PicoScope 5000 Series

PC Oscilloscopes

User's Guide

Contents
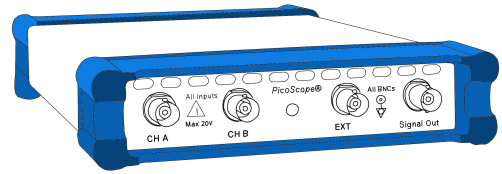
1   Welcome

Thank you for buying a Pico Technology
product!

The PicoScope 5000 Series of PC
Oscilloscopes from Pico Technology is a range
of compact units designed to replace traditional
bench-top oscilloscopes costing many times
the price.

Here are some of the benefits provided by your new PicoScope 5000 Series PC
Oscilloscope:

- Portability: Take the unit with you and plug it in to any Windows PC.
- Performance: Fast sampling up to 1 GS/s, probe-tip bandwidth of 250 MHz, large
  buffer with up to 128 M samples, fast USB 2.0 interface.
- Flexibility: Use it as an oscilloscope, spectrum analyser or high-speed data
  acquisition interface.
- Programmability: The PicoScope 5000 series API lets you write your own
  programs, in your chosen programming language, to control all the features of the
  scope.
- Long-term support: Software upgrades are available to download from our
  website. You can also call our technical specialists for support. You can continue
  to use both of these services free of charge for the lifetime of the product.
- Value for money: You don't have to pay twice for all the features that you
  already have in your PC. The PicoScope 5000 Series scope unit contains the special
  hardware you need and nothing more.
- Convenience: The software makes full use of the large display, storage, user
  interface and networking built in to your PC.

2   Introduction

2.1   Using this guide

You will sometimes see a symbol like this: ⌐2⌐ This is the cross-reference symbol, and it indicates the number of a page on which you can find more information about a topic.

The abbreviations MS/s and GS/s, although not conforming to the SI standard, are used in this guide to mean megasamples per second and gigasamples per second respectively.

2.2   Safety symbols

The following symbols appear on the front panel of the PicoScope 5000 Series PC Oscilloscope.

Symbol 1: Warning triangle

This symbol indicates that a safety hazard exists on the indicated connections if correct precautions are not taken. Read all safety documentation associated with the product before using it.

Symbol 2: Equipotential

This symbol indicates that the outer shells of the indicated BNC connectors are all at the same potential (shorted together). You must therefore take necessary precautions to avoid applying a potential across the return connections of the indicated BNC terminals. Such a potential could cause a large current to flow, resulting in damage to the product or connected equipment, or both.

2.3 Safety warning

We strongly recommend that you read the general safety information below before using your oscilloscope for the first time. Safety protection built in to equipment may cease to function if the equipment is used incorrectly. This could cause damage to your computer, or lead to injury to yourself and others.

Maximum input range

PicoScope 5000 Series PC Oscilloscopes are designed to measure voltages in the range -20 V to +20 V. The A, B and EXT inputs are protected to ±100 V, whilst the AUXIO input can accept 0 V to 10 V. Contact with voltages outside the protection range may cause permanent damage to the unit.

Mains voltages

Pico Technology products are not designed for use with mains voltages. To measure mains, use a differential isolating probe specifically designed for a high source voltage.

Safety grounding

PicoScope 5000 Series PC Oscilloscopes connect directly to the ground of a computer through the USB cable provided to minimise interference.

As with most oscilloscopes, avoid connecting the ground input to any potential other than ground. If in doubt, use a meter to check that there is no significant AC or DC voltage between the ground input of the oscilloscope and the point to which you intend to connect it. Failure to check may cause damage to your computer or injury to yourself and others.

The product does not have a protective safety ground.

Repairs

The oscilloscope contains no user-serviceable parts. Repair or calibration of the oscilloscope requires specialised test equipment and must be performed by Pico Technology.

Cooling fan

The unit contains a low-noise cooling fan that expels air through the holes in the back of the unit (shown in the Connections 9 section). Do not block these holes, as this might cause the unit to overheat. Do not insert anything in the holes, as this could damage the unit or cause injury.

2.4 FCC notice

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his or her own expense.

For safety and maintenance information see the [safety warning] ³ .

2.5 CE notice

The PicoScope 5000 Series PC Oscilloscopes meet the intent of the EMC directive 89/336/EEC and have been designed to EN61326-1 (1997) Class A Emissions and Immunity standard.

PicoScope 5000 Series PC Oscilloscopes also meet the intent of the Low Voltage Directive and have been designed to meet the BS EN 61010-1:2001 IEC 61010-1:2001 (safety requirements for electrical equipment, control, and laboratory use) standard.

2.6   Software licence conditions

The material contained in this software 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 Limited claims the copyright of, and retains the rights to, all material (software, documents etc.) contained in this release. You may copy and distribute the entire release in its original state, but must not copy individual items within the release other than for backup purposes.

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

Because 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 usage in mission-critical applications such as life-support systems.

2.7   Trademarks

Windows is a registered trademark or trademark of Microsoft Corporation in the USA and other countries. Delphi is a registered trademark of Borland Software Corporation.

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

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

       ps5000.en

2.8   Warranty

Pico Technology warrants upon delivery, and for a period of 24 months unless otherwise stated from the date of delivery, that the Goods will be free from defects in material and workmanship.

Pico Technology shall not be liable for a breach of the warranty if the defect has been caused by fair wear and tear, wilful damage, negligence, abnormal working conditions or failure to follow Pico Technology's spoken or written advice on the storage, installation, commissioning, use or maintenance of the Goods or (if no advice has been given) good trade practice; or if the Customer alters or repairs such Goods without the written consent of Pico Technology.

2.9   Company details

Address:	Pico Technology Limited
	The Mill House
	Cambridge Street
	St Neots
	Cambridgeshire
	PE19 1QB
	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

3　Product information

3.1　What do I get?

Your PicoScope 5000 Series PC Oscilloscope kit contains the following items:

Reorder code	Quantity	Description
PR088	1	PicoScope 5203 scope unit (PicoScope 5203 kit only)
PR089	1	PicoScope 5204 scope unit (PicoScope 5204 kit only)
MI145	2	Compensated x1/x10 switchable 250 MHz oscilloscope probe
MI106	1	USB cable, for connection to the USB 1.1 or USB 2.0 port on your PC
PS006	1	Universal power adaptor with UK, US, EU and Australasian plugs
DI042	1	5000 Series software CD
DO115	1	USB Oscilloscope Installation Guide
MI144	1	Carry case

3.2　System requirements

To ensure that your PicoScope 5000 Series 74 PC Oscilloscope operates correctly, you must have a computer with at least the minimum system requirements to run one of the supported operating systems, as shown in the following table. The performance of the software will increase with more powerful PCs, including those with multi-core processors.

Item	Absolute minimum	Recommended minimum	Recommended full specification
Operating system	Windows XP SP2 or Vista (32-bit versions only)		
Processor	As required by Windows	300 MHz	1 GHz
Memory		256 MB	512 MB
Free disk space (Note 1)		1 GB	2 GB
Ports	USB 1.1 compliant port	USB 2.0 compliant port	

Note 1:　The PicoScope software does not use all the disk space specified in the table. The free space is required to make Windows run efficiently.

3.3   Installation instructions

> IMPORTANT
> Do not connect your [PicoScope 5000 Series]⁷⁴
> scope device to the PC before you
> have installed the Pico 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.
- Select one of the mains plugs supplied with your power adapter, according to your location, and fit it to the adaptor.
- Plug the power adaptor supplied with the unit into the mains. Plug the output lead of the adaptor into the scope unit.

Checking the installation

Once you have installed the software and connected the PC Oscilloscope to the PC, start the [PicoScope]⁷⁴ 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 SP2

When you first installed the PicoScope 5000 Series PC Oscilloscope by plugging it into a [USB]⁷⁴ port, Windows associated the Pico [driver]¹² 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

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 5000 series PC Oscilloscope" message. The PC Oscilloscope is then ready for use.

