PicoScope 4000 Series
PC Oscilloscopes

Programmer's Guide

ps4000pg.en-3


pico
Technology

Contents

1 Welcome

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

This manual explains how to use the Application Programming Interface (API) for the PicoScope 4000 Series scopes. For more information on the hardware, see the PicoScope 4000 Series User's Guide available as a separate PDF.

2 Introduction

2.1 Using this guide

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

The abbreviation MS/s is used in this guide to mean megasamples per second.

2.2 Software licence conditions

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

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

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

Copyright. Pico Technology Limited claims the copyright of, and retains the rights to, all material (software, documents etc.) contained in this release. You may copy and distribute the entire release in its original state, but must not copy individual items within the release other than for backup purposes unless permission is explicitly granted by the licensing terms of the items.

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

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

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

2.3 Trademarks

Windows, Excel and Visual Basic are registered trademarks or trademarks of Microsoft Corporation in the USA and other countries. Delphi is a registered trademark of Embarcadero Technologies. Agilent VEE is a registered trademark of Agilent Technologies, Inc. LabView is a registered trademark of National Instruments Corporation.

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

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

2.4 Company details

Address:	Pico Technology James House Colmworth Business Park St Neots Cambridgeshire PE19 8YP United Kingdom
Phone:	+44 (0) 1480 396 395
Fax:	+44 (0) 1480 396 296
Email:	
Technical Support:	support@picotech.com
Sales:	sales@picotech.com
Web site:	www.picotech.com

3 Product information

3.1 System requirements

Using with PicoScope for Windows

To ensure that your PicoScope 4000 Series 68 PC Oscilloscope operates correctly with the PicoScope 68 software, you must have a computer with at least the minimum system requirements to run one of the supported operating systems, as shown in the following table. The performance of the software will increase with more powerful PCs, including those with multi-core processors.

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

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

Using with custom applications

Drivers are available for Windows XP (SP2) and Windows Vista. System specifications for Windows are the same as under "Using with PicoScope for Windows", above.

3.2 Installation instructions

IMPORTANT
Do not connect your PicoScope 4000 Series 68
scope device to the PC before you
have installed the Pico Technology software.
If you do, Windows might not
recognise the scope device correctly.

Procedure

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

Checking the installation

Once you have installed the software and connected the PC Oscilloscope to the PC, start the PicoScope 68 software. PicoScope should now display any signal connected to the scope inputs. If a probe is connected to your oscilloscope, you should see a small 50 or 60 hertz signal in the oscilloscope window when you touch the probe tip with your finger.

Moving your PicoScope PC Oscilloscope to another USB port

- Windows XP SP2

When you first installed the PicoScope 4000 Series PC Oscilloscope by plugging it into a USB 68 port, Windows associated the Pico driver 6 with that port. If you later move the oscilloscope to a different USB port, Windows will display the "New Hardware Found Wizard" again. When this occurs, just click "Next" in the wizard to repeat the installation. If Windows gives a warning about Windows Logo Testing, click "Continue Anyway". As all the software you need is already installed on your computer, there is no need to insert the Pico Software CD again.

- Windows Vista

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

4 Programming with the PicoScope 4000 Series

The ps4000.dll dynamic link library in your PicoScope installation directory allows you to program a PicoScope 4000 Series oscilloscope [68] using standard C function calls. [18]

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

- Open [40] the scope unit.
- Set up the input channels with the required voltage ranges [7] and coupling mode [7].
- Set up triggering [7].
- Start capturing data. (See Sampling modes [8], where programming is discussed in more detail.)
- Wait until the scope unit is ready.
- Stop capturing data.
- Copy data to a buffer.
- Close the scope unit.

Numerous sample programs [62] are installed with your PicoScope software. These show how to use the functions of the driver software in each of the modes available.

4.1 Driver

Your application will communicate with a PicoScope 4000 API driver called ps4000.dll. The driver exports the PicoScope 4000 function definitions [18] in standard C format, but this does not limit you to programming in C. You can use the API with any programming language that supports standard C calls.

The API driver depends on a kernel driver, picopp.sys, which works with Windows XP SP2 and Vista. Your application does not need to call the kernel driver. Once you have installed the PicoScope 6 software, Windows automatically installs the kernel driver when you plug in the PicoScope 4000 Series [68] PC Oscilloscope for the first time.

4.2 System requirements

General requirements

See System Requirements. [4]

USB

The PicoScope 4000 driver offers three different methods [8] of recording data, all of which support both USB 1.1 and USB 2.0, although the fastest transfer rates are achieved between the PC and the PicoScope 4000 using USB 2.0.

4.3 Voltage ranges

You can set a device input channel to any voltage range from ±50 mV to ±100 V with the ps4000SetChannel 47 function. Each sample is scaled from 12 bits to 16 bits, so that the values returned to your application are as follows:

Constant	Voltage	Value returned	
		decimal	hex
PS4000_MIN_VALUE	minimum	-32 764	8004
N/A	zero	0	0000
PS4000_MAX_VALUE	maximum	32 764	7FFC

* In streaming mode, 14 this special value indicates a buffer overrun.

4.4 Channel selection

You can switch each channel on and off, and set its coupling mode to either AC or DC, using the ps4000SetChannel 47 function.

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

4.5 Triggering

PicoScope 4000 Series PC Oscilloscopes can either start collecting data immediately, or be programmed to wait for a trigger event to occur. In both cases you need to use the PicoScope 4000 trigger functions [1] 54 [2] 56 [3] 57. A trigger event can occur when one of the signal or trigger input channels crosses a threshold voltage on either a rising or a falling edge.

The driver supports these triggering methods:

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

4.6 Sampling modes

PicoScope 4000 Series PC Oscilloscopes can run in various sampling modes.

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

- Rapid block mode. This is a variant of block mode that allows you to capture more than one waveform at a time with a minimum of delay between captures. You can use aggregation in this mode if you wish.

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

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

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

4.6.1 Block mode

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

- Block size. The maximum number of values depends upon the size of the oscilloscope's memory. The memory buffer is shared between the enabled channels, so if two channels are enabled, each receives half the memory. These features are handled transparently by the driver. The block size also depends on the number of memory segments in use (see ps4000MemorySegments).

- Sampling rate. A PicoScope 4000 Series PC Oscilloscope can sample at a number of different rates according to the selected timebase and the combination of channels that are enabled. The maximum sampling rate of 80 MS/s can be achieved with a single channel enabled, or with these two-channel combinations: AC, AD, BC and BD. All other combinations limit the scope to a maximum sampling rate of 20 MS/s.

- Setup time. The driver normally performs a number of setup operations, which can take up to 50 milliseconds, before collecting each block of data. If you need to collect data with the minimum time interval between blocks, use rapid block mode and avoid calling setup functions between calls to ps4000RunBlock, ps4000Stop and ps4000GetValues.

- Aggregation. When the data has been collected, you can set an optional aggregation 67 factor and examine the data. Aggregation is a process that reduces the amount of data by combining adjacent samples using a maximum/minimum algorithm. It is useful for zooming in and out of the data without having to repeatedly transfer the entire contents of the scope's buffer to the PC.

- Memory segmentation. The scope's internal memory can be divided into segments so that you can capture several waveforms in succession. Configure this using ps4000MemorySegments. 38

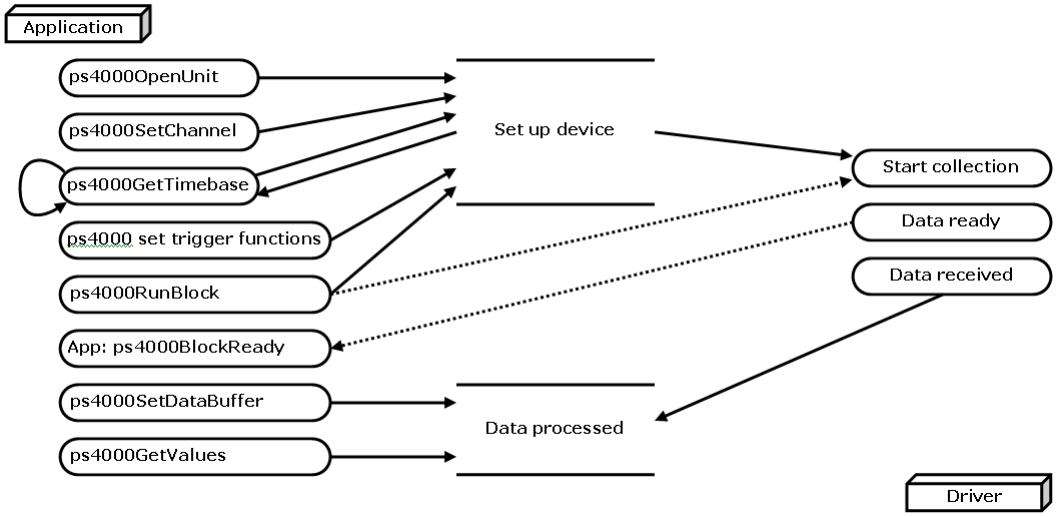
- Data retention. The data is lost when a new run is started in the same segment or the scope is powered down.

