



# **PicoScope 6000 Series**

## **PC Oscilloscopes**

Programmer's Guide



# Contents

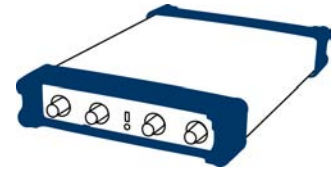
1 Introduction .....	1
<b>1 Welcome</b> .....	1
<b>2 Software licence conditions</b> .....	1
<b>3 Trademarks</b> .....	2
<b>4 Company details</b> .....	2
2 Product information .....	3
<b>1 System requirements</b> .....	3
<b>2 Installation instructions</b> .....	4
3 Programming with the PicoScope 6000 Series .....	5
<b>1 Driver</b> .....	5
<b>2 System requirements</b> .....	5
<b>3 Voltage ranges</b> .....	5
<b>4 Triggering</b> .....	6
<b>5 Sampling modes</b> .....	6
<b>1 Block mode</b> .....	7
<b>2 Rapid block mode</b> .....	9
<b>3 ETS (Equivalent Time Sampling)</b> .....	14
<b>4 Streaming mode</b> .....	16
<b>5 Retrieving stored data</b> .....	17
<b>6 Oversampling</b> .....	17
<b>7 Timebases</b> .....	18
<b>8 Combining several oscilloscopes</b> .....	19
<b>9 API functions</b> .....	20
<b>1 ps6000BlockReady</b> .....	21
<b>2 ps6000CloseUnit</b> .....	22
<b>3 ps6000DataReady</b> .....	23
<b>4 ps6000EnumerateUnits</b> .....	24
<b>5 ps6000FlashLed</b> .....	25
<b>6 ps6000GetAnalogueOffset</b> .....	26
<b>7 ps6000GetMaxDownSampleRatio</b> .....	27
<b>8 ps6000GetNoOfCaptures</b> .....	28
<b>9 ps6000GetStreamingLatestValues</b> .....	29
<b>10 ps6000GetTimebase</b> .....	30
<b>11 ps6000GetTimebase2</b> .....	31
<b>12 ps6000GetTriggerTimeOffset</b> .....	32
<b>13 ps6000GetTriggerTimeOffset64</b> .....	33
<b>14 ps6000GetUnitInfo</b> .....	34
<b>15 ps6000GetValues</b> .....	35
<b>16 ps6000GetValuesAsync</b> .....	37
<b>17 ps6000GetValuesBulk</b> .....	38
<b>18 ps6000GetValuesBulkAsync</b> .....	39
<b>19 ps6000GetValuesOverlapped</b> .....	40
<b>20 ps6000GetValuesOverlappedBulk</b> .....	41
<b>21 ps6000GetValuesTriggerTimeOffsetBulk</b> .....	42
<b>22 ps6000GetValuesTriggerTimeOffsetBulk64</b> .....	43
<b>23 ps6000IsReady</b> .....	44

24	ps6000IsTriggerOrPulseWidthQualifierEnabled	45
25	ps6000MemorySegments	46
26	ps6000NoOfStreamingValues	47
27	ps6000OpenUnit	48
28	ps6000OpenUnitAsync	49
29	ps6000OpenUnitProgress	50
30	ps6000RunBlock	51
31	ps6000RunStreaming	53
32	ps6000SetChannel	55
33	ps6000SetDataBuffer	57
34	ps6000SetDataBufferBulk	58
35	ps6000SetDataBuffers	59
36	ps6000SetDataBuffersBulk	60
37	ps6000SetEts	61
38	ps6000SetEtsTimeBuffer	62
39	ps6000SetEtsTimeBuffers	63
40	ps6000SetExternalClock	64
41	ps6000SetNoOfCaptures	65
42	ps6000SetPulseWidthQualifier	66
43	ps6000SetSigGenArbitrary	69
44	ps6000SetSigGenBuiltIn	72
45	ps6000SetSimpleTrigger	74
46	ps6000SetTriggerChannelConditions	75
47	ps6000SetTriggerChannelDirections	77
48	ps6000SetTriggerChannelProperties	78
49	ps6000SetTriggerDelay	80
50	ps6000SetWaveformLimiter	81
51	ps6000SigGenSoftwareControl	82
52	ps6000Stop	83
53	ps6000StreamingReady	84
10	Programming examples	85
1	C	85
2	Visual Basic	85
3	Excel	85
4	LabView	86
11	Driver status codes	88
12	Enumerated types and constants	91
13	Numeric data types	94
4	Glossary	95
	Index	97

# 1 Introduction

## 1.1 Welcome

The **PicoScope 6000 Series** of oscilloscopes from Pico Technology is a range of compact, high-resolution units designed to replace traditional bench-top oscilloscopes and digitizers.



This manual explains how to use the Application Programming Interface (API) for the PicoScope 6000 Series scopes. For more information on the hardware, see the [PicoScope 6000 Series User's Guide](#) available as a separate PDF.

## 1.2 Software licence conditions

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

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

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

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

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

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

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

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

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

**Upgrades.** We provide upgrades, free of charge, from our web site at [www.picotech.com](http://www.picotech.com). We reserve the right to charge for updates or replacements sent out on physical media.

### 1.3 Trademarks

**Windows**, **Excel** and **Visual Basic** are registered trademarks or trademarks of Microsoft Corporation in the USA and other countries.