3.4   Connections

Standard oscilloscope connectors

PicoScope 5000 Series ⌐74⌐ PC Oscilloscopes have BNC oscilloscope connectors. The inputs have an impedance of 1 MΩ, so they are compatible with all standard scope probes including x10 attenuated types. To ensure that you obtain the rated bandwidth from the scope, use the compensated high-frequency probes supplied with it.
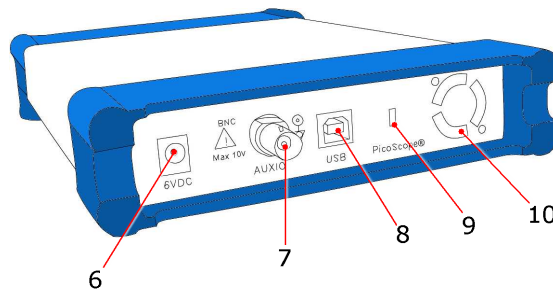
Connector diagrams

Front panel
PicoScope 5203
PicoScope 5204
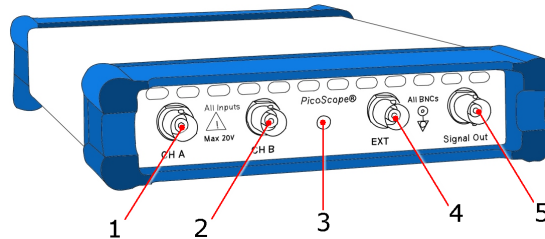
1.   Input channel A
2.   Input channel B
3.   LED: shows when the oscilloscope is sampling data
4.   External trigger input ⌐73⌐
5.   Signal generator output ⌐10⌐

Rear panel
PicoScope 5203
PicoScope 5204

6.   Power socket: for use with the AC adaptor supplied with the unit
7.   Auxiliary input / output. Used as an auxiliary trigger input and output.
8.   USB 2.0 port
9.   Kensington Security Slot. Allows you to attach a standard notebook security cable to the scope.
10. Cooling holes. There is a low-noise fan inside the unit that blows air through these holes.
      Do not block the cooling holes or insert any objects through them, as this could damage the unit or cause injury.

3.4.1   AUXIO connector

The auxiliary input / output connector on the rear panel can be used for two different purposes: -

- When PicoScope 6 is running it functions as a trigger input.
- Programmers can use the <u>API</u> $\boxed{63}$ to configure it as a trigger input or output.
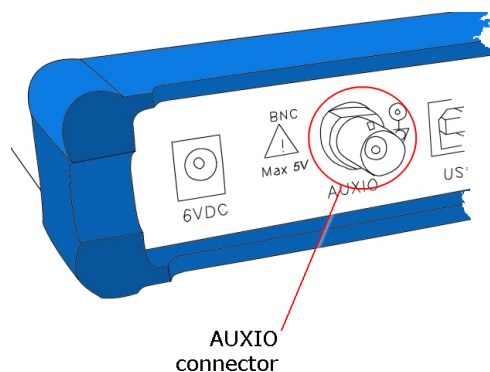
AUXIO connector

AUXIO signal specifications

Refer to the <u>Specifications</u> $\boxed{11}$ table.

3.4.2   Signal Out connector

This signal output connector on the front panel carries the output of the unit's built-in signal generator, which can generate a number of built-in waveforms as well as arbitrary waveforms from a user-defined table of data.

Signal generator output
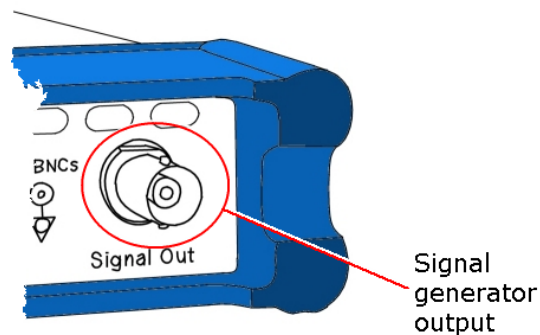
Signal generator output specifications

Refer to the <u>Specifications</u> $\boxed{11}$ table.

3.5   Specifications

Variant	PicoScope 5203	PicoScope 5204
Number of channels	2	
Vertical resolution [74]	8 bits	
Analog bandwidth [73]	250 MHz at probe tip	
Maximum sampling rate [74] (real time) One channel in use Two channels in use	1 GS/s 500 MS/s	
Maximum sampling rate (repetitive signals)	20 GS/s	
Buffer size [73]	32 MS	128 MS
	If two channels in use, buffer shared between channels.	
Inputs Input characteristics Coupling Voltage ranges [74] Accuracy Overload protection	BNC, 1 MΩ in parallel with about 15 pF Selectable AC/DC ±100 mV, ±200 mV, ±500 mV, ±1 V, ±2 V, ±5 V, ±10 V, ±20 V 3% ±100 V	
Timebase Range Accuracy	5 ns/div to 100 s/div 50 ppm	
Signal generator output [10] Standard waveforms Arbitrary waveform buffer Sample rate Output characteristics Resolution Amplitude Offset Maximum range	Sine, square, triangle, ramp (up/down), sin (x)/x, Gaussian, half sine, white noise 8192 samples 125 MS/s BNC, 50 Ω 12 bits ±250 mV to ±2 V ±1 V ±2.5 V	
External trigger [73] Threshold Resolution Input characteristics Bandwidth Overload protection	Variable up to ±20 V 9.8 mV BNC, 1 MΩ 150 MHz ±100 V	
Auxiliary input/output [10] Input impedance Output impedance Input voltage range Input threshold	100 kΩ 600 Ω 0 V to 10 V 1.65 V (nominal)	
Operating environment Temperature range Humidity	0°C to 40°C for normal operation 20°C to 30°C for quoted accuracy 5% to 80% RH, non-condensing	
Storage environment Temperature range Humidity	-20°C to +60°C 5% to 95% RH, non-condensing	
PC connection	USB 2.0 Compatible with USB 1.1	
Power supply Protection AC adaptor	6 V ± 5% @ 2.5 A max. Auto shutdown on excess or reverse voltage Universal adaptor supplied	
Dimensions	170 mm x 255 mm x 40 mm (6.7" x 10.0" x 1.6")	
Weight	0.9 kg (31.7 oz)	
Compliance	European EMC and LVD standards [4] FCC Rules Part 15 Class A [4]	

4 Programming with the PicoScope 5000 Series

The `ps5000.dll` dynamic link library in your PicoScope installation directory allows you to program a PicoScope 5000 Series oscilloscope 74ʼ using standard C function calls. 22ʼ

A typical program for capturing data consists of the following steps:
- Open 39ʼ the scope unit.
- Set up the input channels with the required voltage ranges 13ʼ and coupling mode 13ʼ.
- Set up triggering 13ʼ.
- Set up ETS 16ʼ, if required.
- Start capturing data. (See Sampling modes 14ʼ, 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 69ʼ are installed with your PicoScope software. These show how to use the functions of the driver software in each of the modes available.

4.1 Driver

Your application will communicate with a PicoScope 5000 API driver called `ps5000.dll.` The driver exports the PicoScope 5000 function definitions 22ʼ 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 Windows XP SP2 and Vista. 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 5000 Series 74ʼ PC Oscilloscope for the first time.

4.2 System requirements

General requirements
See System Requirements. 7ʼ

USB

The PicoScope 5000 driver offers three different methods 14ʼ of recording data, all of which support both USB 1.1 and USB 2.0, although the fastest transfer rates are achieved between the PC and the PicoScope 5000 using USB 2.0.

4.3   Voltage ranges

You can set a device input channel to any voltage range from ±100 mV to ±20 V with the ps5000SetChannel 46 function. Each sample is scaled from 8 bits to 16 bits, so that the values returned to your application are as follows:

Constant	Voltage	Value returned	
		decimal	hex
PS5000_LOST_DATA	N/A	-32768*	8000*
PS5000_MIN_VALUE	minimum	-32512	8100
N/A	zero	0	0000
PS5000_MAX_VALUE	maximum	32512	7F00

* In streaming mode, 16 this special value indicates a buffer overrun.

4.4   Channel selection

You can switch each channel on and off, and set its coupling mode to either AC or DC, using the ps5000SetChannel 46 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.

4.5   Triggering

PicoScope 5000 Series PC Oscilloscopes can either start collecting data immediately, or be programmed to wait for a trigger event to occur. In both cases you need to use the PicoScope 5000 trigger functions [1] 60 [2] 62 [3] 63. 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:

	Block mode	ETS mode	Streaming mode
Simple Edge	✓	✓	✓
Advanced Edge	✓		✓
Windowing	✓		✓
Pulse width	✓		✓
Logic	✓		✓
Delay	✓		✓
Drop-out	✓		✓

4.6   Sampling modes

PicoScope 5000 Series PC Oscilloscopes [74] can run in three different sampling modes.

- Block mode. [14] In this mode, the scope stores data in internal RAM and then transfers it to the PC. When the data has been collected it is possible to examine the data, with an optional aggregation [73] factor. The data is lost when a new run is started in the same segment [37], the settings are changed, or the scope is powered down.

- ETS. [16] This mode is similar to block mode, but allows you to achieve higher time resolution by combining data captured in different cycles of a repetitive signal. Aggregation and oversampling [18] are not possible in this mode.

- Streaming mode. [16] In this mode, data is passed directly to the PC without being stored in the scope's internal RAM. This enables long periods of slow data collection for chart recorder and data-logging applications. Streaming mode provides fast streaming at up to 13.33 MS/s (75 ns per sample). Aggregation and triggering are supported in this mode.

