PicoScope 4000 Series

PC Oscilloscopes

Programmer's Guide

ps4000pg.en r8

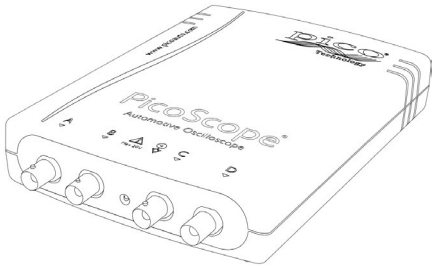

pico
Technology

Contents

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

1.2 Software licence conditions

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

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

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

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

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

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

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

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

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

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

1.3 Trademarks

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

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

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

1.4 Company details

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

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

Email:
Technical Support: support@picotech.com
Sales: sales@picotech.com

Web site: www.picotech.com

2 Product information

2.1 System requirements

Using with PicoScope for Windows

To ensure that your PicoScope 4000 Series PC Oscilloscope operates correctly with the PicoScope software, you must have a computer with at least the minimum system requirements to run one of the supported operating systems, as shown in the following table. The performance of the oscilloscope will be better with a more powerful PC. Please note the PicoScope software is not installed as part of the SDK.

Item	Specification
Operating system	Windows XP SP3, Vista, 7 or 8 32 bit and 64 bit versions supported
Processor	As required by Windows
Memory	
Free disk space	
Ports	USB 2.0 compliant port (recommended) USB 1.1 compliant port (not recommended)

Using with custom applications

Drivers are available for the operating systems mentioned above.

2.2 Installation instructions

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

Procedure

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

Checking the installation

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

Moving your PicoScope PC Oscilloscope to another USB port

- **Windows XP SP3**

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

- **Windows Vista, 7 and 8**

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

3 Programming with the PicoScope 4000 Series

The ps4000.dll dynamic link library in your PicoScope installation directory allows you to program a PicoScope 4000 Series oscilloscope using standard C function calls.

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

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

Numerous sample programs are installed with your PicoScope software. These show how to use the functions of the driver software in each of the modes available.

3.1 Driver

Your application will communicate with a PicoScope 4000 API driver called ps4000.dll. The driver exports the PicoScope 4000 function definitions in standard C format, but this does not limit you to programming in C. You can use the API with any programming language that supports standard C calls.

The API driver depends on a kernel driver, picopp.sys, which works with 32-bit Windows XP SP2, Windows Vista and Windows 7. For 64-bit versions, the API depends on the winusb.sys kernel driver. Your application does not need to call the kernel driver. Once you have installed the PicoScope 6 software, Windows automatically installs the kernel driver when you plug in the PicoScope 4000 Series PC Oscilloscope for the first time.

3.2 System requirements

General requirements

See System Requirements.

USB

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

3.3 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_MAX_VALUE or PS4262_MAX_VALUE	maximum	32 764 32 767	7FFC 7FFF
N/A	zero	0	0000
PS4000_MIN_VALUE or PS4262_MIN_VALUE	minimum	–32 764 –32 767	8004 8001
PS4000_LOST_DATA	Note 1	-32 768	8000

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

3.4 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 analogue bandwidth.
- **AC coupling:** The scope accepts input frequencies from a few hertz up to its maximum analogue bandwidth. The lower -3 dB cutoff frequency is about 1 hertz.

3.5 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 three PicoScope 4000 trigger functions. These can be run collectively by calling ps4000SetSimpleTrigger, or singularly:

- ps4000SetTriggerChannelConditions
- ps4000SetTriggerChannelDirections
- ps4000SetTriggerChannelProperties

A trigger event can occur when one of the signal or trigger input channels crosses a threshold voltage on either a rising or a falling edge.

The driver supports these triggering methods:

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

3.6 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, you can also poll the driver instead of using a callback.

3.6.1 Block mode

In **block mode**, the computer prompts a PicoScope 4000 Series PC Oscilloscope to collect a block of data into its internal memory. When the oscilloscope has collected the whole block, it signals that it is ready and then transfers the whole block to the computer's memory through the USB port.

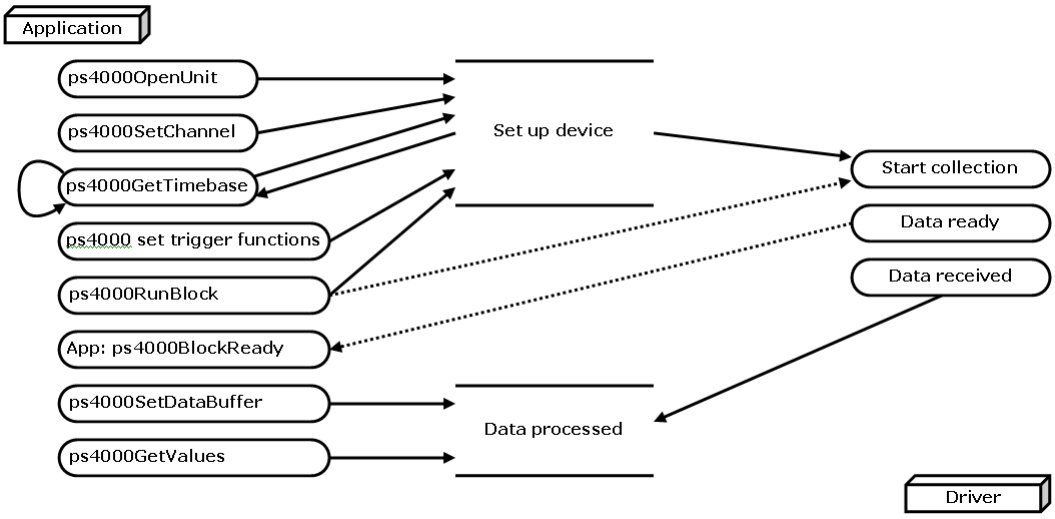
- **Block size.** The maximum number of values depends upon the size of the oscilloscope's memory. The memory buffer is shared between the enabled channels, so if two channels are enabled, each receives half the memory. These features are handled transparently by the driver. The block size also depends on the number of memory segments in use (see `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.

3.6.1.1 Using block mode

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

1. Open the oscilloscope using `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 [1] [2] [3] 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.
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.

3.6.2 Rapid block mode

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

Rapid block mode allows you to sample several waveforms at a time with the minimum time between waveforms. It reduces the gap from milliseconds to about 2.5 microseconds.

See Using rapid block mode for details.

3.6.2.1 Using rapid block mode

You can use **rapid block mode** with or without 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. Using `ps4000GetTimebase`, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions [1] [2] [3] to set up the trigger if required.
5. Set the number of memory segments equal to or greater than the number of captures required using `ps4000MemorySegments`. Use `ps4000SetNoOfCaptures` before each run to specify the number of waveforms to capture.
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.
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`.

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.

3.6.2.2 Rapid block mode example 1: no aggregation

```
#define MAX_SAMPLES 1000
```

Set up the device up as usual.

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

```
// set the number of waveforms to 100
ps4000SetNoOfCaptures (handle,  100);

pParameter = false;
ps4000RunBlock
(
  handle,
  0,                  //noOfPreTriggerSamples,
  10000,              // noOfPostTriggerSamples,
  1,                  // timebase to be used,
  1,                  // oversample
  &timeIndisposedMs,
  1,                  // oversample
  lpReady,
  &pParameter
);
```

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

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

for (int i = 0; i < 10; i++)
{
  for (int c = 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 `toSegementIndex`, 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.

3.6.2.3 Rapid block mode example 2: using aggregation

```
#define MAX_SAMPLES 1000
```

Set up the device up as usual.

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

```
// set the number of waveforms to 100
ps4000SetNoOfCaptures (handle,  100);

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

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

```
for (int 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 (int 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.

3.6.3 ETS (Equivalent Time Sampling)

Note: ETS mode is not supported by the PicoScope 4262 oscilloscope.

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 scope hardware adds a short, variable delay, which is a small fraction of a single sampling interval, between each trigger event and the subsequent sample. This shifts each capture slightly in time so that the samples occur at slightly different times relative to those of the previous capture. The result is a larger set of samples spaced by a small fraction of the original sampling interval. 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	Available in block mode only. Not suitable for one-shot (non-repetitive) signals. Aggregation and oversampling are not supported. Edge-triggering only. Auto trigger delay (`autoTriggerMilliseconds`) is ignored.

3.6.3.1 Using ETS mode

Since ETS mode is a type of block mode, the procedure is the same as the one described in Using block mode.

3.6.4 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.