**LabView** is a registered trademark of National Instruments Corporation.

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

### 1.4 Company details

**Address:** Pico Technology  
James House  
Colmworth Business Park  
ST NEOTS  
Cambridgeshire  
PE19 8YP  
United Kingdom

**Phone:** +44 (0) 1480 396 395

**Fax:** +44 (0) 1480 396 296

**Email:**  
Technical Support: [support@picotech.com](mailto:support@picotech.com)  
Sales: [sales@picotech.com](mailto:sales@picotech.com)

**Web site:** [www.picotech.com](http://www.picotech.com)

## 2 Product information

### 2.1 System requirements

#### Using with PicoScope for Windows

To ensure that your [PicoScope 6000 Series](#) PC Oscilloscope operates correctly, you must have a computer with at least the minimum system requirements to run one of the supported operating systems, as shown in the following table. The performance of the oscilloscope will be better with a more powerful PC, and will benefit from a multi-core processor.

**Please note the PicoScope software is not installed as part of the SDK.**

Item	Absolute minimum	Recommended minimum	Recommended full specification
<b>Operating system</b>	Windows XP SP2 or later Windows Vista Windows 7		
	32 bit and 64* bit versions supported		
<b>Processor</b>	As required by Windows	300 MHz	1 GHz
<b>Memory</b>		256 MB	512 MB
<b>Free disk space**</b>		1.5 GB	2 GB
<b>Ports</b>	USB 1.1 compliant port	USB 2.0 compliant port	

\* While the driver will run on a 64 bit operating system, the driver itself is 32-bit and therefore runs as 32-bit.

\*\* 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 or later), Windows Vista and Windows 7. System specifications for Windows are the same as under "Using with PicoScope for Windows", above.

## 2.2 Installation instructions

**IMPORTANT**  
**Install the PicoScope software before connecting your [PicoScope 6000 Series](#) oscilloscope to the PC for the first time. This will ensure that Windows correctly recognizes the oscilloscope.**

### Procedure

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

### Checking the installation

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

### Moving your PicoScope oscilloscope to another USB port

#### ● Windows XP

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

#### ● Windows Vista/7

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 6000 series oscilloscope" message. The oscilloscope is then ready for use.

## 3 Programming with the PicoScope 6000 Series

The `ps6000.dll` dynamic link library in your PicoScope installation directory allows you to program a [PicoScope 6000 Series oscilloscope](#) using standard C [function calls](#).

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

- [Open](#) the scope unit.
- Set up the input channels with the required [voltage ranges](#) and [coupling 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 [sample programs](#) are installed with your PicoScope software. These show how to use the functions of the driver software in each of the modes available.

### 3.1 Driver

Your application will communicate with a PicoScope 6000 API driver called `ps6000.dll`. The driver exports the PicoScope 6000 [function definitions](#) in standard C format, but this does not limit you to programming in C. You can use the API with any programming language that supports standard C calls.

The API driver depends on a kernel driver, `picopp.sys`, which works with Windows XP, Windows Vista and Windows 7. There is a further low-level driver called `WinUsb.sys`. These low-level drivers are installed by the PicoScope 6 software when you plug the [PicoScope 6000 Series](#) oscilloscope into the computer for the first time. Your application does not call these drivers directly.

### 3.2 System requirements

#### General requirements

See [System Requirements](#).

#### USB

The PicoScope 6000 driver offers [three different methods](#) of recording data, all of which support both USB 1.1 and USB 2.0, although the fastest transfer rates are achieved using USB 2.0.

### 3.3 Voltage ranges

You can set a device input channel to any voltage range from  $\pm 50$  mV to  $\pm 20$  V with the [ps6000SetChannel](#) function. Each sample is scaled to 16 bits so that the values returned to your application are as follows:

Constant	Voltage	Value returned	
		decimal	hex
PS6000_MIN_VALUE	minimum	-32 512	8100
	zero	0	0000
PS6000_MAX_VALUE	maximum	32 512	7F00

### 3.4 Triggering

PicoScope 6000 Series 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 6000 trigger functions [ps6000SetTriggerChannelConditions](#), [ps6000SetTriggerChannelDirections](#) and [ps6000SetTriggerChannelProperties](#). 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.

### 3.5 Sampling modes

[PicoScope 6000 Series 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 downsampling 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.
- **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 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). Downsampling and triggering are supported in this mode.

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

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

In block mode, you can also poll the driver instead of using a callback.

### 3.5.1 Block mode

In **block mode**, the computer prompts a [PicoScope 6000 series](#) 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.

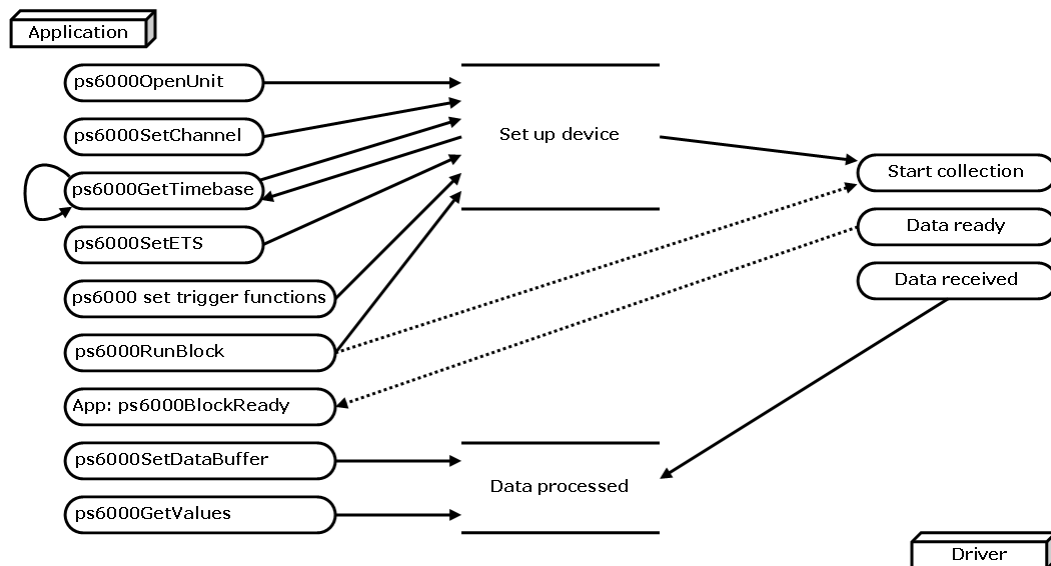
- **Block size.** The maximum number of values depends upon the size of the oscilloscope's memory. The memory buffer is shared between the enabled channels, so if two channels are enabled, each receives half the memory. These features are handled transparently by the driver. The block size also depends on the number of memory segments in use (see [ps6000MemorySegments](#)).
- **Sampling rate.** A PicoScope 6000 Series 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 6000 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 [ps6000RunBlock](#), [ps6000Stop](#) and [ps6000GetValues](#).
- **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 [ps6000MemorySegments](#).
- **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.