In all sampling modes, the driver returns data asynchronously using a *callback.* [73] 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.

4.6.1   Block mode

In block mode, the computer prompts a PicoScope 5000 series [74] 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.

- Sampling rate. A PicoScope 5000 Series PC Oscilloscope can sample at a number of different rates according to the selected timebase. [20] If only one channel is enabled, it can use the A-to-D converter of the disabled channel to double its sampling rate.

- 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, avoid calling setup functions between calls to ps5000RunBlock [42], ps5000Stop [67] and ps5000GetValues [33].

● Aggregation. When the data has been collected, you can set an optional aggregation [73] 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 behaviour using ps5000MemorySegments. [37]

● 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 [15] for programming details.

4.6.1.1   Using block mode

This is the general procedure for reading and displaying data in block mode [14] (including ETS mode [16]), using a single memory segment: [37]

1.   Open the oscilloscope using ps5000OpenUnit. [39]
2.   Select channel ranges and AC/DC coupling using ps5000SetChannel. [46]
3.   Using ps5000GetTimebase [29], select timebases until the required nanoseconds per sample is located.
4.   If you require ETS mode, set it up using ps5000SetEts [49].
5.   If you are using ETS mode, allocate timing information buffers and tell the driver about them using ps5000SetEtsTimeBuffer. [50]
6.   Use the trigger setup functions [1] [60] [2] [62] [3] [63] to set up the trigger if required.
7.   If required, set the signal generator using ps5000SetSigGen. [57]
8.   Start the oscilloscope running using ps5000RunBlock. [42]
9.   Wait until the oscilloscope is ready using the ps5000BlockReady [23] callback.
10.   Use ps5000SetDataBuffer [47] to tell the driver where your memory buffer is.
11.   Transfer the block of data from the oscilloscope using ps5000GetValues. [33]
12.   Display the data.
13.   Repeat steps 8 to 12.
14.   Stop the oscilloscope using ps5000Stop [67].
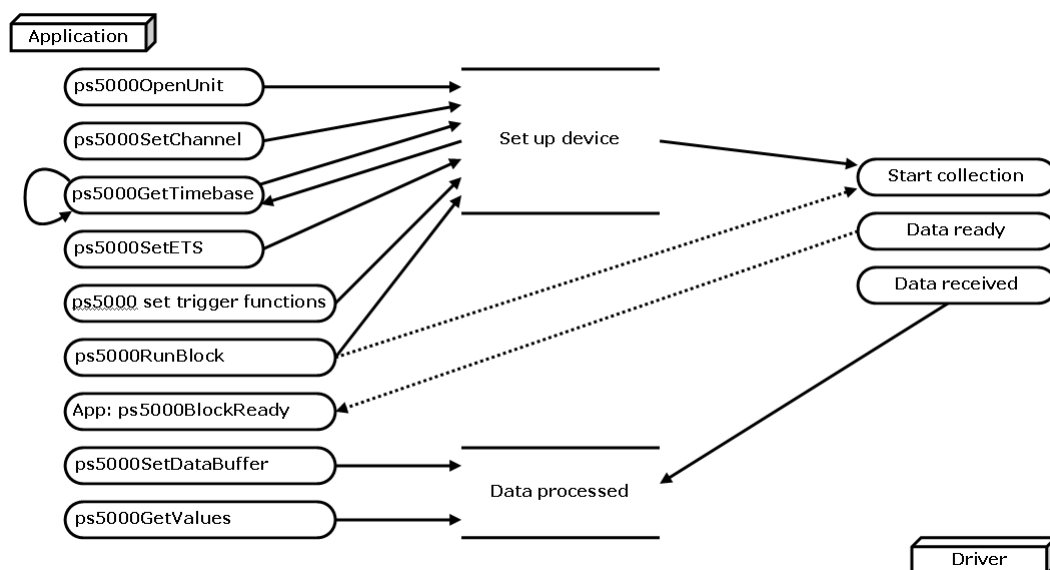
15.   Request new views of stored data using different aggregation parameters: see Retrieving stored data. [18]

4.6.2   ETS (Equivalent Time Sampling)

ETS is a way of increasing the effective sampling rate of the scope when capturing repetitive signals. It is a modified form of block mode, [14] and is controlled by the ps5000SetTrigger [63] and ps5000SetEts [49] 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 Specifications [11] table.

- **Trigger source.** The EXT [9] and AUXIO [10] inputs cannot be used as triggers in ETS mode.

- **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 ps5000BlockReady [23] callback function.

Applicability	Available in block mode [14] only.
	Not suitable for one-shot (non-repetitive) signals.
	Aggregation [73] and oversampling [18] are not supported.
	Edge-triggering [13] only.
	Auto trigger delay [63] (`autoTriggerMilliseconds`) is ignored.

4.6.2.1   Using ETS mode

Since ETS mode [16] is a type of block mode, the procedure is the same as the one described in Using block mode. [15]

4.6.3   Streaming mode

Streaming mode can capture data without the gaps that occur between blocks when using block mode. [14] It can transfer data to the PC at speeds of up to 13.33 million samples per second (75 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 [73] while the device is streaming, which allows your application to zoom in and out of the data with the minimum of programming effort. 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.

● Lost data. If there are gaps in the data values passed to your application, the lost samples are replaced by PS5000_LOST_DATA 13 constants.

● Memory segmentation. The memory can be divided into segments 37 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 17 for programming details.

4.6.3.1   Using streaming mode

This is the general procedure for reading and displaying data in streaming mode 16 using a single memory segment: 37

1. Open the oscilloscope using ps5000OpenUnit. 39
2. Select channels, ranges and AC/DC coupling using ps5000SetChannel. 46
3. Use the trigger setup functions [1] 60 [2] 62 [3] 63 to set up the trigger if required.
4. Call ps5000SetDataBuffer 47 to tell the driver where your data buffer is.
5. Set up aggregation and start the oscilloscope running using PS5000RunStreaming. 44
6. Call ps5000GetStreamingLatestValues 28 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 ps5000Stop, 67 even if Auto Stop is enabled.
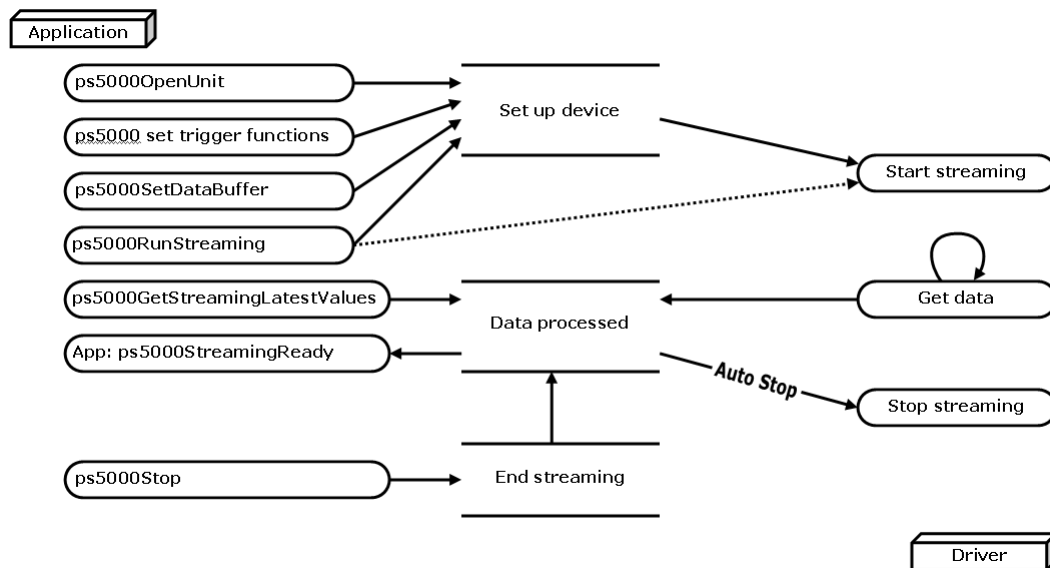
9. Request new views of stored data using different aggregation parameters: see Retrieving stored data. 18

4.6.4 Retrieving stored data

You can collect data from the PicoScope 5000 driver at a different aggregation when ps5000RunBlock [42] or ps5000RunStreaming [44] has already been called and has successfully captured all the data. Use ps5000GetValuesAsync. [34]
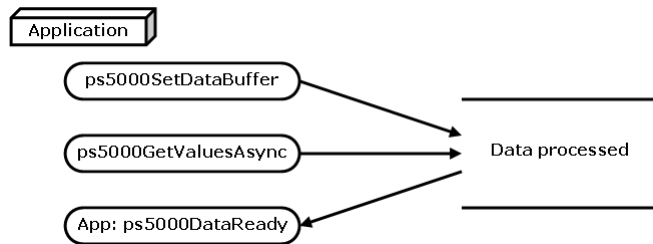
4.7 Oversampling

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 [74] of the oscilloscope by n bits, where n is given approximately by the equation below:

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

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 [14] only, and not in ETS [16] mode.
	Cannot be used at the same time as aggregation. [73]

4.8   Signal generator

The PicoScope 5000 Series PC Oscilloscopes have a built-in <u>signal generator</u> [74].
The generator's clock frequency is fixed, and a phase counter is used to index
successive samples in the waveform buffer. This method is called direct digital
synthesis (DDS), and is discussed in detail under <u>ps5000SetSigGenArbitrary.</u> [54].

