraryMinMaxValues 95
- ps4000SigGenFrequencyToPhase 96
- ps4000SigGenSoftwareControl 97
- ps4000Stop 98
- ps4000StreamingReady 99

H

- Hold-off 48
- Hysteresis 93

I

- IEPE 103
- IEPE mode 67

L

- LED
 - programming 25, 49
- LEVEL constant 93

M

- Maximum sampling rate 104
- Memory in scope 8
- Memory segments 52
- Multi-unit operation 18

O

- One-shot signals 14
- Opening a unit 54, 55, 56, 57, 58

- Operating system 2
- Oversampling 6, 104

P

- PC Oscilloscope 104
- PICO_STATUS enum type 102
- picopp.inf 3
- picopp.sys 3
- PicoScope 4000 Series 1, 104
- PicoScope software 3
- Processor 2
- PS4000_CHANNEL_A 67
- PS4000_CHANNEL_B 67
- PS4000_LOST_DATA 4
- PS4000_MAX_VALUE 4
- PS4000_MIN_VALUE 4
- PS4262_MAX_VALUE 4
- PS4262_MIN_VALUE 4
- Pulse width trigger 79
- PWQ_CONDITIONS structure 80

R

- Rapid block mode 10, 10
- Resolution, vertical 6
- Retrieving data 39, 41
 - stored (API) 17
 - streaming mode 29

S

- Sampling rate
 - maximum 8
- Scaling 4
- Serial numbers 24
- Signal generator 9
 - arbitrary waveforms 81
 - built-in waveforms 86
 - software trigger 97
- Skew, timing 32, 34, 43, 44
- Software license conditions 1
- Status codes 102
- Stopping sampling 98
- Streaming mode 8, 15, 104
 - getting number of values 53
 - retrieving data 29
 - starting 62, 64
 - using (API) 16
- Synchronising units 18
- System memory 2
- System requirements 2

T

- Threshold voltage 5
- Time buffers
 - setting for ETS 75, 76
- Timebase 7, 104
 - setting 30, 31
- Trademarks 2
- Trigger 5
 - conditions 89, 90
 - delay 94
 - directions 91
 - external 4
 - pulse width qualifier 51, 79
 - pulse width qualifier conditions 80
 - setting up 88
 - time offset 32, 34, 35, 37
- Trigger bandwidth 104
- TRIGGER_CHANNEL_PROPERTIES structure 93
- TRIGGER_CONDITIONS structure 90

U

- USB 2
 - hub 18

V

- Vertical resolution 6, 104
- Voltage range 104
- Voltage ranges 4

W

- WINDOW constant 93
- Windows, Microsoft 2

United Kingdom headquarters

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

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

sales@picotech.com
support@picotech.com

www.picotech.com

United States headquarters

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

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