PicoScope 3000 Series (A API)

PC Oscilloscopes and MSOs

Programmer's Guide

ps3000apg.en r13
Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

pico
Technology

Contents

1 Introduction 1
1 Overview 1
2 License agreement 2
2 Programming the PicoScope 3000 Series oscilloscopes 3
1 The ps3000a driver 3
2 Minimum PC requirements 3
3 USB port requirements 4
3 Device features 5
1 Power options 5
2 Voltage ranges 6
3 MSO digital data 6
4 MSO digital connector 7
5 Triggering 7
6 Timebases 8
7 Sampling modes 9
1 Block mode 10
2 Rapid block mode 12
3 ETS (Equivalent Time Sampling) 17
4 Streaming mode 19
5 Retrieving stored data 20
8 Combining several oscilloscopes 20
4 API functions 21
1 ps3000aBlockReady (callback) 23
2 ps3000aChangePowerSource 24
3 ps3000aCloseUnit 25
4 ps3000aCurrentPowerSource 26
5 ps3000aDataReady (callback) 27
6 ps3000aEnumerateUnits 28
7 ps3000aFlashLed 29
8 ps3000aGetAnalogueOffset 30
9 ps3000aGetChannelInformation 31
10 ps3000aGetMaxDownSampleRatio 32
11 ps3000aGetMaxEtsValues 33
12 ps3000aGetMaxSegments 34
13 ps3000aGetNoOfCaptures 35
14 ps3000aGetNoOfProcessedCaptures 36
15 ps3000aGetStreamingLatestValues 37
16 ps3000aGetTimebase 38
17 ps3000aGetTimebase2 39
18 ps3000aGetTriggerInfoBulk 40
19 ps3000aGetTriggerTimeOffset 41

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

20 ps3000aGetTriggerTimeOffset64 42
21 ps3000aGetUnitInfo 43
22 ps3000aGetValues 44
1 Downsampling modes 45
23 ps3000aGetValuesAsync 46
24 ps3000aGetValuesBulk 47
25 ps3000aGetValuesOverlapped 48
26 ps3000aGetValuesOverlappedBulk 49
27 ps3000aGetValuesTriggerTimeOffsetBulk 50
28 ps3000aGetValuesTriggerTimeOffsetBulk64 51
29 ps3000aHoldOff 52
30 ps3000aIsReady 53
31 ps3000aIsTriggerOrPulseWidthQualifierEnabled 54
32 ps3000aMaximumValue 55
33 ps3000aMemorySegments 56
34 ps3000aMinimumValue 57
35 ps3000aNoOfStreamingValues 58
36 ps3000aOpenUnit 59
37 ps3000aOpenUnitAsync 60
38 ps3000aOpenUnitProgress 61
39 ps3000aPingUnit 62
40 ps3000aRunBlock 63
41 ps3000aRunStreaming 65
42 ps3000aSetBandwidthFilter 67
43 ps3000aSetChannel 68
44 ps3000aSetDataBuffer 69
45 ps3000aSetDataBuffers 70
46 ps3000aSetDigitalPort 71
47 ps3000aSetEts 72
48 ps3000aSetEtsTimeBuffer 73
49 ps3000aSetEtsTimeBuffers 74
50 ps3000aSetNoOfCaptures 75
51 ps3000aSetPulseWidthDigitalPortProperties 76
52 ps3000aSetPulseWidthQualifier 77
1 PS3000A_PWQ_CONDITIONS structure 79
53 ps3000aSetPulseWidthQualifierV2 80
1 PS3000A_PWQ_CONDITIONS_V2 structure 82
54 ps3000aSetSigGenArbitrary 83
1 AWG index modes 86
55 ps3000aSetSigGenBuiltIn 87
56 ps3000aSetSigGenBuiltInV2 90
57 ps3000aSetSigGenPropertiesArbitrary 91
58 ps3000aSetSigGenPropertiesBuiltIn 92
59 ps3000aSetSimpleTrigger 93
60 ps3000aSetTriggerChannelConditions 94
1 PS3000A_TRIGGER_CONDITIONS structure 95

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

61 ps3000aSetTriggerChannelConditionsV2 96
1 PS3000A_TRIGGER_CONDITIONS_V2 structure 97
62 ps3000aSetTriggerChannelDirections 98
63 ps3000aSetTriggerChannelProperties 99
1 PS3000A_TRIGGER_CHANNEL_PROPERTIES structure 100
64 ps3000aSetTriggerDelay 101
65 ps3000aSetTriggerDigitalPortProperties 102
1 PS3000A_DIGITAL_CHANNEL_DIRECTIONS structure 103
66 ps3000aSigGenArbitraryMinMaxValues 104
67 ps3000aSigGenFrequencyToPhase 105
68 ps3000aSigGenSoftwareControl 106
69 ps3000aStop 107
70 ps3000aStreamingReady (callback) 108

5 Wrapper functions 109

1 Using the wrapper functions for streaming data capture 109
2 AutoStopped 111
3 AvailableData 112
4 BlockCallback 113
5 ClearTriggerReady 114
6 decrementDeviceCount 115
7 getDeviceCount 116
8 GetStreamingLatestValues 117
9 initWrapUnitInfo 118
10 IsReady 119
11 IsTriggerReady 120
12 resetNextDeviceIndex 121
13 RunBlock 122
14 setAppAndDriverBuffers 123
15 setMaxMinAppAndDriverBuffers 124
16 setAppAndDriverDigiBuffers 125
17 setMaxMinAppAndDriverDigiBuffers 126
18 setChannelCount 127
19 setDigitalPortCount 128
20 setEnabledChannels 129
21 setEnabledDigitalPorts 130
22 SetPulseWidthQualifier 131
23 SetPulseWidthQualifierV2 132
24 SetTriggerConditions 133
25 SetTriggerConditionsV2 135
26 SetTriggerProperties 136
27 StreamingCallback 137

6 Programming examples 138

1 C 138
2 C# 138

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3 Excel 139
4 LabVIEW 139
5 MATLAB 141
6 VB.NET 142

7 Reference 143

1 Numeric data types 143
2 Enumerated types, constants and structures 143
3 Driver status codes 143
4 Glossary 148

Index 151

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

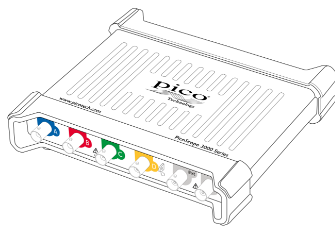
1 Introduction

1.1 Overview

The PicoScope 3000A, 3000B and 3000D Series Oscilloscopes and MSOs from Pico Technology are a range of high-specification, real-time measuring instruments that connect to the USB port of your computer. The series covers various options of portability, deep memory, fast sampling rates and high bandwidth, making it a highly versatile range that suits a wide range of applications. The range includes Hi-Speed USB 2.0 and SuperSpeed USB 3.0 devices.

This manual explains how to use the *ps3000a* API (application programming interface) functions to develop your own programs to collect and analyze data from these oscilloscopes.

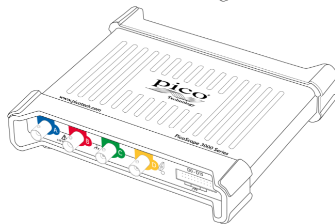
The information in this manual applies to the following oscilloscopes:



PicoScope 3203D to 3206D
PicoScope 3404D to 3406D

USB 3.0 2-channel and 4-channel oscilloscopes

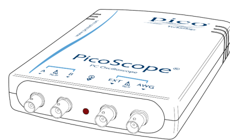
3000D models have an arbitrary waveform generator.



PicoScope 3204D MSO to 3206D MSO
PicoScope 3404D MSO to 3406D MSO

USB 3.0 mixed-signal oscilloscopes

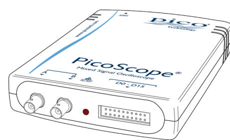
3000D MSO models have 2 or 4 analog inputs, 16 digital inputs and an arbitrary waveform generator.



PicoScope 3204A/B to 3207A/B

High-speed 2-channel oscilloscopes (discontinued)

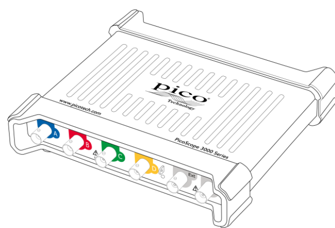
3000A Series models have a function generator; 3000B Series models have an arbitrary waveform generator.



PicoScope 3204 MSO to 3206 MSO

USB 2.0 mixed-signal oscilloscopes (discontinued)

3000 MSO models have 2 or 4 analog inputs, 16 digital inputs and an arbitrary waveform generator.



PicoScope 3404A/B to 3406A/B

High-speed 4-channel oscilloscopes (discontinued)

3000A Series models have a function generator; 3000B Series models have an arbitrary waveform generator.

For information on any of the above oscilloscopes, refer to the data sheets on our website.

For programming information on older PicoScope 3000 Series oscilloscopes and MSOs not listed above, refer to the *PicoScope 3000 Series Programmer's Guide* available from picotech.com.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

1.2 License agreement

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

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

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

Copyright. The software in this release is for use only with Pico products or with data collected using Pico products. You may copy and distribute the SDK without restriction, as long as you do not remove any Pico Technology copyright statements. The example programs in the SDK may be modified, copied and distributed for the purpose of developing programs to collect data using Pico products.

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

Fitness for purpose. No two applications are the same, so Pico cannot guarantee that its equipment or software is suitable for a given application. It is therefore the user's responsibility to ensure that the product is suitable for the user's application.

Mission-critical applications. Because the software runs on a computer that may be running other software products, and may be subject to interference from these other products, this license specifically excludes usage in 'mission-critical' applications, for example life-support systems.

Viruses. This software was continuously monitored for viruses during production. However, the user is responsible for virus checking the software once it is installed.

Support. No software is ever error-free, but if you are dissatisfied with the performance of this software, please contact our technical support staff.

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

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

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

2 Programming the PicoScope 3000 Series oscilloscopes

The `ps3000a.dll` dynamic link library (DLL) in the SDK allows you to program any supported oscilloscope using standard C function calls.

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

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

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

2.1 The ps3000a driver

Your application will communicate with a PicoScope driver called `ps3000a.dll`. This driver is used by all the PicoScopes supported by the *ps3000a* API. The driver exports the *ps3000a* function definitions in standard C format, but this does not limit you to programming in C: you can use the API with any programming language that supports standard C calls.

The API driver depends on another DLL, `PicoIpp.dll`, and a low-level driver, `WinUsb.sys`. These are installed by the SDK when you plug the oscilloscope into the computer for the first time. Your application does not need to call these drivers directly.

2.2 Minimum PC requirements

To ensure that your PicoScope operates correctly, you must have a computer with at least the minimum system requirements to run one of the supported operating systems, as shown in the following table. The performance of the oscilloscope will be better with a more powerful PC, and will benefit from a multicore processor.

Item	Specification
Operating system	Windows XP SP3, Vista, 7 or 8 (32-bit or 64-bit) Or Linux Or OS X (Mac)
Processor	As required by operating system
Memory	As required by operating system
Free disk space	As required by operating system
Ports	USB 2.0 port

Using with custom applications

Drivers are available for the operating systems mentioned above.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

2.3 USB port requirements

The *ps3000a* driver offers four different methods of recording data, all of which support both USB 1.1, USB 2.0, and USB 3.0 connections. The USB 2.0 oscilloscopes are Hi-Speed devices, so transfer rate will not increase by using USB 3.0, but it will decrease when using USB 1.1. The USB 3.0 oscilloscopes are SuperSpeed devices, so should be used with a USB 3.0 port for optimal performance.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3 Device features

3.1 Power options

PicoScope 3000 Series oscilloscopes can be powered in several ways depending on the model:

	USB 2.0 cable	USB 2.0 double-headed cable	USB 3.0 cable	USB 2.0 cable + AC adapter
PicoScope 3200A & 3200B 2-channel USB 2.0 oscilloscopes	✓			
PicoScope 3400A & 3400B 4-channel USB 2.0 oscilloscopes		✓		✓
PicoScope 3207A & 3207B 2-channel USB 3.0 oscilloscopes + **PicoScope 3200D MSO** 2-channel USB 3.0 MSOs + **PicoScope 3200D** 2-channel USB 3.0 oscilloscopes		✓	✓	
PicoScope 3400D MSO 4-channel USB 3.0 MSOs + **PicoScope 3400D** 4-channel USB 3.0 oscilloscopes		✓	✓	✓

Data retention

If the power source is changed (AC adapter connected or disconnected) while the oscilloscope is in operation, the oscilloscope will restart automatically and any unsaved data may be lost.

API functions

The following functions support the flexible power feature:

- ps3000aChangePowerSource()
- ps3000aCurrentPowerSource()

If you want the device to run on USB power only, instruct the driver by calling ps3000aChangePowerSource() after calling ps3000aOpenUnit(). If ps3000aOpenUnit() is called without the power supply connected, the driver returns `PICO_POWER_SUPPLY_NOT_CONNECTED`. If the supply is connected or disconnected during use, the driver will return the relevant status code and you must then call ps3000aChangePowerSource() to continue running the scope. For USB 3.0 scopes, the driver will return `PICO_USB3_0_DEVICE_NON_USB3_0_PORT` if the device is plugged into a non-USB 3.0 port.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.2 Voltage ranges

You can set a device input channel to any voltage range from ±50 mV to ±20 V with ps3000aSetChannel(). Each sample is scaled to 16 bits so that the values returned to your application are as follows:

Function	Voltage	Value returned	
		decimal	hex
ps3000aMinimumValue()	minimum	−32 512	8100
	zero	0	0000
ps3000aMaximumValue()	maximum	32 512	7F00

3.3 MSO digital data

Applicability: mixed-signal oscilloscope (MSO) devices only

A PicoScope MSO has two 8-bit digital ports—PORT0 and PORT1—making a total of 16 digital channels.

The data from each port is returned in a separate buffer that is set up by the ps3000aSetDataBuffer() and ps3000aSetDataBuffers() functions. For compatibility with the analog channels, each buffer is an array of 16-bit words. The 8-bit port data occupies the lower 8 bits of the word, and the upper 8 bits of the word are undefined.

	PORT1 buffer	PORT0 buffer
$Sample_0$	$[XXXXXXXX,D15...D8]_0$	$[XXXXXXXX,D7...D0]_0$
...	...	...
$Sample_{n-1}$	$[XXXXXXXX,D15...D8]_{n-1}$	$[XXXXXXXX,D7...D0]_{n-1}$

Retrieving stored digital data

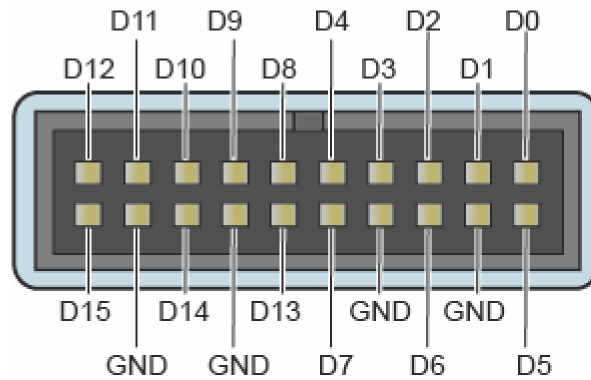
The following C code snippet shows how to combine data from the two 8-bit ports into a single 16-bit word, and then how to extract individual bits from the 16-bit word.

```
// Mask Port 1 values to get lower 8 bits
portValue = 0x00ff & appDigiBuffers[2][i];

// Shift by 8 bits to place in upper 8 bits of 16-bit word
portValue <<= 8;

// Mask Port 0 values to get lower 8 bits
portValue |= 0x00ff & appDigiBuffers[0][i];

for (bit = 0; bit < 16; bit++)
{
  // Shift value (32768 - binary 1000 0000 0000 0000), AND with
  value to get 1  // or 0 for channel
  // Order will be D15 to D8, then D7 to D0

bitValue = (0x8000 >> bit) & portValue? 1 : 0;
}
```

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.4 MSO digital connector

The PicoScope 3000 Series and 3000D Series MSOs have a digital input connector. The following pinout of the 20-pin IDC header plug is drawn as you look at the front panel of the device.



3.5 Triggering

PicoScope oscilloscopes can either start collecting data immediately, or be programmed to wait for a **trigger** event to occur. In both cases you need to use the trigger function ps3000aSetSimpleTrigger(), which in turn calls ps3000aSetTriggerChannelConditions(), ps3000aSetTriggerChannelDirections() and ps3000aSetTriggerChannelProperties() (these can also be called individually, rather than using ps3000aSetSimpleTrigger()). 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.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.6 Timebases

The API allows you to select any of 2^{32} different timebases. 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. Calculate the timebase using the ps3000aGetTimebase() call.

PicoScope 3000A and 3000B Series 2-Channel USB 2.0 Oscilloscopes

Timebase	Sample interval formula	Sample interval	Notes
0	$2^{timebase}$ / 500,000,000	2 ns	Only one channel enabled
1		4 ns	
2		8 ns	
3 ... 2^{32}–1	(timebase–2) / 62,500,000	16 ns ... ~ 68.7 s	

PicoScope 3000 Series USB 2.0 MSOs

Timebase	Sample interval formula	Sample interval	Notes
0	$2^{timebase}$ / 500,000,000	2 ns	No more than one analog channel and one digital port enabled
1		4 ns	
2 ... 2^{32}–1	(timebase–1) / 125,000,000	8 ns ... ~ 34.4 s	

PicoScope 3000A and 3000B Series 4-Channel USB 2.0 Oscilloscopes
PicoScope 3207A and 3207B USB 3.0 Oscilloscopes
PicoScope 3000D Series USB 3.0 Oscilloscopes and MSOs

Timebase	Sample interval formula	Sample interval	Notes
0	$2^{timebase}$ / 1,000,000,000	1 ns	Only one analog channel enabled
1		2 ns	No more than two analog channels or digital ports enabled
2		4 ns	No more than four analog channels or digital ports enabled
3 ... 2^{32}–1	(timebase–2) / 125,000,000	8 ns ... ~ 34.4 s	

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.7 Sampling modes

PicoScope oscilloscopes can run in various **sampling modes:**

- **Block mode.** In this mode, the scope stores data in its buffer memory and then transfers it to the PC. When the data has been collected it is possible to examine the data, with an optional downsampling factor. The data is lost when a new capture is started, the settings are changed, or the scope is powered down.
- **ETS mode.** In this mode, it is possible to increase the effective sampling rate of the scope when capturing repetitive signals. It is a modified form of block mode.
- **Rapid block mode.** This is a variant of block mode that allows you to capture more than one waveform at a time with a minimum of delay between captures. You can use downsampling in this mode if you wish.
- **Streaming mode.** In this mode, data is passed directly to the PC without being stored in the scope's buffer memory. This enables long periods of slow data collection for chart recorder and data-logging applications. Streaming mode supports downsampling and triggering, while providing fast streaming at up these rates:

Number of active channels or ports*	Max. sampling rate (min. sample time)	
	USB 2.0	USB 3.0
1	31.25 MS/s (32 ns)	125 MS/s (8 ns)
2	15.625 MS/s (64 ns)	62.5 MS/s (16 ns)
3 or 4	7.8125 MS/s (128 ns)	31.25 MS/s (32 ns)
More than 4		15.625 MS/s (64 ns)

*Note: A port is a block of 8 digital channels, available on MSOs only.

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

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

In programming environments not supporting callbacks, you may poll the driver in block mode or use one of the wrapper functions provided.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.7.1 Block mode

In **block mode**, the computer prompts the 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. If three or four channels are enabled, each receives a quarter of the memory. These calculations are handled transparently by the driver. The block size also depends on the number of memory segments in use (see ps3000aMemorySegments()).

  For the PicoScope 3000 and 3000D Series MSOs, the memory is shared between the digital ports and analog channels. If one or more analog channels is enabled at the same time as one or more digital ports, the memory per channel is one quarter of the buffer size.

- **Sampling rate.** A *ps3000a* oscilloscope can sample at a number of different rates according to the selected timebase and the combination of channels that are enabled. See the PicoScope 3000 Series User's Guide for the specifications that apply to your scope model.

- **Setup time.** The driver normally performs a number of setup operations, which can take up to 50 milliseconds, before collecting each block of data. If you need to collect data with the minimum time interval between blocks, use rapid block mode and avoid calling setup functions between calls to ps3000aRunBlock(), ps3000aStop() and ps3000aGetValues().

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

- **Memory segmentation.** The scope's internal memory can be divided into segments so that you can capture several waveforms in succession. Configure this using ps3000aMemorySegments().

- **Data retention.** The data is lost when a new run is started in the same segment, the settings are changed, or the scope is powered down or the power source is changed (for flexible power devices).

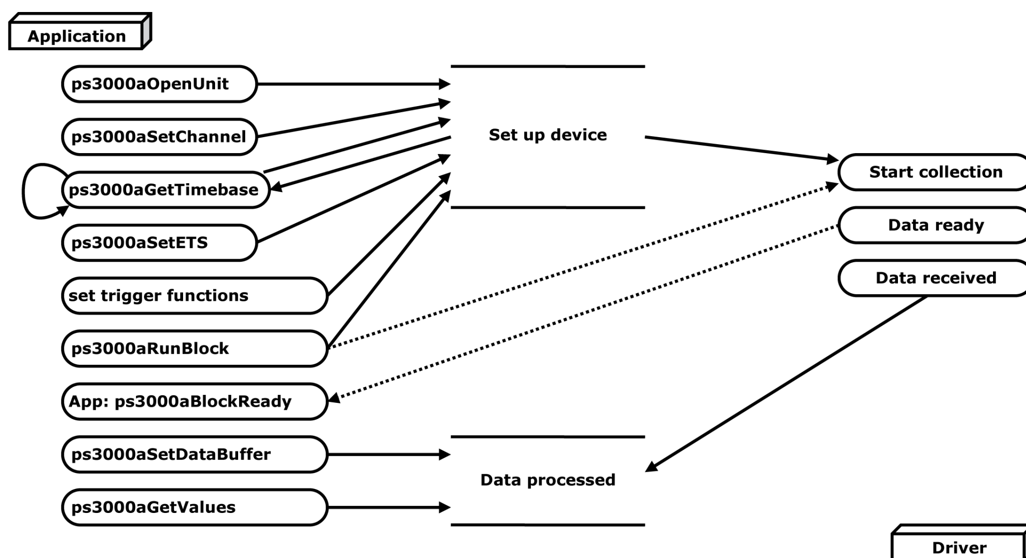
See Using block mode for programming details.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.7.1.1 Using block mode

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

1. Open the oscilloscope using ps3000aOpenUnit().
2. Select channel ranges and AC/DC coupling using ps3000aSetChannel(). All channels are enabled by default, so if you wish to allocate the buffer memory to fewer channels, you must disable those that are not required.
3. *[MSOs only]* Set the digital port using ps3000aSetDigitalPort().
4. Using ps3000aGetTimebase, select timebases until the required nanoseconds per sample is located.
5. Use the trigger setup functions ps3000aSetTriggerChannelConditionsV2(), ps3000aSetTriggerChannelDirections() and ps3000aSetTriggerChannelProperties() to set up the trigger if required.
6. *[MSOs only]* Use the trigger setup functions ps3000aSetTriggerDigitalPortProperties() to set up the digital trigger if required.
7. Start the oscilloscope running using ps3000aRunBlock().
8. Wait until the oscilloscope is ready using the ps3000aBlockReady() callback (or poll using ps3000aIsReady()).
9. Use ps3000aSetDataBuffer() to tell the driver where your memory buffer is.
10. Transfer the block of data from the oscilloscope using ps3000aGetValues().
11. Display the data.
12. Stop the oscilloscope using ps3000aStop().
13. Repeat steps 7 to 11.