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

## 3.5.1.1 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 [ps6000OpenUnit](#).
2. Select channel ranges and AC/DC coupling using [ps6000SetChannel](#).
3. Using [ps6000GetTimebase](#), select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions [ps6000SetTriggerChannelConditions](#), [ps6000SetTriggerChannelDirections](#) and [ps6000SetTriggerChannelProperties](#) to set up the trigger if required.
5. Start the oscilloscope running using [ps6000RunBlock](#).
6. Wait until the oscilloscope is ready using the [ps6000BlockReady](#) callback (or poll using [ps6000IsReady](#)).
7. Use [ps6000SetDataBuffer](#) to tell the driver where your memory buffer is.
8. Transfer the block of data from the oscilloscope using [ps6000GetValues](#).
9. Display the data.
10. Repeat steps 5 to 9.
11. Stop the oscilloscope using [ps6000Stop](#).



12. Request new views of stored data using different downsampling parameters: see [Retrieving stored data](#).

## 3.5.1.2 Asynchronous calls in block mode

The [ps6000GetValues](#) function may take a long time to complete if a large amount of data is being collected. For example, it can take 6 seconds to retrieve the full 1 billion samples from a PicoScope 6403. To avoid hanging the calling thread, it is possible to call [ps6000GetValuesAsync](#) instead. This immediately returns control to the calling thread, which then has the option of waiting for the data or calling [ps6000Stop](#) to abort the operation.

### 3.5.2 Rapid block mode

In normal [block mode](#), the PicoScope 6000 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 less than 1 microsecond.

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

#### 3.5.2.1 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 [ps6000OpenUnit](#).
2. Select channel ranges and AC/DC coupling using [ps6000SetChannel](#).
3. Using [ps6000GetTimebase](#), select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions [ps6000SetTriggerChannelConditions](#), [ps6000SetTriggerChannelDirections](#) and [ps6000SetTriggerChannelProperties](#) 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 [ps6000MemorySegments](#). Use [ps6000SetNoOfCaptures](#) before each run to specify the number of waveforms to capture.
6. Start the oscilloscope running using [ps6000RunBlock](#).
7. Wait until the oscilloscope is ready using the [ps6000BlockReady](#) callback.
8. Use [ps6000SetDataBufferBulk](#) to tell the driver where your memory buffers are.
9. Transfer the blocks of data from the oscilloscope using [ps6000GetValuesBulk](#).
10. Retrieve the time offset for each data segment using [ps6000GetValuesTriggerTimeOffsetBulk64](#).
11. Display the data.
12. Repeat steps 6 to 11 if necessary.
13. Stop the oscilloscope using [ps6000Stop](#).

##### With aggregation

To use rapid block mode with aggregation, follow steps 1 to 7 above and then proceed as follows:

- 8a. Call [ps6000SetDataBuffersBulk](#) to set up one pair of buffers for every waveform segment required.
- 9a. Call [ps6000GetValuesBulk](#) for each pair of buffers.
- 10a. Retrieve the time offset for each data segment using [ps6000GetValuesTriggerTimeOffsetBulk64](#).

Continue from step 11 above.

## 3.5.2.2 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
ps6000SetNoOfCaptures (handle, 100);

pParameter = false;
ps6000RunBlock
(
    handle,
    0,                // noOfPreTriggerSamples
    10000,           // noOfPostTriggerSamples
    1,               // timebase to be used
    1,               // oversample
    &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 (int i = 0; i < 10; i++)
{
    for (int c = PS6000_CHANNEL_A; c <= PS6000_CHANNEL_D; c++)
    {
        ps6000SetDataBufferBulk
        (
            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.

```
ps6000GetValuesBulk
(
    handle,
    &noOfSamples,           // set to MAX_SAMPLES on entering the
    function               // function
    10,                    // fromSegmentIndex
    19,                    // toSegmentIndex
    1,                     // downsampling ratio
    PS6000_RATIO_MODE_NONE, // downsampling ratio mode
    overflow                // an array of size 10 shorts
)
```

Comments: the number of samples could be up to `noOfPreTriggerSamples + noOfPostTriggerSamples`, the values set in `ps6000RunBlock`. The samples are always returned from the first sample taken, unlike the `ps6000GetValues` function which allows the sample index to be set. This function does not support aggregation. The above segments start at 10 and finish at 19 inclusive. It is possible for the `fromSegmentIndex` to wrap around to the `toSegmentIndex`, by setting the `fromSegmentIndex` to 98 and the `toSegmentIndex` to 7.

```
ps6000GetValuesTriggerTimeOffsetBulk64
(
    handle,
    times,
    timeUnits,
    10,
    19
)
```