You can configure these parameters:

- Peak-to-peak voltage
- Frequency
- Offset voltage
- Waveform type

You can pass the signal to be generated to the device as an arbitrary pattern with up
to 8,192 samples, along with the output frequency and voltage level. The signal
generator can sweep the frequency up, down or up-and-down. Use
<u>ps5000SetSigGenArbitrary</u> [54].

You can also use the predefined signals such as square, sine, triangle, ramp up and
ramp down waveforms. Use <u>ps5000SetSigGenBuiltIn</u> [57].

Applicability	Use <u>ps5000SetSigGenArbitrary</u> [54], <u>ps5000SetSigGenBuiltIn</u> [57] and <u>ps5000SigGenSoftwareControl</u> [66] API calls.

4.9   Timebases

The API allows you to select any of 2^{32} different timebases based on a maximum sampling rate of 1 GHz. 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.

The range of timebase values is divided into two subranges, with the subrange 0 to 2 specifying a power of 2, and the subrange greater than 2 specifying a number divided by 125,000,000. The maximum value is $2^{32}-1$.

timebase	sample interval
0 to 2	$2^{timebase}$ / 1,000,000,000 That is: - 0 => 1 ns 1 => 2 ns 2 => 4 ns
> 2	(timebase - 2) / 125,000,000 For example: - 3 => 8 ns 4 => 16 ns 5 => 24 ns … 2^{32}-1 => ~34 s

Applicability	Use ps5000GetTimebase ⌐29⌐ API call.

     Copyright © 2006-2007 Pico Technology Limited. All rights reserved.

4.10   Combining several oscilloscopes

It is possible to collect data using up to 64 PicoScope 5000 Series PC Oscilloscopes 74
at the same time, depending on the capabilities of the PC. Each oscilloscope must be
connected to a separate USB port. The ps5000OpenUnit 39 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 ps5000BlockReady(...)
// define callback function specific to application

handle1 = ps5000OpenUnit()
handle2 = ps5000OpenUnit()

ps5000SetChannel(handle1)
// set up unit 1
ps5000RunBlock(handle1)

ps5000SetChannel(handle2)
// set up unit 2
ps5000RunBlock(handle2)

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

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

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

4.11   API function calls

The PicoScope 5000 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.

4.11.1 ps5000BlockReady

```
typedef void (CALLBACK *ps5000BlockReady)
(
  short        handle,
  PICO_STATUS  status,
  void       * pParameter
)
```

This callback 73 function is part of your application. You register it with the PicoScope 5000 series driver using ps5000RunBlock, 42 and the driver calls it back when block-mode data is ready. You can then download the data using the ps5000GetValues 33 function.

Applicability	Block mode 14 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 ps5000RunBlock 42. The callback function can write to this location to send any data, such as a status flag, back to your application.
Returns	nothing

4.11.2 ps5000CloseUnit

```
PICO_STATUS ps5000CloseUnit
(
   short handle
)
```

This function shuts down a PicoScope 5000 scope device.

Applicability	All modes
Arguments	handle, the handle, returned by ps5000OpenUnit 39, of the scope device to be closed.
Returns 70	PICO_OK PICO_HANDLE_INVALID

4.11.3 ps5000DataReady

```
typedef void (CALLBACK *ps5000DataReady)
(
  short          handle,
  long           noOfSamples,
  short          overflow,
  unsigned long  triggerAt,
  short          triggered,
  void         * pParameter
)
```

This function handles post-collection data returned by the driver after a call to ps5000GetValuesAsync⌐34┐. It is a callback⌐73┐ function that is part of your application. You register it with the PicoScope 5000 series driver using ps5000GetValuesAsync,⌐34┐ 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 ps5000GetValuesAsync.⌐34┐ The callback function can write to this location to send any data, such as a status flag, back to the application. The data type is defined by the application programmer.
Returns	nothing

4.11.4   ps5000FlashLed

```
PICO_STATUS ps5000FlashLed
(
  short handle,
  short start
)
```

This function flashes the LED on the front of the scope without blocking the calling thread. Calls to [ps5000RunStreaming](44) and [ps5000RunBlock](42) 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](70)	PICO_OK PICO_HANDLE_INVALID PICO_BUSY

4.11.5 ps5000GetMaxDownSampleRatio

```
PICO_STATUS ps5000GetMaxDownSampleRatio
(
  short          handle,
  unsigned long  noOfUnaggregatedSamples,
  unsigned long * maxDownSampleRatio,
  short          downSampleRatioMode,
  unsigned short  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 ps5000GetValues 33 `segmentIndex,` the memory segment 37 where the data is stored

Returns 70	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_NULL_PARAMETER`
	`PICO_INVALID_PARAMETER`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_TOO_MANY_SAMPLES`

4.11.6   ps5000GetStreamingLatestValues

```
PICO_STATUS ps5000GetStreamingLatestValues
(
  short                 handle,
  ps5000StreamingReady  lpPs5000Ready,
  void                * pParameter
)
```

This function is used to collect the next block of values while streaming [16] is running. You must call ps5000RunStreaming [44] beforehand to set up streaming.

Applicability	Streaming [16] mode only.
Arguments	handle, the handle of the required device. lpPs5000Ready, a pointer to your ps5000StreamingReady [68] callback function that will return the latest aggregated values. pParameter, a void pointer that will be passed to the ps5000StreamingReady [68] callback function.
Returns [70]	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_INVALID_CALL PICO_BUSY PICO_NOT_RESPONDING

4.11.7 ps5000GetTimebase

```
PICO_STATUS ps5000GetTimebase
(
  short               handle,
  unsigned long       timebase,
  long                noSamples,
  long              * timeIntervalNanoseconds,
  short               oversample,
  long              * maxSamples
  unsigned short      segmentIndex
)
```

This function discovers which [timebases](74) are available on the oscilloscope. You should set up the channels using [ps5000SetChannel](46) and, if required, [ETS](73) mode using [ps5000SetEts,](49) first.

Applicability	All modes
Arguments	`handle,` the handle of the required device. `timebase,` a code between 0 and 2^{32}-1 that specifies the sampling interval ([see timebase guide](20)) `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.](18) `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**](70)	PICO_OK PICO_INVALID_HANDLE PICO_TOO_MANY_SAMPLES PICO_INVALID_CHANNEL PICO_INVALID_TIMEBASE PICO_INVALID_PARAMETER

4.11.8 ps5000GetTriggerTimeOffset

```
PICO_STATUS ps5000GetTriggerTimeOffset
(
  short              handle
  unsigned long    * timeUpper
  unsigned long    * timeLower
  PS5000_TIME_UNITS * timeUnits
  unsigned short     segmentIndex
)
```

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

Applicability	Block mode ⌐14⌐ only.
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 include: -
	`PS5000_NS`
	`PS5000_US`
	`PS5000_MS`
	`PS5000_S`
	`segmentIndex`, the number of the memory segment ⌐37⌐ for which the information is required.
Returns ⌐70⌐	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_DEVICE_SAMPLING`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NULL_PARAMETER`
	`PICO_NO_SAMPLES_AVAILABLE`

4.11.9   ps5000GetTriggerTimeOffset64

```
PICO_STATUS ps5000GetTriggerTimeOffset64
(
  short                handle,
  __int64            * time,
  PS5000_TIME_UNITS * timeUnits,
  unsigned short       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 [14] only

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 `timeUpper` and `timeLower` are measured. The allowable values include: -
	   `PS5000_NS`
	   `PS5000_US`
	   `PS5000_MS`
	   `PS5000_S`
	
	`segmentIndex,` the number of the memory segment [37] for which the information is required

Returns [70]	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_DEVICE_SAMPLING`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NULL_PARAMETER`
	`PICO_NO_SAMPLES_AVAILABLE`