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

3.7.1.2 Asynchronous calls in block mode

ps3000aGetValues() may take a long time to complete if a large amount of data is being collected. For example, it can take 3.5 seconds to retrieve the full 128 Msamples from a PicoScope 3206B using a USB 2.0 connection, or several minutes on USB 1.1. To avoid hanging the calling thread, it is possible to call ps3000aGetValuesAsync() instead. This immediately returns control to the calling thread, which then has the option of waiting for the data or calling ps3000aStop() to abort the operation.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.7.2 Rapid block mode

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

Rapid block mode allows you to sample several waveforms at a time with the minimum time between waveforms. It reduces the gap from milliseconds to less than 2 microseconds (on fastest timebase).

See Using rapid block mode for details.

3.7.2.1 Using rapid block mode

You can use **rapid block mode** with or without aggregation. With aggregation, you need to set up two buffers for each channel to receive the minimum and maximum values.

Without aggregation

1. Open the oscilloscope using ps3000aOpenUnit().
2. Select channel ranges and AC/DC coupling using ps3000aSetChannel().
3. *[MSOs only]* Set the digital port using ps3000aSetDigitalPort().
4. Using ps3000aGetTimebase(), select timebases until the required nanoseconds per sample is located.
5. Use the trigger setup functions ps3000aSetTriggerChannelConditionsV2(), ps3000aSetTriggerChannelDirections() and ps3000aSetTriggerChannelProperties() to set up the trigger if required.
6. *[MSOs only]* Use the trigger setup functions ps3000aSetTriggerDigitalPortProperties() to set up the digital trigger if required.
7. Set the number of memory segments equal to or greater than the number of captures required using ps3000aMemorySegments(). Use ps3000aSetNoOfCaptures() before each run to specify the number of waveforms to capture.
8. Start the oscilloscope running using ps3000aRunBlock().
9. Wait until the oscilloscope is ready using the ps3000aIsReady() or wait on the callback function.
10. Use ps3000aSetDataBuffer() to tell the driver where your memory buffers are.
11. Transfer the blocks of data from the oscilloscope using ps3000aGetValuesBulk().
12. Retrieve the time offset for each data segment using ps3000aGetValuesTriggerTimeOffsetBulk64().
13. Display the data.
14. Repeat steps 7 to 13 if necessary.
15. Stop the oscilloscope using ps3000aStop().

With aggregation

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

10a. Call ps3000aSetDataBuffer() or (ps3000aSetDataBuffers()) to set up one pair of buffers for every waveform segment required.
11a. Call ps3000aGetValuesBulk() for each pair of buffers.
12a. Retrieve the time offset for each data segment using ps3000aGetValuesTriggerTimeOffsetBulk64().

Continue from step 13 above.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.7.2.2 Rapid block mode example 1: no aggregation

```
#define MAX_SAMPLES 1000
```

Set up the device up as usual.

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

```
// set the number of waveforms to 100
ps3000aSetNoOfCaptures (handle, 100);

pParameter = false;
ps3000aRunBlock
(
  handle,
  0,                   // noOfPreTriggerSamples
  10000,               // noOfPostTriggerSamples
  1,                   // timebase to be used
  1,                   // not used
  &timeIndisposedMs,
  1,                   // segment index
  lpReady,
  &pParameter
);
```

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

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

for (int32_t i = 0; i < 10; i++)
{
  for (int32_t c = PS3000A_CHANNEL_A; c <= PS3000A_CHANNEL_B; c+
  +)
  {
    ps3000aSetDataBuffer
    (
      handle,
      c,
      &buffer[c][i],
      MAX_SAMPLES,
      i
      PS3000A_RATIO_MODE_NONE
    );
  }
}
```

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

```
ps3000aGetValuesBulk
(
  handle,
  &noOfSamples,           // set to MAX_SAMPLES on entering the
  function
  10,                     // fromSegmentIndex
  19,                     // toSegmentIndex
  1,                      // downsampling ratio
  PS3000A_RATIO_MODE_NONE, // downsampling ratio mode
  overflow                // an array of size 10 int16_t
)
```

Comments: the number of samples could be up to `noOfPreTriggerSamples + noOfPostTriggerSamples`, the values set in `ps3000aRunBlock`. The samples are always returned from the first sample taken, unlike the `ps3000aGetValues` function which allows the sample index to be set. The above segments start at 10 and finish at 19 inclusive. It is possible for the `fromSegmentIndex` to wrap around to the `toSegmentIndex`, by setting the `fromSegmentIndex` to 98 and the `toSegmentIndex` to 7.

```
ps3000aGetValuesTriggerTimeOffsetBulk64
(
  handle,
  times,
  timeUnits,
  10,
  19
)
```

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.7.2.3 Rapid block mode example 2: using aggregation

```
#define MAX_SAMPLES 1000
```

Set up the device up as usual.

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

```
// set the number of waveforms to 100
ps3000aSetNoOfCaptures (handle, 100);

pParameter = false;
ps3000aRunBlock
(
  handle,
  0,                    // noOfPreTriggerSamples,
  1000000,              // noOfPostTriggerSamples,
  1,                    // timebase to be used,
  1,                    // not used
  &timeIndisposedMs,
  1,                    // segment index
  lpReady,
  &pParameter
);
```

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

```
for (int32_t segment = 10; segment < 20; segment++)
{for (int32_t c = PS3000A_CHANNEL_A; c <= PS3000A_CHANNEL_D; c+
  +)
{
  ps3000aSetDataBuffers
  (
    handle,
    c,
    &bufferMax[c],
    &bufferMin[c]
    MAX_SAMPLES
    Segment,
    PS3000A_RATIO_MODE_AGGREGATE
  );
}
```

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

```
    ps3000aGetValues
    (
      handle,
      0,
      &noOfSamples, // set to MAX_SAMPLES on entering
      1000,
      &downSampleRatioMode, //set to RATIO_MODE_AGGREGATE
      index,
      overflow
    );

    ps3000aGetTriggerTimeOffset64
    (
      handle,
      &time,
      &timeUnits,
      index
    )
  }
```

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

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.7.3 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, and is controlled by the trigger functions and ps3000aSetEts().

- **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 result is a larger set of samples spaced by a small fraction of the original sampling interval. The maximum effective sampling rates that can be achieved with this method are listed in the User's Guide for the scope device.
- **Trigger stability.** Because of the high sensitivity of ETS mode to small time differences, the trigger must be set up to provide a stable waveform that varies as little as possible from one capture to the next.
- **Callback.** ETS mode calls the ps3000aBlockReady() callback function when a new waveform is ready for collection. You then call ps3000aGetValues() to retrieve the waveform.

Applicability	Available in block mode only. Not suitable for one-shot (non-repetitive) signals. Aggregation is not supported. Edge-triggering only. Auto trigger delay (`autoTriggerMilliseconds`) is ignored. Digital ports (on MSOs) cannot be used in ETS mode.

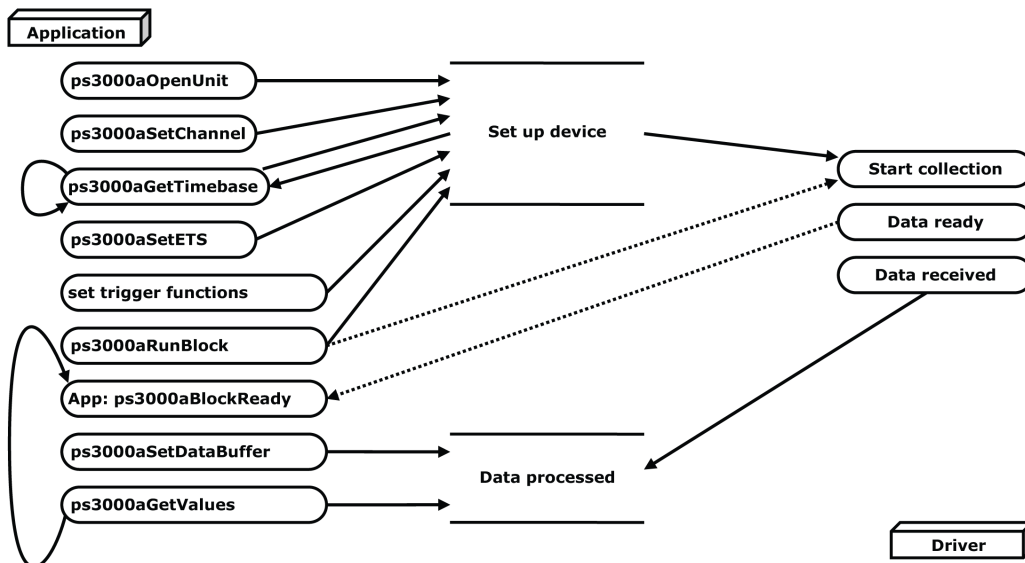
Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.7.3.1 Using ETS mode

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

When using ETS mode you must consider if a digital port has previously been active. If it has, call ps3000aSetDigitalPort() and ps3000aSetTriggerDigitalPortProperties() to ensure these are not active when using ETS.

1. Open the oscilloscope using ps3000aOpenUnit().
2. Select channel ranges and AC/DC coupling using ps3000aSetChannel().
3. Use ps3000aSetEts() to enable ETS and to set the parameters.
4. Using ps3000aGetTimebase(), select timebases until the required nanoseconds per sample is located.
5. Use the trigger setup functions ps3000aSetTriggerChannelConditionsV2(), ps3000aSetTriggerChannelDirections() and ps3000aSetTriggerChannelProperties() to set up the trigger if required.
6. Start the oscilloscope running using ps3000aRunBlock().
7. Wait until the oscilloscope is ready using the ps3000aBlockReady() callback (or poll using ps3000aIsReady()).
8. Use ps3000aSetDataBuffer() to tell the driver where your memory buffer is.
9. Transfer the block of data from the oscilloscope using ps3000aGetValues().
10. Display the data.
11. While you want to collect updated captures, repeat steps 7 to 10.
12. Stop the oscilloscope using ps3000aStop().
13. Repeat steps 6 to 12.



 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.7.4 Streaming mode

Streaming mode can capture data without the gaps that occur between blocks when using block mode. Streaming mode supports downsampling and triggering, while providing fast streaming (for example, with USB 2.0, at up to 31.25 MS/s or 32 ns per sample) when one channel is active, depending on the computer's performance. This makes it suitable for **high-speed data acquisition**, allowing you to capture long data sets limited only by the computer's memory.

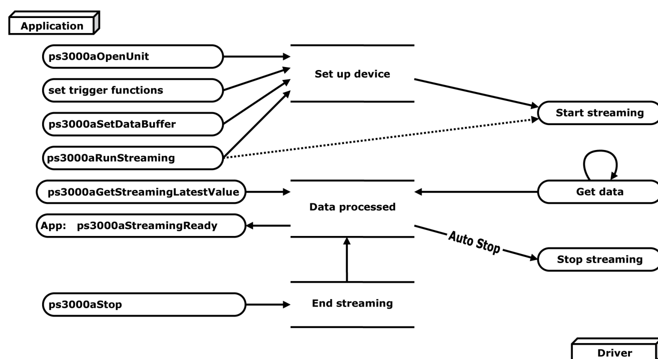
- **Aggregation.** The driver returns aggregated readings while the device is streaming. If aggregation is set to 1 then only one buffer is used per channel. When aggregation is set above 1 then two buffers (maximum and minimum) per channel are used.
- **Memory segmentation.** The memory can be divided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

See Using streaming mode for programming details when using the API. When using the wrapper DLL, see Using the wrapper functions for streaming data capture.

3.7.4.1 Using streaming mode

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

1. Open the oscilloscope using ps3000aOpenUnit().
2. Select channels, ranges and AC/DC coupling using ps3000aSetChannel().
3. *[MSOs only]* Set the digital port using ps3000aSetDigitalPort().
4. Use the trigger setup functions ps3000aSetTriggerChannelConditionsV2(), ps3000aSetTriggerChannelDirections() and ps3000aSetTriggerChannelProperties() to set up the trigger if required.
5. *[MSOs only]* Use the trigger setup functions ps3000aSetTriggerDigitalPortProperties() to set up the digital trigger if required.
6. Call ps3000aSetDataBuffer() to tell the driver where your data buffer is.
7. Set up aggregation and start the oscilloscope running using ps3000aRunStreaming().
8. Call ps3000aGetStreamingLatestValues() to get data.
9. 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.
10. Call ps3000aStop(), even if Auto Stop is enabled.

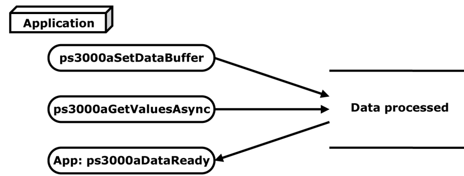


11. Request new views of stored data using different downsampling parameters: see Retrieving stored data.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

3.7.5 Retrieving stored data

You can collect data from the *ps3000a* driver with a different downsampling factor when ps3000aRunBlock() or ps3000aRunStreaming() has already been called and has successfully captured all the data. Use ps3000aGetValuesAsync().



3.8 Combining several oscilloscopes

It is possible to collect data using up to 64 PicoScope oscilloscopes at the same time, depending on the capabilities of the PC. Each oscilloscope must be connected to a separate USB port. ps3000aOpenUnit() 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 ps3000aBlockReady(...)
// define callback function specific to application

handle1 = ps3000aOpenUnit()
handle2 = ps3000aOpenUnit()

ps3000aSetChannel(handle1)
// set up unit 1
ps3000aSetDigitalPort *(when using PicoScope 3000 MSOs only)
ps3000aRunBlock(handle1)

ps3000aSetChannel(handle2)
// set up unit 2
ps3000aSetDigitalPort *(when using PicoScope 3000 MSOs only)
ps3000aRunBlock(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
```

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4 API functions

The *ps3000a* 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. An additional set of wrapper functions is provided for use with programming languages that do not support callbacks.

ps3000aBlockReady	indicate when block-mode data ready
ps3000aChangePowerSource	configure the unit's power source
ps3000aCloseUnit	close a scope device
ps3000aCurrentPowerSource	indicate the current power state of the device
ps3000aDataReady	indicate when post-collection data ready
ps3000aEnumerateUnits	find all connected oscilloscopes
ps3000aFlashLed	flash the front-panel LED
ps3000aGetAnalogueOffset	query the permitted analog offset range
ps3000aGetChannelInformation	query which ranges are available on a device
ps3000aGetMaxDownSampleRatio	query the aggregation ratio for data
ps3000aGetMaxEtsValues	obtain limits for the ETS parameters
ps3000aGetMaxSegments	query the maximum number of segments
ps3000aGetNoOfCaptures	find out how many captures are available
ps3000aGetNoOfProcessedCaptures	query number of captures processed
ps3000aGetStreamingLatestValues	get streaming data while scope is running
ps3000aGetTimebase	find out what timebases are available
ps3000aGetTimebase2	find out what timebases are available
ps3000aGetTriggerInfoBulk	get rapid block trigger timings
ps3000aGetTriggerTimeOffset	find out when trigger occurred (32-bit)
ps3000aGetTriggerTimeOffset64	find out when trigger occurred (64-bit)
ps3000aGetUnitInfo	read information about scope device
ps3000aGetValues	retrieve block-mode data with callback
ps3000aGetValuesAsync	retrieve streaming data with callback
ps3000aGetValuesBulk	retrieve data in rapid block mode
ps3000aGetValuesOverlapped	set up data collection ahead of capture
ps3000aGetValuesOverlappedBulk	set up data collection in rapid block mode
ps3000aGetValuesTriggerTimeOffsetBulk	get rapid-block waveform timings (32-bit)
ps3000aGetValuesTriggerTimeOffsetBulk64	get rapid-block waveform timings (64-bit)
ps3000aHoldOff	not currently used
ps3000aIsReady	poll driver in block mode
ps3000aIsTriggerOrPulseWidthQualifierEnabled	find out whether trigger is enabled
ps3000aMaximumValue	query the max. ADC count in GetValues calls
ps3000aMemorySegments	divide scope memory into segments
ps3000aMinimumValue	query the min. ADC count in GetValues calls
ps3000aNoOfStreamingValues	get number of samples in streaming mode
ps3000aOpenUnit	open a scope device
ps3000aOpenUnitAsync	open a scope device without waiting
ps3000aOpenUnitProgress	check progress of OpenUnit call
ps3000aPingUnit	check communication with device
ps3000aRunBlock	start block mode
ps3000aRunStreaming	start streaming mode
ps3000aSetBandwidthFilter	control the bandwidth limiter
ps3000aSetChannel	set up input channels
ps3000aSetDataBuffer	register data buffer with driver
ps3000aSetDataBuffers	register aggregated data buffers with driver
ps3000aSetDigitalPort	enable the digital port and set the logic level
ps3000aSetEts	set up equivalent-time sampling
ps3000aSetEtsTimeBuffer	set up buffer for ETS timings (64-bit)
ps3000aSetEtsTimeBuffers	set up buffer for ETS timings (32-bit)
ps3000aSetNoOfCaptures	set number of captures to collect in one run
ps3000aSetPulseWidthDigitalPortProperties	set up pulse width triggering on digital port
ps3000aSetPulseWidthQualifier	set up pulse width triggering
ps3000aSetPulseWidthQualifierV2	set up pulse width triggering (digital condition)
ps3000aSetSigGenArbitrary	set up arbitrary waveform generator
ps3000aSetSigGenBuiltIn	set up standard signal generator

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

ps3000aSetSigGenBuiltInV2	set up signal generator (double precision)
ps3000aSetSigGenPropertiesArbitrary	set arbitrary waveform generator properties
ps3000aSetSigGenPropertiesBuiltIn	set signal generator properties
ps3000aSetSimpleTrigger	set up level triggers only
ps3000aSetTriggerChannelConditions	specify which channels to trigger on
ps3000aSetTriggerChannelConditionsV2	specify trigger channels for MSOs
ps3000aSetTriggerChannelDirections	set up signal polarities for triggering
ps3000aSetTriggerChannelProperties	set up trigger thresholds
ps3000aSetTriggerDelay	set up post-trigger delay
ps3000aSetTriggerDigitalPortProperties	set individual digital channels trigger directions
ps3000aSigGenArbitraryMinMaxValues	query AWG parameter limits
ps3000aSigGenFrequencyToPhase	calculate AWG phase from frequency
ps3000aSigGenSoftwareControl	trigger the signal generator
ps3000aStop	stop data capture
ps3000aStreamingReady	indicate when streaming-mode data ready

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.1 ps3000aBlockReady (callback)

```
typedef void (CALLBACK *ps3000aBlockReady)
(
  int16_t       handle,
  PICO_STATUS   status,
  void        * pParameter
)
```

This callback function is part of your application. You register it with the ps3000a driver using ps3000aRunBlock(), and the driver calls it back when block-mode data is ready. You can then download the data using ps3000aGetValues().

Your callback function should do nothing more than copy the data to another buffer within your application. To maintain the best application performance, the function should return as quickly as possible without attempting to process or display the data.

Applicability	Block mode only
Arguments	handle, the handle of the device returning the samples. status, indicates whether an error occurred during collection of the data. * pParameter, a void pointer passed from ps3000aRunBlock(). Your callback function can write to this location to send any data, such as a status flag, back to your application.
Returns	nothing

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.2 ps3000aChangePowerSource

```
PICO_STATUS ps3000aChangePowerSource
(
  int16_t       handle,
  PICO_STATUS   powerstate
)
```

This function selects the power supply mode. You must call this function if any of the following conditions arises:

- USB power is required
- the AC power adapter is connected or disconnected during use
- a USB 3.0 scope is plugged into a USB 2.0 port (indicated if any function returns the PICO_USB3_0_DEVICE_NON_USB3_0_PORT status code)

Applicability	All modes. 4-channel and USB 3.0 oscilloscopes only.
Arguments	handle, the handle of the device. powerstate, the required state of the unit. Either of the following: PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED
Returns	PICO_OK PICO_POWER_SUPPLY_REQUEST_INVALID PICO_INVALID_PARAMETER PICO_NOT_RESPONDING PICO_INVALID_HANDLE

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.3 ps3000aCloseUnit

```
PICO_STATUS ps3000aCloseUnit
(
  int16_t   handle
)
```

This function shuts down an oscilloscope.

Applicability	All modes
Arguments	handle, the handle, returned by ps3000aOpenUnit(), of the scope device to be closed.
Returns	PICO_OK PICO_HANDLE_INVALID PICO_USER_CALLBACK PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.4 ps3000aCurrentPowerSource

```
PICO_STATUS ps3000aCurrentPowerSource
(
  int16_t   handle
)
```

This function returns the current power state of the device.

Applicability	All modes. 4-channel oscilloscopes only.
Arguments	handle, the handle of the device.
Returns	PICO_POWER_SUPPLY_CONNECTED - if the device is powered by the AC adapter. PICO_POWER_SUPPLY_NOT_CONNECTED - if the device is powered by the USB cable.

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.5 ps3000aDataReady (callback)

```
typedef void (CALLBACK *ps3000aDataReady)
(
  int16_t       handle,
  PICO_STATUS   status,
  uint32_t      noOfSamples,
  int16_t       overflow,
  void        * pParameter
)
```

This is a callback function that you write to collect data from the driver. You supply a pointer to the function when you call ps3000aGetValuesAsync(), and the driver calls your function back when the data is ready.

Your callback function should do nothing more than copy the data to another buffer within your application. To maintain the best application performance, the function should return as quickly as possible without attempting to process or display the data.

Applicability	All modes
Arguments	handle, the handle of the device returning the samples. status, a PICO_STATUS code returned by the driver. noOfSamples, the number of samples collected. overflow, a set of flags that indicates whether an overvoltage has occurred and on which channels. It is a bit field with bit 0 representing Channel A. * pParameter, a void pointer passed from ps3000aGetValuesAsync(). 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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.6 ps3000aEnumerateUnits

```
PICO_STATUS ps3000aEnumerateUnits
(
  int16_t * count,
  int8_t  * serials,
  int16_t * serialLth
)
```

This function counts the number of *ps3000a*-compatible scopes connected to the computer, and returns a list of serial numbers as a string.

Applicability	All modes
Arguments	`* count,` on exit, the number of *ps3000a*-compatible units found `* serials,` on exit, a list of serial numbers separated by commas and terminated by a final null. Example: `AQ005/139,VDR61/356,ZOR14/107`. Can be NULL on entry if serial numbers are not required. `* serialLth,` on entry, the length of the `int8_t` buffer pointed to by `serials;` on exit, the length of the string written to `serials`
Returns	PICO_OK PICO_BUSY PICO_NULL_PARAMETER PICO_FW_FAIL PICO_CONFIG_FAIL PICO_MEMORY_FAIL PICO_CONFIG_FAIL_AWG PICO_INITIALISE_FPGA

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.7 ps3000aFlashLed

```
PICO_STATUS ps3000aFlashLed
(
  int16_t   handle,
  int16_t   start
)
```

This function flashes the LED on the front of the scope without blocking the calling thread. Calls to ps3000aRunStreaming() and ps3000aRunBlock() cancel any flashing started by this function. It is not possible to set the LED to be constantly illuminated, as this state is used to indicate that the scope has not been initialized.

Applicability	All modes
Arguments	`handle`, the handle of the scope device `start`, the action required: - < 0 : flash the LED indefinitely. 0 : stop the LED flashing. > 0 : flash the LED `start` times. If the LED is already flashing on entry to this function, the flash count will be reset to `start`.