See Using block mode 9 for programming details.

4.6.1.1 Using block mode

This is the general procedure for reading and displaying data in block mode 8 using a single memory segment: 38

1. Open the oscilloscope using ps4000OpenUnit. 40
2. Select channel ranges and AC/DC coupling using ps4000SetChannel. 47
3. Using ps4000GetTimebase 25, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions [1] 54 [2] 56 [3] 57 to set up the trigger if required.
5. Start the oscilloscope running using ps4000RunBlock. 43
6. Wait until the oscilloscope is ready using the ps4000BlockReady 19 callback.
7. Use ps4000SetDataBuffer 48 to tell the driver where your memory buffer is.
8. Transfer the block of data from the oscilloscope using ps4000GetValues. 30
9. Display the data.
10. Repeat steps 5 to 9.
11. Stop the oscilloscope using ps4000Stop 60.



12. Request new views of stored data using different aggregation parameters: see Retrieving stored data. 15

4.6.2 Rapid block mode

In normal block mode, 8 the PicoScope 4000 series scopes collect one waveform at a time. You start the the device running, wait until all samples are collected by the device, and then download the data to the PC or start another run. There is a time overhead of tens of milliseconds associated with starting a run, causing a gap between waveforms. When you collect data from the device, there is another minimum time overhead which is most noticeable when using a small number of samples.

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

See Using rapid block mode 10 for details.

4.6.2.1 Using rapid block mode

You can use rapid block mode 10 with or without aggregation. 67 The following procedure shows you how to use it without aggregation.

Without aggregation

1. Open the oscilloscope using ps4000OpenUnit. 40
2. Select channel ranges and AC/DC coupling using ps4000SetChannel. 47
3. Using ps4000GetTimebase 25, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions [1] 54 [2] 56 [3] 57 to set up the trigger if required.
5. Set the number of memory segments equal to or greater than the number of captures required using ps4000MemorySegments 38. Use ps4000SetNoOfCaptures 51 before each run to specify the number of waveforms to capture.
6. Start the oscilloscope running using ps4000RunBlock. 43
7. Wait until the oscilloscope is ready using the ps4000BlockReady 19 callback.
8. Use ps4000SetDataBufferBulk 49 to tell the driver where your memory buffers are.
9. Transfer the blocks of data from the oscilloscope using ps4000GetValuesBulk. 32
10. Retrieve the time offset for each data segment using ps4000GetValuesTriggerTimeOffsetBulk64. 34
11. Display the data.
12. Repeat steps 6 to 11 if necessary.
13. Stop the oscilloscope using ps4000Stop 60.

With aggregation

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

10a. Call ps4000SetDataBuffers 50 to set up one pair of buffers for every waveform segment required.
11a. Call ps4000GetValues 30 for each pair of buffers.
12a. Retrieve the time offset for each data segment using ps4000GetTriggerTimeOffset64. 28

Continue from step 13 in the procedure for capturing data without aggregation.

4.6.2.2 Rapid block mode example 1: no aggregation

```
#define MAX_SAMPLES 1000
```

Set up the device up as usual.

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

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

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

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

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

for (int i = 0; i < 10; i++)
{
  for (int c = PS4000_CHANNEL_A; c <= PS4000_CHANNEL_D; c++)
  {
    ps4000SetDataBufferBulk
    (
      handle,
      c,
      &buffer[c][i],
      MAX_SAMPLES,
      i
    );
  }
}
```

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


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

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

```
ps4000GetValuesTriggerTimeOffsetBulk64
(
  handle,
  times,
  timeUnits,
  10,
  19
)
```

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

4.6.2.3 Rapid block mode example 2: using aggregation

```
#define MAX_SAMPLES 1000
```

Set up the device up as usual.

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

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

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

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

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

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

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



```
    );

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

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

4.6.3 Streaming mode

Streaming mode can capture data without the gaps that occur between blocks when using block mode. 8 It can transfer data to the PC at speeds of up to 13.33 million samples per second (75 nanoseconds per sample), depending on the computer's performance. This makes it suitable for high-speed data acquisition, allowing you to capture long data sets limited only by the computer's memory.

- Aggregation. The driver returns aggregated readings 67 while the device is streaming. If aggregation is set to 1 then only one buffer is returned per channel. When aggregation is set above 1 then two buffers (maximum and minimum) per channel are returned.

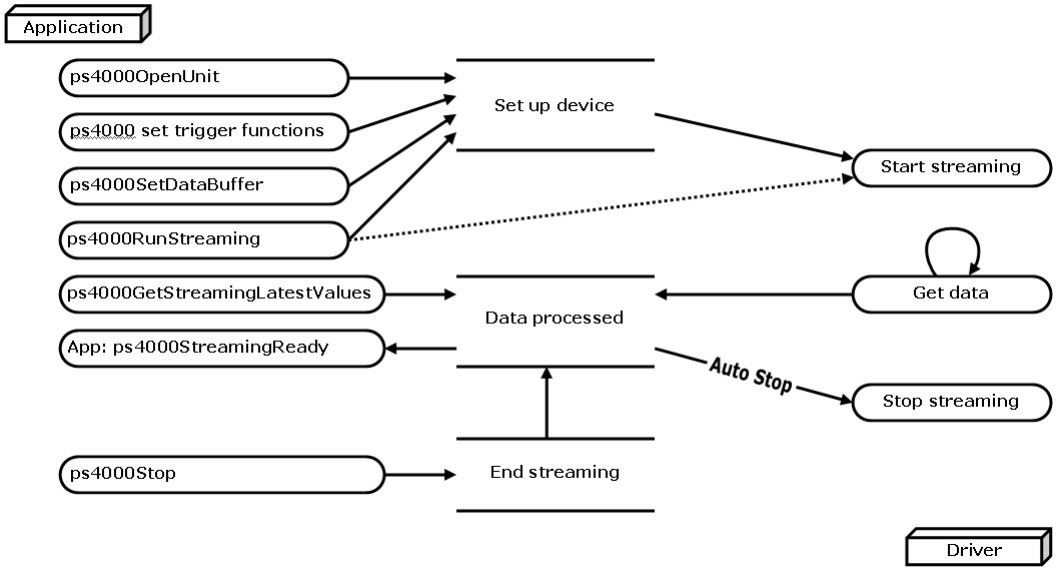
- Memory segmentation. The memory can be divided into segments 38 to reduce the latency of data transfers to the PC. However, this increases the risk of losing data if the PC cannot keep up with the device's sampling rate.

See Using streaming mode 14 for programming details.