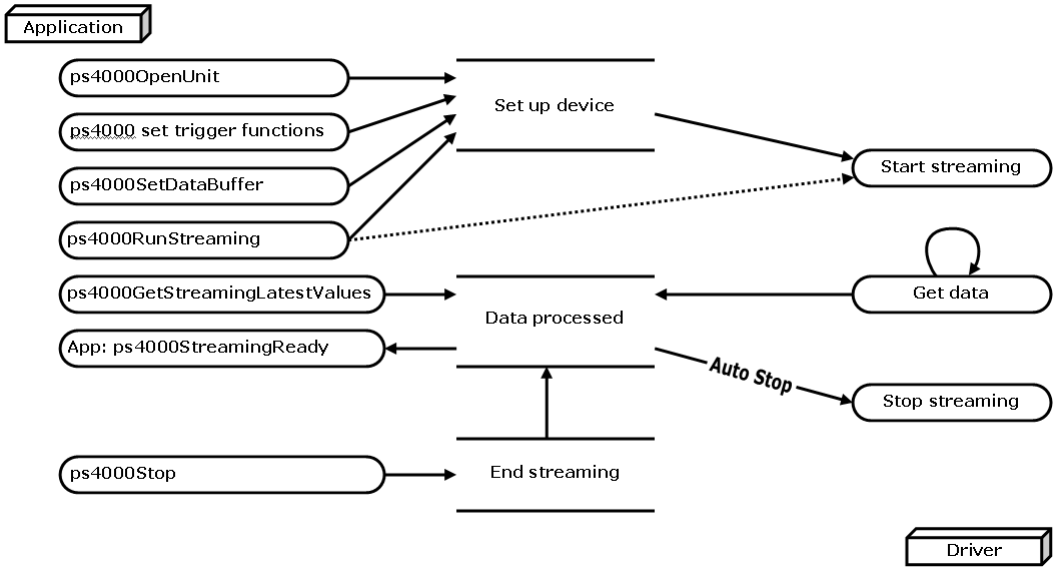
- **Memory segmentation.** The memory can be divided into segments to reduce the latency of data transfers to the PC. However, this increases the risk of losing data if the PC cannot keep up with the device's sampling rate.

See Using streaming mode for programming details.

3.6.4.1 Using streaming mode

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

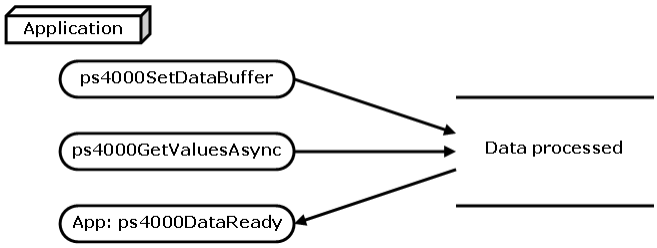
1. Open the oscilloscope using `ps4000OpenUnit`.
2. Select channels, ranges and AC/DC coupling using `ps4000SetChannel`.
3. Use the trigger setup functions [1] [2] [3] to set up the trigger if required.
4. Call `ps4000SetDataBuffer` to tell the driver where your data buffer is.
5. Set up aggregation 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 Auto Stop, so after the driver has received all the data points requested by the application, it stops the device streaming.
8. Call `ps4000Stop`, even if Auto Stop is enabled.



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

3.6.5 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`.



3.7 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.

3.8 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 sample 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 sub-range specifying a power of 2 and the high sub-range specifying a fraction of the clock frequency. The PicoScope 4262 has a single range of timebases specifying a power of 2.

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

Applicability	Use ps4000GetTimebase API call.

3.9 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.

3.10 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 OpenUnit 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
ps4000SigGenSoftwareControl: trigger the signal generator
ps4000Stop: stop data capture
ps4000StreamingReady: indicate when streaming-mode data ready

3.10.1 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	handle, the handle of the device returning the samples. status, indicates whether an error occurred during collection of the data. pParameter, a void pointer passed from ps4000RunBlock. The callback function can write to this location to send any data, such as a status flag, back to your application.
Returns	nothing

3.10.2 ps4000CloseUnit

```
PICO_STATUS ps4000CloseUnit
(
  int16_t handle
)
```

This function shuts down a PicoScope 4000 scope device.

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

3.10.3 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	`handle`, the handle of the device returning the samples. `noOfSamples`, the number of samples collected. `overflow`, returns a flag that indicates whether an overvoltage has occurred on any of the channels. It is a bit pattern with bit 0 denoting Channel A and bit 1 Channel B. `triggerAt`, an index to the buffer indicating the location of the trigger point. This parameter is valid only when `triggered` is non-zero. `triggered`, a flag indicating whether a trigger occurred. If non-zero, a trigger occurred at the location indicated by `triggerAt`. `pParameter`, 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.
Returns	nothing

3.10.4 ps4000EnumerateUnits

```
PICO_STATUS ps4000EnumerateUnits
(
  int16_t   * count,
  int8_t    * serials,
  int16_t   * serialLth
)
```

This function counts the number of PicoScope 4000 units connected to the computer, and returns a list of serial numbers as a string.

Applicability	All modes
Arguments	`* count,` on exit, the number of scopes found `* serials,` on exit, a list of serial numbers separated by commas and terminated by a final null. Example: `AQ005/139,VDR61/356,ZOR14/107`. Can be NULL on entry if serial numbers are not required. `* serialLth,` on entry, the length of the `int8_t` buffer pointed to by `serials`; on exit, the length of the string written to `serials`

Returns	`PICO_OK`
	`PICO_BUSY`
	`PICO_NULL_PARAMETER`
	`PICO_FW_FAIL`
	`PICO_CONFIG_FAIL`
	`PICO_MEMORY_FAIL`
	`PICO_ANALOG_BOARD`
	`PICO_CONFIG_FAIL_AWG`
	`PICO_INITIALISE_FPGA`

3.10.5 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	handle, the handle of the scope device start, the action required: < 0 : flash the LED indefinitely. 0 : stop the LED flashing. > 0 : flash the LED start times. If the LED is already flashing on entry to this function, the flash count will be reset to start.
Returns	PICO_OK PICO_HANDLE_INVALID PICO_BUSY

3.10.6 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	`handle`, the handle of the required device `info`, the type of information required, chosen from the list of `PS4000_CHANNEL_INFO` values `probe`, not used, must be set to 0 `ranges`, an array that will be populated with available ranges for the given value of `info`. May be `NULL`. See `PS4000_RANGE` for possible values. `length`, on entry: the length of the `ranges` array; on exit: the number of elements written to `ranges` or, if `ranges` is `NULL`, the number of elements that would have been written. `channel`, the channel for which the information is required. See `PS4000_CHANNEL` for possible values.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`

3.10.7 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	`handle`, the handle of the required device `noOfUnaggregatedSamples`, the number of unaggregated samples to be used to calculate the maximum downsampling ratio `maxDownSampleRatio`, returns the aggregation ratio `downSampleRatioMode`, see `ps4000GetValues` `segmentIndex`, the memory segment where the data is stored

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

3.10.8 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	handle, the handle of the required device. lpPs4000Ready, a pointer to your ps4000StreamingReady callback function that will return the latest aggregated values. pParameter, a void pointer that will be passed to the ps4000StreamingReady callback function.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_INVALID_CALL PICO_BUSY PICO_NOT_RESPONDING

3.10.9 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	`handle`, the handle of the required device. `timebase`, a code between 0 and $2^{30}-1$ that specifies the sampling interval (see timebase guide). `noSamples`, the number of samples required. This value is used to calculate the most suitable time unit to use. `timeIntervalNanoseconds`, a pointer to the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. `oversample`, 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. `maxSamples`, a pointer to the maximum number of samples available. The maximum samples may vary depending on the number of channels enabled, the timebase chosen and the oversample selected. If this pointer is null, nothing will be written here. `segmentIndex`, the number of the memory segment to use.

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

3.10.10 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	`timeIntervalNanoseconds`, a pointer to the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. All others as in ps4000GetTimebase.
Returns	See ps4000GetTimebase.

3.10.11 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 gets the time, as two 4-byte values, at which the trigger occurred, adjusted for the time skew of the specified channel relative to the trigger source. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture.