Returns	`PICO_OK`
	`PICO_HANDLE_INVALID`
	`PICO_BUSY`
	`PICO_DRIVER_FUNCTION`
	`PICO_NOT_RESPONDING`

4.8 ps3000aGetAnalogueOffset

```
PICO_STATUS ps3000aGetAnalogueOffset
(
  int16_t              handle,
  PS3000A_RANGE        range,
  PS3000A_COUPLING     coupling,
  float              * maximumVoltage,
  float              * minimumVoltage
 )
```

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

Applicability	Al models
Arguments	handle, the value returned from opening the device. range, the voltage range to be used when gathering the min and max information. coupling, the type of AC/DC coupling used. * maximumVoltage, a pointer to a float, an out parameter set to the maximum voltage allowed for the range, may be NULL. * minimumVoltage, a pointer to a float, an out parameter set to the minimum voltage allowed for the range, may be NULL. If both maximumVoltage and minimumVoltage are set to NULL the driver will return PICO_NULL_PARAMETER.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_INVALID_VOLTAGE_RANGE PICO_NULL_PARAMETER

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.9 ps3000aGetChannelInformation

```
PICO_STATUS ps3000aGetChannelInformation
(
  int16_t                handle,
  PS3000A_CHANNEL_INFO   info,
  int32_t                probe,
  int32_t              * ranges,
  int32_t              * length,
  int32_t                channels
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device. `info`, the type of information required. The following value is currently supported: `PS3000A_CI_RANGES` `probe`, not used, must be set to 0. `* ranges`, an array that will be populated with available PS3000A_RANGE values for the given info. If `NULL`, length is set to the number of ranges available. `* length`, on input: the length of the ranges array; on output: the number of elements written to ranges array. `channels`, the channel for which the information is required.

Returns	`PICO_OK`
	`PICO_HANDLE_INVALID`
	`PICO_BUSY`
	`PICO_DRIVER_FUNCTION`
	`PICO_NOT_RESPONDING`
	`PICO_NULL_PARAMETER`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_INFO`

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.10 ps3000aGetMaxDownSampleRatio

```
PICO_STATUS ps3000aGetMaxDownSampleRatio
(
  int16_t                handle,
  uint32_t               noOfUnaggregatedSamples,
  uint32_t             * maxDownSampleRatio,
  PS3000A_RATIO_MODE     downSampleRatioMode,
  uint32_t               segmentIndex
)
```

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

Applicability	All modes
Arguments	handle, the handle of the required device noOfUnaggregatedSamples, the number of unprocessed samples to be downsampled * maxDownSampleRatio, the maximum possible downsampling ratio output downSampleRatioMode, the downsampling mode. See ps3000aGetValues() segmentIndex, the memory segment where the data is stored
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_TOO_MANY_SAMPLES

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.11 ps3000aGetMaxEtsValues

```
PICO_STATUS ps3000aGetMaxEtsValues
(
  int16_t    handle,
  int16_t  * etsCycles,
  int16_t  * etsInterleave
)
```

This function returns the maximum number of cycles and maximum interleaving factor that can be used for the selected scope device in ETS mode. These values are the upper limits for the `etsCycles` and `etsInterleave` arguments supplied to ps3000SetEts().

Applicability	All modes
Arguments	`handle`, the handle of the required device `etsCycles`, the maximum value of the `etsCycles` argument supplied to ps3000SetEts() `etsInterleave`, the maximum value of the `etsInterleave` argument supplied to ps3000SetEts()

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_DRIVER_FUNCTION`
	`PICO_NULL_PARAMETER` - if `etsCycles` and `etsInterleave` are both `NULL`

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.12 ps3000aGetMaxSegments

```
PICO_STATUS ps3000aGetMaxSegments
(
  int16_t    handle,
  uint32_t * maxsegments
 )
```

This function returns the maximum number of segments allowed for the opened device. This number is the maximum value of `nsegments` that can be passed to ps3000aMemorySegments().

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

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.13 ps3000aGetNoOfCaptures

```
PICO_STATUS ps3000aGetNoOfCaptures
(
  int16_t    handle,
  uint32_t * nCaptures
)
```

This function finds out how many captures are available in rapid block mode after ps3000aRunBlock() has been called when either the collection completed or the collection of waveforms was interrupted by calling ps3000aStop(). The returned value (`nCaptures`) can then be used to iterate through the number of segments using ps3000aGetValues(), or in a single call to ps3000aGetValuesBulk() where it is used to calculate the `toSegmentIndex` parameter.

Applicability	Rapid block mode
Arguments	`handle`, handle of the required device. `* nCaptures`, output: the number of available captures that has been collected from calling ps3000aRunBlock().

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.14 ps3000aGetNoOfProcessedCaptures

```
PICO_STATUS ps3000aGetNoOfProcessedCaptures
(
  int16_t    handle,
  uint32_t * nCaptures
)
```

This function finds out how many captures in rapid block mode have been processed after ps3000aRunBlock() has been called when either the collection completed or the collection of waveforms was interrupted by calling ps3000aStop(). The returned value (nCaptures) can then be used to iterate through the number of segments using ps3000aGetValues(), or in a single call to ps3000aGetValuesBulk() where it is used to calculate the toSegmentIndex parameter.

Applicability	Rapid block mode
Arguments	handle, handle of the required device. * nCaptures, output: the number of available captures that has been collected from calling ps3000aRunBlock().
Returns	PICO_OK PICO_DRIVER_FUNCTION PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_TOO_MANY_SAMPLES

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.15 ps3000aGetStreamingLatestValues

```
PICO_STATUS ps3000aGetStreamingLatestValues
(
  int16_t                   handle,
  ps3000aStreamingReady     lpPs3000AReady,
  void                    * pParameter
)
```

This function instructs the driver to return the next block of values to your ps3000aStreamingReady() callback. You must have previously called ps3000aRunStreaming() beforehand to set up streaming.

Applicability	Streaming mode only
Arguments	`handle,` the handle of the required device. `lpPs3000AReady,` a pointer to your ps3000aStreamingReady() callback. `* pParameter,` a void pointer that will be passed to the ps3000aStreamingReady() callback. The callback may optionally use this pointer to return information to the application.

Returns	`PICO_OK`
	`PICO_POWER_SUPPLY_CONNECTED`
	`PICO_POWER_SUPPLY_NOT_CONNECTED`
	`PICO_INVALID_HANDLE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_INVALID_CALL`
	`PICO_BUSY`
	`PICO_NOT_RESPONDING`
	`PICO_DRIVER_FUNCTION`

4.16 ps3000aGetTimebase

```
PICO_STATUS ps3000aGetTimebase
(
  int16_t     handle,
  uint32_t    timebase,
  int32_t     noSamples,
  int32_t   * timeIntervalNanoseconds,
  int16_t     oversample,
  int32_t   * maxSamples,
  uint32_t    segmentIndex
)
```

This function calculates the sampling rate and maximum number of samples for a given timebase under the specified conditions. The result will depend on the number of channels enabled by the last call to ps3000aSetChannel().

This function is provided for use with programming languages that do not support the `float` data type. The value returned in the `timeIntervalNanoseconds` argument is restricted to integers. If your programming language supports the `float` type, then we recommend that you use ps3000aGetTimebase2() instead.

To use ps3000aGetTimebase() or ps3000aGetTimebase2(), first estimate the timebase number that you require using the information in the timebase guide. Next, call one of these functions with the timebase that you have just chosen and verify that the `timeIntervalNanoseconds` argument that the function returns is the value that you require. You may need to iterate this process until you obtain the time interval that you need.

Applicability	All modes
Arguments	`handle`, the handle of the required device. `timebase`, see timebase guide `noSamples`, the number of samples required. `* timeIntervalNanoseconds`, on exit, the time interval between readings at the selected timebase. Use NULL if not required. `oversample`, not used. `* maxSamples`, on exit, the maximum number of samples available. The result may vary depending on the number of channels enabled and the timebase chosen. Use `NULL` if not required. `segmentIndex`, the index of the memory segment to use.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_TOO_MANY_SAMPLES PICO_INVALID_CHANNEL PICO_INVALID_TIMEBASE PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_DRIVER_FUNCTION

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.17 ps3000aGetTimebase2

```
PICO_STATUS ps3000aGetTimebase2
(
  int16_t    handle,
  uint32_t   timebase,
  int32_t    noSamples,
  float    * timeIntervalNanoseconds,
  int16_t    oversample,
  int32_t  * maxSamples,
  uint32_t   segmentIndex
)
```

This function is an upgraded version of ps3000aGetTimebase(), and returns the time interval as a `float` rather than an `int32_t`. This allows it to return sub-nanosecond time intervals. See ps3000aGetTimebase() for a full description.

Applicability	All modes
Arguments	`* timeIntervalNanoseconds`, a pointer to the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. All other arguments: see ps3000aGetTimebase().
Returns	See ps3000aGetTimebase().

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.18 ps3000aGetTriggerInfoBulk

```
PICO_STATUS ps3000aGetTriggerInfoBulk
(
  int16_t                    handle,
  PS3000A_TRIGGER_INFO *     triggerInfo,
  uint32_t                   fromSegmentIndex,
  uint32_t                   toSegmentIndex
)
```

This function returns trigger information in rapid block mode.

Applicability	Rapid block mode. PicoScope 3207A and 3207B only.
Arguments	`handle`, the handle of the required device. `triggerInfo`, an array of pointers to `PS3000A_TRIGGER_INFO` structures that, on exit, will contain information on each trigger event. There will be one structure for each segment in the range `[fromSegmentIndex, toSegmentIndex]`. `fromSegmentIndex`, the number of the first memory segment for which information is required. `toSegmentIndex`, the number of the last memory segment for which information is required.
Returns	PICO_NOT_SUPPORTED_BY_THIS_DEVICE PICO_NO_SAMPLES_AVAILABLE PICO_NULL_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_NOT_USED_IN_THIS_CAPTURE_MODE PICO_ETS_MODE_SET PICO_OK PICO_NOT_RESPONDING PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.19 ps3000aGetTriggerTimeOffset

```
PICO_STATUS ps3000aGetTriggerTimeOffset
(
  int16_t               handle,
  uint32_t            * timeUpper,
  uint32_t            * timeLower,
  PS3000A_TIME_UNITS  * timeUnits,
  uint32_t              segmentIndex
)
```

This function gets the time, as two 4-byte values, at which the trigger occurred. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture. A 64-bit version of this function, ps3000aGetTriggerTimeOffset64(), is also available.

Applicability	Block mode, rapid block mode
Arguments	handle, the handle of the required device * timeUpper, on exit, the upper 32 bits of the time at which the trigger point occurred * timeLower, on exit, the lower 32 bits of the time at which the trigger point occurred * timeUnits, returns the time units in which timeUpper and timeLower are measured. The allowable values are: - PS3000A_FS PS3000A_PS PS3000A_NS PS3000A_US PS3000A_MS PS3000A_S segmentIndex, the number of the memory segment for which the information is required.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NOT_USED_IN_THIS_CAPTURE_MODE PICO_NOT_RESPONDING PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.20 ps3000aGetTriggerTimeOffset64

```
PICO_STATUS ps3000aGetTriggerTimeOffset64
(
  int16_t               handle,
  int64_t             * time,
  PS3000A_TIME_UNITS  * timeUnits,
  uint32_t              segmentIndex
)
```

This function gets the time, as a single 64-bit value, at which the trigger occurred. Call it after block-mode data has been captured or when data has been retrieved from a previous block-mode capture. A 32-bit version of this function, ps3000aGetTriggerTimeOffset(), is also available.

Applicability	Block mode, rapid block mode

Arguments	`handle`, the handle of the required device
	
	`* time`, on exit, the time at which the trigger point occurred
	
	`* timeUnits`, on exit, the time units in which time is measured. The possible values are: -
	`PS3000A_FS`
	`PS3000A_PS`
	`PS3000A_NS`
	`PS3000A_US`
	`PS3000A_MS`
	`PS3000A_S`
	
	`segmentIndex`, the number of the memory segment for which the information is required

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_DEVICE_SAMPLING`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NOT_USED_IN_THIS_CAPTURE_MODE`
	`PICO_NOT_RESPONDING`
	`PICO_NULL_PARAMETER`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_DRIVER_FUNCTION`

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.21 ps3000aGetUnitInfo

```
PICO_STATUS ps3000aGetUnitInfo
(
  int16_t      handle,
  int8_t     * string,
  int16_t      stringLength,
  int16_t    * requiredSize,
  PICO_INFO    info
)
```

This function retrieves information about the specified oscilloscope. If the device fails to open or no device is opened, only the driver version is available.

Applicability	All modes
Arguments	handle, the handle of the device to query. If an invalid handle is passed, only the driver versions can be read. * string, on exit, the information string selected specified by the info argument. If string is NULL, only requiredSize is returned. stringLength, on entry, the maximum number of int8_t that may be written to string. * requiredSize, on exit, the required length of the string array. info, a number specifying what information is required. The possible values are listed in the table below.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_INVALID_INFO PICO_INFO_UNAVAILABLE PICO_DRIVER_FUNCTION

info		**Example**
0	PICO_DRIVER_VERSION Version number of PicoScope 3000A DLL	1,0,0,1
1	PICO_USB_VERSION Type of USB connection to device: 1.1, 2.0 or 3.0	2.0
2	PICO_HARDWARE_VERSION Hardware version of device	1
3	PICO_VARIANT_INFO Variant number of device	3206B
4	PICO_BATCH_AND_SERIAL Batch and serial number of device	KJL87/6
5	PICO_CAL_DATE Calibration date of device	30Sep09
6	PICO_KERNEL_VERSION Version of kernel driver	1,1,2,4
7	PICO_DIGITAL_HARDWARE_VERSION Hardware version of the digital section	1
8	PICO_ANALOGUE_HARDWARE_VERSION Hardware version of the analogue section	1

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.22 ps3000aGetValues

```
PICO_STATUS ps3000aGetValues
(
  int16_t                    handle,
  uint32_t                   startIndex,
  uint32_t                 * noOfSamples,
  uint32_t                   downSampleRatio,
  PS3000A_RATIO_MODE         downSampleRatioMode,
  uint32_t                   segmentIndex,
  int16_t                  * overflow
)
```

This function returns block-mode data, with or without downsampling, starting at the specified sample number. It is used to get the stored data from the driver after data collection has stopped.

Applicability	Block mode, rapid block mode

Arguments	`handle`, the handle of the required device.
	
	`startIndex`, a zero-based index that indicates the start point for data collection. It is measured in sample intervals from the start of the buffer.
	
	`* noOfSamples`, on entry, the number of samples required. On exit, the actual number retrieved. The number of samples retrieved will not be more than the number requested, and the data retrieved starts at `startIndex`.
	
	`downSampleRatio`, the downsampling factor that will be applied to the raw data.
	
	`downSampleRatioMode`, which downsampling mode to use. The available values are: -
	`PS3000A_RATIO_MODE_NONE` (`downSampleRatio` is ignored)
	`PS3000A_RATIO_MODE_AGGREGATE`
	`PS3000A_RATIO_MODE_AVERAGE`
	`PS3000A_RATIO_MODE_DECIMATE`
	
	`AGGREGATE`, `AVERAGE`, `DECIMATE` are single-bit constants that can be ORed to apply multiple downsampling modes to the same data.
	
	`segmentIndex`, the zero-based number of the memory segment where the data is stored.
	
	`* overflow`, on exit, a set of flags that indicate whether an overvoltage has occurred on any of the channels. It is a bit field with bit 0 denoting Channel A.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

Returns	PICO_OK PICO_INVALID_HANDLE PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_NO_SAMPLES_AVAILABLE PICO_DEVICE_SAMPLING PICO_NULL_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_STARTINDEX_INVALID PICO_ETS_NOT_RUNNING PICO_BUFFERS_NOT_SET PICO_INVALID_PARAMETER PICO_TOO_MANY_SAMPLES PICO_DATA_NOT_AVAILABLE PICO_STARTINDEX_INVALID PICO_INVALID_SAMPLERATIO PICO_INVALID_CALL PICO_NOT_RESPONDING PICO_MEMORY PICO_RATIO_MODE_NOT_SUPPORTED PICO_DRIVER_FUNCTION

4.22.1 Downsampling modes

Various methods of data reduction, or **downsampling**, are possible with PicoScope oscilloscopes. The downsampling is done at high speed by dedicated hardware inside the scope, making your application faster and more responsive than if you had to do all the data processing in software.

You specify the downsampling mode when you call one of the data collection functions such as ps3000aGetValues(). The following modes are available:

PS3000A_RATIO_MODE_AGGREGATE	Reduces every block of *n* values to just two values: a minimum and a maximum. The minimum and maximum values are returned in two separate buffers.
PS3000A_RATIO_MODE_DECIMATE	Reduces every block of *n* values to just the first value in the block, discarding all the other values.
PS3000A_RATIO_MODE_AVERAGE	Reduces every block of *n* values to a single value representing the average (arithmetic mean) of all the values.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.23 ps3000aGetValuesAsync

```
PICO_STATUS ps3000aGetValuesAsync
(
  int16_t               handle,
  uint32_t              startIndex,
  uint32_t              noOfSamples,
  uint32_t              downSampleRatio,
  PS3000A_RATIO_MODE    downSampleRatioMode,
  uint32_t              segmentIndex,
  void                * lpDataReady,
  void                * pParameter
)
```

This function returns data either with or without downsampling, starting at the specified sample number. It is used to get the stored data from the scope after data collection has stopped. It returns the data using a callback.

Applicability	Streaming mode and block mode
Arguments	`handle`, the handle of the required device `startIndex`, see ps3000aGetValues() `noOfSamples`, see ps3000aGetValues() `downSampleRatio`, see ps3000aGetValues() `downSampleRatioMode`, see ps3000aGetValues() `segmentIndex`, see ps3000aGetValues() `* lpDataReady`, a pointer to the user-supplied function that will be called when the data is ready. This will be ps3000aDataReady() for block-mode data or ps3000aStreamingReady() for streaming-mode data. `* pParameter`, a void pointer that will be passed to the callback function. The data type is determined by the application.
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_DEVICE_SAMPLING PICO_NULL_PARAMETER PICO_STARTINDEX_INVALID PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_PARAMETER PICO_DATA_NOT_AVAILABLE PICO_INVALID_SAMPLERATIO PICO_INVALID_CALL PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.24 ps3000aGetValuesBulk

```
PICO_STATUS ps3000aGetValuesBulk
(
  int16_t               handle,
  uint32_t            * noOfSamples,
  uint32_t              fromSegmentIndex,
  uint32_t              toSegmentIndex,
  uint32_t              downSampleRatio,
  PS3000A_RATIO_MODE    downSampleRatioMode,
  int16_t             * overflow
)
```

This function retrieves waveforms captured using rapid block mode. The waveforms must have been collected sequentially and in the same run.

Applicability	Rapid block mode
Arguments	handle, the handle of the device * noOfSamples, on entry, the number of samples required; on exit, the actual number retrieved. The number of samples retrieved will not be more than the number requested. The data retrieved always starts with the first sample captured. fromSegmentIndex, the first segment from which the waveform should be retrieved toSegmentIndex, the last segment from which the waveform should be retrieved downSampleRatio, see ps3000aGetValues() downSampleRatioMode, see ps3000aGetValues() * overflow, an array of integers equal to or larger than the number of waveforms to be retrieved. Each segment index has a corresponding entry in the overflow array, with overflow[0] containing the flags for the segment numbered fromSegmentIndex and the last element in the array containing the flags for the segment numbered toSegmentIndex. Each element in the array is a bit field as described under ps3000aGetValues().
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_INVALID_SAMPLERATIO PICO_ETS_NOT_RUNNING PICO_BUFFERS_NOT_SET PICO_TOO_MANY_SAMPLES PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE PICO_NOT_RESPONDING PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.25 ps3000aGetValuesOverlapped

```
PICO_STATUS ps3000aGetValuesOverlapped
(
  int16_t                handle,
  uint32_t               startIndex,
  uint32_t             * noOfSamples,
  uint32_t               downSampleRatio,
  PS3000A_RATIO_MODE     downSampleRatioMode,
  uint32_t               segmentIndex,
  int16_t              * overflow
)
```

This function allows you to make a deferred data-collection request, which will later be executed, and the arguments validated, when you call ps3000aRunBlock() in block mode. The advantage of this function is that the driver makes contact with the scope only once, when you call ps3000aRunBlock(), compared with the two contacts that occur when you use the conventional ps3000aRunBlock(), ps3000aGetValues() calling sequence. This slightly reduces the dead time between successive captures in block mode.

After calling ps3000aRunBlock(), you can optionally use ps3000aGetValues() to request further copies of the data. This might be required if you wish to display the data with different data reduction settings.

Applicability	Block mode
Arguments	handle, the handle of the device startIndex, see ps3000aGetValues() * noOfSamples, see ps3000aGetValues() downSampleRatio, see ps3000aGetValues() downSampleRatioMode, see ps3000aGetValues() segmentIndex, see ps3000aGetValues() * overflow, see ps3000aGetValuesBulk()
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.26 ps3000aGetValuesOverlappedBulk

```
PICO_STATUS ps3000aGetValuesOverlappedBulk
(
  int16_t              handle,
  uint32_t             startIndex,
  uint32_t           * noOfSamples,
  uint32_t             downSampleRatio,
  PS3000A_RATIO_MODE   downSampleRatioMode,
  uint32_t             fromSegmentIndex,
  uint32_t             toSegmentIndex,
  int16_t            * overflow
)
```

This function allows you to make a deferred data-collection request, which will later be executed, and the arguments validated, when you call ps3000aRunBlock() in rapid block mode. The advantage of this method is that the driver makes contact with the scope only once, when you call ps3000aRunBlock(), compared with the two contacts that occur when you use the conventional ps3000aRunBlock(), ps3000aGetValuesBulk() calling sequence. This slightly reduces the dead time between successive captures in rapid block mode.

After calling ps3000aRunBlock(), you can optionally use ps3000aGetValues() to request further copies of the data. This might be required if you wish to display the data with different data reduction settings.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the device `startIndex`, see ps3000aGetValues() `* noOfSamples`, see ps3000aGetValues() `downSampleRatio`, see ps3000aGetValues() `downSampleRatioMode`, see ps3000aGetValues() `fromSegmentIndex`, see ps3000aGetValuesBulk() `toSegmentIndex`, see ps3000aGetValuesBulk() `* overflow`, see ps3000aGetValuesBulk()
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.27 ps3000aGetValuesTriggerTimeOffsetBulk

```
PICO_STATUS ps3000aGetValuesTriggerTimeOffsetBulk
(
  int16_t                handle,
  uint32_t             * timesUpper,
  uint32_t             * timesLower,
  PS3000A_TIME_UNITS   * timeUnits,
  uint32_t               fromSegmentIndex,
  uint32_t               toSegmentIndex
)
```

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

This function is provided for use in programming environments that do not support 64-bit integers. If your programming environment supports this data type, it is easier to use ps3000aGetValuesTriggerTimeOffsetBulk64().

<table>
<tr><td>Applicability</td><td>Rapid block mode</td></tr>
<tr><td colspan="2">Arguments</td></tr>
<tr><td colspan="2">handle, the handle of the device

* timesUpper, an array of integers. On exit, the most significant 32 bits of the time offset for each requested segment index. times[0] will hold the fromSegmentIndex time offset and the last times index will hold the toSegmentIndex time offset. The array must be long enough to hold the number of requested times.

* timesLower, an array of integers. On exit, the least-significant 32 bits of the time offset for each requested segment index. times[0] will hold the fromSegmentIndex time offset and the last times index will hold the toSegmentIndex time offset. The array size must be long enough to hold the number of requested times.

* timeUnits, an array of integers. The array must be long enough to hold the number of requested times. On exit, timeUnits[0] will contain the time unit for fromSegmentIndex and the last element will contain the time unit for toSegmentIndex. Refer to ps3000aGetTriggerTimeOffset() for specific figures

fromSegmentIndex, the first segment for which the time offset is required

toSegmentIndex, the last segment for which the time offset is required. If toSegmentIndex is less than fromSegmentIndex then the driver will wrap around from the last segment to the first.</td></tr>
<tr><td>Returns</td><td>PICO_OK
PICO_POWER_SUPPLY_CONNECTED
PICO_POWER_SUPPLY_NOT_CONNECTED
PICO_INVALID_HANDLE
PICO_NOT_USED_IN_THIS_CAPTURE_MODE
PICO_NOT_RESPONDING
PICO_NULL_PARAMETER
PICO_DEVICE_SAMPLING
PICO_SEGMENT_OUT_OF_RANGE
PICO_NO_SAMPLES_AVAILABLE
PICO_DRIVER_FUNCTION</td></tr>
</table>

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.28 ps3000aGetValuesTriggerTimeOffsetBulk64