4.6.3.1 Using streaming mode

This is the general procedure for reading and displaying data in streaming mode 14 using a single memory segment: 38

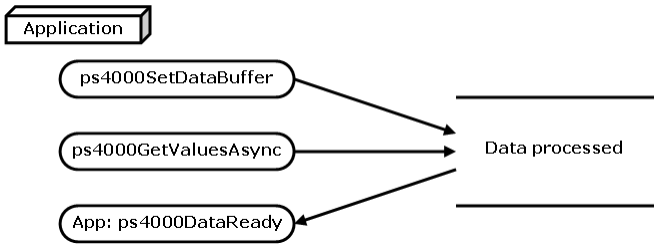
1. Open the oscilloscope using ps4000OpenUnit. 40
2. Select channels, ranges and AC/DC coupling using ps4000SetChannel. 47
3. Use the trigger setup functions [1] 54 [2] 56 [3] 57 to set up the trigger if required.
4. Call ps4000SetDataBuffer 48 to tell the driver where your data buffer is.
5. Set up aggregation and start the oscilloscope running using PS4000RunStreaming. 45
6. Call ps4000GetStreamingLatestValues 24 to get data.
7. Process data returned to your application's function. This example is using Auto Stop, so after the driver has received all the data points requested by the application, it stops the device streaming.
8. Call ps4000Stop. 60 even if Auto Stop is enabled.



9. Request new views of stored data using different aggregation parameters: see Retrieving stored data, 15

4.6.4 Retrieving stored data

You can collect data from the PicoScope 4000 driver with a different aggregation factor when ps4000RunBlock 43 or ps4000RunStreaming 45 has already been called and has successfully captured all the data. Use ps4000GetValuesAsync. 31



4.7 Oversampling

When the oscilloscope is operating at sampling rates less than its maximum, it is possible to oversample. Oversampling is taking more than one measurement during a time interval and returning the average as one sample. The number of measurements per sample is called the oversampling factor. If the signal contains a small amount of Gaussian noise, this technique can increase the effective vertical resolution 68 of the oscilloscope by *n* bits, where *n* is given approximately by the equation below:

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

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

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

Applicability	Available in block mode 8 only. Cannot be used at the same time as aggregation. 67

4.8 Timebases

The API allows you to select any of 2^{30} different timebases based on a maximum sampling rate of 80 MHz. The timebases allow slow enough sampling in block mode to overlap the streaming sample intervals, so that you can make a smooth transition between block mode and streaming mode.

The range of timebase values is divided into two subranges, with the subrange 0 to 2 specifying a power of 2, and the subrange greater than 2 specifying a number divided by 10,000,000. The maximum value is 2^{30} - 1.

t (timebase)	sample interval
0 to 2	2^t / 80,000,000 That is: - 0 => 12.5 ns 1 => 25 ns 2 => 50 ns
> 2	(t - 1) / 20,000,000 For example: - 3 => 100 ns 4 => 150 ns 5 => 200 ns ... 2^{30} - 1 => ~ 53.68 s

Applicability	Use ps4000GetTimebase 25 API call.

4.9 Combining several oscilloscopes

It is possible to collect data using up to 64 PicoScope 4000 Series PC Oscilloscopes 68 at the same time, depending on the capabilities of the PC. Each oscilloscope must be connected to a separate USB port. The ps4000OpenUnit 40 function returns a handle to an oscilloscope. All the other functions require this handle for oscilloscope identification. For example, to collect data from two oscilloscopes at the same time:

```
CALLBACK ps4000BlockReady(...)
// define callback function specific to application

handle1 = ps4000OpenUnit()
handle2 = ps4000OpenUnit()

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

ps4000SetChannel(handle2)
// set up unit 2
ps4000RunBlock(handle2)

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

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

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

4.10 API functions

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

4.10.1 ps4000BlockReady

```
typedef void (CALLBACK *ps4000BlockReady)
(
  short         handle,
  PICO_STATUS   status,
  void        * pParameter
)
```

This callback 67 function is part of your application. You register it with the PicoScope 4000 series driver using ps4000RunBlock, 43 and the driver calls it back when block-mode data is ready. You can then download the data using the ps4000GetValues 30 function.

Applicability	Block mode 8 only
Arguments	`handle,` the handle of the device returning the samples. `status,` indicates whether an error occurred during collection of the data. `pParameter,` a void pointer passed from ps4000RunBlock 43. The callback function can write to this location to send any data, such as a status flag, back to your application.
Returns	nothing

4.10.2 ps4000CloseUnit

```
PICO_STATUS ps4000CloseUnit
(
  short handle
)
```

This function shuts down a PicoScope 4000 scope device.

Applicability	All modes
Arguments	handle, the handle, returned by ps4000OpenUnit 40, of the scope device to be closed.
Returns 64	PICO_OK PICO_HANDLE_INVALID

4.10.3 ps4000DataReady

```
typedef void (CALLBACK *ps4000DataReady)
(
  short           handle,
  long            noOfSamples,
  short           overflow,
  unsigned long   triggerAt,
  short           triggered,
  void          * pParameter
)
```

This function handles post-collection data returned by the driver after a call to ps4000GetValuesAsync 31. It is a callback 67 function that is part of your application. You register it with the PicoScope 4000 series driver using ps4000GetValuesAsync, 31 and the driver calls it back when the data is ready.

Applicability	All modes
Arguments	`handle,` the handle of the device returning the samples. `noOfSamples,` the number of samples collected. `overflow,` returns a flag that indicates whether an overvoltage has occurred on any of the channels. It is a bit pattern with bit 0 denoting Channel A and bit 1 Channel B. `triggerAt,` an index to the buffer indicating the location of the trigger point. This parameter is valid only when `triggered` is non-zero. `triggered,` a flag indicating whether a trigger occurred. If non-zero, a trigger occurred at the location indicated by `triggerAt.` `pParameter,` a void pointer passed from ps4000GetValuesAsync. 31 The callback function can write to this location to send any data, such as a status flag, back to the application. The data type is defined by the application programmer.
Returns	nothing

4.10.4 ps4000FlashLed

```
PICO_STATUS ps4000FlashLed
(
  short handle,
  short start
)
```

This function flashes the LED on the front of the scope without blocking the calling thread. Calls to ps4000RunStreaming 45 and ps4000RunBlock 43 cancel any flashing started by this function.

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

Returns 64	`PICO_OK`
	`PICO_HANDLE_INVALID`
	`PICO_BUSY`

4.10.5 ps4000GetMaxDownSampleRatio

```
PICO_STATUS ps4000GetMaxDownSampleRatio
(
  short           handle,
  unsigned long   noOfUnaggregatedSamples,
  unsigned long * maxDownSampleRatio,
  short           downSampleRatioMode,
  unsigned short  segmentIndex
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device `noOfUnaggregatedSamples`, the number of unaggregated samples to be used to calculate the maximum downsampling ratio `maxDownSampleRatio`: returns the aggregation ratio `downSampleRatioMode`: see ps4000GetValues 30 `segmentIndex`, the memory segment 38 where the data is stored
Returns 64	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_TOO_MANY_SAMPLES

4.10.6 ps4000GetStreamingLatestValues

```
PICO_STATUS ps4000GetStreamingLatestValues
(
  short                   handle,
  ps4000StreamingReady    lpPs4000Ready,
  void                  * pParameter
)
```

This function is used to collect the next block of values while streaming 14 is running. You must call ps4000RunStreaming 45 beforehand to set up streaming.

Applicability	Streaming 14 mode only
Arguments	`handle`, the handle of the required device. `lpPs4000Ready`, a pointer to your ps4000StreamingReady 61 callback function that will return the latest aggregated values. `pParameter`, a void pointer that will be passed to the ps4000StreamingReady 61 callback function.

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_INVALID_CALL`
	`PICO_BUSY`
	`PICO_NOT_RESPONDING`

4.10.7 ps4000GetTimebase

```
PICO_STATUS ps4000GetTimebase
(
  short                 handle,
  unsigned long         timebase,
  long                  noSamples,
  long                * timeIntervalNanoseconds,
  short                 oversample,
  long                * maxSamples
  unsigned short        segmentIndex
)
```

This function discovers which timebases[68] are available on the oscilloscope. You should set up the channels using ps4000SetChannel[47] first.