Applicability	Block mode, rapid block mode
Arguments	`handle`, the handle of the required device `timeUpper`, a pointer to the upper 32 bits of the time at which the trigger point occurred `timeLower`, a pointer to the lower 32 bits of the time at which the trigger point occurred `timeUnits`, returns the time units in which `timeUpper` and `timeLower` are measured. The allowable values are: `PS4000_FS`: femtoseconds `PS4000_PS`: picoseconds `PS4000_NS`: nanoseconds `PS4000_US`: microseconds `PS4000_MS`: milliseconds `PS4000_S`: seconds `segmentIndex`, the number of the memory segment for which the information is required. `channel`, the scope channel for which the information is required

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

3.10.12 ps4000GetTriggerChannelTimeOffset64

```
PICO_STATUS ps4000GetTriggerChannelTimeOffset64
(
  int16_t             handle,
  int64_t           * time,
  PS4000_TIME_UNITS * timeUnits,
  uint16_t            segmentIndex,
  PS4000_CHANNEL      channel
)
```

This function gets the time, as a single 8-byte value, at which the trigger occurred, adjusted for the time skew of the specified channel relative to the trigger source. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture.

Applicability	Block mode, rapid block mode
Arguments	`handle`, the handle of the required device `time`, a pointer to the time at which the trigger point occurred `timeUnits`, returns the time units in which `time` is measured. See ps4000GetTriggerChannelTimeOffset. `segmentIndex`, the number of the memory segment for which the information is required `channel`, the scope channel for which the information is required

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

3.10.13 ps4000GetTriggerTimeOffset

```
PICO_STATUS ps4000GetTriggerTimeOffset
(
  int16_t              handle
  uint32_t           * timeUpper
  uint32_t           * timeLower
  PS4000_TIME_UNITS  * timeUnits
  uint16_t             segmentIndex
)
```

This function gets the time, as two 4-byte values, at which the trigger occurred. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture.

Applicability	Block mode, rapid block mode
Arguments	`handle`, the handle of the required device `timeUpper`, a pointer to the upper 32 bits of the time at which the trigger point occurred `timeLower`, a pointer to the lower 32 bits of the time at which the trigger point occurred `timeUnits`, see ps4000GetTriggerChannelTimeOffset. `segmentIndex`, the number of the memory segment for which the information is required.

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

3.10.14 ps4000GetTriggerTimeOffset64

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

This function gets the time, as a single 8-byte value, at which the trigger occurred. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture.

Applicability	Block mode, rapid block mode
Arguments	`handle`, the handle of the required device `time`, a pointer to the time at which the trigger point occurred `timeUnits`, returns the time units in which `time` is measured. See ps4000GetTriggerChannelTimeOffset. `segmentIndex`, the number of the memory segment for which the information is required

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

3.10.15 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 a character string. If the device fails to open, only the driver version and error code are available to explain why the last open unit call failed.

Applicability	All modes
Arguments	`handle`, the handle of the device from which information is required. If an invalid handle is passed, the error code from the last unit that failed to open is returned. `string`, a pointer to the character string buffer in the calling function where the unit information string (selected with `info`) will be stored. If a null pointer is passed, only the `requiredSize`, pointer to an `int16_t`, of the character string buffer is returned. `stringLength`, used to return the size of the character string buffer. `requiredSize`, used to return the required character string buffer size. `info`, an enumerated type specifying what information is required from the driver.

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

PICO_INFO constant	Example
`0: PICO_DRIVER_VERSION`, version number of PicoScope 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`

3.10.16 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 scope after data collection has stopped.

Applicability	Block mode, rapid block mode
Arguments	`handle`, the handle of the required device. `startIndex`, a zero-based index that indicates the start point for data collection. It is measured in sample intervals from the start of the buffer. `noOfSamples`, on entry: the number of samples requested; on exit, the number of samples actually returned. `downSampleRatio`, the aggregation factor that will be applied to the raw data. `downSampleRatioMode`, 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) `segmentIndex`, the zero-based number of the memory segment where the data is stored. `overflow`, returns a set of flags that indicate whether an overvoltage has occurred on any of the channels. It is a bit pattern with bit 0 denoting Channel A and bit 1 Channel B.

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`

3.10.17 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 scope after data collection has stopped. It returns the data using a callback.

Applicability	Streaming mode only
Arguments	handle, the handle of the required device startIndex, see ps4000GetValues noOfSamples, see ps4000GetValues downSampleRatio, see ps4000GetValues downSampleRatioMode, see ps4000GetValues segmentIndex, see ps4000GetValues lpDataReady, a pointer to the ps4000StreamingReady function that is called when the data is ready 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.
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

3.10.18 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	`handle`, the handle of the device `* noOfSamples`, On entering the API, the number of samples required. On exiting the API, the actual number retrieved. The number of samples retrieved will not be more than the number requested. The data retrieved always starts with the first sample captured. `fromSegmentIndex`, the first segment from which the waveform should be retrieved `toSegmentIndex`, the last segment from which the waveform should be retrieved `* overflow`, equal to or larger than the number of waveforms to be retrieved. Each segment index has a separate `overflow` element, with `overflow[0]` containing the `fromSegmentIndex` and the last index the `toSegmentIndex`.

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

3.10.19 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 time offset, as lower and upper 32-bit values, for a group of waveforms obtained in rapid block mode, adjusted for the time skew relative to the trigger source. The array size for `timesUpper` and `timesLower` must be greater than or equal to the number of waveform time offsets requested. The segment indexes are inclusive.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the device `* timesUpper`, a pointer to 32-bit integers. This will hold the most significant 32 bits of the time offset for each requested segment index. `times[0]` will hold the `fromSegmentIndex` time offset and the last `times` index will hold the `toSegmentIndex` time offset. `* timesLower`, a pointer to 32-bit integers. This will hold the least-significant 32 bits of the time offset for each requested segment index. `times[0]` will hold the `fromSegmentIndex` time offset and the last `times` index will hold the `toSegmentIndex` time offset. `* timeUnits`, a pointer to a range of `PS4000_TIME_UNITS`. This must be equal to 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`. `fromSegmentIndex`, the first segment for which the time offset is required `toSegmentIndex`, the last segment for which the time offset is required. If `toSegmentIndex` is less than `fromSegmentIndex` then the driver will wrap around from the last segment to the first. `channel`, the channel for which the information is required.

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

3.10.20 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 retrieves the time offset, as a 64-bit integer, for a group of waveforms captured in rapid block mode, adjusted for the time skew relative to the trigger source. The array size of `times` must be greater than or equal to the number of waveform time offsets requested. The segment indexes are inclusive.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the device `* 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`. `* 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 `toSegmentIndex`. `fromSegmentIndex`, the first segment for which the time offset is required. The result will be placed in `times[0]` and `timeUnits[0]`. `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. `channel`, the scope channel for which information is required

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

3.10.21 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 time offset, as lower and upper 32-bit values, for a group of waveforms obtained in rapid block mode. The array size for `timesUpper` and `timesLower` must be greater than or equal to the number of waveform time offsets requested. The segment indexes are inclusive.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the device `* timesUpper`, a pointer to 32-bit integers. This will hold the most significant 32 bits of the time offset for each requested segment index. `times[0]` will hold the `fromSegmentIndex` time offset and the last `times` index will hold the `toSegmentIndex` time offset. `* timesLower`, a pointer to 32-bit integers. This will hold the least-significant 32 bits of the time offset for each requested segment index. `times[0]` will hold the `fromSegmentIndex` time offset and the last `times` index will hold the `toSegmentIndex` time offset. `* timeUnits`, a pointer to a range of `PS4000_TIME_UNITS`. This must be equal to 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`. `fromSegmentIndex`, the first segment for which the time offset is required `toSegmentIndex`, the last segment for which the time offset is required. If `toSegmentIndex` is less than `fromSegmentIndex` then the driver will wrap around from the last segment to the first.

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

3.10.22 ps4000GetValuesTriggerTimeOffsetBulk64

```
PICO_STATUS ps4000GetValuesTriggerTimeOffsetBulk64
(
  int16_t             handle,
  int64_t           * times,
  PS4000_TIME_UNITS * timeUnits,
  uint16_t            fromSegmentIndex,
  uint16_t            toSegmentIndex
)
```

This function retrieves the time offset, as a 64-bit integer, for a group of waveforms captured in rapid block mode. The array size of `times` must be greater than or equal to the number of waveform time offsets requested. The segment indexes are inclusive.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the device `* 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`. `* 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 `toSegmentIndex`. `fromSegmentIndex:`, the first segment for which the time offset is required. The result will be placed in `times[0]` and `timeUnits[0]`. `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.