4.11.10 ps5000GetUnitInfo

```
PICO_STATUS ps5000GetUnitInfo
(
    short        handle,
    char      *  string,
    short        stringLength,
    short     *  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 a short, 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](#)₇₀	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_INVALID_INFO PICO_INFO_UNAVAILABLE

info		String returned	Example
0	PICO_DRIVER_VERSION	Version number of PicoScope 5000 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	5204
4	PICO_BATCH_AND_SERIAL	Batch and serial number of device	KJL87/6
5	PICO_CAL_DATE	Calibration date of device	07Feb07
6	PICO_KERNEL_VERSION	Version of kernel driver	1,1,2,4

4.11.11 ps5000GetValues

```
PICO_STATUS ps5000GetValues
(
  short           handle,
  unsigned long   startIndex,
  unsigned long * noOfSamples,
  unsigned long   downSampleRatio,
  short           downSampleRatioMode,
  unsigned short  segmentIndex,
  short         * overflow
)
```

This function returns block-mode data, either with or without aggregation, 73 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 14 only

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`, the number of samples to return.
	
	`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 73)
	
	`segmentIndex`, the zero-based number of the memory segment 37 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 70	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_DEVICE_SAMPLING`
	`PICO_NULL_PARAMETER`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_INVALID_PARAMETER`
	`PICO_TOO_MANY_SAMPLES`
	`PICO_DATA_NOT_AVAILABLE`
	`PICO_STARTINDEX_INVALID`
	`PICO_INVALID_SAMPLERATIO`
	`PICO_INVALID_CALL`
	`PICO_NOT_RESPONDING`
	`PICO_MEMORY`

4.11.12 ps5000GetValuesAsync

```
PICO_STATUS ps5000GetValuesAsync
(
    short            handle,
    unsigned long    startIndex,
    unsigned long    noOfSamples,
    unsigned long    downSampleRatio,
    short            downSampleRatioMode,
    unsigned short   segmentIndex,
    void          *  lpDataReady,
    void          *  pParameter
)
```

This function returns streaming data, either with or without aggregation, [73] 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. [73]

Applicability	Streaming mode [16] only
Arguments	`handle,` the handle of the required device `startIndex:` see ps5000GetValues [33] `noOfSamples:` see ps5000GetValues [33] `downSampleRatio:` see ps5000GetValues [33] `downSampleRatioMode:` see ps5000GetValues [33] `segmentIndex:` see ps5000GetValues [33] `lpDataReady,` a pointer to the ps5000DataReady [25] function that is called when the data is ready `pParameter,` a void pointer that will be passed to the ps5000DataReady [25] callback function. The data type depends on the design of the callback function, which is determined by the application programmer.

Returns [70]	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_DEVICE_SAMPLING` – streaming only
	`PICO_NULL_PARAMETER`
	`PICO_STARTINDEX_INVALID`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_INVALID_PARAMETER`
	`PICO_DATA_NOT_AVAILABLE`
	`PICO_INVALID_SAMPLERATIO`
	`PICO_INVALID_CALL`

4.11.13 ps5000IsLedFlashing

```
PICO_STATUS ps5000IsLedFlashing
(
  short    handle,
  short * 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](70)	`PICO_OK`
	 `PICO_HANDLE_INVALID`
	 `PICO_NULL_PARAMETER`

4.11.14 ps5000IsTriggerOrPulseWidthQualifierEnabled

```
PICO_STATUS ps5000IsTriggerOrPulseWidthQualifierEnabled
(
    short   handle,
    short * triggerEnabled,
    short * pulseWidthQualifierEnabled
)
```

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

Applicability	Call after setting up the trigger and calling ps5000SetEts, [49] and just before calling either ps5000RunBlock [42] or ps5000RunStreaming. [44]
Arguments	`handle,` the handle of the required device `triggerEnabled,` indicates whether the trigger will successfully be set when ps5000RunBlock [42] or ps5000RunStreaming [44] 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 ps5000RunBlock [42] or ps5000RunStreaming [44] is called. A non-zero value indicates that the pulse width qualifier is set, otherwise the pulse width qualifier is not set.

Returns [70]	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`

4.11.15 ps5000MemorySegments

```
PICO_STATUS ps5000MemorySegments
(
  short            handle
  unsigned short   nSegments,
  long           * nMaxSamples
)
```

This function sets the number of <u>memory segments</u> 37 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. The maximum numbers are 8,192 for the PicoScope 5203 and 32,768 for the PicoScope 5204. `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 70	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_TOO_MANY_SEGMENTS`
	`PICO_MEMORY`

4.11.16 ps5000NoOfStreamingValues

```
PICO_STATUS ps5000NoOfStreamingValues
(
  short           handle,
  unsigned long * noOfValues
)
```

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

Applicability	Streaming mode 16. Call after ps5000Stop. 67
Arguments	`handle,` the handle of the required device `noOfValues,` returns the number of samples

Returns 70	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_NOT_USED`
	`PICO_BUSY`

4.11.17 ps5000OpenUnit

```
PICO_STATUS ps5000OpenUnit
(
   short * 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 a short 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.

[Returns](70)	`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`

4.11.18 ps5000OpenUnitAsync

```
PICO_STATUS ps5000OpenUnitAsync
(
   short * status
)
```

This function opens a scope device without blocking the calling thread. You can find out when it has finished by periodically calling ps5000OpenUnitProgress 41 until that function returns a non-zero value.

Applicability	All modes

Arguments	`status`, pointer to a short that indicates:
	    0 if there is already an open operation in progress
	    1 if the open operation is initiated

Returns 70	`PICO_OK`
	`PICO_OPEN_OPERATION_IN_PROGRESS`
	`PICO_OPERATION_FAILED`

4.11.19 ps5000OpenUnitProgress

```
PICO_STATUS ps5000OpenUnitProgress
(
    short * handle,
    short * progressPercent,
    short * complete
)
```

This function checks on the progress of <u>ps5000OpenUnitAsync</u> [40].

Applicability	Use after <u>ps5000OpenUnitAsync</u> [40]
Arguments	`handle`, pointer to a short 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 a short to which the percentage progress is to be written. 100% implies that the open operation is complete. `complete`, pointer to a short that is set to 1 when the open operation has finished

<u>Returns</u> [70]	`PICO_OK`
	`PICO_NULL_PARAMETER`
	`PICO_OPERATION_FAILED`

4.11.20 ps5000RunBlock

```
PICO_STATUS ps5000RunBlock
(
  short                 handle,
  long                  noOfPreTriggerSamples,
  long                  noOfPostTriggerSamples,
  unsigned long         timebase,
  short                 oversample,
  long                * timeIndisposedMs,
  unsigned short        segmentIndex,
  ps5000BlockReady      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 37 referred to by `segmentIndex`.

Applicability	[Block mode only](#) [14]
Arguments	`handle,` the handle of the required device. `noOfPreTriggerSamples,` the number of samples to return before the trigger event. If no trigger has been set then this argument is ignored and `noOfPostTriggerSamples` specifies the maximum number of data points (samples) to collect. `noOfPostTriggerSamples,` the number of samples to be taken after a trigger event. If no trigger event is set then this specifies the maximum number of samples to be taken. If a trigger condition has been set, this specifies the number of data points (samples) to be taken after a trigger has fired, and the number of data points to be collected is: - `noOfPreTriggerSamples +  noOfPostTriggerSamples` `timebase,` a number in the range 0 to 2^{32}-1. See the [guide to calculating timebase values.](#) [20] `oversample,` the [oversampling](#) [18] factor, a number in the range 1 to 256. `timeIndisposedMs,` returns the time, in milliseconds, that the PicoScope5000 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](#) [37] to use. `lpReady,` a pointer to the [ps5000BlockReady](#) [23] callback that the driver will call when the data has been collected. `pParameter,` a void pointer that is passed to the [ps5000BlockReady](#) [23] callback function. The callback can use the pointer to return arbitrary data to your application.

[**Returns**](#) [70]	```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_WARNING_AUX_OUTPUT_CONFLICT```
	```PICO_WARNING_EXT_THRESHOLD_CONFLICT```
	```PICO_TRIGGER_ERROR```

4.11.21 ps5000RunStreaming

```
PICO_STATUS ps5000RunStreaming
(
  short                handle,
  unsigned long      * sampleInterval,
  PS5000_TIME_UNITS    sampleIntervalTimeUnits
  unsigned long        maxPreTriggerSamples,
  unsigned long        maxPostTriggerSamples,
  short                autoStop
  unsigned long        downSampleRatio,
  unsigned long        overviewBufferSize
)
```

This function tells the oscilloscope to start collecting data in <u>streaming mode</u> ⌐16¬. When data has been collected from the device it is <u>aggregated</u> ⌐73¬ and the values returned to the application. Call <u>ps5000GetStreamingLatestValues</u> ⌐28¬ 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](#)[16] 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: -
	   `PS5000_NS`
	   `PS5000_US`
	   `PS5000_MS`
	   `PS5000_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 [ps5000SetDataBuffer.](#)[47]

[**Returns**](#)[70]	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_NULL_PARAMETER`
	`PICO_INVALID_PARAMETER`
	`PICO_STREAMING_FAILED`
	`PICO_NOT_RESPONDING`
	`PICO_TRIGGER_ERROR`
	`PICO_INVALID_SAMPLE_INTERVAL`
	`PICO_INVALID_BUFFER`

4.11.22 ps5000SetChannel

```
PICO_STATUS ps5000SetChannel
(
   short          handle,
   PS5000_CHANNEL channel,
   short          enabled,
   short          dc,
   PS5000_RANGE   range
)
```

This function specifies whether an input channel is to be enabled, the AC/DC coupling
|73ˀ| mode and the voltage range.