```
PICO_STATUS ps3000aGetValuesTriggerTimeOffsetBulk64
(
  int16_t                handle,
  int64_t              * times,
  PS3000A_TIME_UNITS   * timeUnits,
  uint32_t               fromSegmentIndex,
  uint32_t               toSegmentIndex
)
```

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

A 32-bit version of this function, ps3000aGetValuesTriggerTimeOffsetBulk(), is available for use with programming languages that do not support 64-bit integers.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the device `* times`, an array of integers. On exit, this will hold the time offset for each requested segment index. `times[0]` will hold the time offset for `fromSegmentIndex`, and the last `times` index will hold the time offset for `toSegmentIndex`. The array must be long enough to hold the number of times requested. `* timeUnits`, an array of integers long enough to hold the number of requested times. `timeUnits[0]` will contain the time unit for `fromSegmentIndex`, and the last element will contain the `toSegmentIndex`. Refer to ps3000aGetTriggerTimeOffset64() for specific figures. `fromSegmentIndex`, the first segment for which the time offset is required. The results for this segment will be placed in `times[0]` and `timeUnits[0]`. `toSegmentIndex`, the last segment for which the time offset is required. The results for this segment will be placed in the last elements of the `times` and `timeUnits` arrays. If `toSegmentIndex` is less than `fromSegmentIndex` then the driver will wrap around from the last segment to the first.
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_NOT_USED_IN_THIS_CAPTURE_MODE PICO_NOT_RESPONDING PICO_NULL_PARAMETER PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.29 ps3000aHoldOff

```
PICO_STATUS ps3000aHoldOff
(
  int16_t               handle,
  uint64_t              holdoff,
  PS3000A_HOLDOFF_TYPE  type
)
```

This function is for backward compatibility only and is not currently used.

Applicability	None
Arguments	Undefined
Returns	Undefined

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.30 ps3000aIsReady

```
PICO_STATUS ps3000aIsReady
(
  int16_t   handle,
  int16_t * ready
)
```

This function may be used instead of a callback function to receive data from ps3000aRunBlock(). To use this method, pass a NULL pointer as the lpReady argument to ps3000aRunBlock(). You must then poll the driver to see if it has finished collecting the requested samples.

Applicability	Block mode
Arguments	handle, the handle of the required device * ready, output: indicates the state of the collection. If zero, the device is still collecting. If non-zero, the device has finished collecting and ps3000aGetValues() can be used to retrieve the data.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_CANCELLED PICO_NOT_RESPONDING

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.31 ps3000aIsTriggerOrPulseWidthQualifierEnabled

```
PICO_STATUS ps3000aIsTriggerOrPulseWidthQualifierEnabled
(
  int16_t   handle,
  int16_t * triggerEnabled,
  int16_t * pulseWidthQualifierEnabled
)
```

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

Applicability	Call after setting up the trigger, and just before calling either ps3000aRunBlock() or ps3000aRunStreaming().
Arguments	`handle,` the handle of the required device `* triggerEnabled,` on exit, indicates whether the trigger will successfully be set when ps3000aRunBlock() or ps3000aRunStreaming() is called. A non-zero value indicates that the trigger is set, zero that the trigger is not set. `* pulseWidthQualifierEnabled,` on exit, indicates whether the pulse width qualifier will successfully be set when ps3000aRunBlock() or ps3000aRunStreaming() is called. A non-zero value indicates that the pulse width qualifier is set, zero that the pulse width qualifier is not set.

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.32 ps3000aMaximumValue

```
PICO_STATUS ps3000aMaximumValue
(
  int16_t   handle,
  int16_t * value
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device `* value`, returns the maximum ADC value
Returns	PICO_OK PICO_USER_CALLBACK PICO_INVALID_HANDLE PICO_TOO_MANY_SEGMENTS PICO_MEMORY PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.33 ps3000aMemorySegments

```
PICO_STATUS ps3000aMemorySegments
(
  int16_t    handle,
  uint32_t   nSegments,
  int32_t  * nMaxSamples
)
```

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

When the scope is opened, the number of segments defaults to 1, meaning that each capture fills the scope's available memory. This function allows you to divide the memory into a number of segments so that the scope can store several waveforms sequentially.

Applicability	All modes
Arguments	`handle`, the handle of the required device. `nSegments`, the number of segments required, from 1 to the value of `maxsegments` returned by ps3000aGetMaxSegments(). `* nMaxSamples`, on exit, the number of samples available in each segment. This is the total number over all channels, so if more than one channel is in use then the number of samples available to each channel is `nMaxSamples` divided by the number of channels.

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

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.34 ps3000aMinimumValue

```
PICO_STATUS ps3000aMinimumValue
(
  int16_t   handle,
  int16_t * value
)
```

This function returns the minimum ADC count returned by calls to ps3000aGetValues() and related functions

Applicability	All modes
Arguments	`handle`, the handle of the required device `* value`, returns the minimum ADC value

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.35 ps3000aNoOfStreamingValues

```
PICO_STATUS ps3000aNoOfStreamingValues
(
  int16_t    handle,
  uint32_t * noOfValues
)
```

This function returns the number of samples available after data collection in streaming mode. Call it after calling ps3000aStop().

Applicability	Streaming mode
Arguments	handle, the handle of the required device * noOfValues, on exit, the number of samples
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_NOT_USED PICO_BUSY PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.36 ps3000aOpenUnit

```
PICO_STATUS ps3000aOpenUnit
(
  int16_t * handle,
  int8_t  * serial
)
```

This function opens a PicoScope 3000 Series oscilloscope attached to the computer. The maximum number of units that can be opened depends on the operating system, the kernel driver and the computer. If ps3000aOpenUnit() is called without the power supply connected, the driver returns PICO_POWER_SUPPLY_NOT_CONNECTED.

Applicability	All modes
Arguments	* handle, on exit, the result of the attempt to open a scope: -1 : if the scope fails to open 0 : if no scope is found > 0 : a number that uniquely identifies the scope If a valid handle is returned, it must be used in all subsequent calls to API functions to identify this scope. * serial, on entry, a null-terminated string containing the serial number of the scope to be opened. If serial is NULL then the function opens the first scope found; otherwise, it tries to open the scope that matches the string.
Returns	PICO_OK PICO_OS_NOT_SUPPORTED PICO_OPEN_OPERATION_IN_PROGRESS PICO_EEPROM_CORRUPT PICO_KERNEL_DRIVER_TOO_OLD PICO_FPGA_FAIL PICO_MEMORY_CLOCK_FREQUENCY PICO_FW_FAIL PICO_MAX_UNITS_OPENED PICO_NOT_FOUND (if the specified unit was not found) PICO_NOT_RESPONDING PICO_MEMORY_FAIL PICO_ANALOG_BOARD PICO_CONFIG_FAIL_AWG PICO_INITIALISE_FPGA PICO_POWER_SUPPLY_NOT_CONNECTED

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.37 ps3000aOpenUnitAsync

```
PICO_STATUS ps3000aOpenUnitAsync
(
  int16_t * status,
  int8_t  * serial
)
```

This function opens a scope without blocking the calling thread. You can find out when it has finished by periodically calling ps3000aOpenUnitProgress() until that function returns a non-zero value.

Applicability	All modes
Arguments	* status, a status code: 0 if the open operation was disallowed because another open operation is in progress 1 if the open operation was successfully started * serial, see ps3000aOpenUnit()
Returns	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_OPERATION_FAILED

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.38 ps3000aOpenUnitProgress

```
PICO_STATUS ps3000aOpenUnitProgress
(
  int16_t * handle,
  int16_t * progressPercent,
  int16_t * complete
)
```

This function checks on the progress of a request made to ps3000aOpenUnitAsync() to open a scope.

Applicability	Use after ps3000aOpenUnitAsync()
Arguments	`* handle,` see ps3000aOpenUnit(). This handle is valid only if the function returns `PICO_OK`. `* progressPercent,` on exit, the percentage progress towards opening the scope. 100% implies that the open operation is complete. `* complete,` set to 1 when the open operation has finished

Returns	`PICO_OK`
	`PICO_NULL_PARAMETER`
	`PICO_OPERATION_FAILED`

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.39 ps3000aPingUnit

```
PICO_STATUS ps3000aPingUnit
(
  int16_t handle
)
```

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

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

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.40 ps3000aRunBlock

```
PICO_STATUS ps3000aRunBlock
(
  int16_t             handle,
  int32_t             noOfPreTriggerSamples,
  int32_t             noOfPostTriggerSamples,
  uint32_t            timebase,
  int16_t             oversample,
  int32_t           * timeIndisposedMs,
  uint32_t            segmentIndex,
  ps3000aBlockReady   lpReady,
  void              * pParameter
)
```

This function starts collecting data in block mode. For a step-by-step guide to this process, see Using block mode.

The number of samples is determined by `noOfPreTriggerSamples` and `noOfPostTriggerSamples` (see below for details). The total number of samples must not be more than the size of the segment referred to by `segmentIndex`.

<table>
<tr><td>Applicability</td><td>Block mode, rapid block mode</td></tr>
<tr><td colspan="2">Arguments</td></tr>
<tr><td colspan="2">
<code>handle,</code> the handle of the required device.

<code>noOfPreTriggerSamples,</code> the number of samples to return before the trigger event. If no trigger has been set then this argument is ignored and <code>noOfPostTriggerSamples</code> specifies the maximum number of samples to collect.

<code>noOfPostTriggerSamples,</code> the number of samples to be taken after a trigger event. If no trigger event has been set then this specifies the maximum number of samples to be taken. If a trigger condition has been set, this specifies the number of samples to be taken after a trigger has fired, and the number of samples to be collected is then: -

<code>noOfPreTriggerSamples + noOfPostTriggerSamples</code>

<code>timebase,</code> a number in the range 0 to $2^{32}-1$. See the guide to calculating timebase values.

<code>oversample,</code> not used.

<code>* timeIndisposedMs,</code> on exit, the time, in milliseconds, that the scope will spend collecting samples. This does not include any auto trigger timeout. If this pointer is null, nothing will be written here.

<code>segmentIndex,</code> zero-based, specifies which memory segment to use.

<code>lpReady,</code> a pointer to the ps3000aBlockReady() callback function that the driver will call when the data has been collected. To use the ps3000aIsReady() polling method instead of a callback function, set this pointer to NULL.

<code>* pParameter,</code> a void pointer that is passed to the ps3000aBlockReady() callback function. The callback can use this pointer to return arbitrary data to the application.
</td></tr>
<tr><td>Returns</td><td><code>PICO_OK</code>
<code>PICO_POWER_SUPPLY_CONNECTED</code></td></tr>
</table>

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

	PICO_POWER_SUPPLY_NOT_CONNECTED PICO_BUFFERS_NOT_SET (in overlapped mode) PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_CHANNEL PICO_INVALID_TRIGGER_CHANNEL PICO_INVALID_CONDITION_CHANNEL PICO_TOO_MANY_SAMPLES PICO_INVALID_TIMEBASE PICO_NOT_RESPONDING PICO_CONFIG_FAIL PICO_INVALID_PARAMETER PICO_NOT_RESPONDING PICO_TRIGGER_ERROR PICO_DRIVER_FUNCTION PICO_FW_FAIL PICO_NOT_ENOUGH_SEGMENTS (in bulk mode) PICO_PULSE_WIDTH_QUALIFIER PICO_SEGMENT_OUT_OF_RANGE (in overlapped mode) PICO_STARTINDEX_INVALID (in overlapped mode) PICO_INVALID_SAMPLERATIO (in overlapped mode) PICO_CONFIG_FAIL

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.41 ps3000aRunStreaming

```
PICO_STATUS ps3000aRunStreaming
(
  int16_t                handle,
  uint32_t             * sampleInterval,
  PS3000A_TIME_UNITS     sampleIntervalTimeUnits,
  uint32_t               maxPreTriggerSamples,
  uint32_t               maxPostTriggerSamples,
  int16_t                autoStop,
  uint32_t               downSampleRatio,
  PS3000A_RATIO_MODE     downSampleRatioMode,
  uint32_t               overviewBufferSize
)
```

This function tells the oscilloscope to start collecting data in streaming mode. When data has been collected from the device it is downsampled if necessary and then delivered to the application. Call ps3000aGetStreamingLatestValues() to retrieve the data. See Using streaming mode for a step-by-step guide to this process.

When a trigger is set, the total number of samples stored in the driver is the sum of maxPreTriggerSamples and maxPostTriggerSamples. If autoStop is false then this will become the maximum number of samples without downsampling.

Applicability	Streaming mode
Arguments	

handle, the handle of the required device.

* sampleInterval, on entry, the requested time interval between samples; on exit, the actual time interval used.

sampleIntervalTimeUnits, the unit of time used for sampleInterval. Use one of these enumerated types:
- PS3000A_FS
- PS3000A_PS
- PS3000A_NS
- PS3000A_US
- PS3000A_MS
- PS3000A_S

maxPreTriggerSamples, the maximum number of raw samples before a trigger event for each enabled channel. If no trigger condition is set this argument is ignored.

maxPostTriggerSamples, the maximum number of raw samples after a trigger event for each enabled channel. If no trigger condition is set, this argument states the maximum number of samples to be stored.

autoStop, a flag that specifies if the streaming should stop when all of maxSamples have been captured.

downSampleRatio: see ps3000aGetValues()
downSampleRatioMode: see ps3000aGetValues()

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 ps3000aSetDataBuffer().

Returns	PICO_OK PICO_INVALID_HANDLE PICO_ETS_MODE_SET PICO_USER_CALLBACK PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_STREAMING_FAILED PICO_NOT_RESPONDING PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_TRIGGER_ERROR PICO_INVALID_SAMPLE_INTERVAL PICO_INVALID_BUFFER PICO_DRIVER_FUNCTION PICO_FW_FAIL PICO_MEMORY

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.42 ps3000aSetBandwidthFilter

```
PICO_STATUS ps3000aSetBandwidthFilter
(
  int16_t                     handle,
  PS3000A_CHANNEL             channel,
  PS3000A_BANDWIDTH_LIMITER bandwidth
)
```

This function sets the bandwidth limiter for a specified channel.

Applicability	All modes. PicoScope 3400 and 3000D MSO Series scopes only.

Arguments	`handle`, the handle of the required device
	
	`channel`, the channel to be configured. Use one of the following enumerated types:
	`PS3000A_CHANNEL_A`: Channel A input
	`PS3000A_CHANNEL_B`: Channel B input
	`PS3000A_CHANNEL_C`: Channel C input (if present)
	`PS3000A_CHANNEL_D`: Channel D input (if present)
	
	`bandwidth`, either one of these values:
	`PS3000A_BW_FULL`
	`PS3000A_BW_20MHZ`

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_BANDWIDTH`

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.43 ps3000aSetChannel

```
PICO_STATUS ps3000aSetChannel
(
  int16_t            handle,
  PS3000A_CHANNEL    channel,
  int16_t            enabled,
  PS3000A_COUPLING   type,
  PS3000A_RANGE      range,
  float              analogueOffset
)
```

This function specifies whether an input channel is to be enabled, its input coupling type, voltage range and analog offset.

Applicability	All modes
Arguments	`handle`, the handle of the required device `channel`, the channel to be configured. Use one of the following enumerated types: `PS3000A_CHANNEL_A`: Channel A input `PS3000A_CHANNEL_B`: Channel B input `PS3000A_CHANNEL_C`: Channel C input `PS3000A_CHANNEL_D`: Channel D input `enabled`, whether or not to enable the channel (`TRUE` or `FALSE`) `type`, the impedance and coupling type. The values are: `PS3000A_AC`: 1 megohm impedance, AC coupling. The channel accepts input frequencies from about 1 hertz up to its maximum −3 dB analog bandwidth. `PS3000A_DC`: 1 megohm impedance, DC coupling. The scope accepts all input frequencies from zero (DC) up to its maximum −3 dB analog bandwidth. `range`, the input voltage range, one of these enumerated types: `PS3000A_50MV`: ±50 mV `PS3000A_100MV`: ±100 mV `PS3000A_200MV`: ±200 mV `PS3000A_500MV`: ±500 mV `PS3000A_1V`: ±1 V `PS3000A_2V`: ±2 V `PS3000A_5V`: ±5 V `PS3000A_10V`: ±10 V `PS3000A_20V`: ±20 V `analogueOffset`, a voltage to add to the input channel before digitization. The allowable range of offsets depends on the input range selected for the channel, as obtained from ps3000aGetAnalogueOffset().

Returns	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_VOLTAGE_RANGE`
	`PICO_INVALID_COUPLING`
	`PICO_INVALID_ANALOGUE_OFFSET`
	`PICO_DRIVER_FUNCTION`

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.44 ps3000aSetDataBuffer

```
PICO_STATUS ps3000aSetDataBuffer
(
  int16_t              handle,
  PS3000A_CHANNEL      channel,
  int16_t            * buffer,
  int32_t              bufferLth,
  uint32_t             segmentIndex,
  PS3000A_RATIO_MODE   mode
)
```

This function tells the driver where to store the data, either unprocessed or downsampled, that will be returned after the next call to one of the `GetValues` functions. The function allows you to specify only a single buffer, so for aggregation mode, which requires two buffers, you need to call ps3000aSetDataBuffers() instead.

You must allocate memory for the buffer before calling this function.

Applicability	Block, rapid block and streaming modes. All downsampling modes except aggregation.

Arguments	`handle`, the handle of the required device
	
	`channel`, the channel you want to use with the buffer. Use one of these enumerated types:
	`PS3000A_CHANNEL_A`
	`PS3000A_CHANNEL_B`
	`PS3000A_CHANNEL_C`
	`PS3000A_CHANNEL_D`
	
	To set the buffer for a digital port, use one of these enumerated types:
	`PS3000A_DIGITAL_PORT0 = 0x80`
	`PS3000A_DIGITAL_PORT1 = 0x81`
	
	`* buffer`, the location of the buffer
	
	`bufferLth`, the size of the `buffer` array
	
	`segmentIndex`, the number of the memory segment to be used
	
	`mode`, the downsampling mode. See ps3000aGetValues() for the available modes, but note that a single call to ps3000aSetDataBuffer() can only associate one buffer with one downsampling mode. If you intend to call ps3000aGetValues() with more than one downsampling mode activated, then you must call ps3000aSetDataBuffer() several times to associate a separate buffer with each downsampling mode.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_RATIO_MODE_NOT_SUPPORTED`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_DRIVER_FUNCTION`
	`PICO_INVALID_PARAMETER`

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.45 ps3000aSetDataBuffers

```
PICO_STATUS ps3000aSetDataBuffers
(
  int16_t                 handle,
  PS3000A_CHANNEL         channel,
  int16_t               * bufferMax,
  int16_t               * bufferMin,
  int32_t                 bufferLth,
  uint32_t                segmentIndex,
  PS3000A_RATIO_MODE      mode
)
```

This function tells the driver the location of one or two buffers for receiving data. You need to allocate memory for the buffers before calling this function. If you do not need two buffers, because you are not using aggregate mode, then you can optionally use ps3000aSetDataBuffer() instead.

Applicability	Block and streaming modes with aggregation.

Arguments	`handle`, the handle of the required device.
	
	`channel`, the channel for which you want to set the buffers. Use one of these constants:
	`PS3000A_CHANNEL_A`
	`PS3000A_CHANNEL_B`
	`PS3000A_CHANNEL_C`
	`PS3000A_CHANNEL_D`
	
	To set the buffer for a digital port, use one of these enumerated types:
	`PS3000A_DIGITAL_PORT0 = 0x80`
	`PS3000A_DIGITAL_PORT1 = 0x81`
	
	`* bufferMax`, a buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise.
	
	`* bufferMin`, a buffer to receive the minimum aggregated data values. Not used in other downsampling modes.
	
	`bufferLth`, the size of the `bufferMax` and `bufferMin` arrays.
	
	`segmentIndex`, the number of the memory segment to be used
	
	`mode`, see ps3000aGetValues()

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_RATIO_MODE_NOT_SUPPORTED`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_DRIVER_FUNCTION`
	`PICO_INVALID_PARAMETER`

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.46 ps3000aSetDigitalPort

```
PICO_STATUS ps3000aSetDigitalPort
(
  int16_t                  handle,
  PS3000A_DIGITAL_PORT     port,
  int16_t                  enabled,
  int16_t                  logiclevel
)
```

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

Applicability	Block and streaming modes with aggregation. MSOs only.
Arguments	`handle`, the handle of the required device. `port`, identifies the port for digital data: `PS3000A_DIGITAL_PORT0 = 0x80` (digital channels 0–7) `PS3000A_DIGITAL_PORT1 = 0x81` (digital channels 8–15) `enabled`, whether or not to enable the channel. The values are: `TRUE`: enable `FALSE`: do not enable `logiclevel`, the voltage at which the state transitions between 0 and 1. Range: –32767 (–5 V) to 32767 (5 V).

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_RATIO_MODE_NOT_SUPPORTED`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_DRIVER_FUNCTION`
	`PICO_INVALID_PARAMETER`

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.47 ps3000aSetEts

```
PICO_STATUS ps3000aSetEts
(
  int16_t              handle,
  PS3000A_ETS_MODE     mode,
  int16_t              etsCycles,
  int16_t              etsInterleave,
  int32_t            * sampleTimePicoseconds
)
```

This function is used to enable or disable ETS (equivalent-time sampling) and to set the ETS parameters. See ETS overview for an explanation of ETS mode.

Applicability	Block mode
Arguments	
`handle`, the handle of the required device `mode`, the ETS mode. Use one of these values: `PS3000A_ETS_OFF` – disables ETS `PS3000A_ETS_FAST` – enables ETS and provides `etsCycles` of data, which may contain data from previously returned cycles `PS3000A_ETS_SLOW` – enables ETS and provides fresh data every `etsCycles`. This mode takes longer to provide each data set, but the data sets are more stable and are guaranteed to contain only new data. `etsCycles`, the number of cycles to store: the driver then selects `etsInterleave` cycles to give the most uniform spread of samples. Range: between two and five times the value of `etsInterleave`, and not more than the `etsCycles` value returned by ps3000aGetMaxEtsValues(). `etsInterleave`, the number of waveforms to combine into a single ETS capture. The maximum allowed value for the selected device is returned by ps3000aGetMaxEtsValues() in the `etsInterleave` argument. `* sampleTimePicoseconds`, on exit, the effective sampling interval of the ETS data. For example, if the captured sample time is 4 ns and `etsInterleave` is 10, the effective sample time in ETS mode is 400 ps.	

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.48 ps3000aSetEtsTimeBuffer

```
PICO_STATUS ps3000aSetEtsTimeBuffer
(
  int16_t   handle,
  int64_t * buffer,
  int32_t   bufferLth
)
```

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

Applicability	ETS mode only. If your programming language does not support 64-bit data, use the 32-bit version ps3000aSetEtsTimeBuffers() instead.
Arguments	`handle`, the handle of the required device `* buffer`, an array of 64-bit words, each representing the time in picoseconds at which the sample was captured `bufferLth`, the size of the buffer array

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.49 ps3000aSetEtsTimeBuffers

```
PICO_STATUS ps3000aSetEtsTimeBuffers
(
  int16_t    handle,
  uint32_t * timeUpper,
  uint32_t * timeLower,
  int32_t    bufferLth
)
```

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

Applicability	ETS mode only. If your programming language supports 64-bit data then you can use ps3000aSetEtsTimeBuffer() instead.
Arguments	handle, the handle of the required device * timeUpper, an array of 32-bit words, each representing the upper 32 bits of the time in picoseconds at which the sample was captured * timeLower, an array of 32-bit words, each representing the lower 32 bits of the time in picoseconds at which the sample was captured bufferLth, the size of the timeUpper and timeLower arrays
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_DRIVER_FUNCTION

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.50 ps3000aSetNoOfCaptures

```
PICO_STATUS ps3000aSetNoOfCaptures
(
  int16_t  handle,
  uint32_t nCaptures
)
```

This function sets the number of captures to be collected in one run of rapid block mode. If you do not call this function before a run, the driver will capture only one waveform. Once a value has been set, the value remains constant unless changed.