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

3.10.23 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	`holdoff`, the number of samples between trigger events. The time is calculated by multiplying the sample interval by the holdoff. `type`, the type of hold-off. Only holdoff by time is currently supported: `PS4000_TIME`

Returns	`PICO_OK` - success
	`PICO_DRIVER_FUNCTION`
	`PICO_INVALID_PARAMETER`

3.10.24 ps4000IsLedFlashing

```
PICO_STATUS ps4000IsLedFlashing
(
  int16_t   handle,
  int16_t * status
)
```

This function reports whether or not the LED is flashing.

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

Returns	`PICO_OK`
	`PICO_HANDLE_INVALID`
	`PICO_NULL_PARAMETER`

3.10.25 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	handle, the handle of the required device ready, on exit, indicates the state of the collection. If zero, the device is still collecting. If non-zero, the device has finished collecting and ps4000GetValues can be used to retrieve the data.
Returns	

3.10.26 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	handle, the handle of the required device triggerEnabled, 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. pulseWidthQualifierEnabled, 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.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER

3.10.27 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	`handle`, the handle of the required device `nSegments`, the number of segments to be used, from `1` to `8192` `nMaxSamples`, 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 `nMaxSamples` divided by the number of channels.

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

3.10.28 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	handle, the handle of the required device noOfValues, returns the number of samples
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_NOT_USED PICO_BUSY

3.10.29 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	`handle`, pointer to an `int16_t` 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 of the device opened) The handle number must be used in all subsequent calls to API functions to identify this scope device.

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`

3.10.30 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	status, pointer to an int16_t 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

3.10.31 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	status, pointer to a int16_t that indicates: 0 if there is already an open operation in progress 1 if the open operation is initiated serial, the serial number of the device to be opened. A null-terminated string.
Returns	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_OPERATION_FAILED

3.10.32 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	`handle`, pointer to an `int16_t` 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. `serial`, the serial number of the device to be opened. A null-terminated string.

Returns	`PICO_OK`
	`PICO_OS_NOT_SUPPORTED`
	`PICO_OPEN_OPERATION_IN_PROGRESS`
	`PICO_EEPROM_CORRUPT`
	`PICO_KERNEL_DRIVER_TOO_OLD`
	`PICO_FW_FAIL`
	`PICO_MAX_UNITS_OPENED`
	`PICO_NOT_FOUND`
	`PICO_NOT_RESPONDING`

3.10.33 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	handle, pointer to an int16_t 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. **Note:** This handle is not valid unless the function returns PICO_OK. progressPercent, pointer to an int16_t to which the percentage progress is to be written. 100% implies that the open operation is complete. complete, pointer to an int16_t that is set to 1 when the open operation has finished
Returns	PICO_OK PICO_NULL_PARAMETER PICO_OPERATION_FAILED

3.10.34 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	handle, the handle of the required device. noOfPreTriggerSamples, the number of samples to return before the trigger event. If no trigger has been set then this argument is ignored and noOfPostTriggerSamples specifies the maximum number of data points (samples) to collect. noOfPostTriggerSamples, the number of samples to be taken after a trigger event. If no trigger event is set then this specifies the maximum number of samples to be taken. If a trigger condition has been set, this specifies the number of data points (samples) to be taken after a trigger has fired, and the number of data points to be collected is: noOfPreTriggerSamples + noOfPostTriggerSamples timebase, a number in the range 0 to $2^{30}-1$. See the guide to calculating timebase values. oversample, the oversampling factor, a number in the range 1 to 16. timeIndisposedMs, returns the time, in milliseconds, that the PicoScope4000 will spend collecting samples. This does not include any auto trigger timeout. If this pointer is null, nothing will be written here. segmentIndex, zero-based, specifies which memory segment to use. lpReady, a pointer to the 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. pParameter, a void pointer that is passed to the ps4000BlockReady callback function. The callback can use the pointer to return arbitrary data to your application.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_CHANNEL PICO_INVALID_TRIGGER_CHANNEL PICO_INVALID_CONDITION_CHANNEL PICO_TOO_MANY_SAMPLES PICO_INVALID_TIMEBASE PICO_NOT_RESPONDING PICO_CONFIG_FAIL PICO_INVALID_PARAMETER PICO_NOT_RESPONDING PICO_TRIGGER_ERROR

3.10.35 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	handle, the handle of the required device. sampleInterval, a pointer to the requested time interval between data points on entry and the actual time interval assigned on exit. sampleIntervalTimeUnits, the unit of time that the sampleInterval is set to. Use one of these values: PS4000_FS PS4000_PS PS4000_NS PS4000_US PS4000_MS PS4000_S maxPreTriggerSamples, the maximum number of raw samples before a trigger condition for each enabled channel. If no trigger condition is set this argument is ignored. maxPostTriggerSamples, the maximum number of raw samples after a trigger condition for each enabled channel. If no trigger condition is set this argument states the maximum number of samples to be stored. autoStop, a flag to specify if the streaming should stop when all of maxSamples have been taken. downSampleRatio, the number of raw values to each aggregated value. overviewBufferSize, the size of the overview buffers. These are temporary buffers used for storing the data before returning it to the application. The size is the same as the bufferLth value passed to ps4000SetDataBuffer.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_STREAMING_FAILED PICO_NOT_RESPONDING PICO_TRIGGER_ERROR PICO_INVALID_SAMPLE_INTERVAL PICO_INVALID_BUFFER

3.10.36 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	handle, the handle of the required device. sampleInterval, a pointer to the requested time interval between data points on entry and the actual time interval assigned on exit. sampleIntervalTimeUnits, the unit of time that the sampleInterval is set to. Use one of these values: PS4000_FS PS4000_PS PS4000_NS PS4000_US PS4000_MS PS4000_S maxPreTriggerSamples, the maximum number of raw samples before a trigger condition for each enabled channel. If no trigger condition is set this argument is ignored. maxPostTriggerSamples, the maximum number of raw samples after a trigger condition for each enabled channel. If no trigger condition is set this argument states the maximum number of samples to be stored. autoStop, a flag to specify if the streaming should stop when all of maxSamples have been taken. downSampleRatio, the number of raw values to each aggregated value. downSampleRatioMode, the data reduction mode to use. overviewBufferSize, the size of the overview buffers. These are temporary buffers used for storing the data before returning it to the application. The size is the same as the bufferLth value passed to ps4000SetDataBuffer.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_STREAMING_FAILED PICO_NOT_RESPONDING PICO_TRIGGER_ERROR PICO_INVALID_SAMPLE_INTERVAL PICO_INVALID_BUFFER

3.10.37 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	`handle`, the handle of the required device
	
	`channel`, an enumerated type. The values are:
	`PS4000_CHANNEL_A`
	`PS4000_CHANNEL_B`
	
	`enable`, whether to enable or disable the filter:
	`TRUE` = enable
	`FALSE` = disable

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

3.10.38 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	`handle`, the handle of the required device
	
	`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)
	
	`enabled`, specifies if the channel is active. The values are: -
	`TRUE` = active
	`FALSE` = inactive
	
	`dc`, specifies the AC/DC coupling mode. The values are: -
	`TRUE` = DC
	`FALSE` = AC
	
	`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_`.

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

3.10.39 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	`handle`, the handle of the required device
	
	`channel`, 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)
	
	`buffer`, a buffer to receive the data values
	
	`bufferLth`, the size of the `buffer` array

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`

3.10.40 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	`handle`, the handle of the device `channel`, 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. `* buffer`, an array in which the captured data is stored `bufferLth`, the size of the buffer `waveform`, an index to the waveform number, between `0` and `nCaptures-1`

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_PARAMETER`

3.10.41 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	`handle`, the handle of the required device.
	
	`channel`, 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)
	
	`bufferMax`, a buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise.
	
	`bufferMin`, a buffer to receive the minimum data values when `downSampleRatio` > 1. Not used when `downSampleRatio` is 1.
	
	`bufferLth`, specifies the size of the `bufferMax` and `bufferMin` arrays.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`

3.10.42 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	`handle`, the handle of the required device.
	
	`channel`, 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)
	
	`bufferMax`, a buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise.
	
	`bufferMin`, a buffer to receive the minimum data values when `downSampleRatio` > 1. Not used when `downSampleRatio` is 1.
	
	`bufferLth`, specifies the size of the `bufferMax` and `bufferMin` arrays.
	