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

## 3.5.2.3 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
ps6000SetNoOfCaptures (handle, 100);

pParameter = false;
ps6000RunBlock
(
    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 = PS6000_CHANNEL_A; c <= PS6000_CHANNEL_D; c++)
{
    ps6000SetDataBuffers
    (
        handle,
        c,
        &bufferMax[c],
        &bufferMin[c]
        MAX_SAMPLES,
        PS6000_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.

```
for (int segment = 10; segment < 20; segment++)
{
    ps6000GetValues
    (
        handle,
        0,
        &noOfSamples, // set to MAX_SAMPLES on entering
        1000,
        &downSampleRatioMode, //set to RATIO_MODE_AGGREGATE
        index,
        overflow
    );

    ps6000GetTriggerTimeOffset64
    (
        handle,
        &time,
        &timeUnits,
        index
    )
}
```

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

### 3.5.3 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 [ps6000SetTrigger](#) and [ps6000SetEts](#) functions.

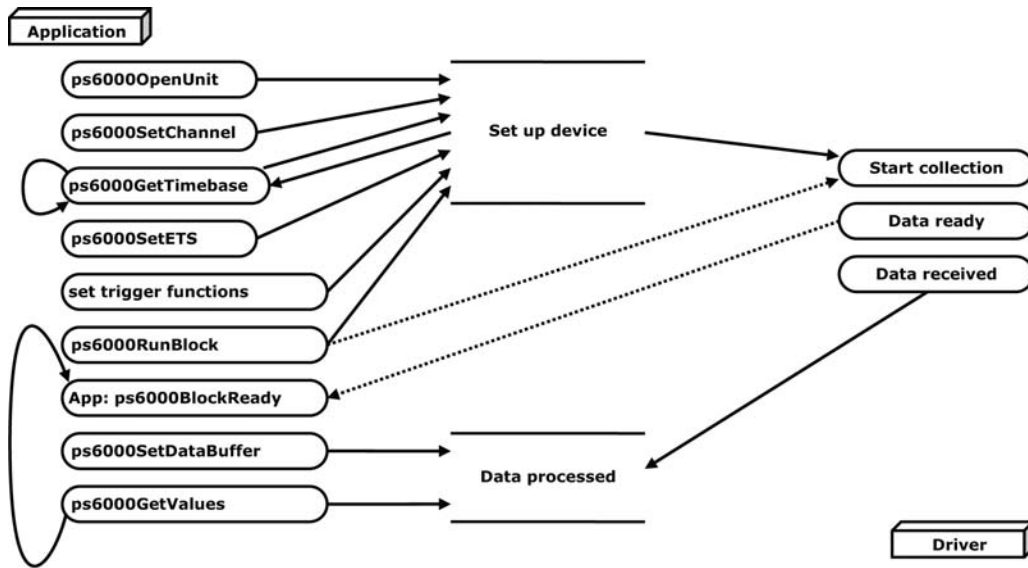
- **Overview.** ETS works by capturing several cycles of a repetitive waveform, then combining them to produce a composite waveform that has a higher effective sampling rate than the individual captures. The scope hardware adds a short, variable delay, which is a small fraction of a single sampling interval, between each trigger event and the subsequent sample. This shifts each capture slightly in time so that the samples occur at slightly different times relative to those of the previous capture. The result is a larger set of samples spaced by a small fraction of the original sampling interval. The maximum effective sampling rates that can be achieved with this method are listed in the User's Guide for the scope device.
- **Trigger stability.** Because of the high sensitivity of ETS mode to small time differences, the trigger must be set up to provide a stable waveform that varies as little as possible from one capture to the next.
- **Callback.** ETS mode returns data to your application using the [ps6000BlockReady](#) callback function.

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

#### 3.5.3.1 Using ETS mode

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

1. Open the oscilloscope using [ps6000OpenUnit](#).
2. Select channel ranges and AC/DC coupling using [ps6000SetChannel](#).
3. Using [ps6000GetTimebase](#), select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions [ps6000SetTriggerChannelConditions](#), [ps6000SetTriggerChannelDirections](#) and [ps6000SetTriggerChannelProperties](#) to set up the trigger if required.
5. Start the oscilloscope running using [ps6000RunBlock](#).
6. Wait until the oscilloscope is ready using the [ps6000BlockReady](#) callback (or poll using [ps6000IsReady](#)).
7. Use [ps6000SetDataBuffer](#) to tell the driver where your memory buffer is.
8. Transfer the block of data from the oscilloscope using [ps6000GetValues](#).
9. Display the data.
10. While you want to collect updated captures, repeat steps 6-9.
11. Stop the oscilloscope using [ps6000Stop](#).
12. Repeat steps 5 to 11.



### 3.5.4 Streaming mode

**Streaming mode** can capture data without the gaps that occur between blocks when using [block mode](#). It can transfer data to the PC at speeds of at least 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.

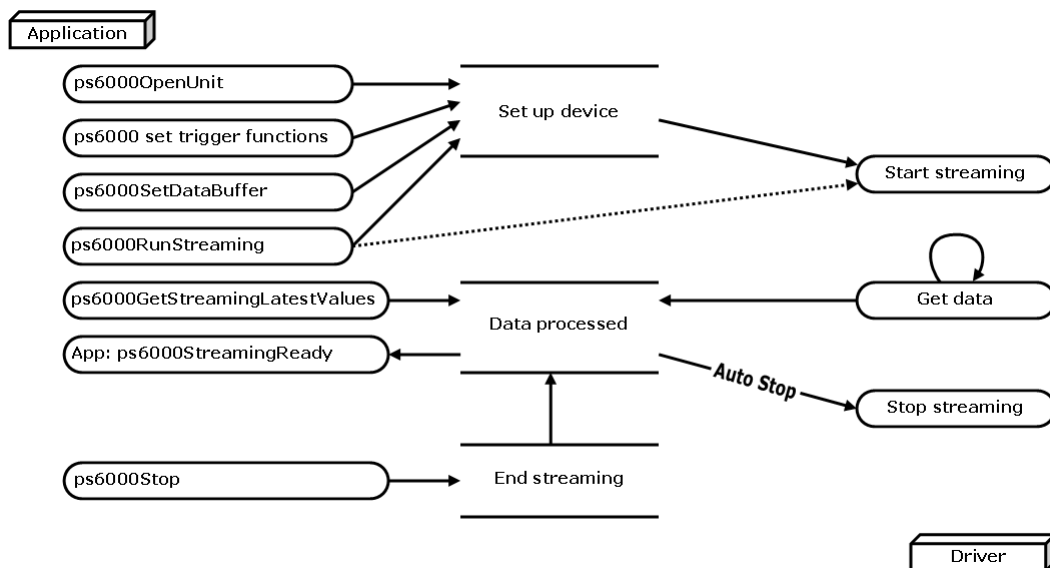
- **Aggregation.** The driver returns [aggregated readings](#) while the device is streaming. If aggregation is set to 1 then only one buffer is returned per channel. When aggregation is set above 1 then two buffers (maximum and minimum) per channel are returned.
- **Memory segmentation.** The memory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