Applicability	Rapid block mode
Arguments	handle, the handle of the device nCaptures, the number of waveforms to capture in one run
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.51 ps3000aSetPulseWidthDigitalPortProperties

```
PICO_STATUS ps3000aSetPulseWidthDigitalPortProperties
(
  int16_t                              handle,
  PS3000A_DIGITAL_CHANNEL_DIRECTIONS * directions
  int16_t                              nDirections
)
```

This function will set the individual digital channels' pulse-width trigger directions. Each trigger direction consists of a channel name and a direction. If the channel is not included in the array of PS3000A_DIGITAL_CHANNEL_DIRECTIONS the driver assumes the digital channel's pulse-width trigger direction is PS3000A_DIGITAL_DONT_CARE.

Applicability	All modes
Arguments	handle, the handle of the required device. * directions, a pointer to an array of PS3000A_DIGITAL_CHANNEL_DIRECTIONS structures describing the requested properties. The array can contain a single element describing the properties of one channel, or a number of elements describing several digital channels. If directions is NULL, digital pulse-width triggering is switched off. A digital channel that is not included in the array will be set to PS3000A_DIGITAL_DONT_CARE. nDirections, the number of digital channel directions being passed to the driver.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_INVALID_DIGITAL_CHANNEL PICO_INVALID_DIGITAL_TRIGGER_DIRECTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.52 ps3000aSetPulseWidthQualifier

```
PICO_STATUS ps3000aSetPulseWidthQualifier
(
  int16_t                        handle,
  PS3000A_PWQ_CONDITIONS       * conditions,
  int16_t                        nConditions,
  PS3000A_THRESHOLD_DIRECTION    direction,
  uint32_t                       lower,
  uint32_t                       upper,
  PS3000A_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 level triggering or window triggering to produce more complex triggers. The pulse-width qualifier is set by defining one or more structures that are then ORed together. Each structure is itself the AND of the states of one or more of the inputs. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

Applicability	All modes
Arguments	

handle, the handle of the required device

* conditions, an array of PS3000A_PWQ_CONDITIONS structures* specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there are several elements, the overall trigger condition is the logical OR of all the elements. If conditions is NULL then the pulse-width qualifier is not used.

nConditions, the number of elements in the conditions array. If nConditions is zero then the pulse-width qualifier is not used.
Range: 0 to PS3000A_MAX_PULSE_WIDTH_QUALIFIER_COUNT.

direction, the direction of the signal required for the pulse width trigger to fire. See PS3000A_THRESHOLD_DIRECTION constants for the list of possible values. Each channel of the oscilloscope (except the EXT input) has two thresholds for each direction—for example, PS3000A_RISING and PS3000A_RISING_LOWER—so that one can be used for the pulse-width qualifier and the other for the level trigger. The driver will not let you use the same threshold for both triggers; so, for example, you cannot use PS3000A_RISING as the direction argument for both ps3000aSetTriggerConditions() and ps3000aSetPulseWidthQualifier() at the same time. There is no such restriction when using window triggers.

lower, the lower limit of the pulse-width counter, measured in samples.

upper, the upper limit of the pulse-width counter, measured in samples. This parameter is used only when the type is set to PS3000A_PW_TYPE_IN_RANGE or PS3000A_PW_TYPE_OUT_OF_RANGE.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

Arguments	type, the pulse-width type, one of these constants: PS3000A_PW_TYPE_NONE: do not use the pulse width qualifier PS3000A_PW_TYPE_LESS_THAN: pulse width less than lower PS3000A_PW_TYPE_GREATER_THAN: pulse width greater than lower PS3000A_PW_TYPE_IN_RANGE: pulse width between lower and upper PS3000A_PW_TYPE_OUT_OF_RANGE: pulse width not between lower and upper
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_PULSE_WIDTH_QUALIFIER PICO_DRIVER_FUNCTION

*Note: using this function the driver will convert the PS3000A_PWQ_CONDITIONS into a PS3000A_PWQ_CONDITIONS_V2 and will set the condition for digital to PS3000A_DIGITAL_DONT_CARE.

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.52.1 PS3000A_PWQ_CONDITIONS structure

A structure of this type is passed to ps3000aSetPulseWidthQualifier() in the `conditions` argument to specify the trigger conditions. It is defined as follows:

```
typedef struct tPS3000APwqConditions
{
  PS3000A_TRIGGER_STATE channelA;
  PS3000A_TRIGGER_STATE channelB;
  PS3000A_TRIGGER_STATE channelC;
  PS3000A_TRIGGER_STATE channelD;
  PS3000A_TRIGGER_STATE external;
  PS3000A_TRIGGER_STATE aux;
} PS3000A_PWQ_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The ps3000aSetPulseWidthQualifier() function can OR together a number of these structures to produce the final pulse width qualifier, which can therefore be any possible Boolean function of the scope's inputs.

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

Applicability	All models*

Elements	`channelA, channelB, channelC**, channelD**, external,` the type of condition that should be applied to each channel. Use these constants: -
	`PS3000A_CONDITION_DONT_CARE`
	`PS3000A_CONDITION_TRUE`
	`PS3000A_CONDITION_FALSE`
	
	The channels that are set to `PS3000A_CONDITION_TRUE` or `PS3000A_CONDITION_FALSE` must all meet their conditions simultaneously to produce a trigger. Channels set to `PS3000A_CONDITION_DONT_CARE` are ignored.
	
	`aux,` not used

*Note: using this function the driver will convert the `PS3000A_PWQ_CONDITIONS` into a `PS3000A_PWQ_CONDITIONS_V2` and will set the condition for digital to `PS3000A_DIGITAL_DONT_CARE.`

**Note: applicable to 4-channel oscilloscopes only.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.53 ps3000aSetPulseWidthQualifierV2

```
PICO_STATUS ps3000aSetPulseWidthQualifierV2
(
  int16_t                        handle,
  PS3000A_PWQ_CONDITIONS_V2    * conditions,
  int16_t                        nConditions,
  PS3000A_THRESHOLD_DIRECTION    direction,
  uint32_t                       lower,
  uint32_t                       upper,
  PS3000A_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 level triggering or window triggering to produce more complex triggers. The pulse-width qualifier is set by defining one or more structures that are then ORed together. Each structure is itself the AND of the states of one or more of the inputs. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

Applicability	All modes
Arguments	

handle, the handle of the required device

* conditions, an array of PS3000A_PWQ_CONDITIONS_V2 structures specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there are several elements, the overall trigger condition is the logical OR of all the elements. If conditions is NULL then the pulse-width qualifier is not used.

nConditions, the number of elements in the conditions array. If nConditions is zero then the pulse-width qualifier is not used.
Range: 0 to PS3000A_MAX_PULSE_WIDTH_QUALIFIER_COUNT.

direction, the direction of the signal required for the pulse width trigger to fire. See PS3000A_THRESHOLD_DIRECTION constants for the list of possible values. Each channel of the oscilloscope (except the EXT input) has two thresholds for each direction—for example, PS3000A_RISING and PS3000A_RISING_LOWER—so that one can be used for the pulse-width qualifier and the other for the level trigger. The driver will not let you use the same threshold for both triggers; so, for example, you cannot use PS3000A_RISING as the direction argument for both ps3000aSetTriggerConditionsV2() and ps3000aSetPulseWidthQualifierV2() at the same time. There is no such restriction when using window triggers.

lower, the lower limit of the pulse-width counter, measured in samples.

upper, the upper limit of the pulse-width counter, measured in samples. This parameter is used only when the type is set to PS3000A_PW_TYPE_IN_RANGE or PS3000A_PW_TYPE_OUT_OF_RANGE.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

Arguments	type, the pulse-width type, one of these constants: PS3000A_PW_TYPE_NONE: do not use the pulse width qualifier PS3000A_PW_TYPE_LESS_THAN: pulse width less than lower PS3000A_PW_TYPE_GREATER_THAN: pulse width greater than lower PS3000A_PW_TYPE_IN_RANGE: pulse width between lower and upper PS3000A_PW_TYPE_OUT_OF_RANGE: pulse width not between lower and upper
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_PULSE_WIDTH_QUALIFIER PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.53.1 PS3000A_PWQ_CONDITIONS_V2 structure

A structure of this type is passed to ps3000aSetPulseWidthQualifierV2() in the `conditions` argument to specify the trigger conditions. It is defined as follows:

```
typedef struct tPS3000APwqConditionsV2
{
  PS3000A_TRIGGER_STATE channelA;
  PS3000A_TRIGGER_STATE channelB;
  PS3000A_TRIGGER_STATE channelC;
  PS3000A_TRIGGER_STATE channelD;
  PS3000A_TRIGGER_STATE external;
  PS3000A_TRIGGER_STATE aux;
  PS3000A_TRIGGER_STATE digital;
} PS3000A_PWQ_CONDITIONS_V2
```

Each structure is the logical AND of the states of the scope's inputs. The ps3000aSetPulseWidthQualifierV2() function can OR together a number of these structures to produce the final pulse width qualifier, which can therefore be any possible Boolean function of the scope's inputs.

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

Applicability	All models

Elements	`channelA, channelB, channelC*, channelD*, external,` the type of condition that should be applied to each channel. Use these constants: -
	`PS3000A_CONDITION_DONT_CARE`
	`PS3000A_CONDITION_TRUE`
	`PS3000A_CONDITION_FALSE`
	
	The channels that are set to `PS3000A_CONDITION_TRUE` or `PS3000A_CONDITION_FALSE` must all meet their conditions simultaneously to produce a trigger. Channels set to `PS3000A_CONDITION_DONT_CARE` are ignored.
	
	`aux,` not used

*Note: applicable to 4-channel analog devices only.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.54 ps3000aSetSigGenArbitrary

```
PICO_STATUS ps3000aSetSigGenArbitrary
(
  int16_t                          handle,
  int32_t                          offsetVoltage,
  uint32_t                         pkToPk,
  uint32_t                         startDeltaPhase,
  uint32_t                         stopDeltaPhase,
  uint32_t                         deltaPhaseIncrement,
  uint32_t                         dwellCount,
  int16_t                        * arbitraryWaveform,
  int32_t                          arbitraryWaveformSize,
  PS3000A_SWEEP_TYPE               sweepType,
  PS3000A_EXTRA_OPERATIONS         operation,
  PS3000A_INDEX_MODE               indexMode,
  uint32_t                         shots,
  uint32_t                         sweeps,
  PS3000A_SIGGEN_TRIG_TYPE         triggerType,
  PS3000A_SIGGEN_TRIG_SOURCE       triggerSource,
  int16_t                          extInThreshold
)
```

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

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

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

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

where:

outputFrequency	=	repetition rate of the complete arbitrary waveform
dacFrequency	=	update rate of AWG DAC (see table below)
deltaPhase	=	calculated from `startDeltaPhase` and `deltaPhaseIncrement` (use ps3000aSigGenFrequencyToPhase() to do the calculation for you)
phaseAccumulatorSize	=	maximum count of phase accumulator (see table below)
awgBufferSize	=	maximum AWG buffer size (see table below)
arbitraryWaveformSize	=	length in samples of the user-defined waveform

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

Parameter	PicoScope 3204B 3204 MSO 3205B 3205 MSO 3404B 3405B	PicoScope 3206B 3206 MSO 3406B	PicoScope All 3000D All 3000D MSO	PicoScope 3207B
dacFrequency	20 MHz			100 MHz
dacPeriod (= 1/*dacFrequency*)	50 ns			10 ns
phaseAccumulatorSize	4,294,967,296 (2^{32})			
awgBufferSize	8192 (2^{13})	16,384 (2^{14})	32,768 (2^{15})	

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

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

Applicability All modes. All models with AWG.

Arguments

`handle`, the handle of the required device.

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

`pkToPk`, the peak-to-peak voltage, in microvolts, of the waveform signal.
Note that if the signal voltages described by the combination of `offsetVoltage` and `pkToPk` extend outside the voltage range of the signal generator, the output waveform will be clipped.

`startDeltaPhase`, the initial value added to the phase accumulator as the generator begins to step through the waveform buffer. Calculate this value from the information above, or use ps3000aSigGenFrequencyToPhase().

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

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

`dwellCount`, the time, in units of *dacPeriod,* between successive additions of `deltaPhaseIncrement` to the delta phase accumulator. This determines the rate at which the generator sweeps the output frequency.

Minimum value: `PS3000A_MIN_DWELL_COUNT`

`* arbitraryWaveform`, a buffer that holds the waveform pattern as a set of samples equally spaced in time. If `pkToPk` is set to its maximum (4 V) and `offsetVoltage` is set to 0 V:

a sample of `minArbitraryWaveformValue` corresponds to –2 V
a sample of `maxArbitraryWaveformValue` corresponds to +2 V

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

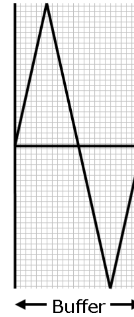
where minArbitraryWaveformValue and maxArbitraryWaveformValue are the values returned by ps3000aSigGenArbitraryMinMaxValues(). arbitraryWaveformSize, the size of the arbitrary waveform buffer, in samples, in the range: [minArbitraryWaveformSize, maxArbitraryWaveformSize] where minArbitraryWaveformSize and maxArbitraryWaveformSize are the values returned by ps3000aSigGenArbitraryMinMaxValues(). 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 enumerated types: - PS3000A_UP PS3000A_DOWN PS3000A_UPDOWN PS3000A_DOWNUP operation, the type of waveform to be produced, specified by one of the following enumerated types: PS3000A_ES_OFF, normal signal generator operation specified by wavetype. PS3000A_WHITENOISE, the signal generator produces white noise and ignores all settings except pkToPk and offsetVoltage. PS3000A_PRBS, produces a pseudorandom random binary sequence with a bit rate specified by the start and stop frequency. indexMode, specifies how the signal will be formed from the arbitrary waveform data. Single and dual index modes are possible. Use one of these constants: PS3000A_SINGLE PS3000A_DUAL shots, see ps3000aSigGenBuiltIn() sweeps, see ps3000aSigGenBuiltIn() triggerType, see ps3000aSigGenBuiltIn() triggerSource, see ps3000aSigGenBuiltIn() extInThreshold, see ps3000aSigGenBuiltIn()	
Returns	PICO_OK PICO_AWG_NOT_SUPPORTED PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_BUSY PICO_INVALID_HANDLE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING PICO_NOT_RESPONDING PICO_WARNING_EXT_THRESHOLD_CONFLICT PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_OFFSET_VOLTAGE PICO_SIGGEN_PK_TO_PK PICO_SIGGEN_OUTPUT_OVER_VOLTAGE PICO_DRIVER_FUNCTION PICO_SIGGEN_WAVEFORM_SETUP_FAILED

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

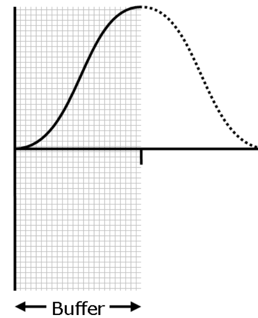
4.54.1 AWG index modes

The arbitrary waveform generator supports **single** and **dual** index modes to help you make the best use of the waveform buffer.

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



Dual mode. The generator outputs the contents of the buffer from beginning to end, and then does a second pass in the reverse direction through the buffer. This allows you to specify only the first half of a waveform with twofold symmetry, such as a Gaussian function, and let the generator fill in the other half.



 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.55 ps3000aSetSigGenBuiltIn

```
PICO_STATUS ps3000aSetSigGenBuiltIn
(
  int16_t                      handle,
  int32_t                      offsetVoltage,
  uint32_t                     pkToPk,
  PS3000A_WAVE_TYPE            waveType,
  float                        startFrequency,
  float                        stopFrequency,
  float                        increment,
  float                        dwellTime,
  PS3000A_SWEEP_TYPE           sweepType,
  PS3000A_EXTRA_OPERATIONS     operation,
  uint32_t                     shots,
  uint32_t                     sweeps,
  PS3000A_SIGGEN_TRIG_TYPE     triggerType,
  PS3000A_SIGGEN_TRIG_SOURCE   triggerSource,
  int16_t                      extInThreshold
)
```

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

Applicability	All models

Arguments

handle, the handle of the required device

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

pkToPk, the peak-to-peak voltage, in microvolts, of the waveform signal. Note that if the signal voltages described by the combination of offsetVoltage and pkToPk extend outside the voltage range of the signal generator, the output waveform will be clipped.

waveType, the type of waveform to be generated.

PS3000A_SINE	sine wave
PS3000A_SQUARE	square wave
PS3000A_TRIANGLE	triangle wave
PS3000A_DC_VOLTAGE	DC voltage

The following waveTypes apply to B and MSO models only.

PS3000A_RAMP_UP	rising sawtooth
PS3000A_RAMP_DOWN	falling sawtooth
PS3000A_SINC	sin (x)/x
PS3000A_GAUSSIAN	Gaussian
PS3000A_HALF_SINE	half (full-wave rectified) sine

startFrequency, the frequency that the signal generator will initially produce. For allowable values see PS3000A_SINE_MAX_FREQUENCY and related values.

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

<table>
<tr><td>Arguments</td><td>
dwellTime, the time for which the sweep stays at each frequency, in seconds

sweepType, whether the frequency will sweep from startFrequency to stopFrequency, or in the opposite direction, or repeatedly reverse direction. Use one of these constants:
PS3000A_UP
PS3000A_DOWN
PS3000A_UPDOWN
PS3000A_DOWNUP

operation, the type of waveform to be produced, specified by one of the following enumerated types (MSO and B models only):
PS3000A_ES_OFF, normal signal generator operation specified by wavetype.
PS3000A_WHITENOISE, the signal generator produces white noise and ignores all settings except pkToPk and offsetVoltage.
PS3000A_PRBS, produces a pseudorandom binary sequence with bit rate specified by the start and stop frequencies.

shots,
0: sweep the frequency as specified by sweeps
1...PS3000A_MAX_SWEEPS_SHOTS: the number of cycles of the waveform to be produced after a trigger event. sweeps must be zero.
PS3000A_SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN: start and run continuously after trigger occurs

sweeps,
0: produce number of cycles specified by shots
1..PS3000A_MAX_SWEEPS_SHOTS: the number of times to sweep the frequency after a trigger event, according to sweepType. shots must be zero.
PS3000A_SHOT_SWEEP_TRIGGER_CONTINUOUS_RUN: start a sweep and continue after trigger occurs

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

<table>
<tr><td>PS3000A_SIGGEN_RISING</td><td>trigger on rising edge</td></tr>
<tr><td>PS3000A_SIGGEN_FALLING</td><td>trigger on falling edge</td></tr>
<tr><td>PS3000A_SIGGEN_GATE_HIGH</td><td>run while trigger is high</td></tr>
<tr><td>PS3000A_SIGGEN_GATE_LOW</td><td>run while trigger is low</td></tr>
</table>

triggerSource, the source that will trigger the signal generator:

<table>
<tr><td>PS3000A_SIGGEN_NONE</td><td>run without waiting for trigger</td></tr>
<tr><td>PS3000A_SIGGEN_SCOPE_TRIG</td><td>use scope trigger</td></tr>
<tr><td>PS3000A_SIGGEN_EXT_IN</td><td>use EXT input</td></tr>
<tr><td>PS3000A_SIGGEN_SOFT_TRIG</td><td>wait for software trigger provided by ps3000aSigGenSoftwareControl()</td></tr>
<tr><td>PS3000A_SIGGEN_TRIGGER_RAW</td><td>reserved</td></tr>
</table>
</td></tr>
</table>

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

Arguments	If a trigger source other than P3000A_SIGGEN_NONE is specified, then either shots or sweeps, but not both, must be non-zero. extInThreshold, used to set trigger level for external trigger.
Returns	PICO_OK PICO_BUSY PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING PICO_NOT_RESPONDING PICO_WARNING_AUX_OUTPUT_CONFLICT PICO_WARNING_EXT_THRESHOLD_CONFLICT PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_OFFSET_VOLTAGE PICO_SIGGEN_PK_TO_PK PICO_SIGGEN_OUTPUT_OVER_VOLTAGE PICO_DRIVER_FUNCTION PICO_SIGGEN_WAVEFORM_SETUP_FAILED PICO_NOT_RESPONDING

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.56 ps3000aSetSigGenBuiltInV2

```
PICO_STATUS ps3000aSetSigGenBuiltInV2
(
  int16_t                     handle,
  int32_t                     offsetVoltage,
  uint32_t                    pkToPk,
  PS3000A_WAVE_TYPE           waveType,
  double                      startFrequency,
  double                      stopFrequency,
  double                      increment,
  double                      dwellTime,
  PS3000A_SWEEP_TYPE          sweepType,
  PS3000A_EXTRA_OPERATIONS    operation,
  uint32_t                    shots,
  uint32_t                    sweeps,
  PS3000A_SIGGEN_TRIG_TYPE    triggerType,
  PS3000A_SIGGEN_TRIG_SOURCE  triggerSource,
  int16_t                     extInThreshold
)
```

This function is an upgraded version of ps3000aSetSigGenBuiltIn() with double-precision frequency arguments for more precise control at low frequencies.

Applicability	All models
Arguments	See ps3000aSetSigGenBuiltIn()
Returns	See ps3000aSetSigGenBuiltIn()

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.57 ps3000aSetSigGenPropertiesArbitrary

```
PICO_STATUS ps3000aSetSigGenPropertiesArbitrary
(
  int16_t                       handle,
  uint32_t                      startDeltaPhase,
  uint32_t                      stopDeltaPhase,
  uint32_t                      deltaPhaseIncrement,
  uint32_t                      dwellCount,
  PS3000A_SWEEP_TYPE            sweepType,
  uint32_t                      shots,
  uint32_t                      sweeps,
  PS3000A_SIGGEN_TRIG_TYPE      triggerType,
  PS3000A_SIGGEN_TRIG_SOURCE    triggerSource,
  int16_t                       extInThreshold
)
```

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

Applicability	All modes
Arguments	See ps3000aSetSigGenArbitrary()
Returns	0: if successful. Error code: if failed

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.58 ps3000aSetSigGenPropertiesBuiltIn

```
PICO_STATUS ps3000aSetSigGenPropertiesBuiltIn
(
  int16_t                       handle,
  double                        startFrequency,
  double                        stopFrequency,
  double                        increment,
  double                        dwellTime,
  PS3000A_SWEEP_TYPE            sweepType,
  uint32_t                      shots,
  uint32_t                      sweeps,
  PS3000A_SIGGEN_TRIG_TYPE      triggerType,
  PS3000A_SIGGEN_TRIG_SOURCE    triggerSource,
  int16_t                       extInThreshold
)
```

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

Applicability	All modes
Arguments	See ps3000aSetSigGenBuiltIn()
Returns	0: if successful. Error code: if failed

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.59 ps3000aSetSimpleTrigger

```
PICO_STATUS ps3000aSetSimpleTrigger
(
  int16_t                        handle,
  int16_t                        enable,
  PS3000A_CHANNEL                source,
  int16_t                        threshold,
  PS3000A_THRESHOLD_DIRECTION    direction,
  uint32_t                       delay,
  int16_t                        autoTrigger_ms
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device. `enable`, zero to disable the trigger, any non-zero value to set the trigger. `source`, the channel on which to trigger. `threshold`, the ADC count at which the trigger will fire. `direction`, the direction in which the signal must move to cause a trigger. The following directions are supported: `ABOVE`, `BELOW`, `RISING`, `FALLING` and `RISING_OR_FALLING`. `delay`, the time between the trigger occurring and the first sample. For example, if `delay = 100`, the scope would wait 100 sample periods before sampling. At a timebase of 500 MS/s, or 2 ns per sample, the total delay would then be 100 x 2 ns = 200 ns. Range: 0 to `MAX_DELAY_COUNT`. `autoTrigger_ms`, the number of milliseconds the device will wait if no trigger occurs. If this is set to zero, the scope device will wait indefinitely for a trigger.