	`mode`, the data reduction mode to use

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`

3.10.43 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	`handle`, the handle of the required device
	
	`channel`, 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)
	
	`buffer`, a buffer to receive the data values
	
	`bufferLth`, the size of the `buffer` array
	
	`mode`, the type of data reduction to use

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`

3.10.44 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 not supported by PicoScope 4262.
Arguments	`handle`, the handle of the required device `mode`, the ETS mode. Use one of these values: - `PS4000_ETS_OFF` disables ETS `PS4000_ETS_FAST` enables ETS and provides `ets_cycles` cycles of data, which may contain data from previously returned cycles `PS4000_ETS_SLOW` enables ETS and provides fresh data every `ets_cycles` 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. `ets_cycles`, the number of cycles to store: the computer can then select `ets_interleave` cycles to give the most uniform spread of samples. `ets_cycles` should be between two and five times the value of `ets_interleave`. `ets_interleave`, 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. `sampleTimePicoseconds`, returns the effective sample time used by the function

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

3.10.45 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 mode is not supported by the PicoScope 4262 oscilloscope. If your programming language does not support 64-bit data, use the 32-bit version ps4000SetEtsTimeBuffers instead.
Arguments	handle, the handle of the required device buffer, a pointer to a set of 8-byte words, the time in nanoseconds at which the first data point occurred bufferLth, the size of the buffer array
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER

3.10.46 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.

Note: ETS mode is not supported by the PicoScope 4262 oscilloscope.

Applicability	ETS mode only. If your programming language supports 64-bit data, then you can use `ps4000SetEtsTimeBuffer` instead.
Arguments	`handle`, the handle of the required device `timeUpper`, a pointer to a set of 4-byte words, the time in nanoseconds at which the first data point occurred, top 32 bits only `timeLower`, a pointer to a set of 4-byte words, the time in nanoseconds at which the first data point occurred, bottom 32 bits only `bufferLth`, the size of the `timeUpper` and `timeLower` arrays

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`

3.10.47 ps4000SetExtTriggerRange

```
PICO_STATUS ps4000SetExtTriggerRange
(
  int16_t           handle,
  PS4000_RANGE      extRange
)
```

This function sets the range of the external trigger.

Applicability	PicoScope 4262 only
Arguments	`handle`, the handle of the required oscilloscope `extRange`, specifies the range for the external trigger (±500 mV or ±5 V)
Returns	`PICO_OK` `PICO_INVALID_PARAMETER`

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

3.10.48 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`, the handle of the device `nCaptures`, the number of waveforms to be captured in one run

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`

3.10.49 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 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	`handle`, the handle of the required device `conditions`, 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 `conditions` is set to `null` then the pulse width qualifier is not used. `nConditions`, the number of elements in the `conditions` array. If `nConditions` is zero then the pulse width qualifier is not used. `direction`, the direction of the signal required for the trigger to fire `lower`, the lower limit of the pulse width counter `upper`, the upper limit of the pulse width counter. This parameter is used only when the type is set to `PW_TYPE_IN_RANGE` or `PW_TYPE_OUT_OF_RANGE`. `type`, the pulse width type, one of these constants: - `PW_TYPE_NONE` (do not use the pulse width qualifier) `PW_TYPE_LESS_THAN` (pulse width less than `lower`) `PW_TYPE_GREATER_THAN` (pulse width greater than `lower`) `PW_TYPE_IN_RANGE` (pulse width between `lower` and `upper`) `PW_TYPE_OUT_OF_RANGE` (pulse width not between `lower` and `upper`)

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

3.10.49.1 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.

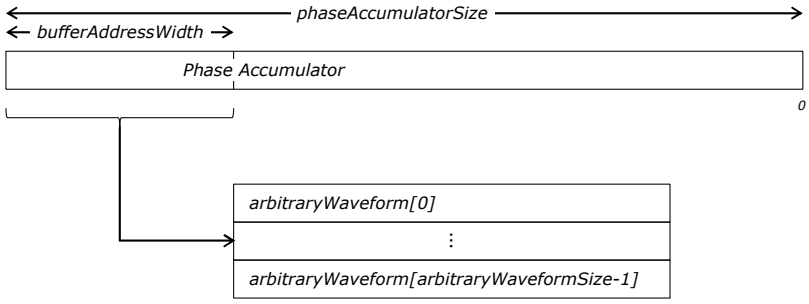
Elements	channelA, channelB, channelC, channelD: the type of condition that should be applied to each channel. Use these constants: - CONDITION_DONT_CARE CONDITION_TRUE CONDITION_FALSE The channels that are set to CONDITION_TRUE or CONDITION_FALSE must all meet their conditions simultaneously to produce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

3.10.50 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 functions instructs the signal generator to produce an arbitrary waveform.

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



The generator steps through the waveform by adding a *deltaPhase* value between 1 and $2^{phaseAccumulatorSize}-1$ to the phase accumulator every clock period (*dacPeriod*). If the *deltaPhase* is constant, the generator produces a waveform at a constant frequency that can be calculated as follows:

$$outputFrequency = \frac{dacFrequency}{arbitraryWaveformSize} \times \frac{deltaPhase}{2^{(phaseAccumulatorSize-bufferAddressWidth)}}$$

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
phaseAccumulatorSize	32 bits	32 bits
bufferAddressWidth	13 bits	12 bits
dacFrequency	20 MHz	192 kHz
dacPeriod (= 1/*dacFrequency*)	50 ns	≈ 5.208 µs

Applicability	PicoScope 4226, 4227 and 4262 only

Arguments

`handle`, the handle of the required device.

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

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

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

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

`deltaPhaseIncrement`, the amount added to the delta phase value every time the `dwellCount` period expires. This determines the amount by which the generator sweeps the output frequency in each dwell period. When frequency sweeping is not required, set to zero.

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

PicoScope 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. 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_AUX_IN`:	the AUX input
`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:

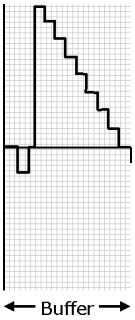
PicoScope 4226 & 4227:	±20 V
PicoScope 4262:	±500 mV or ±5 V depending on range selected by ps4000SetExtTriggerRange()

Returns	0: if successful. Error code: if failed

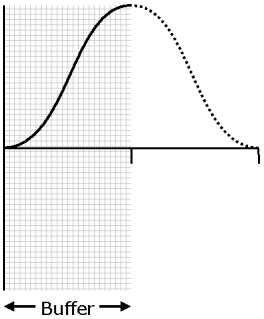
3.10.50.1 AWG index modes

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

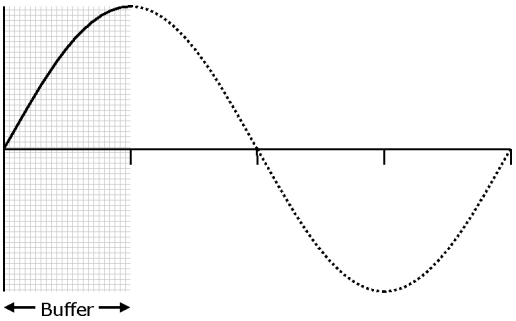
`SINGLE` **mode**. The generator outputs the raw contents of the buffer repeatedly. This mode is the only one that can generate asymmetrical waveforms. You can also use this mode for symmetrical waveforms, but the dual and quad modes make more efficient use of the buffer memory.



`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.



3.10.51 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	`handle`, the handle of the required oscilloscope. `offsetVoltage`, the voltage offset, in microvolts, to be applied to the waveform. `pkToPk`, the peak-to-peak voltage, in microvolts, of the waveform signal. `waveType`, the type of waveform to be generated by the oscilloscope: `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 `startFrequency`, the frequency at which the signal generator should begin. Range: `MIN_SIG_GEN_FREQ` to `MAX_SIG_GEN_FREQ`. `stopFrequency`, 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`. `increment`, the amount by which the frequency rises or falls every `dwellTime` seconds in sweep mode.

	`dwellTime`, the time in seconds between frequency changes in sweep mode. `sweepType`, see ps4000SetSigGenArbitrary `operationType`, see ps4000SetSigGenArbitrary `shots`, see ps4000SigGenArbitrary `sweeps`, see ps4000SigGenArbitrary `triggerType`, see ps4000SigGenArbitrary `triggerSource`, see ps4000SigGenArbitrary `extInThreshold`, see ps4000SigGenArbitrary
Returns	0: if successful. Error code: if failed.

3.10.52 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 cancelled.

Applicability	All modes
Arguments	`handle`, the handle of the required device. `enabled`, zero to disable the trigger, any non-zero value to set the trigger. `source`, the channel on which to trigger. `threshold`, the ADC count at which the trigger will fire. `direction`, the direction in which the signal must move to cause a trigger. The following directions are supported: `ABOVE`, `BELOW`, `RISING`, `FALLING` and `RISING_OR_FALLING`. `delay`, the time, in sample periods, between the trigger occurring and the first sample being taken. `autoTrigger_ms`, the number of milliseconds the device will wait if no trigger occurs.