Applicability	All modes

Arguments	`handle`, the handle of the required device
	
	`channel`, an enumerated type. The values are: -
	    `PS5000_CHANNEL_A`
	    `PS5000_CHANNEL_B`
	
	`enabled`, specifies if the channel is active. The values are: -
	    `TRUE` = active
	    `FALSE` = inactive
	
	`dc`, specifies the AC/DC coupling |73ˀ| mode. The values are: -
	    `TRUE` = DC
	    `FALSE` = AC
	
	`range`, a number between 3 and 10 that specifies the voltage range. See the table |46ˀ| below.

Returns |70ˀ|	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_VOLTAGE_RANGE`

	range	Voltage range
3	`PS5000_100MV`	±100 mV
4	`PS5000_200MV`	±200 mV
5	`PS5000_500MV`	±500 mV
6	`PS5000_1V`	±1 V
7	`PS5000_2V`	±2 V
8	`PS5000_5V`	±5 V
9	`PS5000_10V`	±10 V
10	`PS5000_20V`	±20 V

4.11.23 ps5000SetDataBuffer

```
PICO_STATUS ps5000SetDataBuffer
(
  short            handle,
  PS5000_CHANNEL   channel,
  short          * buffer,
  long             bufferLth
)
```

This function registers your data buffer, for non-aggregated [73] data, with the PicoScope 5000 driver. You need to allocate the buffer before calling this function.

Applicability	All modes. For aggregated data, use ps5000SetDataBuffers [48] 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: -
	   `PS5000_CHANNEL_A`
	   `PS5000_CHANNEL_B`
	
	`buffer,` a buffer to receive the data values
	
	`bufferLth,` the size of the `buffer` array

Returns [70]	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`

4.11.24 ps5000SetDataBuffers

```
PICO_STATUS ps5000SetDataBuffers
(
   short             handle,
   PS5000_CHANNEL    channel,
   short           * bufferMax,
   short           * bufferMin,
   long              bufferLth
)
```

This function registers your data buffers, for receiving [aggregated](73) data, with the PicoScope 5000 driver. You need to allocate memory for the buffers before calling this function.

Applicability	All sampling modes. For non-aggregated data, use [ps5000SetDataBuffer](47) 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: -
	   `PS5000_CHANNEL_A`
	   `PS5000_CHANNEL_B`
	
	`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**](70)	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`

4.11.25 ps5000SetEts

```
PICO_STATUS ps5000SetEts
(
   short             handle,
   PS5000_ETS_MODE   mode,
   short             etsCycles,
   short             etsInterleave,
   long            * sampleTimePicoseconds
)
```

This function is used to enable or disable ETS [16] (equivalent time sampling) and to set the ETS parameters.

Applicability	Block mode [14]

Arguments	`handle,` the handle of the required device
	
	`mode,` the ETS mode. Use one of these values: -
	  `PS5000_ETS_OFF` - disables ETS
	  `PS5000_ETS_FAST` - enables ETS and provides `ets_cycles` cycles of data, which may contain data from previously returned cycles
	  `PS5000_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 [70]	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`

4.11.26 ps5000SetEtsTimeBuffer

```
PICO_STATUS ps5000SetEtsTimeBuffer
(
  short      handle,
  __int64 * buffer,
  long       bufferLth
)
```

This function tells the PicoScope 5000 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 16 only. If your programming language does not support 64-bit data, use the 32-bit version ps5000SetEtsTimeBuffers 51 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 70	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`

4.11.27 ps5000SetEtsTimeBuffers

```
PICO_STATUS ps5000SetEtsTimeBuffers
(
  short           handle,
  unsigned long * timeUpper,
  unsigned long * timeLower,
  long            bufferLth
)
```

This function tells the PicoScope 5000 driver where to find your application's ETS time buffers. These buffers contain the timing information for each ETS sample after you run a block-mode ETS capture. There are two buffers containing the upper and lower 32-bit parts of the timing information, to allow programming languages that do not support 64-bit data to retrieve the timings correctly.

Applicability	ETS mode 16 only. If your programming language supports 64-bit data, then you can use ps5000SetEtsTimeBuffer 50 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 70	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`

4.11.28 ps5000SetPulseWidthQualifier

```
PICO_STATUS ps5000SetPulseWidthQualifier
(
   short                 handle,
   struct PWQ_CONDITIONS * conditions,
   short                 nConditions,
   THRESHOLD_DIRECTION   direction,
   unsigned long         lower,
   unsigned long         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](53) 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**](70)	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_PULSE_WIDTH_QUALIFIER

4.11.28.1 PWQ_CONDITIONS structure

A structure of this type is passed to ps5000SetPulseWidthQualifier 52 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 ps5000SetPulseWidthQualifier 52 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, external, aux:` 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.

4.11.29 ps5000SetSigGenArbitrary

```
PICO_STATUS ps5000SetSigGenArbitrary
(
    short                 handle,
    long                  offsetVoltage,
    unsigned long         pkToPk
    unsigned long         startDeltaPhase,
    unsigned long         stopDeltaPhase,
    unsigned long         deltaPhaseIncrement,
    unsigned long         dwellCount,
    short               * arbitraryWaveform,
    long                  arbitraryWaveformSize,
    SWEEP_TYPE            sweepType,
    short                 whiteNoise,
    INDEX_MODE            indexMode,
    unsigned long         shots,
    unsigned long         sweeps,
    SIGGEN_TRIG_TYPE      triggerType,
    SIGGEN_TRIG_SOURCE    triggerSource,
    short                 extInThreshold
)
```

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

The arbitrary waveform generator uses direct digital synthesis (DDS). It maintains a 32-bit phase counter that indicates the present location in the waveform. The top 11 bits of the counter are used as an index into a buffer containing the arbitrary waveform.

The generator steps through the waveform by adding a "delta phase" between 1 and 2^{32}-1 to the phase counter every 8 ns. If the delta phase is constant, then the generator produces a waveform at a constant frequency. It is also possible to sweep the frequency by continually modifying the delta phase. This is done by setting up a "delta phase increment" which is added to the delta phase at specified intervals.

Applicability	All modes
Arguments	handle, the handle of the required device offsetVoltage, the voltage offset, in microvolts, to be applied to the waveform pkToPk, the peak-to-peak voltage, in microvolts, of the waveform signal 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 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.

| **Arguments**	`dwellCount`, the time, in 8 ns steps, between successive additions of `deltaPhaseIncrement` to the delta phase counter. This determines the rate at which the generator sweeps the output frequency.
	`arbitraryWaveform`, a buffer that holds the waveform pattern as a set of samples equally spaced in time.
	`arbitraryWaveformSize`, the size of the arbitrary waveform buffer.
	`sweepType`, determines whether the `startDeltaPhase` is swept up to the `stopDeltaPhase`, or down to it, or repeatedly swept up and down. Use one of these values: - 　　UP 　　DOWN 　　UPDOWN 　　DOWNUP
	`whiteNoise`. If `TRUE`, the signal generator produces white noise and ignores all settings except `pkToPk` and `offsetVoltage`. If `FALSE`, the generator produces the arbitrary waveform.
	`indexMode`, specifies how the signal will be formed from the arbitrary waveform data. [Single, dual and quad index modes](#) [56] are possible. Use one of the constants: 　　SINGLE 　　DUAL 　　QUAD
	`shots`, see [ps5000SigGenBuiltIn](#) [57] `sweeps`, see [ps5000SigGenBuiltIn](#) [57] `triggerType`, see [ps5000SigGenBuiltIn](#) [57] `triggerSource`, see [ps5000SigGenBuiltIn](#) [57] `extInThreshold`, see [ps5000SigGenBuiltIn](#) [57]
[Returns](#) [70]	PICO_OK PICO_INVALID_HANDLE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING PICO_NOT_RESPONDING PICO_WARNING_AUX_OUTPUT_CONFLICT PICO_WARNING_EXT_THRESHOLD_CONFLICT PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_OFFSET_VOLTAGE PICO_SIGGEN_PK_TO_PK PICO_SIGGEN_OUTPUT_OVER_VOLTAGE

4.11.29.1 AWG index modes

The [arbitrary waveform generator](#) 54 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.
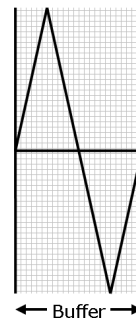
← Buffer →

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.
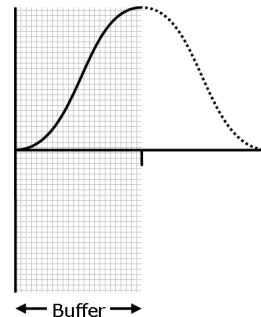
← Buffer →

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. 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.

← Buffer →
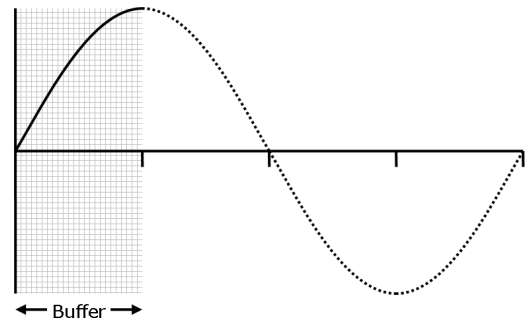
4.11.30 ps5000SetSigGenBuiltIn