Returns	`PICO_OK`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_PARAMETER`
	`PICO_MEMORY`
	`PICO_CONDITIONS`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_DRIVER_FUNCTION`

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.60 ps3000aSetTriggerChannelConditions

```
PICO_STATUS ps3000aSetTriggerChannelConditions
(
  int16_t                        handle,
  PS3000A_TRIGGER_CONDITIONS *  conditions,
  int16_t                        nConditions
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is defined by one or more PS3000A_TRIGGER_CONDITIONS structures that are then ORed together. Each structure is itself the AND of the states of one or more of the inputs. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

If complex triggering is not required, use ps3000aSetSimpleTrigger().

Applicability	All modes
Arguments	handle, the handle of the required device. * conditions, an array of PS3000A_TRIGGER_CONDITIONS structures* specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there is more than one element, the overall trigger condition is the logical OR of all the elements. nConditions, the number of elements in the conditions array. If nConditions is zero then triggering is switched off.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_MEMORY PICO_DRIVER_FUNCTION

*Note: using this function the driver will convert the PS3000A_TRIGGER_CONDITIONS into a PS3000A_TRIGGER_CONDITIONS_V2 and will set the condition for digital to PS3000A_DIGITAL_DONT_CARE.

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.60.1 PS3000A_TRIGGER_CONDITIONS structure

A structure of this type is passed to ps3000aSetTriggerChannelConditions() in the `conditions` argument to specify the trigger conditions, and is defined as follows: -

```
typedef struct tPS3000ATriggerConditions
{
  PS3000A_TRIGGER_STATE channelA;
  PS3000A_TRIGGER_STATE channelB;
  PS3000A_TRIGGER_STATE channelC;
  PS3000A_TRIGGER_STATE channelD;
  PS3000A_TRIGGER_STATE external;
  PS3000A_TRIGGER_STATE aux;
  PS3000A_TRIGGER_STATE pulseWidthQualifier;
} PS3000A_TRIGGER_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The ps3000aSetTriggerChannelConditions() function can OR together a number of these structures to produce the final trigger condition, which can be any possible Boolean function of the scope's inputs.

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

Elements	`channelA, channelB, channelC, channelD, external, pulseWidthQualifier,` the type of condition that should be applied to each channel. Use these constants:
	`PS3000A_CONDITION_DONT_CARE`
	`PS3000A_CONDITION_TRUE`
	`PS3000A_CONDITION_FALSE`
	
	The channels that are set to `PS3000A_CONDITION_TRUE` or `PS3000A_CONDITION_FALSE` must all meet their conditions simultaneously to produce a trigger. Channels set to `PS3000A_CONDITION_DONT_CARE` are ignored.
	
	`aux,` not used

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.61 ps3000aSetTriggerChannelConditionsV2

```
PICO_STATUS ps3000aSetTriggerChannelConditionsV2
(
  int16_t                          handle,
  PS3000A_TRIGGER_CONDITIONS_V2 *  conditions,
  int16_t                          nConditions
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is defined by one or more PS3000A_TRIGGER_CONDITIONS_V2 structures that are then ORed together. Each structure is itself the AND of the states of one or more of the inputs. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

If complex triggering is not required, use ps3000aSetSimpleTrigger().

Applicability	All modes
Arguments	handle, the handle of the required device. * conditions, an array of PS3000A_TRIGGER_CONDITIONS_V2 structures specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there is more than one element, the overall trigger condition is the logical OR of all the elements. nConditions, the number of elements in the conditions array. If nConditions is zero then triggering is switched off.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_MEMORY PICO_DRIVER_FUNCTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.61.1 PS3000A_TRIGGER_CONDITIONS_V2 structure

A structure of this type is passed to ps3000aSetTriggerChannelConditionsV2() in the `conditions` argument to specify the trigger conditions, and is defined as follows: -

```
typedef struct tPS3000ATriggerConditionsV2
{
  PS3000A_TRIGGER_STATE channelA;
  PS3000A_TRIGGER_STATE channelB;
  PS3000A_TRIGGER_STATE channelC;
  PS3000A_TRIGGER_STATE channelD;
  PS3000A_TRIGGER_STATE external;
  PS3000A_TRIGGER_STATE aux;
  PS3000A_TRIGGER_STATE pulseWidthQualifier;
  PS3000A_TRIGGER_STATE digital;
} PS3000A_TRIGGER_CONDITIONS_V2;
```

Each structure is the logical AND of the states of the scope's inputs. ps3000aSetTriggerChannelConditionsV2() can OR together a number of these structures to produce the final trigger condition, which can be any possible Boolean function of the scope's inputs.

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

Elements	`channelA, channelB, channelC, channelD, external, pulseWidthQualifier,` the type of condition that should be applied to each channel. Use these constants:
	`PS3000A_CONDITION_DONT_CARE`
	`PS3000A_CONDITION_TRUE`
	`PS3000A_CONDITION_FALSE`
	
	The channels that are set to `PS3000A_CONDITION_TRUE` or `PS3000A_CONDITION_FALSE` must all meet their conditions simultaneously to produce a trigger. Channels set to `PS3000A_CONDITION_DONT_CARE` are ignored.
	
	`aux,` not used

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.62 ps3000aSetTriggerChannelDirections

```
PICO_STATUS ps3000aSetTriggerChannelDirections
(
  int16_t                      handle,
  PS3000A_THRESHOLD_DIRECTION channelA,
  PS3000A_THRESHOLD_DIRECTION channelB,
  PS3000A_THRESHOLD_DIRECTION channelC,
  PS3000A_THRESHOLD_DIRECTION channelD,
  PS3000A_THRESHOLD_DIRECTION ext,
  PS3000A_THRESHOLD_DIRECTION aux
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device `channelA`, `channelB`, `channelC`, `channelD`, `ext`, the direction in which the signal must pass through the threshold to activate the trigger. See the table below for allowable values. If using a level trigger in conjunction with a pulse-width trigger, see the description of the `direction` argument to ps3000aSetPulseWidthQualifierV2() for more information. `aux`, not used

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

`PS3000A_THRESHOLD_DIRECTION` **constants**

`PS3000A_ABOVE`	for gated triggers: above the upper threshold
`PS3000A_ABOVE_LOWER`	for gated triggers: above the lower threshold
`PS3000A_BELOW`	for gated triggers: below the upper threshold
`PS3000A_BELOW_LOWER`	for gated triggers: below the lower threshold
`PS3000A_RISING`	for threshold triggers: rising edge, using upper threshold
`PS3000A_RISING_LOWER`	for threshold triggers: rising edge, using lower threshold
`PS3000A_FALLING`	for threshold triggers: falling edge, using upper threshold
`PS3000A_FALLING_LOWER`	for threshold triggers: falling edge, using lower threshold
`PS3000A_RISING_OR_FALLING`	for threshold triggers: either edge
`PS3000A_INSIDE`	for window-qualified triggers: inside window
`PS3000A_OUTSIDE`	for window-qualified triggers: outside window
`PS3000A_ENTER`	for window triggers: entering the window
`PS3000A_EXIT`	for window triggers: leaving the window
`PS3000A_ENTER_OR_EXIT`	for window triggers: either entering or leaving the window
`PS3000A_POSITIVE_RUNT`	for window-qualified triggers
`PS3000A_NEGATIVE_RUNT`	for window-qualified triggers
`PS3000A_NONE`	no trigger

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.63 ps3000aSetTriggerChannelProperties

```
PICO_STATUS ps3000aSetTriggerChannelProperties
(
  int16_t                              handle,
  PS3000A_TRIGGER_CHANNEL_PROPERTIES * channelProperties,
  int16_t                              nChannelProperties,
  int16_t                              auxOutputEnable,
  int32_t                              autoTriggerMilliseconds
)
```

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

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.63.1 PS3000A_TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to ps3000aSetTriggerChannelProperties() in the channelProperties argument to specify the trigger mechanism, and is defined as follows: -

```
typedef struct tPS3000ATriggerChannelProperties
{
  int16_t                thresholdUpper;
  uint16_t               thresholdUpperHysteresis;
  int16_t                thresholdLower;
  uint16_t               thresholdLowerHysteresis;
  PS3000A_CHANNEL        channel;
  PS3000A_THRESHOLD_MODE thresholdMode;
} PS3000A_TRIGGER_CHANNEL_PROPERTIES
```

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

Elements	`thresholdUpper`, the upper threshold at which the trigger must fire. This is scaled in 16-bit ADC counts at the currently selected range for that channel.
	
	`thresholdUpperHysteresis`, the hysteresis by which the trigger must exceed the upper threshold before it will fire. It is scaled in 16-bit counts.
	
	`thresholdLower`, the lower threshold at which the trigger must fire. This is scaled in 16-bit ADC counts at the currently selected range for that channel.
	
	`thresholdLowerHysteresis`, the hysteresis by which the trigger must exceed the lower threshold before it will fire. It is scaled in 16-bit counts.
	
	`channel`, the channel to which the properties apply. This can be one of the four input channels listed under ps3000aSetChannel(), or PS3000A_TRIGGER_AUX for the AUX input.
	
	`thresholdMode`, either a level or window trigger. Use one of these constants: -
	`PS3000A_LEVEL`
	`PS3000A_WINDOW`

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.64 ps3000aSetTriggerDelay

```
PICO_STATUS ps3000aSetTriggerDelay
(
  int16_t  handle,
  uint32_t delay
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device `delay`, the time between the trigger occurring and the first sample. For example, if `delay=100` then the scope would wait 100 sample periods before sampling. At a timebase of 500 MS/s, or 2 ns per sample, the total delay would then be 100 x 2 ns = 200 ns. Range: 0 to `MAX_DELAY_COUNT`

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

4.65 ps3000aSetTriggerDigitalPortProperties

```
PICO_STATUS ps3000aSetTriggerDigitalPortProperties
(
  int16_t                                  handle,
  PS3000A_DIGITAL_CHANNEL_DIRECTIONS *  directions
  int16_t                                  nDirections
)
```

This function will set the individual digital channels' trigger directions. Each trigger direction consists of a channel name and a direction. If the channel is not included in the array of PS3000A_DIGITAL_CHANNEL_DIRECTIONS the driver assumes the digital channel's trigger direction is PS3000A_DIGITAL_DONT_CARE.

Applicability	All modes
Arguments	handle, the handle of the required device. * directions, a pointer to an array of PS3000A_DIGITAL_CHANNEL_DIRECTIONS structures describing the requested properties. The array can contain a single element describing the properties of one channel, or a number of elements describing several digital channels. If directions is NULL, digital triggering is switched off. A digital channel that is not included in the array will be set to PS3000A_DIGITAL_DONT_CARE. nDirections, the number of digital channel directions being passed to the driver.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_INVALID_DIGITAL_CHANNEL PICO_INVALID_DIGITAL_TRIGGER_DIRECTION

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.65.1 PS3000A_DIGITAL_CHANNEL_DIRECTIONS structure

A structure of this type is passed to ps3000aSetTriggerDigitalPortProperties() in the `directions` argument to specify the trigger mechanism, and is defined as follows: -

```
pragma pack(1)
typedef struct tPS3000ADigitalChannelDirections
{
  PS3000A_DIGITAL_CHANNEL   channel;
  PS3000A_DIGITAL_DIRECTION direction;
} PS3000A_DIGITAL_CHANNEL_DIRECTIONS;
#pragma pack()

typedef enum enPS3000ADigitalChannel
{
  PS3000A_DIGITAL_CHANNEL_0,
  PS3000A_DIGITAL_CHANNEL_1,
  PS3000A_DIGITAL_CHANNEL_2,
  PS3000A_DIGITAL_CHANNEL_3,
  PS3000A_DIGITAL_CHANNEL_4,
  PS3000A_DIGITAL_CHANNEL_5,
  PS3000A_DIGITAL_CHANNEL_6,
  PS3000A_DIGITAL_CHANNEL_7,
  PS3000A_DIGITAL_CHANNEL_8,
  PS3000A_DIGITAL_CHANNEL_9,
  PS3000A_DIGITAL_CHANNEL_10,
  PS3000A_DIGITAL_CHANNEL_11,
  PS3000A_DIGITAL_CHANNEL_12,
  PS3000A_DIGITAL_CHANNEL_13,
  PS3000A_DIGITAL_CHANNEL_14,
  PS3000A_DIGITAL_CHANNEL_15,
  PS3000A_DIGITAL_CHANNEL_16,
  PS3000A_DIGITAL_CHANNEL_17,
  PS3000A_DIGITAL_CHANNEL_18,
  PS3000A_DIGITAL_CHANNEL_19,
  PS3000A_DIGITAL_CHANNEL_20,
  PS3000A_DIGITAL_CHANNEL_21,
  PS3000A_DIGITAL_CHANNEL_22,
  PS3000A_DIGITAL_CHANNEL_23,
  PS3000A_DIGITAL_CHANNEL_24,
  PS3000A_DIGITAL_CHANNEL_25,
  PS3000A_DIGITAL_CHANNEL_26,
  PS3000A_DIGITAL_CHANNEL_27,
  PS3000A_DIGITAL_CHANNEL_28,
  PS3000A_DIGITAL_CHANNEL_29,
  PS3000A_DIGITAL_CHANNEL_30,
  PS3000A_DIGITAL_CHANNEL_31,
  PS3000A_MAX_DIGITAL_CHANNELS
} PS3000A_DIGITAL_CHANNEL;

typedef enum enPS3000ADigitalDirection
{
  PS3000A_DIGITAL_DONT_CARE,
  PS3000A_DIGITAL_DIRECTION_LOW,
  PS3000A_DIGITAL_DIRECTION_HIGH,
  PS3000A_DIGITAL_DIRECTION_RISING,
  PS3000A_DIGITAL_DIRECTION_FALLING,
  PS3000A_DIGITAL_DIRECTION_RISING_OR_FALLING,
  PS3000A_DIGITAL_MAX_DIRECTION
} PS3000A_DIGITAL_DIRECTION;
```

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.66 ps3000aSigGenArbitraryMinMaxValues

```
PICO_STATUS ps3000aSigGenArbitraryMinMaxValues
(
  int16_t    handle,
  int16_t  * minArbitraryWaveformValue,
  int16_t  * maxArbitraryWaveformValue,
  uint32_t * minArbitraryWaveformSize,
  uint32_t * maxArbitraryWaveformSize
)
```

This function returns the range of possible sample values and waveform buffer sizes that can be supplied to ps3000aSetSignGenArbitrary() for setting up the arbitrary waveform generator (AWG). These values vary between different models in the PicoScope 3000 Series.

Applicability	All models with AWG
Arguments	`handle`, the handle of the required device. `minArbitraryWaveformValue`, on exit, the lowest sample value allowed in the `arbitraryWaveform` buffer supplied to ps3000aSetSignGenArbitrary(). `maxArbitraryWaveformValue`, on exit, the highest sample value allowed in the `arbitraryWaveform` buffer supplied to ps3000aSetSignGenArbitrary(). `minArbitraryWaveformSize`, on exit, the minimum value allowed for the `arbitraryWaveformSize` argument supplied to ps3000aSetSignGenArbitrary(). `maxArbitraryWaveformSize`, on exit, the maximum value allowed for the `arbitraryWaveformSize` argument supplied to ps3000aSetSignGenArbitrary().

Returns	`PICO_OK`
	`PICO_NOT_SUPPORTED_BY_THIS_DEVICE`, if the device does not have an arbitrary waveform generator.
	`PICO_NULL_PARAMETER`, if all the parameter pointers are NULL.
	`PICO_INVALID_HANDLE`
	`PICO_DRIVER_FUNCTION`

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.67 ps3000aSigGenFrequencyToPhase

```
PICO_STATUS ps3000aSigGenFrequencyToPhase
(
  int16_t              handle,
  double               frequency,
  PS3000A_INDEX_MODE   indexMode,
  uint32_t             bufferLength,
  uint32_t           * phase
)
```

This function converts a frequency to a phase count for use with the arbitrary waveform generator (AWG). The value returned depends on the length of the buffer, the index mode passed and the device model. The phase count can then be sent to the driver through ps3000aSetSigGenArbitrary() or ps3000aSetSigGenPropertiesArbitrary().

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.68 ps3000aSigGenSoftwareControl

```
PICO_STATUS ps3000aSigGenSoftwareControl
(
  int16_t handle,
  int16_t state
)
```

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

Applicability	Use with ps3000aSetSigGenBuiltIn() or ps3000aSetSigGenArbitrary().
Arguments	handle, the handle of the required device state, sets the trigger gate high or low when the trigger type is set to either SIGGEN_GATE_HIGH or SIGGEN_GATE_LOW. Ignored for other trigger types.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_TRIGGER_SOURCE PICO_DRIVER_FUNCTION PICO_NOT_RESPONDING

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.69 ps3000aStop

```
PICO_STATUS ps3000aStop
(
  int16_t handle
)
```

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

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

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

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

4.70 ps3000aStreamingReady (callback)

```
typedef void (CALLBACK *ps3000aStreamingReady)
(
  int16_t     handle,
  int32_t     noOfSamples,
  uint32_t    startIndex,
  int16_t     overflow,
  uint32_t    triggerAt,
  int16_t     triggered,
  int16_t     autoStop,
  void      * pParameter
)
```

This callback function is part of your application. You register it with the driver using ps3000aGetStreamingLatestValues(), and the driver calls it back when streaming-mode data is ready. You can then download the data using the ps3000aGetValuesAsync() function.

Your callback function should do nothing more than copy the data to another buffer within your application. To maintain the best application performance, the function should return as quickly as possible without attempting to process or display the data.

Applicability	Streaming mode only
Arguments	`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 ps3000aSetDataBuffer(). `overflow`, returns a set of flags that indicate whether an overvoltage has occurred on any of the channels. It is a bit pattern with bit 0 denoting Channel A. `triggerAt`, an index to the buffer indicating the location of the trigger point relative to `startIndex`. This parameter is valid only when `triggered` is non-zero. `triggered`, a flag indicating whether a trigger occurred. If non-zero, a trigger occurred at the location indicated by `triggerAt`. `autoStop`, the flag that was set in the call to ps3000aRunStreaming(). `* pParameter`, a void pointer passed from ps3000aGetStreamingLatestValues(). The callback function can write to this location to send any data, such as a status flag, back to the application.
Returns	nothing

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5 Wrapper functions

The wrapper functions are for use with programming languages that do not support features of C such as callback functions. To use the wrapper functions you must include the `ps3000aWrap.dll` library, which is supplied in the SDK, in your project.

For all other functions, see the list of API functions.

5.1 Using the wrapper functions for streaming data capture

1. Open the oscilloscope using ps3000aOpenUnit().
   1a. Register the handle with the wrapper and obtain a device index for use with some wrapper function calls by calling initWrapUnitInfo().
   1b. Inform the wrapper of the number of channels on the device by calling setChannelCount().
   1c. [MSOs only] Inform the wrapper of the number of digital ports on the device by calling setDigitalPortCount().

2. Select channels, ranges and AC/DC coupling using ps3000aSetChannel().
   2a. Inform the wrapper which channels have been enabled by calling setEnabledChannels().
   2b. [MSOs only] Inform the wrapper which digital ports have been enabled by calling setEnabledDigitalPorts().

3. [MSOs only] Set the digital port using ps3000aSetDigitalPort().

4. Use the trigger setup functions ps3000aSetTriggerChannelConditionsV2(), ps3000aSetTriggerChannelDirections() and ps3000aSetTriggerChannelProperties() to set up the trigger if required. For programming languages that do not support structures, use the wrapper's SetTriggerConditionsV2() in place of ps3000aSetTriggerCHannelConditionsV2() and SetTriggerProperties() in place of ps3000aSetTriggerChannelProperties().

5. [MSOs only] Use the trigger setup function ps3000aSetTriggerDigitalPortProperties() to set up the digital trigger if required.

6. Call ps3000aSetDataBuffer() to tell the driver where your data buffer is.
   6a. Register the data buffer(s) with the wrapper and set the application buffer into which the data will be copied.
   - For analog channels: Call setAppAndDriverBuffers() or setMaxMinAppAndDriverBuffers().
   - [MSOs Only] For digital ports: Call setAppAndDriverDigiBuffers() or setMaxMinAppAndDriverDigiBuffers().

7. Set up aggregation and start the oscilloscope running using ps3000aRunStreaming().

8. Loop and call GetStreamingLatestValues() and IsReady() to get data and flag when the wrapper is ready for data to be retrieved.
   8a. Call the wrapper's AvailableData() function to obtain information on the number of samples collected and the start index in the buffer.
   8b. Call the wrapper's IsTriggerReady() function for information on whether a trigger has occurred and the trigger index relative to the start index in the buffer.

9. Process data returned to your application's function.

10. Call ps3000aStop(), even if Auto Stop is enabled.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

11. To disconnect a device, call ps3000aCloseUnit() followed by the wrapper's decrementDeviceCount() function.

12. Call the resetNextDeviceIndex() wrapper function.

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.2 AutoStopped

```
int16_t AutoStopped
(
  uint16_t deviceIndex
)
```

This function indicates if the device has stopped after collecting of the number of samples specified in the call to ps3000aRunStreaming(). This occurs only if the ps3000aRunStreaming() function's autostop flag is set.

Applicability	Streaming mode
Arguments	deviceIndex, identifies the required device
Returns	0 – if streaming has not stopped or deviceIndex is out of range <> 0 – if streaming has stopped automatically

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.3 AvailableData

```
uint32_t AvailableData
(
  uint16_t   deviceIndex,
  uint32_t * startIndex
)
```

This function indicates the number of samples returned from the driver and shows the start index of the data in the buffer when collecting data in streaming mode.

Applicability	Streaming mode
Arguments	deviceIndex, identifies the required device startIndex, on exit, an index to the first valid sample in the buffer (when data is available)
Returns	0 – data is not yet available or the device index is invalid <>0 – the number of samples returned from the driver

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.4 BlockCallback

```
void BlockCallback
(
  int16_t       handle,
  PICO_STATUS   status,
  void        * pParameter
)
```

This is a wrapper for the ps3000aBlockReady() callback. The driver calls it back when block-mode data is ready.

Applicability	Block mode
Arguments	See ps3000aBlockReady()
Returns	Nothing

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.5 ClearTriggerReady

```
PICO_STATUS ClearTriggerReady
(
  uint16_t deviceIndex
)
```

This function clears the `triggered` and `triggeredAt` flags for use with streaming-mode capture.

Applicability	Streaming mode
Arguments	`deviceIndex,` identifies the device to use
Returns	`PICO_OK,` if successful `PICO_INVALID_PARAMETER,` if `deviceIndex` is out of bounds

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.6 decrementDeviceCount

```
PICO_STATUS decrementDeviceCount
(
  uint16_t deviceIndex
)
```

Reduces the count of the number of PicoScope devices being controlled by the application.

Note: This function does not close the connection to the device being controlled. Use the ps3000aCloseUnit() function for this.

Applicability	All modes
Arguments	`deviceIndex`, identifies the device to use
Returns	`PICO_OK`, if successful `PICO_INVALID_PARAMETER`, if `deviceIndex` is out of bounds

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.7 getDeviceCount

```
uint16_t getDeviceCount
(
  void
)
```

This function returns the number of PicoScope 3000 Series devices being controlled by the application.

Applicability	All modes
Arguments	None
Returns	The number of PicoScope 3000 Series devices being controlled

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.8 GetStreamingLatestValues

```
PICO_STATUS GetStreamingLatestValues
(
  uint16_t deviceIndex
)
```

This function returns the next block of values to your application when capturing data in streaming mode. Use with programming languages that do not support callback functions.

Applicability	Streaming mode
Arguments	`deviceIndex`, identifies the required device
Returns	`PICO_INVALID_PARAMETER`, if `deviceIndex` is invalid See also ps3000aGetStreamingLatestValues() return values

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.9 initWrapUnitInfo