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

3.10.53 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	`handle`, the handle of the required device. `conditions`, 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. `nConditions`, the number of elements in the `conditions` array. If `nConditions` is zero then triggering is switched off.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_MEMORY_FAIL

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

3.10.54 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	`handle`, the handle of the required device `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. `aux`, not used

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

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

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

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

3.10.55.1 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	thresholdUpper, the upper threshold at which the trigger must fire. This is scaled in 16-bit ADC counts at the currently selected range for that channel. thresholdUpperHysteresis, the hysteresis by which the trigger must exceed the upper threshold before it will fire. It is scaled in 16-bit counts. thresholdLower, the lower threshold at which the trigger must fire. This is scaled in 16-bit ADC counts at the currently selected range for that channel. thresholdLowerHysteresis, the hysteresis by which the trigger must exceed the lower threshold before it will fire. It is scaled in 16-bit counts. channel, the channel to which the properties apply. See ps4000SetChannel for possible values. thresholdMode, either a level or window trigger. Use one of these constants: - LEVEL WINDOW

3.10.56 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	All modes
Arguments	`handle`, the handle of the required device `delay`, the time between the trigger occurring and the first sample, in sample periods. For example, if `delay`=100 then the scope would wait 100 sample periods before sampling. Example: with the PicoScope 4224, at a timebase of 80 MS/s, or 12.5 ns per sample `(timebase=0)` the total delay would then be 100 x 12.5 ns = 1.25 µs.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`

3.10.57 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	handle, the handle of the required device state, sets the trigger gate high or low when the trigger type is set to either SIGGEN_GATE_HIGH or SIGGEN_GATE_LOW. Ignored for other trigger types.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_TRIGGER_SOURCE

3.10.58 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.

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

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

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`

3.10.59 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.

Applicability	Streaming mode only
Arguments	`handle`, the handle of the device returning the samples. `noOfSamples`, the number of samples to collect. `startIndex`, an index to the first valid sample in the buffer. This is the buffer that was previously passed to ps4000SetDataBuffer. `overflow`, returns a set of flags that indicate whether an overvoltage has occurred on any of the channels. It is a bit pattern with bit 0 corresponding to Channel A and so on. `triggerAt`, an index to the buffer indicating the location of the trigger point relative to `startIndex`. This parameter is valid only when `triggered` is non-zero. `triggered`, a flag indicating whether a trigger occurred. If non-zero, a trigger occurred at the location indicated by `triggerAt`. `autoStop`, the flag that was set in the call to ps4000RunStreaming. `pParameter`, 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.
Returns	nothing

3.11 Enumerated types and constants

The following types and constants are defined in the file `ps4000Api.h`, which is included in the SDK.

```
#define PS4000_MAX_OVERSAMPLE_12BIT     16
#define PS4000_MAX_OVERSAMPLE_8BIT      256

#define PS4XXX_MAX_ETS_CYCLES           400
#define PS4XXX_MAX_INTERLEAVE           80

#define PS4000_MAX_VALUE                32764
#define PS4000_MIN_VALUE                -32764
#define PS4000_LOST_DATA                -32768

#define PS4262_MAX_VALUE                32767
#define PS4262_MIN_VALUE                -32767

#define PS4000_EXT_MAX_VALUE            32767
#define PS4000_EXT_MIN_VALUE            -32767

#define MAX_PULSE_WIDTH_QUALIFIER_COUNT 16777215L
#define MAX_DELAY_COUNT                 8388607L

#define MIN_SIG_GEN_FREQ                0.0f
#define MAX_SIG_GEN_FREQ                100000.0f
#define MAX_SIG_GEN_FREQ_4262           20000.0f

#define MAX_SIG_GEN_BUFFER_SIZE         8192
#define PS4262_MAX_WAVEFORM_BUFFER_SIZE 4096
#define MIN_SIG_GEN_BUFFER_SIZE         1
#define MIN_DWELL_COUNT                 10
#define PS4262_MIN_DWELL_COUNT          3
#define MAX_SWEEPS_SHOTS                ((1 << 30) - 1)

typedef enum enChannelBufferIndex
{
  PS4000_CHANNEL_A_MAX,
  PS4000_CHANNEL_A_MIN,
  PS4000_CHANNEL_B_MAX,
  PS4000_CHANNEL_B_MIN,
  PS4000_CHANNEL_C_MAX,
  PS4000_CHANNEL_C_MIN,
  PS4000_CHANNEL_D_MAX,
  PS4000_CHANNEL_D_MIN,
  PS4000_MAX_CHANNEL_BUFFERS
} PS4000_CHANNEL_BUFFER_INDEX;

typedef enum enPS4000Channel
{
  PS4000_CHANNEL_A,
  PS4000_CHANNEL_B,
  PS4000_CHANNEL_C,
  PS4000_CHANNEL_D,
  PS4000_EXTERNAL,
  PS4000_MAX_CHANNELS = PS4000_EXTERNAL,
  PS4000_TRIGGER_AUX,
  PS4000_MAX_TRIGGER_SOURCES
```

```
} PS4000_CHANNEL;

typedef enum enPS4000Range
{
      PS4000_10MV,
      PS4000_20MV,
      PS4000_50MV,
      PS4000_100MV,
      PS4000_200MV,
      PS4000_500MV,
      PS4000_1V,
      PS4000_2V,
      PS4000_5V,
      PS4000_10V,
      PS4000_20V,
      PS4000_50V,
      PS4000_100V,
      PS4000_MAX_RANGES,

      PS4000_RESISTANCE_100R = PS4000_MAX_RANGES,
      PS4000_RESISTANCE_1K,
      PS4000_RESISTANCE_10K,
      PS4000_RESISTANCE_100K,
      PS4000_RESISTANCE_1M,
      PS4000_MAX_RESISTANCES,

      PS4000_ACCELEROMETER_10MV = PS4000_MAX_RESISTANCES,
      PS4000_ACCELEROMETER_20MV,
      PS4000_ACCELEROMETER_50MV,
      PS4000_ACCELEROMETER_100MV,
      PS4000_ACCELEROMETER_200MV,
      PS4000_ACCELEROMETER_500MV,
      PS4000_ACCELEROMETER_1V,
      PS4000_ACCELEROMETER_2V,
      PS4000_ACCELEROMETER_5V,
      PS4000_ACCELEROMETER_10V,
      PS4000_ACCELEROMETER_20V,
      PS4000_ACCELEROMETER_50V,
      PS4000_ACCELEROMETER_100V,
      PS4000_MAX_ACCELEROMETER,

      PS4000_TEMPERATURE_UPTO_40 = PS4000_MAX_ACCELEROMETER,
      PS4000_TEMPERATURE_UPTO_70,
      PS4000_TEMPERATURE_UPTO_100,
      PS4000_TEMPERATURE_UPTO_130,
      PS4000_MAX_TEMPERATURES,

      PS4000_RESISTANCE_5K = PS4000_MAX_TEMPERATURES,
      PS4000_RESISTANCE_25K,
      PS4000_RESISTANCE_50K,
      PS4000_MAX_EXTRA_RESISTANCES
} PS4000_RANGE;

typedef enum enPS4000Probe
{
  P_NONE,
  P_CURRENT_CLAMP_10A,
  P_CURRENT_CLAMP_1000A,
  P_TEMPERATURE_SENSOR,
```



```
  P_CURRENT_MEASURING_DEVICE,
  P_PRESSURE_SENSOR_50BAR,
  P_PRESSURE_SENSOR_5BAR,
  P_OPTICAL_SWITCH,
  P_UNKNOWN,
  P_MAX_PROBES = P_UNKNOWN
} PS4000_PROBE;

typedef enum enPS4000ChannelInfo
{
  CI_RANGES,
  CI_RESISTANCES,
  CI_ACCELEROMETER,
  CI_PROBES,
  CI_TEMPERATURES
} PS4000_CHANNEL_INFO;

typedef enum enPS4000EtsMode
  {
  PS4000_ETS_OFF,             // ETS disabled
  PS4000_ETS_FAST,
  PS4000_ETS_SLOW,
  PS4000_ETS_MODES_MAX
  } PS4000_ETS_MODE;

typedef enum enPS4000TimeUnits
  {
  PS4000_FS,
  PS4000_PS,
  PS4000_NS,
  PS4000_US,
  PS4000_MS,
  PS4000_S,
  PS4000_MAX_TIME_UNITS,
  } PS4000_TIME_UNITS;

typedef enum enSweepType
{
  UP,
  DOWN,
  UPDOWN,
  DOWNUP,
  MAX_SWEEP_TYPES
} SWEEP_TYPE;

typedef enum enPS4000OperationTypes
{
  PS4000_OP_NONE,
  PS4000_WHITENOISE,
  PS4000_PRBS
} PS4000_OPERATION_TYPES;

typedef enum enWaveType
{
  PS4000_SINE,
  PS4000_SQUARE,
  PS4000_TRIANGLE,
  PS4000_RAMP_UP,
  PS4000_RAMP_DOWN,
```

```
  PS4000_SINC,
  PS4000_GAUSSIAN,
  PS4000_HALF_SINE,
  PS4000_DC_VOLTAGE,
  PS4000_WHITE_NOISE,
  MAX_WAVE_TYPES
} WAVE_TYPE;

typedef enum enSigGenTrigType
{
  SIGGEN_RISING,
  SIGGEN_FALLING,
  SIGGEN_GATE_HIGH,
  SIGGEN_GATE_LOW
} SIGGEN_TRIG_TYPE;

typedef enum enSigGenTrigSource
{
  SIGGEN_NONE,
  SIGGEN_SCOPE_TRIG,
  SIGGEN_AUX_IN,
  SIGGEN_EXT_IN,
  SIGGEN_SOFT_TRIG
} SIGGEN_TRIG_SOURCE;

typedef enum enIndexMode
{
  SINGLE,
  DUAL,
  QUAD,
  MAX_INDEX_MODES
} INDEX_MODE;

typedef enum enThresholdMode
{
  LEVEL,
  WINDOW
} THRESHOLD_MODE;

typedef enum enThresholdDirection
{
  ABOVE,             // using upper threshold
  BELOW,
  RISING,            // using upper threshold
  FALLING,           // using upper threshold
  RISING_OR_FALLING, // using both threshold
  ABOVE_LOWER,       // using lower threshold
  BELOW_LOWER,       // using lower threshold
  RISING_LOWER,      // using upper threshold
  FALLING_LOWER,     // using upper threshold

  // Windowing using both thresholds
  INSIDE = ABOVE,
  OUTSIDE = BELOW,
  ENTER = RISING,
  EXIT = FALLING,
  ENTER_OR_EXIT = RISING_OR_FALLING,
  POSITIVE_RUNT = 9,
  NEGATIVE_RUNT,
```



```
  // no trigger set
  NONE = RISING
} THRESHOLD_DIRECTION;

typedef enum enTriggerState
{
  CONDITION_DONT_CARE,
  CONDITION_TRUE,
  CONDITION_FALSE,
  CONDITION_MAX
} TRIGGER_STATE;

#pragma pack(1)
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;
#pragma pack()

#pragma pack(1)
typedef struct tPwqConditions
{
  TRIGGER_STATE channelA;
  TRIGGER_STATE channelB;
  TRIGGER_STATE channelC;
  TRIGGER_STATE channelD;
  TRIGGER_STATE external;
  TRIGGER_STATE aux;
} PWQ_CONDITIONS;
#pragma pack()

#pragma pack(1)
typedef struct tTriggerChannelProperties
{
  int16_t          thresholdUpper;
  uint16_t thresholdUpperHysteresis;
  int16_t          thresholdLower;
  uint16_t thresholdLowerHysteresis;
  PS4000_CHANNEL channel;
  THRESHOLD_MODE thresholdMode;
} TRIGGER_CHANNEL_PROPERTIES;
#pragma pack()

typedef enum enRatioMode
{
  RATIO_MODE_NONE,
  RATIO_MODE_AGGREGATE = 1,
  RATIO_MODE_AVERAGE = 2
} RATIO_MODE;

typedef enum enPulseWidthType
{
```

```
  PW_TYPE_NONE,
  PW_TYPE_LESS_THAN,
  PW_TYPE_GREATER_THAN,
  PW_TYPE_IN_RANGE,
  PW_TYPE_OUT_OF_RANGE
} PULSE_WIDTH_TYPE;

typedef enum enPs4000HoldOffType
{
  PS4000_TIME,
  PS4000_MAX_HOLDOFF_TYPE
} PS4000_HOLDOFF_TYPE;

typedef enum enPS4000FrequencyCounterRange
{
  FC_2K,
  FC_20K,
  FC_20,
  FC_200,
  FC_MAX
} PS4000_FREQUENCY_COUNTER_RANGE;
```