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

#### 3.5.4.1 Using streaming mode

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

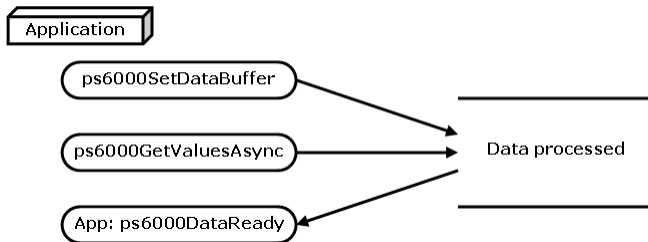
1. Open the oscilloscope using [ps6000OpenUnit](#).
2. Select channels, ranges and AC/DC coupling using [ps6000SetChannel](#).
3. Use the trigger setup functions [ps6000SetTriggerChannelConditions](#), [ps6000SetTriggerChannelDirections](#) and [ps6000SetTriggerChannelProperties](#) to set up the trigger if required.
4. Call [ps6000SetDataBuffer](#) to tell the driver where your data buffer is.
5. Set up aggregation and start the oscilloscope running using [ps6000RunStreaming](#).
6. Call [ps6000GetStreamingLatestValues](#) 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 [ps6000Stop](#), even if Auto Stop is enabled.



9. Request new views of stored data using different downsampling parameters: see [Retrieving stored data](#).

### 3.5.5 Retrieving stored data

You can collect data from the PicoScope 6000 driver with a different [downsampling](#) factor when [ps6000RunBlock](#) or [ps6000RunStreaming](#) has already been called and has successfully captured all the data. Use [ps6000GetValuesAsync](#).



## 3.6 Oversampling

*Note: This feature is provided for backward-compatibility only. The same effect can be obtained more efficiently with the PicoScope 6000 Series using the hardware averaging feature (see [Downsampling modes](#)).*

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 wideband noise (strictly speaking, *Gaussian noise*), this technique can increase the effective [vertical resolution](#) of the oscilloscope by  $n$  bits, where  $n$  is given approximately by the equation below:

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

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

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

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.

<b>Applicability</b>	Available in <a href="#">block mode</a> only.  Cannot be used at the same time as <a href="#">downsampling</a> .
----------------------	--

### 3.7 Timebases

The API allows you to select any of  $2^{32}$  different timebases based on a maximum sampling rate of 5 GHz. The timebases allow slow enough sampling in block mode to overlap the streaming sample intervals, so that you can make a smooth transition between block mode and streaming mode.

<b>timebase</b>	<b>sample interval formula</b>	<b>sample interval examples</b>
0 to 4	$2^{\text{timebase}} / 5,000,000,000$	0 => 200 ps 1 => 400 ps 2 => 800 ps 3 => 1.6 ns 4 => 3.2 ns
5 to $2^{32}-1$	$(\text{timebase} - 4) / 156,250,000$	5 => 6.4 ns ... $2^{32}-1$ => ~ 6.87 s
<b>Applicability</b>	Use <a href="#">ps6000GetTimebase</a> API call.	

### 3.8 Combining several oscilloscopes

It is possible to collect data using up to 64 [PicoScope 6000 Series oscilloscopes](#) at the same time, depending on the capabilities of the PC. Each oscilloscope must be connected to a separate USB port. The [ps6000OpenUnit](#) 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 ps6000BlockReady(...)
// define callback function specific to application

handle1 = ps6000OpenUnit()
handle2 = ps6000OpenUnit()

ps6000SetChannel(handle1)
// set up unit 1
ps6000RunBlock(handle1)

ps6000SetChannel(handle2)
// set up unit 2
ps6000RunBlock(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: an [external clock](#) may be fed into the AUX input to provide some degree of synchronisation between multiple oscilloscopes.

### 3.9 API functions

The PicoScope 6000 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.