```
PICO_STATUS initWrapUnitInfo
(
  int16_t    handle,
  uint16_t * deviceIndex
)
```

This function initializes a WRAP_UNIT_INFO structure for a PicoScope 3000 Series device and places it in the g_deviceInfo array at the next available index.

The wrapper supports a maximum of 4 devices.

Your main application should map the handle to the index starting with the first handle corresponding to index 0.

Applicability	All modes
Arguments	deviceIndex, on exit, the index at which the WRAP_UNIT_INFO structure will be stored in the g_deviceInfo array
Returns	PICO_OK, if successful PICO_INVALID_HANDLE, if the handle is less than or equal to 0 PICO_MAX_UNITS_OPENED, if the wrapper already has records for the maximum number of devices that it will support

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.10 IsReady

```
int16_t IsReady
(
  uint16_t deviceIndex
)
```

This function polls the driver to verify that streaming data is ready to be received. The RunBlock() or GetStreamingLatestValues() function must have been called before calling this function.

Applicability	Streaming mode. (In block mode, we recommend using ps3000aIsReady() instead.)
Arguments	`deviceIndex`, the index assigned by the wrapper corresponding to the required device
Returns	`0` – data is not yet available or `deviceIndex` is out of range `<>0` – data is ready to be collected

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.11 IsTriggerReady

```
int16_t IsTriggerReady
(
  uint16_t   deviceIndex
  uint32_t * triggeredAt
)
```

This function indicates whether a trigger has occurred when collecting data in streaming mode, and provides the location of the trigger point in the buffer.

Applicability	Streaming mode
Arguments	deviceIndex, the index assigned by the wrapper corresponding to the required device triggeredAt, on exit, the index of the sample in the buffer where the trigger occurred, relative to the first valid sample index. This value is set to 0 when the function returns 0.
Returns	0 – the device has not triggered, or deviceIndex is invalid <>0 – the device has been triggered

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.12 resetNextDeviceIndex

```
PICO_STATUS resetNextDeviceIndex
(
  void
)
```

This function is used to reset the index used to determine the next point at which to store a WRAP_UNIT_INFO structure.

Call this function only after the devices have been disconnected.

Applicability	All modes
Arguments	None
Returns	PICO_OK

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.13 RunBlock

```
PICO_STATUS RunBlock
(
  uint16_t   deviceIndex,
  int32_t    preTriggerSamples,
  int32_t    postTriggerSamples,
  uint32_t   timebase,
  uint32_t    segmentIndex
)
```

This function starts collecting data in block mode without the requirement for specifying callback functions. Use the IsReady function to poll the driver once this function has been called.

Applicability	Block mode
Arguments	`deviceIndex`, the index assigned by the wrapper corresponding to the required device `preTriggerSamples`, see `noOfPreTriggerSamples` in ps3000aRunBlock() `postTriggerSamples`, see `noOfPreTriggerSamples` in ps3000aRunBlock() `timebase`, see ps3000aRunBlock() `segmentIndex`, see ps3000aRunBlock()
Returns	See ps3000aRunBlock() return values

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.14 setAppAndDriverBuffers

```
PICO_STATUS setAppAndDriverBuffers
(
  uint16_t   deviceIndex,
  int16_t    channel,
  int16_t  * appBuffer,
  int16_t  * driverBuffer,
  uint32_t   bufferLength
)
```

This function sets the application buffer and corresponding driver buffer in order for the streaming callback to copy the data for the analog channel from the driver buffer to the application buffer.

Applicability	Streaming mode
Arguments	`deviceIndex,` the index assigned by the wrapper corresponding to the required device `channel,` the channel number (should be a numerical value corresponding to a `PS3000A_CHANNEL` enumeration value) `appBuffer,` the application buffer `driverBuffer,` the buffer set by the driver `bufferLength,` the length of the buffers (the lengths of the buffers must be equal)
Returns	`PICO_OK,` if successful `PICO_INVALID_PARAMETER,` if `deviceIndex` is out of bounds `PICO_INVALID_CHANNEL,` if channel is not valid

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.15 setMaxMinAppAndDriverBuffers

```
PICO_STATUS setMaxMinAppAndDriverBuffers
(
  uint16_t   deviceIndex,
  int16_t    channel,
  int16_t  * appMaxBuffer,
  int16_t  * appMinBuffer,
  int16_t  * driverMaxBuffer,
  int16_t  * driverMinBuffer,
  uint32_t   bufferLength
)
```

Set the application buffer and corresponding driver buffer in order for the streaming callback to copy the data for the analog channel from the driver maximum and minimum buffers to the respective application buffers for aggregated data collection.

Applicability	Streaming mode
Arguments	`deviceIndex`, the index assigned by the wrapper corresponding to the required device `channel`, the channel number (should be a numerical value corresponding to a `PS3000A_CHANNEL` enumeration value) `appMaxBuffer`, the application buffer for maximum values (the 'max buffer') `appMinBuffer`, the application buffer for minimum values (the 'min buffer') `driverMaxBuffer`, the max buffer set by the driver `driverMinBuffer`, the min buffer set by the driver `bufferLength`, the length of the buffers (the lengths of the buffers must be equal)
Returns	`PICO_OK`, if successful `PICO_INVALID_PARAMETER`, if `deviceIndex` is out of bounds `PICO_INVALID_CHANNEL`, if channel is not valid

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.16 setAppAndDriverDigiBuffers

```
PICO_STATUS setAppAndDriverDigiBuffers
(
  uint16_t   deviceIndex,
  int16_t    digiPort,
  int16_t  * appDigiBuffer,
  int16_t  * driverDigiBuffer,
  uint32_t   bufferLength
)
```

This function sets the application buffer and corresponding driver buffer in order for the streaming callback to copy the data for the digital port from the driver buffer to the application buffer.

Applicability	Streaming mode. PicoScope 3000 MSO and 3000D MSO models only.
Arguments	`deviceIndex`, the index assigned by the wrapper corresponding to the required device `digiPort`, the digital port number (0 or 1) `appDigiBuffer`, the application buffer for the digital port `driverDigitalBuffer`, the buffer for the digital port set by the driver `bufferLength`, the length of the buffers (the lengths of the buffers must be equal)
Returns	`PICO_OK`, if successful `PICO_INVALID_PARAMETER`, if `deviceIndex` is out of bounds `PICO_INVALID_DIGITAL_PORT`, if `digiPort` is not 0 (Port 0) or 1 (Port 1)

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.17 setMaxMinAppAndDriverDigiBuffers

```
PICO_STATUS setMaxMinAppAndDriverDigiBuffers
(
  uint16_t   deviceIndex,
  int16_t    digiPort,
  int16_t  * appMaxDigiBuffer,
  int16_t  * appMinDigiBuffer,
  int16_t  * driverMaxDigiBuffer,
  int16_t  * driverMinDigiBuffer,
  uint32_t   bufferLength
)
```

This functions sets the application buffers and corresponding driver buffers in order for the streaming callback to copy the data for the digital port from the driver 'max' and 'min' buffers to the respective application buffers for aggregated data collection.

Applicability	Streaming mode. PicoScope 3000 MSO and 3000D models only.
Arguments	`deviceIndex`, the index assigned by the wrapper corresponding to the required device `digiPort`, the digital port number (0 or 1) `appMaxDigiBuffer`, the application max. buffer for the digital port `appMinDigiBuffer`, the application min. buffer for the digital port `driverMaxDigiBuffer`, the max. buffer set by the driver for the digital port `driverMinDigiBuffer`, the min. buffer set by the driver for the digital port `bufferLength`, the length of the buffers (the lengths of the buffers must be equal)
Returns	`PICO_OK`, if successful `PICO_INVALID_PARAMETER`, if `deviceIndex` is out of bounds `PICO_INVALID_DIGITAL_PORT`, if `digiPort` is not 0 (Port 0) or 1 (Port 1)

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.18 setChannelCount

```
PICO_STATUS setChannelCount
(
  uint16_t deviceIndex,
  int16_t  channelCount
)
```

This function sets the number of analog channels on the device. This is used to assist with copying data in the streaming callback.

The initWrapUnitInfo() must have been called before this function is called.

Applicability	Streaming mode
Arguments	deviceIndex, the index assigned by the wrapper corresponding to the required device channelCount, the number of channels on the device
Returns	PICO_OK, if successful PICO_INVALID_PARAMETER, if deviceIndex is out of bounds or channelCount is not 2 or 4

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.19 setDigitalPortCount

```
PICO_STATUS setDigitalPortCount
(
  uint16_t deviceIndex,
  int16_t  digitalPortCount
)
```

Set the number of digital ports on the device. This is used to assist with copying data in the streaming callback.

You must call initWrapUnitInfo() before calling this function.

Applicability	Streaming mode
Arguments	`deviceIndex`, the index assigned by the wrapper corresponding to the required device `digitalPortCount`, the number of digital ports on the device. Set to 2 for the PicoScope 3000 MSO and 3000D MSO devices and 0 for other models.
Returns	`PICO_OK`, if successful `PICO_INVALID_PARAMETER`, `deviceIndex` is out of bounds or `digitalPortCount` is invalid

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.20 setEnabledChannels

```
PICO_STATUS setEnabledChannels
(
  uint16_t   deviceIndex,
  int16_t  * enabledChannels
)
```

Set the number of enabled analog channels on the device. This is used to assist with copying data in the streaming callback.

You must call setChannelCount() before calling this function.

Applicability	Streaming mode
Arguments	`deviceIndex`, the index assigned by the wrapper corresponding to the required device `enabledChannels`, an array of 4 elements representing the channel states
Returns	`PICO_OK`, if successful `PICO_INVALID_PARAMETER`, if `deviceIndex` is out of bounds or `channelCount` is not 2 or 4

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.21 setEnabledDigitalPorts

```
PICO_STATUS setEnabledDigitalPorts
(
  uint16_t   deviceIndex,
  int16_t  * enabledDigitalPorts
)
```

This function sets the number of enabled digital ports on the device. This is used to assist with copying data in the streaming callback.

For PicoScope 3000 MSO and 3000D MSO models, you must call setDigitalPortCount() first.

Applicability	Streaming mode
Arguments	`deviceIndex`, the index assigned by the wrapper corresponding to the required device `enabledDigitalPorts`, an array of 4 elements representing the digital port states
Returns	`PICO_OK`, if successful `PICO_INVALID_PARAMETER`, if `deviceIndex` is out of bounds, or `digitalPortCount` is invalid

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.22 SetPulseWidthQualifier

```
PICO_STATUS SetPulseWidthQualifier
(
  int16_t    handle,
  uint32_t * pwqConditionsArray,
  int16_t    nConditions,
  uint32_t   direction,
  uint32_t   lower,
  uint32_t   upper,
  uint32_t   type
)
```

This function sets up pulse-width qualification, which can be used on its own for pulse-width triggering or combined with level triggering or window triggering to produce more complex triggers.

The pulse-width qualifier is defined by one or more sets of integers corresponding to `PS3000A_PWQ_CONDITIONS` structures which are then converted and passed to ps3000aSetPulseWidthQualifier().

Use this function with programming languages that do not support structs.

Applicability	Analog-input models only (for MSOs, use SetPulseWidthQualifierV2())
Arguments	`handle`, the handle of the required device `pwqConditionsArray`, an array of integer values specifying the conditions for each channel `nConditions`, the number that will be passed after the wrapper code has created its structures (i.e. the number of `pwqConditionsArray` elements / 6) `direction`, the direction of the signal required for the pulse width trigger to fire (see `PS3000A_THRESHOLD_DIRECTION` enumerations) `lower`, the lower limit of the pulse-width counter, measured in samples `upper`, the upper limit of the pulse-width counter, measured in samples `type`, the pulse-width type (see `PS3000A_PULSE_WIDTH_TYPE` enumerations)
Returns	See ps3000aSetPulseWidthQualifier() return values

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.23 SetPulseWidthQualifierV2

```
PICO_STATUS SetPulseWidthQualifierV2
(
  int16_t    handle,
  uint32_t * pwqConditionsArrayV2,
  int16_t    nConditions,
  uint32_t   direction,
  uint32_t   lower,
  uint32_t   upper,
  uint32_t   type
)
```

This function sets up pulse-width qualification, which can be used on its own for pulse-width triggering or combined with level triggering or window triggering to produce more complex triggers.

The pulse-width qualifier is defined by one or more sets of integers corresponding to `PS3000A_PWQ_CONDITIONS_V2` structures which are then converted and passed to ps3000aSetPulseWidthQualifierV2().

Use this function with programming languages that do not support structs.

Applicability	All models
Arguments	`handle`, the handle of the required device `pwqConditionsArray`, an array of integer values specifying the conditions for each channel `nConditions`, the number that will be passed after the wrapper code has created its structures (i.e. the number of `pwqConditionsArrayV2` elements / 6) `direction`, the direction of the signal required for the pulse width trigger to fire (see `PS3000A_THRESHOLD_DIRECTION` enumerations) `lower`, the lower limit of the pulse-width counter, measured in samples `upper`, the upper limit of the pulse-width counter, measured in samples `type`, the pulse-width type (see `PS3000A_PULSE_WIDTH_TYPE` enumerations)
Returns	See ps3000aSetPulseWidthQualifier() return values

 Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.24 SetTriggerConditions

```
PICO_STATUS SetTriggerConditions
(
  int16_t   handle,
  int32_t * conditionsArray,
  int16_t   nConditions
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is defined by one or more sets of integers corresponding to PS3000A_TRIGGER_CONDITIONS structures which are then converted and passed to ps3000aSetTriggerChannelConditions().

Use this function with programming languages that do not support structs.

Applicability	Analog-input models only (for MSOs use SetTriggerConditionsV2())
Arguments	handle, the handle of the required device conditionsArray, an array of integer values specifying the conditions for each channel nConditions, the number that will be passed after the wrapper code has created its structures (i.e. the number of conditionsArray elements divided by 7)
Returns	See ps3000aSetTriggerChannelConditions() return values

Examples

Below are examples for using the function in Visual Basic.

To trigger off channels A OR B

```
Dim conditionsArray(13) As Integer
conditionsArray(0) = 1          ' channel A
conditionsArray(1) = 0          ' channel B
conditionsArray(2) = 0          ' channel C
conditionsArray(3) = 0          ' channel D
conditionsArray(4) = 0          ' external
conditionsArray(5) = 0          ' aux
conditionsArray(6) = 0          ' pulse width qualifier

' *** OR'ed with

conditionsArray(7) = 0          ' channel A
conditionsArray(8) = 1          ' channel B
conditionsArray(9) = 0          ' channel C
conditionsArray(10) = 0         ' channel D
conditionsArray(11) = 0         ' external
conditionsArray(12) = 0         ' aux
conditionsArray(13) = 0         ' pulse width qualifier
status = SetTriggerConditions(handle, conditionsArray(0), 2)
```

To trigger off channels A AND B

```
Dim conditionsArray(6) As Integer
conditionsArray(0) = 1          ' channel A
conditionsArray(1) = 1          ' channel B
conditionsArray(2) = 0          ' channel C
conditionsArray(3) = 0          ' channel D
```

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

```
conditionsArray(4) = 0          ' external
conditionsArray(5) = 0          ' aux
conditionsArray(6) = 0          ' pulse width qualifier

status = SetTriggerConditions(handle, conditionsArray(0), 1)
```


Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.25 SetTriggerConditionsV2

```
PICO_STATUS SetTriggerConditionsV2
(
  int16_t   handle,
  int32_t * conditionsArrayV2,
  int16_t   nConditions
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is defined by one or more sets of integers corresponding to PS3000A_TRIGGER_CONDITIONS_V2 structures which are then converted and passed to ps3000aSetTriggerChannelConditionsV2().

Use this function with programming languages that do not support structs.

Applicability	All models
Arguments	handle, the handle of the required device conditionsArrayV2, an array of integer values specifying the conditions for each channel nConditions, the number that will be passed after the wrapper code has created its structures (i.e. the number of conditionsArray elements divided by 8)
Returns	See ps3000aSetTriggerChannelConditionsV2() return values

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.26 SetTriggerProperties

```
PICO_STATUS SetTriggerProperties
(
  int16_t   handle,
  int32_t * propertiesArray,
  int16_t   nProperties,
  int32_t   autoTrig
)
```

This function is used to enable or disable triggering and set its parameters. This is done by assigning the values from the propertiesArray to an array of PS3000A_TRIGGER_CHANNEL_PROPERTIES structures which are then passed to the ps3000aSetTriggerChannelProperties() function with the other parameters.

Use this function with programming languages that do not support structs.

Applicability	All modes
Arguments	handle, the handle of the required device propertiesArray, an array of sets of integers corresponding to PS3000A_TRIGGER_CHANNEL_PROPERTIES structures describing the required properties to be set. See also channelProperties in ps3000aSetTriggerChannelProperties(). nProperties, the number that will be passed after the wrapper code has created its structures (i.e. the number of propertiesArray elements divided by 6) autoTrig, see autoTriggerMilliseconds in ps3000aSetTriggerChannelProperties()
Returns	See ps3000aSetTriggerChannelProperties() return values

Example

Here is an example for using the function in Visual Basic:

```
Dim propertiesArray(11) As Integer

'channel A
propertiesArray(0) = 1500 ' Upper
propertiesArray(1) = 300  ' UpperHysteresis
propertiesArray(2) = 0    ' Lower
propertiesArray(3) = 0    ' LowerHysteresis
propertiesArray(4) = 0    ' channel (0=ChA, 1=ChB, 2=ChC, 3=ChD)
propertiesArray(5) = 0    ' thresholdMode (Level=0, Window=1)

'channel B
propertiesArray(6) = 1500 ' Upper
propertiesArray(7) = 300  ' UpperHysteresis
propertiesArray(8) = 0    ' Lower
propertiesArray(9) = 0    ' LowerHysteresis
propertiesArray(10) = 1   ' channel (0=ChA, 1=ChB, 2=ChC, 3=ChD)
propertiesArray(11) = 0   ' thresholdMode (Level=0, Window=1)