ps4000Api.h issue: 1.36 8/9/2011, updated 12/9/12

3.12 Driver error codes

This description of the **driver error codes** is aimed at those people who intend to write their own programs for use with the driver. Every function in the ps4000.dll driver returns an error code from the following list of PICO_STATUS values. They are declared in the picoStatus.h header file supplied with the SDK.

Code (hex)	Enum
00	PICO_OK. The PicoScope 4000 is functioning correctly.
01	PICO_MAX_UNITS_OPENED. An attempt has been made to open more than PS4000_MAX_UNITS. (Reserved)
02	PICO_MEMORY_FAIL. Not enough memory could be allocated on the host machine.
03	PICO_NOT_FOUND. No PicoScope 4000 could be found.
04	PICO_FW_FAIL. Unable to download firmware.
05	PICO_OPEN_OPERATION_IN_PROGRESS. The driver is busy opening a device.
06	PICO_OPERATION_FAILED. An unspecified error occurred.
07	PICO_NOT_RESPONDING. The PicoScope 4000 is not responding to commands from the PC.
08	PICO_CONFIG_FAIL. The configuration information in the PicoScope 4000 has become corrupt or is missing.
09	PICO_KERNEL_DRIVER_TOO_OLD. The picopp.sys file is too old to be used with the device driver.
0A	PICO_EEPROM_CORRUPT. The EEPROM has become corrupt, so the device will use a default setting.
0B	PICO_OS_NOT_SUPPORTED. The operating system on the PC is not supported by this driver.
0C	PICO_INVALID_HANDLE. There is no device with the handle value passed.
0D	PICO_INVALID_PARAMETER. A parameter value is not valid.
0E	PICO_INVALID_TIMEBASE. The time base is not supported or is invalid.
0F	PICO_INVALID_VOLTAGE_RANGE. The voltage range is not supported or is invalid.
10	PICO_INVALID_CHANNEL. The channel number is not valid on this device or no channels have been set.
11	PICO_INVALID_TRIGGER_CHANNEL. The channel set for a trigger is not available on this device.
12	PICO_INVALID_CONDITION_CHANNEL. The channel set for a condition is not available on this device.
13	PICO_NO_SIGNAL_GENERATOR. The device does not have a signal generator.
14	PICO_STREAMING_FAILED. Streaming has failed to start or has stopped without user request.
15	PICO_BLOCK_MODE_FAILED. Block failed to start - a parameter may have been set wrongly.
16	PICO_NULL_PARAMETER. A parameter that was required is NULL.
17	PICO_ETS_MODE_SET. The function call failed because ETS mode is being used.
18	PICO_DATA_NOT_AVAILABLE. No data is available from a run block call.
19	PICO_STRING_BUFFER_TOO_SMALL. The buffer passed was too small for the string to be returned.
1A	PICO_ETS_NOT_SUPPORTED. ETS is not supported on this device variant.
1B	PICO_AUTO_TRIGGER_TIME_TOO_SHORT. The auto trigger time is less than the time it will take to collect the data.
1C	PICO_BUFFER_STALL. The collection of data has stalled as unread data would be overwritten.
1D	PICO_TOO_MANY_SAMPLES. Number of samples requested is more than available in the current memory segment.