<a href="#">ps6000BlockReady</a>	indicate when block-mode data ready
<a href="#">ps6000CloseUnit</a>	close a scope device
<a href="#">ps6000DataReady</a>	indicate when post-collection data ready
<a href="#">ps6000EnumerateUnits</a>	find all connected oscilloscopes
<a href="#">ps6000FlashLed</a>	flash the front-panel LED
<a href="#">ps6000GetAnalyseOffset</a>	get min/max allowable analogue offset
<a href="#">ps6000GetMaxDownSampleRatio</a>	find out aggregation ratio for data
<a href="#">ps6000GetStreamingLatestValues</a>	get streaming data while scope is running
<a href="#">ps6000GetTimebase</a>	find out what timebases are available
<a href="#">ps6000GetTimebase2</a>	find out what timebases are available
<a href="#">ps6000GetTriggerTimeOffset</a>	find out when trigger occurred (32-bit)
<a href="#">ps6000GetTriggerTimeOffset64</a>	find out when trigger occurred (64-bit)
<a href="#">ps6000GetUnitInfo</a>	read information about scope device
<a href="#">ps6000GetValues</a>	get block-mode data with callback
<a href="#">ps6000GetValuesAsync</a>	get streaming data with callback
<a href="#">ps6000GetValuesBulk</a>	get data in rapid block mode
<a href="#">ps6000GetValuesBulkAsync</a>	get data in rapid block mode using callback
<a href="#">ps6000GetValuesOverlapped</a>	set up data collection ahead of capture
<a href="#">ps6000GetValuesOverlappedBulk</a>	set up data collection in rapid block mode
<a href="#">ps6000GetValuesTriggerTimeOffsetBulk</a>	get rapid-block waveform timings (32-bit)
<a href="#">ps6000GetValuesTriggerTimeOffsetBulk64</a>	get rapid-block waveform timings (64-bit)
<a href="#">ps6000IsReady</a>	poll driver in block mode
<a href="#">ps6000IsTriggerOrPulseWidthQualifierEnabled</a>	find out whether trigger is enabled
<a href="#">ps6000MemorySegments</a>	divide scope memory into segments
<a href="#">ps6000NoOfStreamingValues</a>	get number of samples in streaming mode
<a href="#">ps6000OpenUnit</a>	open a scope device
<a href="#">ps6000OpenUnitAsync</a>	open a scope device without waiting
<a href="#">ps6000OpenUnitProgress</a>	check progress of OpenUnit call
<a href="#">ps6000RunBlock</a>	start block mode
<a href="#">ps6000RunStreaming</a>	start streaming mode
<a href="#">ps6000SetChannel</a>	set up input channels
<a href="#">ps6000SetDataBuffer</a>	register data buffer with driver
<a href="#">ps6000SetDataBufferBulk</a>	set the buffers for each waveform
<a href="#">ps6000SetDataBuffers</a>	register aggregated data buffers with driver
<a href="#">ps6000SetDataBuffersBulk</a>	register data buffers for rapid block mode
<a href="#">ps6000SetEts</a>	set up equivalent-time sampling
<a href="#">ps6000SetEtsTimeBuffer</a>	set up buffer for ETS timings (64-bit)
<a href="#">ps6000SetEtsTimeBuffers</a>	set up buffer for ETS timings (32-bit)
<a href="#">ps6000SetExternalClock</a>	set AUX input to receive external clock
<a href="#">ps6000SetNoOfCaptures</a>	set number of captures to collect in one run
<a href="#">ps6000SetPulseWidthQualifier</a>	set up pulse width triggering
<a href="#">ps6000SetSigGenArbitrary</a>	set up arbitrary waveform generator
<a href="#">ps6000SetSigGenBuiltIn</a>	set up standard signal generator
<a href="#">ps6000SetSimpleTrigger</a>	set up level triggers only
<a href="#">ps6000SetTriggerChannelConditions</a>	specify which channels to trigger on
<a href="#">ps6000SetTriggerChannelDirections</a>	set up signal polarities for triggering
<a href="#">ps6000SetTriggerChannelProperties</a>	set up trigger thresholds
<a href="#">ps6000SetTriggerDelay</a>	set up post-trigger delay
<a href="#">ps6000SetWaveformLimiter</a>	limit rapid block transfer rate
<a href="#">ps6000SigGenSoftwareControl</a>	trigger the signal generator
<a href="#">ps6000Stop</a>	stop data capture
<a href="#">ps6000StreamingReady</a>	indicate when streaming-mode data ready

## 3.9.1 ps6000BlockReady

```
typedef void (CALLBACK *ps6000BlockReady)
(
    short        handle,
    PICO_STATUS  status,
    void         * pParameter
)
```

This [callback](#) function is part of your application. You register it with the PicoScope 6000 series driver using [ps6000RunBlock](#), and the driver calls it back when block-mode data is ready. You can then download the data using the [ps6000GetValues](#) function.

<b>Applicability</b>	<a href="#">Block mode</a> only
<b>Arguments</b>	<p><code>handle</code>, the handle of the device returning the samples.</p> <p><code>status</code>, indicates whether an error occurred during collection of the data.</p> <p><code>pParameter</code>, a void pointer passed from <a href="#">ps6000RunBlock</a>. Your callback function can write to this location to send any data, such as a status flag, back to your application.</p>
<b>Returns</b>	nothing

```
3.9.2 ps6000CloseUnit
      PICO_STATUS ps6000CloseUnit
      (
        short handle
      )
```

This function shuts down a PicoScope 6000 oscilloscope.

<b>Applicability</b>	All modes
<b>Arguments</b>	<code>handle</code> , the handle, returned by <a href="#">ps6000OpenUnit</a> , of the scope device to be closed.
<b>Returns</b>	PICO_OK PICO_HANDLE_INVALID PICO_USER_CALLBACK PICO_DRIVER_FUNCTION

## 3.9.3 ps6000DataReady

```
typedef void (CALLBACK *ps6000DataReady)
(
    short          handle,
    PICO_STATUS    status,
    unsigned long  noOfSamples,
    short          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 [ps6000GetValuesAsync](#), and the driver calls your function back when the data is ready.

<b>Applicability</b>	All modes
<b>Arguments</b>	<p><code>handle</code>, the handle of the device returning the samples.</p> <p><code>status</code>, a <a href="#">PICO_STATUS</a> code returned by the driver.</p> <p><code>noOfSamples</code>, the number of samples collected.</p> <p><code>overflow</code>, 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.</p> <p><code>pParameter</code>, a void pointer passed from <a href="#">ps6000GetValuesAsync</a>. The callback function can write to this location to send any data, such as a status flag, back to the application. The data type is defined by the application programmer.</p>
<b>Returns</b>	nothing

## 3.9.4 ps6000EnumerateUnits