status = SetTriggerProperties(handle, propertiesArray(0), 2, 0,
  1000)
```


Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

5.27 StreamingCallback

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

This is a wrapper for the ps3000aStreamingReady() callback. The driver calls it back when streaming-mode data is ready.

Applicability	Streaming mode
Arguments	See ps3000aStreamingReady()
Returns	Nothing

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

6 Programming examples

Example code is provided in a number of programming languages. You may freely modify this code for your own applications.

6.1 C

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

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

- `ps3000acon.c`
- `ps3000a.lib` (Microsoft Visual C 32-bit applications)

The following files must be in the compilation directory:

- `ps3000aApi.h`
- `picoStatus.h`

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

- `ps3000a.dll`
- `PicoIpp.dll`

An example Microsoft Visual C++ 2010 Express project is included in the SDK in the `C_Console` folder. 64-bit versions of `ps3000a.dll`, `PicoIpp.dll` and `ps3000a.lib` are provided in the SDK's `x64` directory.

6.2 C#

The following files, located in the SDK's `PS3000ACSConsole` folder, are required:

- `AssemblyInfo.cs`
- `PS3000ACSConsole.cs`
- `PS3000AImports.cs`
- `PS3000APinnedArray.cs`
- `ps3000a.dll`
- `PicoIpp.dll`

To build the Windows Console application from the Microsoft Visual Studio IDE (2010 Express or later):

- Load the `PS3000ACSConsole.sln` solution file into the IDE.
- Press F6 to build the solution or click *Debug > Build Solution*.

Ensure that the `ps3000a.dll` and `PicoIpp.dll` files are in the search path.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

6.3 Excel

The examples are located in the `Excel` folder of the SDK.

1. Load the spreadsheet `ps3000a.xlsm`
2. Select **Tools | Macro**
3. Select **GetData**
4. Select **Run**

A 64-bit version (`ps3000aV2_x64.xlsm`) is also included. The examples are compatible with Microsoft Office 2007 and later. To run the examples, click the **Get Block** or **Run Streaming** buttons. To edit the examples:

1. Click **View > Macros > View Macros**
2. Select **GetData** or **StreamingData** and click **Edit**

A `Legacy` folder contains the old version of the example.

Note: The Excel macro language is similar to Visual Basic. The functions which return a `TRUE`/`FALSE` value, return 0 for `FALSE` and `1` for `TRUE`, whereas Visual Basic expects 65535 for `TRUE`. Check for `>0` rather than `=TRUE`.

As Excel VBA does not support the callback features of the PicoScope API, additional wrapper functions are provided.

6.4 LabVIEW

The SDK contains a library of VIs that can be used to control the oscilloscope. It also includes some simple examples of using these VIs in streaming mode, block mode and rapid block mode, and for controlling the function generator and arbitrary waveform generator.

As LabVIEW does not support the callback features of the PicoScope API, additional wrapper functions are provided.

Versions of the data acquisition examples for mixed-signal oscilloscopes are also provided, as is a 64-bit block mode capture example.

The LabVIEW library (`PicoScope3000a.llb`) can be placed in the `user.lib` subdirectory to make the VIs available on the 'User Libraries' palette. You must also copy `ps3000a.dll, PicoIpp.dll` and `ps3000awrap.dll` to the LabVIEW installation's resource folder.

The library contains the following VIs:

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

- `PicoScope3000aAdvancedTriggerSettings.vi` – an interface for the advanced trigger features of the oscilloscope.

  This VI is not required for setting up simple triggers, which are configured using `PicoScope3000aSettings.vi`.

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

  ps3000aSetTriggerChannelConditionsV2()

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

[ps3000aSetTriggerChannelDirectionsV2()](#)
[ps3000aSetTriggerChannelProperties()](#)
[ps3000aSetTriggerDigitalPortProperties()](#)
[ps3000aSetPulseWidthQualifier()](#)
[ps3000aSetTriggerDelay()](#)

- `PicoScope3000aAWG.vi` – controls the arbitrary waveform generator.

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

  When selecting an arbitrary waveform, it is necessary to specify a text file containing the waveform. This text file should have a single value on each line in the range −1 to +1. For further information on the settings, see descriptions of ps3000aSetSigGenBuiltIn() and ps3000aSetSigGenArbitrary().

- `PicoScope3000aClose.vi` – closes the oscilloscope.

  Should be called before exiting an application.

- `PicoScope3000aGetBlock.vi` – collects a block of data from the oscilloscope.

  This can be called in a loop in order to continually collect blocks of data. The oscilloscope should first be set up by using `PicoScope3000aSettings.vi`. The VI outputs data arrays in two clusters (max and min). If not using aggregation, 'Min Buffers' is not used.

- `PicoScope3000AGetRapidBlock.vi` – collects a set of data blocks or captures from the oscilloscope in rapid block mode.

  This VI is similar to `PicoScope3000AGetBlock.vi`. It outputs two-dimensional arrays for each channel that contain data from all the requested number of captures.

- `PicoScope3000aGetRapidBlockBulk.vi` - similar to `PicoScope3000aGetRapidBlock.vi` but retrieves all the data using ps3000aGetValuesBulk().

- `PicoScope3000aGetStreamingValues.vi` – used in streaming mode to get the latest values from the driver.

  This VI should be called in a loop after the oscilloscope has been set up using `PicoScope3000aSettings.vi` and streaming has been started by calling `PicoScope3000aStartStreaming.vi`. The VI outputs the number of samples available and the start index of these samples in the array output by `PicoScope3000aStartStreaming.vi`.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

- `PicoScope3000aOpen.vi` – opens a PicoScope 3000 Series (A API) oscilloscope and returns a handle to the device.
- `PicoScope3000aPowerSource.vi` - changes the power settings of a PicoScope 3000 Series device, where applicable.
- `PicoScope3000aSettings.vi` – sets up the oscilloscope.

  The inputs are clusters for setting up channels and simple triggers. Advanced triggers can be set up using `PicoScope3000aAdvancedTriggerSettings.vi`.
- `PicoScope3000aStartStreaming.vi` – starts the oscilloscope streaming.

  It outputs arrays that will contain samples once `PicoScope3000aGetStreamingValues.vi` has returned.
- `PicoScope3000aWrap.vi` - retrieves a unique identifier from `ps3000aWrap.dll` in order to support multiple devices and is also used to inform the wrapper DLL of the number of analog channels and digital ports on the device as well as which channels and ports are enabled.
- `PicoStatus.vi` – checks the status value returned by calls to the driver.

  If the driver returns an error, the status member of the error cluster is set to 'true' and the error code and source are set.
- `PicoScope3000aUnitInfo.vi` – displays the device information for the oscilloscope.

  This VI can be called after opening the device and outputs a cluster containing the information returned by calls to ps3000aGetUnitInfo() for each information type.

6.5 MATLAB

The MATLAB® examples consist of a generic Instrument Driver and accompanying scripts demonstrating how to call the functions in order to operate the scope in different modes. For further information, refer to the *MATLAB Instrument Driver for PicoScope 3000A/B Series – Guide to Functions* document included in the SDK.

The following files will also be required:

- `ps3000a.dll`
- `ps3000aWrap.dll`
- `PicoIpp.dll`

The examples supplied can be used with MATLAB 2012a or later. Version 3.1 or later of the Instrument Control Toolbox will also be required.

As MATLAB does not support the callback features of the PicoScope API, additional wrapper functions are provided.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

6.6 VB.NET

A basic VB.NET Console application is provided in the VB.NET folder in the SDK. The following files will also be required:

- `ps3000a.dll`
- `PicoIpp.dll`

64-bit versions may be found in the SDK's `x64` folder.

Build the project and place the DLL files in the same directory as the executable or ensure that the location is listed in the Windows `PATH` environment variable.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

7 Reference

7.1 Numeric data types

Here is a list of the sizes and ranges of the numeric data types used in the *ps3000a* API.

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

7.2 Enumerated types, constants and structures

The enumerated types, constants and structures used in the *ps3000a* API are defined in the file ps3000aApi.h. We recommend that you refer to these constants by name unless your programming language allows only numerical values.

7.3 Driver status codes

Every function in the *ps3000a* driver returns a **driver status code** from the following list of PICO_STATUS values. These definitions can also be found in the file picoStatus.h, which is included in the *ps3000a* SDK. Not all codes apply to the *ps3000a* SDK.

Code (hex)	Symbol and meaning
00	PICO_OK The PicoScope is functioning correctly
01	PICO_MAX_UNITS_OPENED An attempt has been made to open more than PS3000A_MAX_UNITS.
02	PICO_MEMORY_FAIL Not enough memory could be allocated on the host machine
03	PICO_NOT_FOUND No PicoScope could be found
04	PICO_FW_FAIL Unable to download firmware
05	PICO_OPEN_OPERATION_IN_PROGRESS
06	PICO_OPERATION_FAILED
07	PICO_NOT_RESPONDING The PicoScope is not responding to commands from the PC
08	PICO_CONFIG_FAIL The configuration information in the PicoScope 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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

Code	Description
0D	PICO_INVALID_PARAMETER A parameter value is not valid
0E	PICO_INVALID_TIMEBASE The timebase is not supported or is invalid
0F	PICO_INVALID_VOLTAGE_RANGE The voltage range is not supported or is invalid
10	PICO_INVALID_CHANNEL The channel number is not valid on this device or no channels have been set
11	PICO_INVALID_TRIGGER_CHANNEL The channel set for a trigger is not available on this device
12	PICO_INVALID_CONDITION_CHANNEL The channel set for a condition is not available on this device
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
18	PICO_DATA_NOT_AVAILABLE No data is available from a run block call
19	PICO_STRING_BUFFER_TOO_SMALL The buffer passed for the information was too small
1A	PICO_ETS_NOT_SUPPORTED ETS is not supported on this device
1B	PICO_AUTO_TRIGGER_TIME_TOO_SHORT The auto trigger time is less than the time it will take to collect the pre-trigger data
1C	PICO_BUFFER_STALL The collection of data has stalled as unread data would be overwritten
1D	PICO_TOO_MANY_SAMPLES Number of samples requested is more than available in the current memory segment
1E	PICO_TOO_MANY_SEGMENTS Not possible to create number of segments requested
1F	PICO_PULSE_WIDTH_QUALIFIER A null pointer has been passed in the trigger function or one of the parameters is out of range
20	PICO_DELAY One or more of the hold-off parameters are out of range
21	PICO_SOURCE_DETAILS One or more of the source details are incorrect
22	PICO_CONDITIONS One or more of the conditions are incorrect
23	PICO_USER_CALLBACK The driver's thread is currently in the ps3000a...Ready callback function and therefore the action cannot be carried out
24	PICO_DEVICE_SAMPLING An attempt is being made to get stored data while streaming. Either stop streaming by calling ps3000aStop, or use ps3000aGetStreamingLatestValues
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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

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
2D	PICO_MEMORY Driver cannot allocate memory
2E	PICO_SIG_GEN_PARAM Incorrect parameter passed to the signal generator
2F	PICO_SHOTS_SWEEPS_WARNING Conflict between the shots and sweeps parameters sent to the signal generator
33	PICO_WARNING_EXT_THRESHOLD_CONFLICT Attempt to set different EXT input thresholds set for signal generator and oscilloscope trigger
35	PICO_SIGGEN_OUTPUT_OVER_VOLTAGE The combined peak to peak voltage and the analog offset voltage exceed the allowable voltage the signal generator can produce
36	PICO_DELAY_NULL NULL pointer passed as delay parameter
37	PICO_INVALID_BUFFER The buffers for overview data have not been set while streaming
38	PICO_SIGGEN_OFFSET_VOLTAGE The analog offset voltage is out of range
39	PICO_SIGGEN_PK_TO_PK The analog peak to peak voltage is out of range
3A	PICO_CANCELLED A block collection has been cancelled
3B	PICO_SEGMENT_NOT_USED The segment index is not currently being used
3C	PICO_INVALID_CALL The wrong GetValues function has been called for the collection mode in use
3F	PICO_NOT_USED The function is not available
40	PICO_INVALID_SAMPLERATIO The aggregation ratio requested is out of range
41	PICO_INVALID_STATE Device is in an invalid state
42	PICO_NOT_ENOUGH_SEGMENTS The number of segments allocated is fewer than the number of captures requested
43	PICO_DRIVER_FUNCTION You called a driver function while another driver function was still being processed
44	PICO_RESERVED
45	PICO_INVALID_COUPLING An invalid coupling type was specified in ps3000aSetChannel
46	PICO_BUFFERS_NOT_SET An attempt was made to get data before a data buffer was defined

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

47	PICO_RATIO_MODE_NOT_SUPPORTED The selected downsampling mode (used for data reduction) is not allowed
49	PICO_INVALID_TRIGGER_PROPERTY An invalid parameter was passed to ps3000aSetTriggerChannelProperties
4A	PICO_INTERFACE_NOT_CONNECTED The driver was unable to contact the oscilloscope
4D	PICO_SIGGEN_WAVEFORM_SETUP_FAILED A problem occurred in ps3000aSetSigGenBuiltIn or ps3000aSetSigGenArbitrary
4E	PICO_FPGA_FAIL
4F	PICO_POWER_MANAGER
50	PICO_INVALID_ANALOGUE_OFFSET An impossible analogue offset value was specified in ps3000aSetChannel
51	PICO_PLL_LOCK_FAILED Unable to configure the PicoScope
52	PICO_ANALOG_BOARD The oscilloscope's analog board is not detected, or is not connected to the digital board
53	PICO_CONFIG_FAIL_AWG Unable to configure the signal generator
54	PICO_INITIALISE_FPGA The FPGA cannot be initialized, so unit cannot be opened
56	PICO_EXTERNAL_FREQUENCY_INVALID The frequency for the external clock is not within ±5% of the stated value
57	PICO_CLOCK_CHANGE_ERROR The FPGA could not lock the clock signal
58	PICO_TRIGGER_AND_EXTERNAL_CLOCK_CLASH You are trying to configure the AUX input as both a trigger and a reference clock
59	PICO_PWQ_AND_EXTERNAL_CLOCK_CLASH You are trying to congfigure the AUX input as both a pulse width qualifier and a reference clock
5A	PICO_UNABLE_TO_OPEN_SCALING_FILE The scaling file set can not be opened.
5B	PICO_MEMORY_CLOCK_FREQUENCY The frequency of the memory is reporting incorrectly.
5C	PICO_I2C_NOT_RESPONDING The I2C that is being actioned is not responding to requests.
5D	PICO_NO_CAPTURES_AVAILABLE There are no captures available and therefore no data can be returned.
5E	PICO_NOT_USED_IN_THIS_CAPTURE_MODE The capture mode the device is currently running in does not support the current request.
103	PICO_GET_DATA_ACTIVE Reserved
104	PICO_IP_NETWORKED The device is currently connected via the IP Network socket and thus the call made is not supported.
105	PICO_INVALID_IP_ADDRESS An IP address that is not correct has been passed to the driver.
106	PICO_IPSOCKET_FAILED The IP socket has failed.
107	PICO_IPSOCKET_TIMEDOUT The IP socket has timed out.
108	PICO_SETTINGS_FAILED The settings requested have failed to be set.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

109	PICO_NETWORK_FAILED The network connection has failed.
10A	PICO_WS2_32_DLL_NOT_LOADED Unable to load the WS2 dll.
10B	PICO_INVALID_IP_PORT The IP port is invalid
10C	PICO_COUPLING_NOT_SUPPORTED The type of coupling requested is not supported on the opened device.
10D	PICO_BANDWIDTH_NOT_SUPPORTED Bandwidth limit is not supported on the opened device.
10E	PICO_INVALID_BANDWIDTH The value requested for the bandwidth limit is out of range.
10F	PICO_AWG_NOT_SUPPORTED The arbitrary waveform generator is not supported by the opened device.
110	PICO_ETS_NOT_RUNNING Data has been requested with ETS mode set but run block has not been called, or stop has been called.
111	PICO_SIG_GEN_WHITENOISE_NOT_SUPPORTED White noise is not supported on the opened device.
112	PICO_SIG_GEN_WAVETYPE_NOT_SUPPORTED The wave type requested is not supported by the opened device.
113	PICO_INVALID_DIGITAL_PORT A port number that does not evaluate to either PS3000A_DIGITAL_PORT0 or PS3000A_DIGITAL_PORT1, the ports that are supported.
114	PICO_INVALID_DIGITAL_CHANNEL The digital channel is not in the range PS3000A_DIGITAL_CHANNEL0 to PS3000_DIGITAL_CHANNEL15, the digital channels that are supported.
115	PICO_INVALID_DIGITAL_TRIGGER_DIRECTION The digital trigger direction is not a valid trigger direction and should be equal in value to one of the PS3000A_DIGITAL_DIRECTION enumerations.
116	PICO_SIG_GEN_PRBS_NOT_SUPPORTED Siggen does not generate pseudo-random bit stream.
117	PICO_ETS_NOT_AVAILABLE_WITH_LOGIC_CHANNELS When a digital port is enabled, ETS sample mode is not available for use.
118	PICO_WARNING_REPEAT_VALUE Not applicable to this device.
119	PICO_POWER_SUPPLY_CONNECTED 4-Channel only - The DC power supply is connected.
11A	PICO_POWER_SUPPLY_NOT_CONNECTED 4-Channel only - The DC power supply isn’t connected.
11B	PICO_POWER_SUPPLY_REQUEST_INVALID Incorrect power mode passed for current power source.
11C	PICO_POWER_SUPPLY_UNDERVOLTAGE The supply voltage from the USB source is too low.
11D	PICO_CAPTURING_DATA The oscilloscope is in the process of capturing data.
11E	PICO_USB3_0_DEVICE_NON_USB3_0_PORT A USB 3.0 device is connected to a non-USB 3.0 port.

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

7.4 Glossary

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

Aggregation. The PicoScope 3000 driver can use a method called aggregation to reduce the amount of data your application needs to process. This means that for every block of consecutive samples, it stores only the minimum and maximum values. You can set the number of samples in each block, called the aggregation parameter, when you call ps3000aRunStreaming() for real-time capture, and when you call ps3000aGetStreamingLatestValues() to obtain post-processed data.

Aliasing. An effect that can cause digital oscilloscopes to display fast-moving waveforms incorrectly, by showing spurious low-frequency signals ("aliases") that do not exist in the input. To avoid this problem, choose a sampling rate that is at least twice the frequency of the fastest-changing input signal.

Analog bandwidth. All oscilloscopes have an upper limit to the range of frequencies at which they can measure accurately. The analog bandwidth of an oscilloscope is defined as the frequency at which a displayed sine wave has half the power of the input sine wave (or, equivalently, about 71% of the amplitude).

AWG. Arbitrary waveform generator. On selected models, the signal generator output marked **GEN** or **AWG** can produce an arbitrary waveform defined by the user. Define this waveform by calling ps3000SetSigGenArbitrary() and related functions.

Block mode. A sampling mode in which the computer prompts the oscilloscope to collect a block of data into its internal memory before stopping the oscilloscope and transferring the whole block into computer memory. This mode of operation is effective when the input signal being sampled is high frequency. Note: To avoid aliasing effects, the maximum input frequency must be less than half the sampling rate.

Buffer size. The size, in samples, of the oscilloscope buffer memory. The buffer memory is used by the oscilloscope to temporarily store data before transferring it to the PC.

ETS. Equivalent Time Sampling. ETS constructs a picture of a repetitive signal by accumulating information over many similar wave cycles. This means the oscilloscope can capture fast-repeating signals that have a higher frequency than the maximum sampling rate. Note: ETS should not be used for one-shot or non-repetitive signals.

External trigger. This is the BNC socket marked **EXT** or **Ext**. It can be used as a signal to start data capture, but not as an analog input.

Flexible power. The 4-channel 3000 Series oscilloscopes can be powered by either the USB port or the AC adapter supplied. A two-headed USB cable is supplied for obtaining power from two USB ports.

Maximum sampling rate. A figure indicating the maximum number of samples the oscilloscope is capable of acquiring per second. Maximum sample rates are given in MS/s (megasamples per second). The higher the sampling capability of the oscilloscope, the more accurate the representation of the high frequencies in a fast signal.

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

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

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

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

Signal generator. This is a feature of some oscilloscopes which allows a signal to be generated without an external input device being present. The signal generator output is the BNC socket marked **GEN** or **Gen** on the oscilloscope. If you connect a BNC cable between this and one of the channel inputs, you can send a signal into one of the channels. It can generate a sine, square or triangle wave that can be swept back and forth.

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

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

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

USB 1.1. USB (Universal Serial Bus) is a standard port that enables you to connect external devices to PCs. A USB 1.1 port supports a data transfer rate of 12 Mbps (12 megabits per second), much faster than an RS-232 port.

USB 2.0. A USB 2.0 port supports a data transfer rate of 480 Mbps and is backward-compatible with USB 1.1.

USB 3.0. A USB 3.0 port supports a data transfer rate of 5 Gbps and is backwards-compatible with USB 2.0 and USB 1.1.

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

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

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

pico
Technology

Index

A

AC adapter 5
AC/DC coupling 68
Access 2
ADC count 55, 57
Aggregation 19
Analog offset 30, 68
Arbitrary waveform generator 83, 86
AWG
  buffer lengths 104
  sample values 104

B

Bandwidth limiter 68
Block mode 7, 9, 10, 11
  asynchronous call 11
  callback 23
  polling status 53
  running 63

C

C programming 138
C# programming 138
Callback function 9, 17
  block mode 23
  for data 27
  streaming mode 108
Channels
  enabling 68
  settings 68
Closing units 25
Communication 62
Connection 62
Constants 143
Copyright 2
Coupling type, setting 68

D

Data acquisition 19
Data buffers
  declaring 69
  declaring, aggregation mode 70
Data retention 5, 10
Digital connector 7
Digital data 6
Digital port 6
Downsampling 10, 44
  maximum ratio 32, 33
  modes 45
Driver 3

E

Enabling channels 68
Enumerated types 143
Enumerating oscilloscopes 28
ETS 9
  overview 17
  setting time buffers 73, 74
  setting up 72
  using 18
Excel macros 139

F

Fitness for purpose 2
Functions
  list of 21
  ps3000aBlockReady 23
  ps3000aChangePowerSource 24
  ps3000aCloseUnit 25
  ps3000aCurrentPowerSource 26
  ps3000aDataReady 27
  ps3000aEnumerateUnits 28
  ps3000aFlashLed 29
  ps3000aGetAnalogueOffset 30
  ps3000aGetChannelInformation 31
  ps3000aGetMaxDownSampleRatio 32
  ps3000aGetMaxEtsValues 33
  ps3000aGetMaxSegments 34
  ps3000aGetNoOfCaptures 35, 36
  ps3000aGetStreamingLatestValues 37
  ps3000aGetTimebase 8, 38
  ps3000aGetTimebase2 39
  ps3000aGetTriggerInfoBulk 40
  ps3000aGetTriggerTimeOffset 41
  ps3000aGetTriggerTimeOffset64 42
  ps3000aGetUnitInfo 43
  ps3000aGetValues 11, 44
  ps3000aGetValuesAsync 11, 46
  ps3000aGetValuesBulk 47
  ps3000aGetValuesOverlapped 48
  ps3000aGetValuesOverlappedBulk 49
  ps3000aGetValuesTriggerTimeOffsetBulk 50
  ps3000aGetValuesTriggerTimeOffsetBulk64 51
  ps3000aHoldOff 52
  ps3000aIsReady 53

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

Functions
ps3000aIsTriggerOrPulseWidthQualifierEnabled 54
ps3000aMaximumValue 6, 55
ps3000aMemorySegments 56
ps3000aMinimumValue 6, 57
ps3000aNoOfStreamingValues 58
ps3000aOpenUnit 59
ps3000aOpenUnitAsync 60
ps3000aOpenUnitProgress 61
ps3000aPingUnit 62
ps3000aRunBlock 63
ps3000aRunStreaming 65
ps3000aSetChannel 6, 68
ps3000aSetDataBuffer 69
ps3000aSetDataBuffers 70
ps3000aSetDigitalPort 71
ps3000aSetEts 17, 72
ps3000aSetEtsTimeBuffer 73
ps3000aSetEtsTimeBuffers 74
ps3000aSetNoOfCaptures 75
ps3000aSetPulseWidthDigitalPortProperties 76
ps3000aSetPulseWidthQualifier 77
ps3000aSetPulseWidthQualifierV2 80
ps3000aSetSigGenArbitrary 83
ps3000aSetSigGenBuiltIn 87
ps3000aSetSigGenBuiltInV2 90
ps3000aSetSigGenPropertiesArbitrary 91
ps3000aSetSigGenPropertiesBuiltIn 92
ps3000aSetSimpleTrigger 7, 93
ps3000aSetTriggerChannelConditions 7, 94
ps3000aSetTriggerChannelConditionsV2 96
ps3000aSetTriggerChannelDirections 7, 98
ps3000aSetTriggerChannelProperties 7, 99
ps3000aSetTriggerDelay 101
ps3000aSetTriggerDigitalPortProperties 102
ps3000aSigGenArbitraryMinMaxValues 104
ps3000aSigGenFrequencyToPhase 105
ps3000aSigGenSoftwareControl 106
ps3000aStop 11, 107
ps3000aStreamingReady 108

H

Hysteresis 100, 103

I

Index modes 86
Information, reading from units 43
Input range, selecting 68
Intended use 1

L

LabVIEW 139
LED
flashing 29
Legal information 2
Liability 2

M

Macros in Excel 139
MATLAB 141
Memory in scope 10
Memory segments 10, 11, 19, 56
Mission-critical applications 2
Multi-unit operation 20

N

Numeric data types 143

O

One-shot signals 17
Opening a unit 59
checking progress 61
without blocking 60

P

PC oscilloscope 1
PC requirements 3
PICO_STATUS enum type 143
PicoScope 3000 MSO Series 1
PicoScope 3000A Series 1
PicoScope 3000B Series 1
PicoScope 3000D MSO Series 1
PicoScope 3000D Series 1
PicoScope software 1, 3, 143
Ports
enabling 71
PORT0, PORT1 6
settings 71
Power source 5, 24, 26
ps3000a API 3
ps3000a.dll 3
PS3000A_CONDITION_ constants 79
PS3000A_CONDITION_V2 constants 82
PS3000A_LEVEL constant 100, 103
PS3000A_PWQ_CONDITIONS structure 79
PS3000A_PWQ_CONDITIONS_V2 structure 82
PS3000A_RATIO_MODE_AGGREGATE 45

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

PS3000A_RATIO_MODE_AVERAGE 45
PS3000A_RATIO_MODE_DECIMATE 45
PS3000A_TIME_UNITS constant 41, 42
PS3000A_TRIGGER_CHANNEL_PROPERTIES structure 100, 103
PS3000A_TRIGGER_CONDITION constants 95
PS3000A_TRIGGER_CONDITION_V2 constants 97
PS3000A_TRIGGER_CONDITIONS 94
PS3000A_TRIGGER_CONDITIONS structure 95
PS3000A_TRIGGER_CONDITIONS_V2 96
PS3000A_TRIGGER_CONDITIONS_V2 structure 97
PS3000A_WINDOW constant 100, 103
Pulse-width qualifier 77
    conditions 79
    requesting status 54
Pulse-width qualifierV2 80
    conditions 82

R

Ranges 31
Rapid block mode 9, 12, 35, 36
    aggregation 15
    no aggregation 13
    setting number of captures 75
Retrieving data 44, 46
    block mode, deferred 48
    rapid block mode 47
    rapid block mode, deferred 49
    stored 20
    streaming mode 37
Retrieving times
    rapid block mode 50, 51

S

Sampling rate
    block mode 10
    streaming mode 9
Scaling 6
Serial numbers 28
Setup time 10
Signal generator
    arbitrary waveforms 83
    built-in waveforms 87, 90
    calculating phase 105
    software trigger 106
Spectrum analyzer 1
Status codes 143
Stopping sampling 107
Streaming mode 9, 19
    callback 108
    getting number of samples 58
    retrieving data 37
    running 65
    using 19
Structures 143
Support 2

T

Threshold voltage 7
Time buffers
    setting for ETS 73, 74
Timebase 8
    calculating 38, 39
Trademarks 2
Trigger 7
    channel properties 76, 99, 102
    conditions 94, 95, 96, 97
    delay 101
    digital port pulse width 76
    digital ports 102
    directions 98
    pulse-width qualifier 77
    pulse-width qualifier conditions 79
    pulse-width qualifierV2 80
    pulse-width qualifierV2 conditions 82
    requesting status 54
    setting up 93
    stability 17
    time offset 41, 42
    time offsets in rapid mode 40

U

Upgrades 2
Usage 2
USB 1, 3, 4
    hub 20
    powering 5

V

VB.NET 142
Viruses 2
Voltage range 6
    selecting 68

W

WinUsb.sys 3
Wrapper functions
    AutoStopped 111
    AvailableData 112

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

Wrapper functions
ClearTriggerReady 114
decrementDeviceCount 115
GetStreamingLatestValues 117
IsReady 119
IsTriggerReady 120
resetNextDeviceIndex 121
RunBlock 122
setAppAndDriverBuffers 123
setAppAndDriverDigiBuffers 125
setChannelCount 127
setDigitalPortCount 128
setEnabledChannels 129
setEnabledDigitalPorts 130
setMaxMinAppAndDriverBuffers 124
setMaxMinAppAndDriverDigiBuffers 126
SetPulseWidthQualifier 131
SetPulseWidthQualifierV2 132
SetTriggerConditions 133
SetTriggerConditionsV2 135
SetTriggerProperties 136
StreamingCallback 137
using 109

Copyright © 2011-2015 Pico Technology Limited. All rights reserved.

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

ps3000apg.en r13 04/12/2014
Copyright © 2011-2015 Pico Technology Limited. All rights reserved.