1E	PICO_TOO_MANY_SEGMENTS. Not possible to create number of segments requested.
1F	PICO_PULSE_WIDTH_QUALIFIER. A null pointer has been passed in the trigger function or one of the parameters is out of range.
20	PICO_DELAY. One or more of the hold-off parameters are out of range.
21	PICO_SOURCE_DETAILS. One or more of the source details are incorrect.
22	PICO_CONDITIONS. One or more of the conditions are incorrect.
23	PICO_USER_CALLBACK. The driver's thread is currently in the ps4000...Ready callback function and therefore the action cannot be carried out.
24	PICO_DEVICE_SAMPLING. An attempt is being made to get stored data while streaming. Either stop streaming by calling ps4000Stop, or use ps4000GetStreamingLatestValues.
25	PICO_NO_SAMPLES_AVAILABLE. ...because a run has not been completed.
26	PICO_SEGMENT_OUT_OF_RANGE. The memory index is out of range.
27	PICO_BUSY. The driver cannot return data yet.
28	PICO_STARTINDEX_INVALID. The start time to get stored data is out of range.
29	PICO_INVALID_INFO. The information number requested is not a valid number.
2A	PICO_INFO_UNAVAILABLE. The handle is invalid so no information is available about the device. Only PICO_DRIVER_VERSION is available.
2B	PICO_INVALID_SAMPLE_INTERVAL. The sample interval selected for streaming is out of range.
2C	PICO_TRIGGER_ERROR. ETS is set but no trigger has been set. A trigger setting is required for ETS.
2D	PICO_MEMORY. Driver cannot allocate memory
2E	PICO_SIG_GEN_PARAM. Error in signal generator parameter
2F	PICO_SHOTS_SWEEPS_WARNING. The signal generator will output the signal required but sweeps and shots will be ignored. Only one parameter can be non-zero.
30	PICO_SIGGEN_TRIGGER_SOURCE. A software trigger has been sent but the trigger source is not a software trigger.
31	PICO_AUX_OUTPUT_CONFLICT. A ps4000SetTrigger... call has found a conflict between the trigger source and the AUX output enable.
32	PICO_AUX_OUTPUT_ETS_CONFLICT. ETS mode is being used and AUX is set as an input.
33	PICO_WARNING_EXT_THRESHOLD_CONFLICT. The EXT threshold is being set in both a ps4000SetTrigger... function and in the signal generator but the threshold values differ. The last value set will be used.
34	PICO_WARNING_AUX_OUTPUT_CONFLICT. A ps4000SetTrigger... function has set AUX as an output and the signal generator is using it as a trigger.
35	PICO_SIGGEN_OUTPUT_OVER_VOLTAGE. The requested voltage and offset levels combine to give an overvoltage.
36	PICO_DELAY_NULL. NULL pointer passed as delay parameter.
37	PICO_INVALID_BUFFER. The buffers for overview data have not been set while streaming.
38	PICO_SIGGEN_OFFSET_VOLTAGE. The signal generator offset voltage is higher than allowed.
39	PICO_SIGGEN_PK_TO_PK. The signal generator peak-to-peak voltage is higher than allowed.
3A	PICO_CANCELLED. A block collection has been cancelled.
3B	PICO_SEGMENT_NOT_USED. The specified segment index is not in use.
3C	PICO_INVALID_CALL. The wrong GetValues function has been called for the collection mode in use.
3D	PICO_GET_VALUES_INTERRUPTED
3F	PICO_NOT_USED. The function is not available.

40	PICO_INVALID_SAMPLERATIO. The aggregation ratio requested is out of range.
41	PICO_INVALID_STATE. Device is in an invalid state.
42	PICO_NOT_ENOUGH_SEGMENTS. The number of segments allocated is fewer than the number of captures requested.
43	PICO_DRIVER_FUNCTION. You called a driver function while another driver function was still being processed.
44	PICO_RESERVED
45	PICO_INVALID_COUPLING. The dc argument passed to ps4000SetChannel was invalid.
46	PICO_BUFFERS_NOT_SET. Memory buffers were not set up before calling one of the ps4000Run... functions.
47	PICO_RATIO_MODE_NOT_SUPPORTED. downSampleRatioMode is not valid for the connected device.
48	PICO_RAPID_NOT_SUPPORT_AGGREGATION. Aggregation was requested in rapid block mode.
49	PICO_INVALID_TRIGGER_PROPERTY. An incorrect value was passed to ps4000SetTriggerChannelProperties.

picoStatus.h revision 1.36, 8/9/2011

3.13 Programming examples

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

- C
- Excel
- LabView

3.13.1 C

The SDK includes a console-mode program (`ps4000con.c`) that demonstrates how to use the PicoScope 4000 driver in Windows. The program demonstrates the following procedures:

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

To build this application:

- Set up a project for a 32-bit console mode application
- Add `ps4000con.c` to the project
- Add `ps4000.lib` to the project (Microsoft C only)
- Add `ps4000Api.h` and `picoStatus.h` to the project
- Build the project

3.13.2 Excel

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

1. Copy the following files from the SDK to a location that is on your Windows execution path (for example, C:\windows\system32):

- `ps4000wrap.dll`
- `ps4000.dll`
- `PicoIpp.dll`

2. Load the spreadsheet `ps4000.xls`
3. Select **Tools > Macro**
4. Select **GetData**
5. Select **Run**

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

3.13.3 LabVIEW

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

The LabVIEW library (`PicoScope4000.llb`) can be placed in the `user.lib` sub-directory to make the VIs available on the 'User Libraries' palette. You must also copy `ps4000.dll` and `ps4000wrap.dll` to the folder containing your LabView project.

The library contains the following VIs:

- `PicoErrorHandler.vi` - takes an error cluster and, if an error has occurred, displays a message box indicating the source of the error and the status code returned by the driver
- `PicoScope4000AdvancedTriggerSettings.vi` - an interface for the advanced trigger features of the oscilloscope

  This VI is not required for setting up simple triggers, which are configured using `PicoScope4000Settings.vi`.

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

  ```
  ps4000SetTriggerChannelConditions
  ps4000SetTriggerChannelDirections
  ps4000SetTriggerChannelProperties
  ps4000SetPulseWidthQualifier
  ps4000SetTriggerDelay
  ```

- `PicoScope4000AWG.vi` - controls the arbitrary waveform generator

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

  When selecting an arbitrary waveform, it is necessary to specify a text file containing the waveform. This text file should have a single value on each line in the range -1 to 1. For further information on the settings, see descriptions of `ps4000SetSigGenBuiltIn` and `ps4000SetSigGenArbitrary`.

- `PicoScope4000Close.vi` - closes the oscilloscope

  Should be called before exiting an application.

- `PicoScope4000GetBlock.vi` - collects a block of data from the oscilloscope

  This can be called in a loop in order to continually collect blocks of data. The oscilloscope should first be set up by using `PicoScope4000Settings.vi`. The VI outputs data arrays in two clusters (max and min). If not using aggregation, 'Min Buffers' is not used.

- `PicoScope4000GetRapidBlock.vi` - collects a set of data blocks or captures from the oscilloscope in rapid block mode

  This VI is similar to `PicoScope4000GetBlock.vi`. It outputs two-dimensional arrays for each channel that contain data from all the requested number of captures.

- `PicoScope4000GetStreamingValues.vi` - used in streaming mode to get the latest values from the driver

  This VI should be called in a loop after the oscilloscope has been set up using `PicoScope4000Settings.vi` and streaming has been started by calling `PicoScope4000StartStreaming.vi`. The VI outputs the number of samples available and the start index of these samples in the array output by `PicoScope4000StartStreaming.vi`.

- `PicoScope4000Open.vi` - opens a PicoScope 4000 and returns a handle to the device

- `PicoScope4000Settings.vi` - sets up the oscilloscope

  The inputs are clusters for setting up channels and simple triggers. Advanced triggers can be set up using `PicoScope4000AdvancedTriggerSettings.vi`.

- `PicoScope4000StartStreaming.vi` - starts the oscilloscope streaming

  It outputs arrays that will contain samples once `PicoScope4000GetStreamingValues.vi` has returned.

- `PicoStatus.vi` - checks the status value returned by calls to the driver

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

4 Glossary

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

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

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.

Device Manager. Device Manager is a Windows program that displays the current hardware configuration of your computer. On Windows XP, Vista or 7, right-click **My Computer**, choose **Properties**, then click the **Hardware** tab and the **Device Manager** button. In Windows 8 it is directly accessible from the **Start** menu.

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

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

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

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. Megasample (million samples) per second.

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 formed by connecting a PicoScope 4000 Series scope unit to a computer running the PicoScope software.

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

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

Streaming mode. A sampling mode in which the oscilloscope samples data and returns it to the computer in an unbroken stream. This mode allows the capture of data sets whose size is not limited by the size of the scope's memory buffer, at sampling rates up to 6.6 million samples per second.

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

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

USB 1.1. Universal Serial Bus (Full Speed). This is a standard port used to connect external devices to PCs. 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.

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.


pico
Technology

Index

A

B

C

D

E

F

H

I

L

M

O

P

R

S

T

U

V

W


pico
Technology

Pico Technology

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

ps4000pg.en r8 2014-05-13