```
PICO_STATUS ps6000EnumerateUnits
(
    short * count,
    char * serials,
    short * serialLth
)
```

This function counts the number of PicoScope 6000 units connected to the computer, and returns a list of serial numbers as a string.

<b>Applicability</b>	All modes
<b>Arguments</b>	<p>* <code>count</code>, on exit, the number of PicoScope 6000 units found</p> <p>* <code>serials</code>, on exit, a list of serial numbers separated by commas and terminated by a final null. Example: AQ005/139,VDR61/356,ZOR14/107. Can be NULL on entry if serial numbers are not required.</p> <p>* <code>serialLth</code>, on entry, the length of the char buffer pointed to by <code>serials</code>; on exit, the length of the string written to <code>serials</code></p>
<b>Returns</b>	PICO_OK PICO_BUSY PICO_NULL_PARAMETER PICO_FW_FAIL PICO_CONFIG_FAIL PICO_MEMORY_FAIL PICO_ANALOG_BOARD PICO_CONFIG_FAIL_AWG PICO_INITIALISE_FPGA

## 3.9.5 ps6000FlashLed

```
PICO_STATUS ps6000FlashLed
(
    short handle,
    short start
)
```

This function flashes the LED on the front of the scope without blocking the calling thread. Calls to [ps6000RunStreaming](#) and [ps6000RunBlock](#) 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.

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

## 3.9.6 ps6000GetAnalogueOffset

```

PICO_STATUS ps6000GetAnalogueOffset
(
    short          handle,
    PS6000_RANGE, range
    PS6000_COUPLING coupling
    float          * maximumVoltage,
    float          * minimumVoltage
)

```

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

<b>Applicability</b>	6402/6403/6404 models only
<b>Arguments</b>	<p>handle, the value returned from opening the device.</p> <p>range, the voltage range to be used when gathering the min and max information.</p> <p>coupling, the type of AC/DC coupling used.</p> <p>* maximumVoltage, a pointer to a float, an out parameter set to the maximum voltage allowed for the range, may be NULL.</p> <p>* minimumVoltage, a pointer to a float, an out parameter set to the minimum voltage allowed for the range, may be NULL.</p> <p>If both maximumVoltage and minimumVoltage are set to NULL the driver will return PICO_NULL_PARAMETER.</p>
<b>Returns</b>	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_INVALID_VOLTAGE_RANGE PICO_NULL_PARAMETER

## 3.9.7 ps6000GetMaxDownSampleRatio

```
PICO_STATUS ps6000GetMaxDownSampleRatio
(
    short          handle,
    unsigned long  noOfUnaggregatedSamples,
    unsigned long  * maxDownSampleRatio,
    PS6000_RATIO_MODE downSampleRatioMode,
    unsigned long  segmentIndex
)
```

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

<b>Applicability</b>	All modes
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>noOfUnaggregatedSamples</code>, the number of unprocessed samples to be downsampled</p> <p><code>maxDownSampleRatio</code>: the maximum possible downsampling ratio</p> <p><code>downSampleRatioMode</code>: the downsampling mode. See <a href="#">ps6000GetValues</a>.</p> <p><code>segmentIndex</code>, the <a href="#">memory segment</a> where the data is stored</p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_NO_SAMPLES_AVAILABLE</p> <p>PICO_NULL_PARAMETER</p> <p>PICO_INVALID_PARAMETER</p> <p>PICO_SEGMENT_OUT_OF_RANGE</p> <p>PICO_TOO_MANY_SAMPLES</p>

```

3.9.8 ps6000GetNoOfCaptures
      PICO_STATUS ps6000GetNoOfCaptures
      (
        short      handle,
        unsigned long * nCaptures
      )

```

This function finds out how many captures are available after [ps6000RunBlock](#) has been called when either the collection completed or the collection of waveforms was interrupted by calling [ps6000Stop](#). The returned value (`nCaptures`) can then be used to iterate through the number of segments using [ps6000GetValues](#), or in a single call to [ps6000GetValuesBulk](#) where it is used to calculate the `toSegmentIndex` parameter.

<b>Applicability</b>	All modes
<b>Arguments</b>	<p><code>handle</code>: handle of the required device.</p> <p><code>nCaptures</code>, output: the number of available captures that has been collected from calling <a href="#">ps6000RunBlock</a>.</p>
<b>Returns</b>	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_TOO_MANY_SAMPLES

## 3.9.9 ps6000GetStreamingLatestValues

```
PICO_STATUS ps6000GetStreamingLatestValues
(
    short          handle,
    ps6000StreamingReady lpPs6000Ready,
    void          * pParameter
)
```

This function instructs the driver to return the next block of values to your [ps6000StreamingReady](#) callback function. You must have previously called [ps6000RunStreaming](#) beforehand to set up [streaming](#).

<b>Applicability</b>	<a href="#">Streaming</a> mode only
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device.</p> <p><code>lpPs6000Ready</code>, a pointer to your <a href="#">ps6000StreamingReady</a> callback function.</p> <p><code>pParameter</code>, a void pointer that will be passed to the <a href="#">ps6000StreamingReady</a> callback function. The callback function may optionally use this pointer to return information to the application.</p>
<b>Returns</b>	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_INVALID_CALL PICO_BUSY PICO_NOT_RESPONDING PICO_DRIVER_FUNCTION PICO_STARTINDEX_INVALID

## 3.9.10 ps6000GetTimebase

```

PICO_STATUS ps6000GetTimebase
(
    short          handle,
    unsigned long  timebase,
    unsigned long  noSamples,
    long           * timeIntervalNanoseconds,
    short          oversample,
    unsigned long  * maxSamples
    unsigned long  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 [ps6000SetChannel](#).