Applicability	All modes
Arguments	`handle,` the handle of the required device. `timebase,` a code between 0 and 2^{30}-1 that specifies the sampling interval (see timebase guide[16]) `noSamples,` the number of samples required. This value is used to calculate the most suitable time unit to use. `timeIntervalNanoseconds,` a pointer to the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. `oversample,` the amount of oversample required. An oversample of 4, for example, would quadruple the time interval and quarter the maximum samples, and at the same time would increase the effective resolution by one bit. See the topic on oversampling.[15] `maxSamples,` a pointer to the maximum number of samples available. The maximum samples may vary depending on the number of channels enabled, the timebase chosen and the oversample selected. If this pointer is null, nothing will be written here. `segmentIndex,` the number of the memory segment to use.
Returns[64]	PICO_OK PICO_INVALID_HANDLE PICO_TOO_MANY_SAMPLES PICO_INVALID_CHANNEL PICO_INVALID_TIMEBASE PICO_INVALID_PARAMETER

4.10.8 ps4000GetTimebase2

```
PICO_STATUS ps4000GetTimebase2
(
  short            handle,
  unsigned long    timebase,
  long             noSamples,
  float          * timeIntervalNanoseconds,
  short            oversample,
  long           * maxSamples
  unsigned short   segmentIndex
)
```

This function differs from ps4000GetTimebase 25 only in the `float *` type of the `timeIntervalNanoseconds` argument.

Applicability	All modes
Arguments	`timeIntervalNanoseconds`, a pointer to the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. All others as in ps4000GetTimebase 25.
Returns 64	See ps4000GetTimebase 25.

4.10.9 ps4000GetTriggerTimeOffset

```
PICO_STATUS ps4000GetTriggerTimeOffset
(
  short                 handle
  unsigned long     *   timeUpper
  unsigned long     *   timeLower
  PS4000_TIME_UNITS *   timeUnits
  unsigned short        segmentIndex
)
```

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

Applicability	Block mode, 8 rapid block mode 10

Arguments	`handle`, the handle of the required device
	
	`timeUpper`, a pointer to the upper 32 bits of the time at which the trigger point occurred
	
	`timeLower`, a pointer to the lower 32 bits of the time at which the trigger point occurred
	
	`timeUnits`, returns the time units in which `timeUpper` and `timeLower` are measured. The allowable values are: -
	`PS4000_FS`
	`PS4000_PS`
	`PS4000_NS`
	`PS4000_US`
	`PS4000_MS`
	`PS4000_S`
	
	`segmentIndex`, the number of the memory segment 38 for which the information is required.

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_DEVICE_SAMPLING`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NULL_PARAMETER`
	`PICO_NO_SAMPLES_AVAILABLE`

4.10.10 ps4000GetTriggerTimeOffset64

```
PICO_STATUS ps4000GetTriggerTimeOffset64
(
  short              handle,
  __int64          * time,
  PS4000_TIME_UNITS * timeUnits,
  unsigned short     segmentIndex
)
```

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

Applicability	Block mode, 8 rapid block mode 10

Arguments	`handle`, the handle of the required device
	
	`time`, a pointer to the time at which the trigger point occurred
	
	`timeUnits`, returns the time units in which `time` is measured. The allowable values are: -
	`PS4000_FS`
	`PS4000_PS`
	`PS4000_NS`
	`PS4000_US`
	`PS4000_MS`
	`PS4000_S`
	
	`segmentIndex`, the number of the memory segment 38 for which the information is required

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_DEVICE_SAMPLING`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NULL_PARAMETER`
	`PICO_NO_SAMPLES_AVAILABLE`

4.10.11 ps4000GetUnitInfo

```
PICO_STATUS ps4000GetUnitInfo
(
  short       handle,
  char      * string,
  short       stringLength,
  short     * requiredSize
  PICO_INFO   info
)
```

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

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

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`
	`PICO_INVALID_INFO`
	`PICO_INFO_UNAVAILABLE`

info		String returned	Example
0	PICO_DRIVER_VERSION	Version number of PicoScope 4000 DLL	1,0,0,1
1	PICO_USB_VERSION	Type of USB connection to device: 1.1 or 2.0	2.0
2	PICO_HARDWARE_VERSION	Hardware version of device	1
3	PICO_VARIANT_INFO	Variant number of device	4224
4	PICO_BATCH_AND_SERIAL	Batch and serial number of device	KJL87/6
5	PICO_CAL_DATE	Calibration date of device	11Nov08
6	PICO_KERNEL_VERSION	Version of kernel driver	1,1,2,4

4.10.12 ps4000GetValues

```
PICO_STATUS ps4000GetValues
(
  short            handle,
  unsigned long    startIndex,
  unsigned long  * noOfSamples,
  unsigned long    downSampleRatio,
  short            downSampleRatioMode,
  unsigned short   segmentIndex,
  short          * overflow
)
```

This function returns block-mode data, either with or without aggregation, 67 starting at the specified sample number. It is used to get the stored data from the scope after data collection has stopped.

Applicability	Block mode, 8 rapid block mode 10
Arguments	`handle`, the handle of the required device. `startIndex`, a zero-based index that indicates the start point for data collection. It is measured in sample intervals from the start of the buffer. `noOfSamples`, on entry: the number of samples requested; on exit, the number of samples actually returned. `downSampleRatio`, the aggregation factor that will be applied to the raw data. `downSampleRatioMode`, whether to use aggregation to reduce the amount of data. The available values are: - `RATIO_MODE_NONE` (`downSampleRatio` is ignored) `RATIO_MODE_AGGREGATE` (uses aggregation 67) `segmentIndex`, the zero-based number of the memory segment 38 where the data is stored. `overflow`, returns a set of flags that indicate whether an overvoltage has occurred on any of the channels. It is a bit pattern with bit 0 denoting Channel A and bit 1 Channel B.

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_DEVICE_SAMPLING`
	`PICO_NULL_PARAMETER`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_INVALID_PARAMETER`
	`PICO_TOO_MANY_SAMPLES`
	`PICO_DATA_NOT_AVAILABLE`
	`PICO_STARTINDEX_INVALID`
	`PICO_INVALID_SAMPLERATIO`
	`PICO_INVALID_CALL`
	`PICO_NOT_RESPONDING`
	`PICO_MEMORY`

4.10.13 ps4000GetValuesAsync

```
PICO_STATUS ps4000GetValuesAsync
(
  short            handle,
  unsigned long    startIndex,
  unsigned long    noOfSamples,
  unsigned long    downSampleRatio,
  short            downSampleRatioMode,
  unsigned short   segmentIndex,
  void           * lpDataReady,
  void           * pParameter
)
```

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

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

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

4.10.14 ps4000GetValuesBulk

```
PICO_STATUS ps4000GetValuesBulk
(
  short            handle,
  unsigned long  * noOfSamples,
  unsigned short   fromSegmentIndex,
  unsigned short   toSegmentIndex,
  short          * overflow
)
```

This function allows more than one waveform to be retrieved at a time in rapid block mode. [10] The waveforms must have been collected sequentially and in the same run. This method of collection does not support aggregation. [67]

Applicability	Rapid block mode [10]
Arguments	`handle`, the handle of the device `* noOfSamples`. On entering the API, the number of samples required. On exiting the API, the actual number retrieved. The number of samples retrieved will not be more than the number requested. The data retrieved always starts with the first sample captured. `fromSegmentIndex`, the first segment from which the waveform should be retrieved `toSegmentIndex`, the last segment from which the waveform should be retrieved `* overflow`, equal to or larger than the number of waveforms to be retrieved. Each segment index has a separate `overflow` element, with `overflow[0]` containing the `fromSegmentIndex` and the last index the `toSegmentIndex`.

Returns [64]	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_STARTINDEX_INVALID`
	`PICO_NOT_RESPONDING`

4.10.15 ps4000GetValuesTriggerTimeOffsetBulk

```
PICO_STATUS ps4000GetValuesTriggerTimeOffsetBulk
(
  short                 handle,
  unsigned long     *   timesUpper,
  unsigned long     *   timesLower,
  PS4000_TIME_UNITS *   timeUnits,
  unsigned short        fromSegmentIndex,
  unsigned short        toSegmentIndex
)
```

This function retrieves the time offset, as lower and upper 32-bit values, for a group of waveforms obtained in rapid block mode. 10 The array size for `timesUpper` and `timesLower` must be greater than or equal to the number of waveform time offsets requested. The segment indexes are inclusive.