```
PICO_STATUS ps5000SetSigGenBuiltIn
(
  short               handle,
  long                offsetVoltage,
  unsigned long       pkToPk
  short               waveType
  float               startFrequency,
  float               stopFrequency,
  float               increment,
  float               dwellTime,
  SWEEP_TYPE          sweepType,
  short               whiteNoise,
  unsigned long       shots,
  unsigned long       sweeps
  SIGGEN_TRIG_TYPE    triggerType,
  SIGGEN_TRIG_SOURCE  triggerSource,
  short               extInThreshold
)
```

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

Applicability	All modes
Arguments	`handle,` the handle of the required device
	`offsetVoltage,` the voltage offset, in microvolts, to be applied to the waveform
	`pkToPk,` the peak-to-peak voltage, in microvolts, of the waveform signal
	`waveType,` the type of waveform to be generated by the scope device. See the [table]58 below.
	`startFrequency,` the frequency that the signal generator will initially produce. For allowable values see `ps5000api.h` - look for `PS5000_SINE_MAX_FREQUENCY` and so on.
	`stopFrequency,` the frequency at which the sweep reverses direction or returns to the initial frequency
	`increment,` the amount of frequency increase or decrease in sweep mode
	`dwellTime,` the time for which the sweep stays at each frequency, in seconds
	`sweepType,` specifies whether the frequency will sweep from `startFrequency` to `stopFrequency,` or in the opposite direction, or repeatedly reverse direction. Use one of these constants: - UP DOWN

	UPDOWN DOWNUP
Arguments	`whiteNoise.` If `TRUE,` the signal generator produces white noise and ignores all settings except `offsetVoltage` and `pkTopk.` If `FALSE,` the signal generator produces the waveform specified by `waveType.` `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 <= shots <= 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 <= sweeps <= MAX_SWEEPS_SHOTS`), then `shots` must be set to zero. `triggerType,` the type of trigger that will be applied to the signal generator. See the table of triggerType values 58 below. `triggerSource,` the source that will trigger the signal generator. See the table of triggerSource values 59 below. 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.

Returns 70	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_SIG_GEN_PARAM`
	`PICO_SHOTS_SWEEPS_WARNING`
	`PICO_NOT_RESPONDING`
	`PICO_WARNING_AUX_OUTPUT_CONFLICT`
	`PICO_WARNING_EXT_THRESHOLD_CONFLICT`
	`PICO_NO_SIGNAL_GENERATOR`
	`PICO_SIGGEN_OFFSET_VOLTAGE`
	`PICO_SIGGEN_PK_TO_PK`
	`PICO_SIGGEN_OUTPUT_OVER_VOLTAGE`

`waveType` values

`PS5000_SINE`	sine wave
`PS5000_SQUARE`	square wave
`PS5000_TRIANGLE`	triangle wave
`PS5000_RAMP_UP`	rising sawtooth
`PS5000_RAMP_DOWN`	falling sawtooth
`PS5000_SINC`	sin (x)/x
`PS5000_GAUSSIAN`	Gaussian
`PS5000_HALF_SINE`	half (full-wave rectified) sine
`PS5000_DC_VOLTAGE`	DC voltage
`PS5000_WHITE_NOISE`	white noise

`triggerType` values

`SIGGEN_RISING`	trigger on rising edge
`SIGGEN_FALLING`	trigger on falling edge

```
SIGGEN_GATE_HIGH          run while trigger is high
SIGGEN_GATE_LOW           run while trigger is low
```

`triggerSource` values

```
SIGGEN_NONE               run without waiting for trigger
SIGGEN_SCOPE_TRIG         use scope trigger
SIGGEN_AUX_IN             use AUXIO input 10
SIGGEN_EXT_IN             use EXT input 73
SIGGEN_SOFT_TRIG          wait for software trigger provided by
                          ps5000SigGenSoftwareControl 66
```

4.11.31 ps5000SetTriggerChannelConditions

```
PICO_STATUS ps5000SetTriggerChannelConditions
(
  short                    handle,
  struct TRIGGER_CONDITIONS * conditions,
  short                    nConditions
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is set up by defining one or more TRIGGER_CONDITIONS 61 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 61 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 70	`PICO_OK`
	 `PICO_INVALID_HANDLE`
	 `PICO_USER_CALLBACK`
	 `PICO_CONDITIONS`
	 `PICO_MEMORY_FAIL`

4.11.31.1 TRIGGER_CONDITIONS structure

A structure of this type is passed to ps5000SetTriggerChannelConditions 60 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
ps5000SetTriggerChannelConditions 60 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, aux, 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. The EXT and AUX inputs are ignored in ETS 16 mode.

4.11.32 ps5000SetTriggerChannelDirections

```
PICO_STATUS ps5000SetTriggerChannelDirections
(
   short                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. When using ETS, ⌐16⌐ only channel A and channel B can be used as a trigger, and only level triggers are available.
Arguments	`handle,` the handle of the required device `channelA, channelB, channelC, channelD, ext, aux` all specify the direction in which the signal must pass through the threshold to activate the trigger. See the table⌐62⌐ below.
Returns ⌐70⌐	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_INVALID_PARAMETER

Trigger direction constants

`ABOVE`	for gated triggers: above a threshold
`BELOW`	for gated triggers: below a threshold
`RISING`	for threshold triggers: rising edge
`FALLING`	for threshold triggers: falling edge
`RISING_OR_FALLING`	for threshold triggers: either edge
`INSIDE`	for window-qualified triggers: inside window
`OUTSIDE`	for window-qualified triggers: outside window
`ENTER`	for window triggers: entering the window
`EXIT`	for window triggers: leaving the window
`ENTER_OR_EXIT`	for window triggers: either entering or leaving the window
`NONE`	no trigger

4.11.33 ps5000SetTriggerChannelProperties

```
PICO_STATUS ps5000SetTriggerChannelProperties
(
  short                                handle,
  struct TRIGGER_CHANNEL_PROPERTIES *  channelProperties
  short                                nChannelProperties
  short                                auxOutputEnable,
  long                                 autoTriggerMilliseconds
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device. `channelProperties`, a pointer to an array of <u>TRIGGER_CHANNEL_PROPERTIES</u>⌐64⌐ 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`. Zero configures the <u>AUXIO connector</u>⌐10⌐ as a trigger input. Any other value configures it as a trigger output. This argument is ignored in <u>ETS</u>⌐16⌐ mode, when the AUXIO connector becomes high-impedance and is 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. This parameter is ignored in <u>ETS</u>⌐16⌐ mode.
<u>**Returns**</u>⌐70⌐	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_TRIGGER_ERROR PICO_MEMORY_FAIL

4.11.33.1 TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to ps5000SetTriggerChannelProperties⌐63⌐ in the
`channelProperties` argument to specify the trigger mechanism, and is defined as
follows: -

```
typedef struct tTriggerChannelProperties
{
   short            thresholdMajor;
   short            thresholdMinor;
   unsigned short hysteresis;
   PS5000_CHANNEL channel;
   THRESHOLD_MODE thresholdMode;
} TRIGGER_CHANNEL_PROPERTIES
```

Elements	`thresholdMajor`, the upper threshold at which the trigger event is to take place. This is scaled in 16-bit ADC counts⌐13⌐ at the currently selected range for that channel. If an external trigger is enabled, the range is fixed at ±20 V.
	
	`thresholdMinor`, the lower threshold at which the trigger event is to take place. This is scaled in 16-bit ADC counts⌐13⌐ at the currently selected range for that channel. If an external trigger is enabled, the range is fixed at ±20 V. Used only with window triggering types.
	