Applicability	Rapid block mode 10
Arguments	`handle`, the handle of the device `* timesUpper`, a pointer to 32-bit integers. This will hold the most significant 32 bits of the time offset for each requested segment index. `times[0]` will hold the `fromSegmentIndex` time offset and the last `times` index will hold the `toSegmentIndex` time offset. `* timesLower`, a pointer to 32-bit integers. This will hold the least-significant 32 bits of the time offset for each requested segment index. `times[0]` will hold the `fromSegmentIndex` time offset and the last `times` index will hold the `toSegmentIndex` time offset. `* timeUnits`, a pointer to a range of `PS4000_TIME_UNITS`. This must be equal to or larger than the number of requested times. `timeUnits[0]` will contain the time unit for `fromSegmentIndex` and the last index will contain the time unit for `toSegmentIndex`. `fromSegmentIndex`, the first segment for which the time offset is required `toSegmentIndex`, the last segment for which the time offset is required. If `toSegmentIndex` is less than `fromSegmentIndex` then the driver will wrap around from the last segment to the first.

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`
	`PICO_DEVICE_SAMPLING`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NO_SAMPLES_AVAILABLE`

4.10.16 ps4000GetValuesTriggerTimeOffsetBulk64

```
PICO_STATUS ps4000GetValuesTriggerTimeOffsetBulk64
(
  short                 handle,
  __int64             * times,
  PS4000_TIME_UNITS   * timeUnits,
  unsigned short        fromSegmentIndex,
  unsigned short        toSegmentIndex
)
```

This function retrieves the time offset, as a 64-bit integer, for a group of waveforms captured in rapid block mode. [10] The array size of `times` must be greater than or equal to the number of waveform time offsets requested. The segment indexes are inclusive.

Applicability	Rapid block mode [10]
Arguments	`handle`, the handle of the device `* times`, a pointer to 64-bit integers. This will hold the time offset for each requested segment index. `times[0]` will hold the time offset for `fromSegmentIndex`, and the last `times` index will hold the time offset for `toSegmentIndex`. `* timeUnits`, a pointer to a range of `PS4000_TIME_UNITS`. This must be equal or larger than the number of requested times. `timeUnits[0]` will contain the time unit for `fromSegmentIndex`, and the last index will contain the `toSegmentIndex`. `fromSegmentIndex`, the first segment for which the time offset is required. The result will be placed in `times[0]` and `timeUnits[0]`. `toSegmentIndex`, the last segment for which the time offset is required. The result will be placed in the last elements of the `times` and `timeUnits` arrays. If `toSegmentIndex` is less than `fromSegmentIndex` then the driver will wrap around from the last segment to the first.

Returns [64]	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_NULL_PARAMETER`
	`PICO_DEVICE_SAMPLING`
	`PICO_SEGMENT_OUT_OF_RANGE`
	`PICO_NO_SAMPLES_AVAILABLE`

4.10.17 ps4000HoldOff

```
PICO_STATUS ps4000HoldOff
(
  short                handle,
  u_int64_t            holdoff,
  PS4000_HOLDOFF_TYPE  type
)
```

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

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

Returns 64	`PICO_OK` - success
	`PICO_DRIVER_FUNCTION`
	`PICO_INVALID_PARAMETER`

4.10.18 ps4000IsLedFlashing

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

This function reports whether or not the LED is flashing.

Applicability	All modes
Arguments	`handle,` the handle of the scope device `status,` returns a flag indicating the status of the LED: - <> 0 : flashing 0 : not flashing
Returns 64	PICO_OK PICO_HANDLE_INVALID PICO_NULL_PARAMETER

4.10.19 ps4000IsTriggerOrPulseWidthQualifierEnabled

```
PICO_STATUS ps4000IsTriggerOrPulseWidthQualifierEnabled
(
  short   handle,
  short * triggerEnabled,
  short * pulseWidthQualifierEnabled
)
```

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

Applicability	Call after setting up the trigger, and just before calling either ps4000RunBlock 43 or ps4000RunStreaming. 45
Arguments	`handle,` the handle of the required device `triggerEnabled,` indicates whether the trigger will successfully be set when ps4000RunBlock 43 or ps4000RunStreaming 45 is called. A non-zero value indicates that the trigger is set, otherwise the trigger is not set. `pulseWidthQualifierEnabled,` indicates whether the pulse width qualifier will successfully be set when ps4000RunBlock 43 or ps4000RunStreaming 45 is called. A non-zero value indicates that the pulse width qualifier is set, otherwise the pulse width qualifier is not set.
Returns 64	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER

4.10.20 ps4000MemorySegments

```
PICO_STATUS ps4000MemorySegments
(
  short            handle
  unsigned short   nSegments,
  long           * nMaxSamples
)
```

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

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

Applicability	All modes
Arguments	`handle`, the handle of the required device `nSegments`, the number of segments to be used, from `1` to `8,192` `nMaxSamples`, returns the number of samples that are available in each segment. This is independent of the number of channels, so if more than one channel is in use then the number of samples available to each channel is `nMaxSamples` divided by the number of channels.

Returns 64	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_TOO_MANY_SEGMENTS`
	`PICO_MEMORY`

4.10.21 ps4000NoOfStreamingValues

```
PICO_STATUS ps4000NoOfStreamingValues
(
  short           handle,
  unsigned long * noOfValues
)
```

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

Applicability	Streaming mode 14. Call after ps4000Stop. 60
Arguments	handle, the handle of the required device noOfValues, returns the number of samples
Returns 64	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_NOT_USED PICO_BUSY

4.10.22 ps4000OpenUnit

```
PICO_STATUS ps4000OpenUnit
(
  short * handle
)
```

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

Applicability	All modes
Arguments	handle, pointer to a short that receives the handle number: -1 : if the unit fails to open, 0 : if no unit is found or > 0 : if successful (value is handle to the device opened) The handle number must be used in all subsequent calls to API functions to identify this scope device.
Returns 64	PICO_OK PICO_OS_NOT_SUPPORTED PICO_OPEN_OPERATION_IN_PROGRESS PICO_EEPROM_CORRUPT PICO_KERNEL_DRIVER_TOO_OLD PICO_FW_FAIL PICO_MAX_UNITS_OPENED PICO_NOT_FOUND PICO_NOT_RESPONDING

4.10.23 ps4000OpenUnitAsync

```
PICO_STATUS ps4000OpenUnitAsync
(
  short * status
)
```

This function opens a scope device without blocking the calling thread. You can find out when it has finished by periodically calling ps4000OpenUnitProgress[42] until that function returns a non-zero value.

Applicability	All modes
Arguments	status, pointer to a short that indicates: 0 if there is already an open operation in progress 1 if the open operation is initiated
Returns[64]	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_OPERATION_FAILED

4.10.24 ps4000OpenUnitProgress

```
PICO_STATUS ps4000OpenUnitProgress
(
  short * handle,
  short * progressPercent,
  short * complete
)
```

This function checks on the progress of ps4000OpenUnitAsync 41.

Applicability	Use after ps4000OpenUnitAsync 41
Arguments	handle, pointer to a short where the unit handle is to be written. -1 if the unit fails to open, 0 if no unit is found or a non-zero handle to the device. Note: This handle is not valid unless the function returns PICO_OK. progressPercent, pointer to a short to which the percentage progress is to be written. 100% implies that the open operation is complete. complete, pointer to a short that is set to 1 when the open operation has finished
Returns 64	PICO_OK PICO_NULL_PARAMETER PICO_OPERATION_FAILED

4.10.25 ps4000RunBlock

```
PICO_STATUS ps4000RunBlock
(
  short               handle,
  long                noOfPreTriggerSamples,
  long                noOfPostTriggerSamples,
  unsigned long       timebase,
  short               oversample,
  long              * timeIndisposedMs,
  unsigned short      segmentIndex,
  ps4000BlockReady    lpReady,
  void              * pParameter
)
```

This function starts a collection of data points (samples) in block mode.

The number of samples is determined by `noOfPreTriggerSamples` and `noOfPostTriggerSamples` (see below for details). The total number of samples must not be more than the memory depth of the segment 38 referred to by `segmentIndex`.

Applicability	Block mode, 8 rapid block mode 10
Arguments	handle, the handle of the required device. noOfPreTriggerSamples, the number of samples to return before the trigger event. If no trigger has been set then this argument is ignored and noOfPostTriggerSamples specifies the maximum number of data points (samples) to collect. noOfPostTriggerSamples, the number of samples to be taken after a trigger event. If no trigger event is set then this specifies the maximum number of samples to be taken. If a trigger condition has been set, this specifies the number of data points (samples) to be taken after a trigger has fired, and the number of data points to be collected is: - noOfPreTriggerSamples + noOfPostTriggerSamples timebase, a number in the range 0 to 2^{30}-1. See the guide to calculating timebase values. 16 oversample, the oversampling 15 factor, a number in the range 1 to 16. timeIndisposedMs, returns the time, in milliseconds, that the PicoScope4000 will spend collecting samples. This does not include any auto trigger timeout. If this pointer is null, nothing will be written here. segmentIndex, zero-based, specifies which memory segment 38 to use. lpReady, a pointer to the ps4000BlockReady 19 callback that the driver will call when the data has been collected. pParameter, a void pointer that is passed to the ps4000BlockReady 19 callback function. The callback can use the pointer to return arbitrary data to your application.
Returns 64	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_CHANNEL PICO_INVALID_TRIGGER_CHANNEL PICO_INVALID_CONDITION_CHANNEL PICO_TOO_MANY_SAMPLES PICO_INVALID_TIMEBASE PICO_NOT_RESPONDING PICO_CONFIG_FAIL PICO_INVALID_PARAMETER PICO_NOT_RESPONDING PICO_TRIGGER_ERROR

4.10.26 ps4000RunStreaming

```
PICO_STATUS ps4000RunStreaming
(
  short               handle,
  unsigned long     * sampleInterval,
  PS4000_TIME_UNITS   sampleIntervalTimeUnits
  unsigned long       maxPreTriggerSamples,
  unsigned long       maxPostTriggerSamples,
  short               autoStop
  unsigned long       downSampleRatio,
  unsigned long       overviewBufferSize
)
```

This function tells the oscilloscope to start collecting data in streaming mode 14. When data has been collected from the device it is aggregated 67 and the values returned to the application. Call ps4000GetStreamingLatestValues 24 to retrieve the data.

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

Applicability	Streaming mode 14 only
Arguments	handle, the handle of the required device. sampleInterval, a pointer to the requested time interval between data points on entry and the actual time interval assigned on exit. sampleIntervalTimeUnits, the unit of time that the sampleInterval is set to. Use one of these values: - PS4000_FS PS4000_PS PS4000_NS PS4000_US PS4000_MS PS4000_S maxPreTriggerSamples, the maximum number of raw samples before a trigger condition for each enabled channel. If no trigger condition is set this argument is ignored. maxPostTriggerSamples, the maximum number of raw samples after a trigger condition for each enabled channel. If no trigger condition is set this argument states the maximum number of samples to be stored. autoStop, a flag to specify if the streaming should stop when all of maxSamples have been taken. downSampleRatio, the number of raw values to each aggregated value. overviewBufferSize, the size of the overview buffers. These are temporary buffers used for storing the data before returning it to the application. The size is the same as the bufferLth value passed to ps4000SetDataBuffer. 48
Returns 64	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_STREAMING_FAILED PICO_NOT_RESPONDING PICO_TRIGGER_ERROR PICO_INVALID_SAMPLE_INTERVAL PICO_INVALID_BUFFER

4.10.27 ps4000SetChannel

```
PICO_STATUS ps4000SetChannel
(
  short          handle,
  PS4000_CHANNEL channel,
  short          enabled,
  short          dc,
  PS4000_RANGE   range
)
```

This function specifies whether an input channel is to be enabled, the AC/DC coupling 67 mode and the voltage range.

Applicability	All modes
Arguments	handle, the handle of the required device channel, an enumerated type. The values are: - PS4000_CHANNEL_A PS4000_CHANNEL_B PS4000_CHANNEL_C (4-channel scopes only) PS4000_CHANNEL_D (4-channel scopes only) enabled, specifies if the channel is active. The values are: - TRUE = active FALSE = inactive dc, specifies the AC/DC coupling 67 mode. The values are: - TRUE = DC FALSE = AC range, a number between 2 and 12 that specifies the voltage range. See the table 47 below.
Returns 64	PICO_OK PICO_USER_CALLBACK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_INVALID_VOLTAGE_RANGE

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

4.10.28 ps4000SetDataBuffer

```
PICO_STATUS ps4000SetDataBuffer
(
  short            handle,
  PS4000_CHANNEL   channel,
  short          * buffer,
  long             bufferLth
)
```

This function registers your data buffer, for non-aggregated 67 data, with the PicoScope 4000 driver. You need to allocate the buffer before calling this function.

Applicability	All modes. For aggregated data, use ps4000SetDataBuffers 50 instead.
Arguments	handle, the handle of the required device channel, the channel for which you want to set the buffers. Use one of these values: - PS4000_CHANNEL_A PS4000_CHANNEL_B PS4000_CHANNEL_C (4-channel scopes only) PS4000_CHANNEL_D (4-channel scopes only) buffer, a buffer to receive the data values bufferLth, the size of the buffer array
Returns 64	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL

4.10.29 ps4000SetDataBufferBulk

```
PICO_STATUS ps4000SetDataBufferBulk
(
short            handle,
PS4000_CHANNEL   channel,
short          * buffer,
long             bufferLth,
unsigned short   waveform
)
```

This function allows the buffers to be set for each waveform in rapid block mode. 10 The number of waveforms captured is determined by the `nCaptures` argument sent to ps4000SetNoOfCaptures. 51 There is only one buffer for each waveform, because bulk collection does not support aggregation. 67

Applicability	Rapid block mode 10
Arguments	`handle`, the handle of the device `channel`, the scope channel with which the buffer is to be associated. The data should be retrieved from this channel by calling one of the GetValues 30 functions. `* buffer`, an array in which the captured data is stored `bufferLth`, the size of the buffer waveform, an index to the waveform number, between `0` and `nCaptures` - 1

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_PARAMETER`

4.10.30 ps4000SetDataBuffers

```
PICO_STATUS ps4000SetDataBuffers
(
  short            handle,
  PS4000_CHANNEL   channel,
  short          * bufferMax,
  short          * bufferMin,
  long             bufferLth
)
```

This function registers your data buffers, for receiving aggregated 67 data, with the PicoScope 4000 driver. You need to allocate memory for the buffers before calling this function.

Applicability	All sampling modes. For non-aggregated data, use ps4000SetDataBuffer 48 instead.

Arguments	`handle`, the handle of the required device.
	
	`channel`, the channel for which you want to set the buffers. Use one of these constants: -
	`PS4000_CHANNEL_A`
	`PS4000_CHANNEL_B`
	`PS4000_CHANNEL_C` (4-channel scopes only)
	`PS4000_CHANNEL_D` (4-channel scopes only)
	
	`bufferMax`, a buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise.
	
	`bufferMin`, a buffer to receive the minimum data values when `downSampleRatio` > 1. Not used when `downSampleRatio` is 1.
	
	`bufferLth`, specifies the size of the `bufferMax` and `bufferMin` arrays.

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`

4.10.31 ps4000SetNoOfCaptures

```
PICO_STATUS ps4000SetNoOfCaptures
(
  short          handle,
  unsigned short nCaptures
)
```

This function sets the number of captures to be collected in one run of rapid block mode. 10 If you do not call this function before a run, the driver will capture one waveform.