	`hysteresis`, the hysteresis that the trigger has to exceed before it will fire. It is scaled in 16-bit counts.
	
	`channel`, the channel to which the properties apply. See ps5000SetChannel⌐46⌐ for possible values.
	
	`thresholdMode`, either a level or window trigger. Use one of these constants: -
	   LEVEL
	   WINDOW

4.11.34 ps5000SetTriggerDelay

```
PICO_STATUS ps5000SetTriggerDelay
(
   short         handle,
   unsigned long delay
)
```

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

Applicability	All modes
Arguments	handle, the handle of the required device delay, the time between the trigger occurring and the first sample, in multiples of eight sample periods. For example, if delay=100 then the scope would wait 800 sample periods before sampling. At a timebase [20] of 500 MS/s, or 2 ns per sample (timebase=1), the total delay would then be 800 x 2 ns = 1.6 µs.
Returns [70]	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK

4.11.35 ps5000SigGenSoftwareControl

```
PICO_STATUS ps5000SigGenSoftwareControl
(
  short    handle,
  short    state
)
```

This function causes a trigger event, or starts and stops gating. It is used when the signal generator is set to SIGGEN_SOFT_TRIG. 59

Applicability	Use with ps5000SetSigGenBuiltIn 57 or ps5000SetSigGenArbitrary. 54
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 70	PICO_OK PICO_INVALID_HANDLE PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_TRIGGER_SOURCE

4.11.36 ps5000Stop

```
PICO_STATUS ps5000Stop
(
   short   handle
)
```

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

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

Applicability	All modes
Arguments	`handle,` the handle of the required device.
Returns [70]	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK

4.11.37 ps5000StreamingReady

```
typedef void (CALLBACK *ps5000StreamingReady)
(
    short           handle,
    long            noOfSamples,
    unsigned long   startIndex,
    short           overflow,
    unsigned long   triggerAt,
    short           triggered,
    short           autoStop,
    void          * pParameter
)
```

This callback[73] function is part of your application. You register it with the PicoScope 5000 series driver using ps5000GetStreamingLatestValues,[28] and the driver calls it back when streaming-mode data is ready. You can then download the data using the ps5000GetValuesAsync[34] function.

Applicability	Streaming mode[16] 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 ps5000SetDataBuffer.[47] `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. `triggerAt,` an index to the buffer indicating the location of the trigger point. This parameter is valid only when `triggered` is non-zero. `triggered,` a flag indicating whether a trigger occurred. If non-zero, a trigger occurred at the location indicated by `triggerAt`. `autoStop,` the flag that was set in the call to ps5000RunStreaming.[44] `pParameter,` a void pointer passed from ps5000GetStreamingLatestValues.[28] The callback function can write to this location to send any data, such as a status flag, back to the application.
Returns	nothing

4.12   Programming examples

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

- C ⌐69⌐
- Delphi ⌐69⌐

4.12.1 C

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

The console example program is a generic windows application - that is, it does not use Borland AppExpert or Microsoft AppWizard. To compile the program, create a new project for an Application containing the following files: -

- `ps5000con.c` and

- `ps5000bc.lib`   (Borland 32-bit applications) or
- `ps5000.lib`   (Microsoft Visual C 32-bit applications)

The following files must be in the compilation directory: -

- `ps5000Api.h`
- `picoStatus.h`

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

- `ps5000.dll`

4.12.2 Delphi

The program

- `ps5000.dpr`

in the `Examples/ps5000/` subdirectory of your PicoScope installation demonstrates how to operate PicoScope 5000 Series PC Oscilloscopes. ⌐74⌐ The file

- `ps5000.inc`

contains procedure prototypes that you can include in your own programs. Other required files are:

- `ps5000.res`
- `ps5000fm.dfm`
- `ps5000fm.pas`

This has been tested with Delphi version 3.

4.13   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 ps5000 driver returns an error code from the following list of `PICO_STATUS` values.

Code (hex)	Enum	Description
00	PICO_OK	The PicoScope 5000 is functioning correctly
01	PICO_MAX_UNITS_OPENED	An attempt has been made to open more than `PS5000_MAX_UNITS`. Reserved.
02	PICO_MEMORY_FAIL	Not enough memory could be allocated on the host machine
03	PICO_NOT_FOUND	No PicoScope 5000 could be found
04	PICO_FW_FAIL	Unable to download firmware
05	PICO_OPEN_OPERATION_IN_PROGRESS	
06	PICO_OPERATION_FAILED	
07	PICO_NOT_RESPONDING	The PicoScope 5000 is not responding to commands from the PC
08	PICO_CONFIG_FAIL	The configuration information in the PicoScope 5000 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	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 for the information was too small
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 ps5000...Ready 23 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 ps5000Stop, 67 or use ps5000GetStreamingLatestValues 28
25	PICO_NO_SAMPLES_AVAILABLE	...because a run has not been completed
26	PICO_SEGMENT_OUT_OF_RANGE	The memory index is out of range
27	PICO_BUSY	Data cannot be returned yet
28	PICO_STARTINDEX_INVALID	The start time to get stored data is out of range
29	PICO_INVALID_INFO	The information number requested is not a valid number
2A	PICO_INFO_UNAVAILABLE	The handle is invalid so no information is available about the device. Only PICO_DRIVER_VERSION is available.
2B	PICO_INVALID_SAMPLE_INTERVAL	The sample interval selected for streaming is out of range
2C	PICO_TRIGGER_ERROR	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	One or more signal generator parameters are out of range
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	ps5000SetTrigger 63 has found a conflict between the trigger source and the AUXIO output enable
32	PICO_AUX_OUTPUT_ETS_CONFLICT	ETS mode is being used and AUXIO is set as an input

33	PICO_WARNING_EXT_THRESHOLD_CONFLICT	The EXT threshold is being set in both a ps5000SetTrigger 63 function and in the signal generator but the threshold values differ. The last value set will be used.
34	PICO_WARNING_AUX_OUTPUT_CONFLICT	ps5000SetTrigger has set AUXIO 10 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 offset is higher than allowed
39	PICO_SIGGEN_PK_TO_PK	The peak-to-peak voltage is higher than allowed
3A	PICO_CANCELLED	A block collection has been cancelled
3B	PICO_SEGMENT_NOT_USED	The segment index is not currently being used
3C	PICO_INVALID_CALL	The wrong GetValues 33 function has been called for the collection mode in use
3F	PICO_NOT_USED	
40	PICO_INVALID_SAMPLERATIO	The aggregation 73 ratio requested is out of range
41	PICO_INVALID_STATE	Device is in an invalid state

5 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.

Aggregation. The PicoScope 5000 74 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 PS5000RunStreaming 44 for real-time capture, and when you call ps5000GetStreamingLatestValues 28 to obtain post-processed data.

Analog bandwidth. The input frequency at which the measured signal amplitude is 3 decibels below the true signal amplitude.

Block mode. A sampling mode in which the computer prompts the oscilloscope to collect a block of data into its internal memory before stopping the oscilloscope and transferring the whole block into computer memory. 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 5000 driver uses to communicate asynchronously with your application. At design time, you add a function (a *callback* function) to your application to deal with captured data. At run time, when you request captured data from the driver, you also pass it a pointer to your function. The driver then returns control to your application, allowing it to perform other tasks until the data is ready. When this happens, the driver calls your function in a new thread to signal that the data is ready. It is then up to your function to communicate this fact to the rest of your application.

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

Driver. A program that controls a piece of hardware. The driver for the PicoScope 5000 Series PC Oscilloscopes is supplied in the form of a 32-bit Windows DLL, `ps5000.dll.` This is used by the PicoScope software, and by user-designed applications, to control the oscilloscopes.

ETS. Equivalent Time Sampling. Constructs a picture of a repetitive signal by accumulating information over many similar wave cycles. This allows the oscilloscope to create a composite cycle that has more samples, and therefore better time resolution, than a single cycle. Note: cannot be used for one-shot signals.

External trigger. The BNC socket marked EXT on the PicoScope 5000 Series PC Oscilloscopes. It can be used as a trigger input.

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. "GS/s" is an abbreviation for gigasamples (1,000,000,000 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 [74] of the oscilloscope.

PC Oscilloscope. A virtual instrument formed by connecting a PicoScope 5000 Series scope unit to a computer running the PicoScope software.

PicoScope 5000 Series. Pico Technology's fifth generation of PC Oscilloscopes.

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

Signal generator. Generates a waveform and outputs it on the BNC socket marked Signal Out on the oscilloscope. This output can be used to drive a test signal through a BNC cable into one of the scope's input channels. The PicoScope software and the API allow the generator to output regular waveforms, such as sine and square waves, or arbitrary waveforms defined by the user.

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 13.3 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 [18] (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 ±100 V.

Index