Applicability	Rapid block mode 10
Arguments	`handle`, the handle of the device `nCaptures`, the number of waveforms to be captured in one run

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`

4.10.32 ps4000SetPulseWidthQualifier

```
PICO_STATUS ps4000SetPulseWidthQualifier
(
  short                    handle,
  struct PWQ_CONDITIONS *  conditions,
  short                    nConditions,
  THRESHOLD_DIRECTION      direction,
  unsigned long            lower,
  unsigned long            upper,
  PULSE_WIDTH_TYPE         type
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device `conditions`, a pointer to an array of PWQ_CONDITIONS 53 structures specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there are several elements, the overall trigger condition is the logical OR of all the elements. If `conditions` is set to `null` then the pulse width qualifier is not used. `nConditions`, the number of elements in the `conditions` array. If `nConditions` is zero then the pulse width qualifier is not used. `direction`, the direction of the signal required for the trigger to fire `lower`, the lower limit of the pulse width counter `upper`, the upper limit of the pulse width counter. This parameter is used only when the type is set to `PW_TYPE_IN_RANGE` or `PW_TYPE_OUT_OF_RANGE`. `type`, the pulse width type, one of these constants: - `PW_TYPE_NONE` (do not use the pulse width qualifier) `PW_TYPE_LESS_THAN` (pulse width less than `lower`) `PW_TYPE_GREATER_THAN` (pulse width greater than `lower`) `PW_TYPE_IN_RANGE` (pulse width between `lower` and `upper`) `PW_TYPE_OUT_OF_RANGE` (pulse width not between `lower` and `upper`)

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_CONDITIONS`
	`PICO_PULSE_WIDTH_QUALIFIER`

4.10.32.1 PWQ_CONDITIONS structure

A structure of this type is passed to ps4000SetPulseWidthQualifier 52 in the `conditions` argument to specify the trigger conditions, and is defined as follows: -

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

Each structure is the logical AND of the states of the scope's inputs. The ps4000SetPulseWidthQualifier 52 function can OR together a number of these structures to produce the final pulse width qualifier, which can be any possible Boolean function of the scope's inputs.

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

4.10.33 ps4000SetTriggerChannelConditions

```
PICO_STATUS ps4000SetTriggerChannelConditions
(
  short                          handle,
  struct TRIGGER_CONDITIONS *    conditions,
  short                          nConditions
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is set up by defining one or more TRIGGER_CONDITIONS 55 structures that are then ORed together. Each structure is itself the AND of the states of one or more of the inputs. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

Applicability	All modes
Arguments	`handle`, the handle of the required device. `conditions`, a pointer to an array of TRIGGER_CONDITIONS 55 structures specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there are several elements, the overall trigger condition is the logical OR of all the elements. `nConditions`, the number of elements in the `conditions` array. If `nConditions` is zero then triggering is switched off.
Returns 64	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_MEMORY_FAIL

4.10.33.1 TRIGGER_CONDITIONS structure

A structure of this type is passed to ps4000SetTriggerChannelConditions[54] in the `conditions` argument to specify the trigger conditions, and is defined as follows: -

```
typedef struct tTriggerConditions
{
  TRIGGER_STATE channelA;
  TRIGGER_STATE channelB;
  TRIGGER_STATE channelC;
  TRIGGER_STATE channelD;
  TRIGGER_STATE external;
  TRIGGER_STATE aux;
  TRIGGER_STATE pulseWidthQualifier;
} TRIGGER_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The ps4000SetTriggerChannelConditions[54] function can OR together a number of these structures to produce the final trigger condition, which can be any possible Boolean function of the scope's inputs.

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

4.10.34 ps4000SetTriggerChannelDirections

```
PICO_STATUS ps4000SetTriggerChannelDirections
(
  short                handle,
  THRESHOLD_DIRECTION channelA,
  THRESHOLD_DIRECTION channelB,
  THRESHOLD_DIRECTION channelC,
  THRESHOLD_DIRECTION channelD,
  THRESHOLD_DIRECTION ext,
  THRESHOLD_DIRECTION aux
)
```

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

Applicability	All modes.
Arguments	`handle`, the handle of the required device `channelA`, `channelB`, `channelC`, `channelD` all specify the direction in which the signal must pass through the threshold to activate the trigger. See the table 56 below. `ext`, `aux`: not used
Returns 64	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_INVALID_PARAMETER

Trigger direction constants

ABOVE	for gated triggers: above the upper threshold
ABOVE_LOWER	for gated triggers: above the lower threshold
BELOW	for gated triggers: below the upper threshold
BELOW_LOWER	for gated triggers: below the lower threshold
RISING	for threshold triggers: rising edge, using upper threshold
RISING_LOWER	for threshold triggers: rising edge, using lower threshold
FALLING	for threshold triggers: falling edge, using upper threshold
FALLING_LOWER	for threshold triggers: falling edge, using lower threshold
RISING_OR_FALLING	for threshold triggers: either edge
INSIDE	for window-qualified triggers: inside window
OUTSIDE	for window-qualified triggers: outside window
ENTER	for window triggers: entering the window
EXIT	for window triggers: leaving the window
ENTER_OR_EXIT	for window triggers: either entering or leaving the window
POSITIVE_RUNT	for window-qualified triggers
NEGATIVE_RUNT	for window-qualified triggers
NONE	no trigger

4.10.35 ps4000SetTriggerChannelProperties

```
PICO_STATUS ps4000SetTriggerChannelProperties
(
  short                                handle,
  struct TRIGGER_CHANNEL_PROPERTIES *  channelProperties
  short                                nChannelProperties
  short                                auxOutputEnable,
  long                                 autoTriggerMilliseconds
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device. `channelProperties`, a pointer to an array of TRIGGER_CHANNEL_PROPERTIES 58 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 64	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_TRIGGER_ERROR PICO_MEMORY_FAIL PICO_INVALID_TRIGGER_PROPERTY

4.10.35.1 TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to ps4000SetTriggerChannelProperties[57] in the `channelProperties` argument to specify the trigger mechanism, and is defined as follows: -

```
typedef struct tTriggerChannelProperties
{
  short          thresholdUpper;
  unsigned short thresholdUpperHysteresis;
  short          thresholdLower;
  unsigned short thresholdLowerHysteresis;
  PS4000_CHANNEL channel;
  THRESHOLD_MODE thresholdMode;
} TRIGGER_CHANNEL_PROPERTIES
```

Elements	`thresholdUpper`, the upper threshold at which the trigger must fire. This is scaled in 16-bit ADC counts[7] 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[7] at the currently selected range for that channel.
	
	`thresholdLowerHysteresis`, the hysteresis by which the trigger must exceed the lower threshold before it will fire. It is scaled in 16-bit counts.
	
	`channel`, the channel to which the properties apply. See ps4000SetChannel[47] for possible values.
	
	`thresholdMode`, either a level or window trigger. Use one of these constants: -
	`LEVEL`
	`WINDOW`

4.10.36 ps4000SetTriggerDelay

```
PICO_STATUS ps4000SetTriggerDelay
(
  short         handle,
  unsigned long delay
)
```

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

Applicability	All modes
Arguments	`handle`, the handle of the required device `delay`, the time between the trigger occurring and the first sample, in multiples of eight sample periods. For example, if `delay`=100 then the scope would wait 800 sample periods before sampling. At a timebase 16 of 80 MS/s, or 12.5 ns per sample (`timebase` = 0), the total delay would then be 800 x 12.5 ns = 10 µs.

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`

4.10.37 ps4000Stop

```
PICO_STATUS ps4000Stop
(
  short  handle
)
```

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

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

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

Returns 64	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`

4.10.38 ps4000StreamingReady

```
typedef void (CALLBACK *ps4000StreamingReady)
(
  short           handle,
  long            noOfSamples,
  unsigned long   startIndex,
  short           overflow,
  unsigned long   triggerAt,
  short           triggered,
  short           autoStop,
  void          * pParameter
)
```

This callback 67 function is part of your application. You register it with the PicoScope 4000 series driver using ps4000GetStreamingLatestValues, 24 and the driver calls it back when streaming-mode data is ready. You can then download the data using the ps4000GetValuesAsync 31 function.

Applicability	Streaming mode 14 only
Arguments	`handle,` the handle of the device returning the samples. `noOfSamples,` the number of samples to collect. `startIndex,` an index to the first valid sample in the buffer. This is the buffer that was previously passed to ps4000SetDataBuffer. 48 `overflow,` returns a set of flags that indicate whether an overvoltage has occurred on any of the channels. It is a bit pattern with bit 0 denoting Channel A and bit 1 Channel B. `triggerAt,` an index to the buffer indicating the location of the trigger point. This parameter is valid only when `triggered` is non-zero. `triggered,` a flag indicating whether a trigger occurred. If non-zero, a trigger occurred at the location indicated by `triggerAt.` `autoStop,` the flag that was set in the call to ps4000RunStreaming. 45 `pParameter,` a void pointer passed from ps4000GetStreamingLatestValues. 24 The callback function can write to this location to send any data, such as a status flag, back to the application.
Returns	nothing

4.11 Programming examples

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

- C 62

4.11.1 C

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

- Open a PicoScope 4223, 4224, 4423 or 4424
- Collect a block of samples immediately
- Collect a block of samples when a trigger event occurs
- Collect a stream of data immediately
- Collect a stream of data when a trigger event occurs

To build this application:

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

4.11.2 Excel

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

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

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

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

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

4.11.3 LabView

Two example VIs are supplied in the PicoScope 4000 Series Software Development Kit:

- `ps4000_block.vi`
- `ps4000_stream.vi`

These demonstrate the use of block mode 8 and streaming mode 14 respectively.

You will also need:

- `ps4000.dll`

Copy all the relevant files into your LabVIEW directory.

4.12 Driver error codes

This description of the driver error codes is aimed at those people who intend to write their own programs for use with the driver. Every function in the ps4000.dll driver returns an error code from the following list of PICO_STATUS values.

Code (hex)	Enum	Description
00	PICO_OK	The PicoScope 4000 is functioning correctly
01	PICO_MAX_UNITS_OPENED	An attempt has been made to open more than PS4000_MAX_UNITS. Reserved.
02	PICO_MEMORY_FAIL	Not enough memory could be allocated on the host machine
03	PICO_NOT_FOUND	No PicoScope 4000 could be found
04	PICO_FW_FAIL	Unable to download firmware
05	PICO_OPEN_OPERATION_IN_PROGRESS	
06	PICO_OPERATION_FAILED	
07	PICO_NOT_RESPONDING	The PicoScope 4000 is not responding to commands from the PC
08	PICO_CONFIG_FAIL	The configuration information in the PicoScope 4000 has become corrupt or is missing
09	PICO_KERNEL_DRIVER_TOO_OLD	The picopp.sys file is too old to be used with the device driver
0A	PICO_EEPROM_CORRUPT	The EEPROM has become corrupt, so the device will use a default setting
0B	PICO_OS_NOT_SUPPORTED	The operating system on the PC is not supported by this driver
0C	PICO_INVALID_HANDLE	There is no device with the handle value passed
0D	PICO_INVALID_PARAMETER	A parameter value is not valid
0E	PICO_INVALID_TIMEBASE	The time base is not supported or is invalid
0F	PICO_INVALID_VOLTAGE_RANGE	The voltage range is not supported or is invalid
10	PICO_INVALID_CHANNEL	The channel number is not valid on this device or no channels have been set
11	PICO_INVALID_TRIGGER_CHANNEL	The channel set for a trigger is not available on this device
12	PICO_INVALID_CONDITION_CHANNEL	The channel set for a condition is not available on this device
13	PICO_NO_SIGNAL_GENERATOR	The device does not have a signal generator
14	PICO_STREAMING_FAILED	Streaming has failed to start or has stopped without user request
15	PICO_BLOCK_MODE_FAILED	Block failed to start - a parameter may have been set wrongly

16	PICO_NULL_PARAMETER	A parameter that was required is NULL
18	PICO_DATA_NOT_AVAILABLE	No data is available from a run block call
19	PICO_STRING_BUFFER_TOO_SMALL	The buffer passed for the information was too small
1A	PICO_ETS_NOT_SUPPORTED	ETS is not supported on this device variant
1B	PICO_AUTO_TRIGGER_TIME_TOO_SHORT	The auto trigger time is less than the time it will take to collect the data
1C	PICO_BUFFER_STALL	The collection of data has stalled as unread data would be overwritten
1D	PICO_TOO_MANY_SAMPLES	Number of samples requested is more than available in the current memory segment
1E	PICO_TOO_MANY_SEGMENTS	Not possible to create number of segments requested
1F	PICO_PULSE_WIDTH_QUALIFIER	A null pointer has been passed in the trigger function or one of the parameters is out of range
20	PICO_DELAY	One or more of the hold-off parameters are out of range
21	PICO_SOURCE_DETAILS	One or more of the source details are incorrect
22	PICO_CONDITIONS	One or more of the conditions are incorrect
23	PICO_USER_CALLBACK	The driver's thread is currently in the ps4000...Ready 19 callback function and therefore the action cannot be carried out
24	PICO_DEVICE_SAMPLING	An attempt is being made to get stored data while streaming. Either stop streaming by calling ps4000Stop, 60 or use ps4000GetStreamingLatestValues 24
25	PICO_NO_SAMPLES_AVAILABLE	...because a run has not been completed
26	PICO_SEGMENT_OUT_OF_RANGE	The memory index is out of range
27	PICO_BUSY	Data cannot be returned yet
28	PICO_STARTINDEX_INVALID	The start time to get stored data is out of range
29	PICO_INVALID_INFO	The information number requested is not a valid number
2A	PICO_INFO_UNAVAILABLE	The handle is invalid so no information is available about the device. Only PICO_DRIVER_VERSION is available.
2B	PICO_INVALID_SAMPLE_INTERVAL	The sample interval selected for streaming is out of range
2C	PICO_TRIGGER_ERROR	Not used
2D	PICO_MEMORY	Driver cannot allocate memory

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
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 30 function has been called for the collection mode in use
3D	PICO_GET_VALUES_INTERRUPTED	
3F	PICO_NOT_USED	The function is not available
40	PICO_INVALID_SAMPLERATIO	The aggregation 67 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_DRIVE_FUNCTION	You called a driver function while another driver function was still being processed
45	PICO_INVALID_COUPLING	The dc argument passed to ps4000SetChannel 47 was invalid
46	PICO_BUFFERS_NOT_SET	Memory buffers were not set up before calling one of the ps4000Run... 43 functions
47	PICO_RATIO_MODE_NOT_SUPPORTED	downSampleRatioMode 30 is not valid for the connected device
48	PICO_RAPID_NOT_SUPPORT_AGGREGATION	Aggregation was requested in rapid block mode 10
49	PICO_INVALID_TRIGGER_PROPERTY	An incorrect value was passed to ps4000SetTriggerChannelProperties 57

5 Glossary

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

Aggregation. The PicoScope 4000 68 driver can use this method to reduce the amount of data your application needs to process. This means that for every block of consecutive samples, it stores only the minimum and maximum values. You can set the number of samples in each block, called the aggregation parameter, when you call PS4000RunStreaming 45 for real-time capture, and when you call ps4000GetStreamingLatestValues 24 to obtain post-processed data.

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

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

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

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

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

Maximum sampling rate. A figure indicating the maximum number of samples the oscilloscope can acquire per second. The higher the sampling rate of the oscilloscope, the more accurate the representation of the high-frequency details in a fast signal. "MS/s" is an abbreviation for megasamples (millions of samples) per second.

Oversampling. Oversampling is taking measurements more frequently than the requested sample rate, and then combining them to produce the required number of samples. If, as is usually the case, the signal contains a small amount of noise, this technique can increase the effective vertical resolution 68 of the oscilloscope.

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

PicoScope 4000 Series. A range of high-resolution PC Oscilloscopes from Pico Technology. The part number is made up as follows:

PicoScope	4	4	2	4
	4 = 4000 Series	2 = 2 channels 4 = 4 channels	2 = 12-bit resolution	3 = automotive variants 4 = standard variants

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

Streaming mode. A sampling mode in which the oscilloscope samples data and returns it to the computer in an unbroken stream. This mode allows the capture of data sets whose size is not limited by the size of the scope's memory buffer, at sampling rates up to 13.3 million samples per second.

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

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

USB 1.1. Universal Serial Bus (Full Speed). This is a standard port used to connect external devices to PCs. A typical USB 1.1 port supports a data transfer rate of 12 megabits per second, so is much faster than an RS232 COM port.

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

Vertical resolution. A value, in bits, indicating the precision with which the oscilloscope converts input voltages to digital values. Oversampling 15 (see above) can improve the effective vertical resolution.

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

Index

A

B

C

D

E

F

H

I

L

M

O

P

R

S

T

U

V

W

Pico Technology

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

ps4000pg.en-3

7.1.10