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 [ps6000GetTimebase2](#) instead.

To use [ps6000GetTimebase](#) or [ps6000GetTimebase2](#), first estimate the timebase number that you require using the information in the [timebase guide](#). Pass this timebase to the `GetTimebase` function and check the returned `timeIntervalNanoseconds` argument. If necessary, repeat until you obtain the time interval that you need.

<b>Applicability</b>	All modes
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device.</p> <p><code>timebase</code>, <a href="#">see timebase guide</a></p> <p><code>noSamples</code>, the number of samples required. This value is used to calculate the most suitable time unit to use.</p> <p><code>timeIntervalNanoseconds</code>, on exit, the time interval between readings at the selected timebase. Use NULL if not required.</p> <p><code>oversample</code>, the amount of <a href="#">oversample</a> required. Range: 0 to <a href="#">PS6000_MAX_OVERSAMPLE_8BIT</a>.</p> <p><code>maxSamples</code>, on exit, the maximum number of samples available. The result may vary depending on the number of channels enabled, the timebase chosen and the oversample selected. Use NULL if not required.</p> <p><code>segmentIndex</code>, the index of the memory segment to use.</p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_TOO_MANY_SAMPLES</p> <p>PICO_INVALID_CHANNEL</p> <p>PICO_INVALID_TIMEBASE</p> <p>PICO_INVALID_PARAMETER</p> <p>PICO_SEGMENT_OUT_OF_RANGE</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.11 ps6000GetTimebase2

```
PICO_STATUS ps6000GetTimebase2
(
    short          handle,
    unsigned long  timebase,
    unsigned long  noSamples,
    float          * timeIntervalNanoseconds,
    short          oversample,
    unsigned long  * maxSamples
    unsigned long  segmentIndex
)
```

This function is an upgraded version of [ps6000GetTimebase](#), and returns the time interval as a `float` rather than a `long`. This allows it to return sub-nanosecond time intervals. See [ps6000GetTimebase](#) for a full description.

<b>Applicability</b>	All modes
<b>Arguments</b>	<code>timeIntervalNanoseconds</code> , a pointer to the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here.  All other arguments: see <a href="#">ps6000GetTimebase</a> .
<b>Returns</b>	See <a href="#">ps6000GetTimebase</a> .

## 3.9.12 ps6000GetTriggerTimeOffset

```
PICO_STATUS ps6000GetTriggerTimeOffset
(
    short          handle
    unsigned long  * timeUpper
    unsigned long  * timeLower
    PS6000_TIME_UNITS * timeUnits
    unsigned long  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, [ps6000GetTriggerTimeOffset64](#), is also available.

<b>Applicability</b>	<a href="#">Block mode</a> , <a href="#">rapid block mode</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>timeUpper</code>, on exit, the upper 32 bits of the time at which the trigger point occurred</p> <p><code>timeLower</code>, on exit, the lower 32 bits of the time at which the trigger point occurred</p> <p><code>timeUnits</code>, returns the time units in which <code>timeUpper</code> and <code>timeLower</code> are measured. The allowable values are: -</p> <ul style="list-style-type: none"> <li><a href="#">PS6000_FS</a></li> <li><a href="#">PS6000_PS</a></li> <li><a href="#">PS6000_NS</a></li> <li><a href="#">PS6000_US</a></li> <li><a href="#">PS6000_MS</a></li> <li><a href="#">PS6000_S</a></li> </ul> <p><code>segmentIndex</code>, the number of the <a href="#">memory segment</a> for which the information is required.</p>
<b>Returns</b>	<ul style="list-style-type: none"> <li>PICO_OK</li> <li>PICO_INVALID_HANDLE</li> <li>PICO_DEVICE_SAMPLING</li> <li>PICO_SEGMENT_OUT_OF_RANGE</li> <li>PICO_NULL_PARAMETER</li> <li>PICO_NO_SAMPLES_AVAILABLE</li> <li>PICO_DRIVER_FUNCTION</li> </ul>

## 3.9.13 ps6000GetTriggerTimeOffset64

```
PICO_STATUS ps6000GetTriggerTimeOffset64
(
    short          handle,
    __int64       * time,
    PS6000_TIME_UNITS * timeUnits,
    unsigned long 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, [ps6000GetTriggerTimeOffset](#), is also available.

<b>Applicability</b>	<a href="#">Block mode</a> , <a href="#">rapid block mode</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>time</code>, on exit, the time at which the trigger point occurred</p> <p><code>timeUnits</code>, on exit, the time units in which time is measured. The possible values are: -</p> <ul style="list-style-type: none"> <li><a href="#">PS6000_FS</a></li> <li><a href="#">PS6000_PS</a></li> <li><a href="#">PS6000_NS</a></li> <li><a href="#">PS6000_US</a></li> <li><a href="#">PS6000_MS</a></li> <li><a href="#">PS6000_S</a></li> </ul> <p><code>segmentIndex</code>, the number of the <a href="#">memory segment</a> for which the information is required</p>
<b>Returns</b>	PICO_OK PICO_INVALID_HANDLE PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_DRIVER_FUNCTION

## 3.9.14 ps6000GetUnitInfo

```
PICO_STATUS ps6000GetUnitInfo
(
    short    handle,
    char     * string,
    short    stringLength,
    short    * requiredSize
    PICO_INFO info
)
```

This function retrieves information about the specified oscilloscope. If the device fails to open, only the driver version and error code are available to explain why the last open unit call failed.

<b>Applicability</b>	All modes
<b>Arguments</b>	<p><code>handle</code>, 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.</p> <p><code>string</code>, on exit, the unit information string selected specified by the <code>info</code> argument. If <code>string</code> is NULL, only <code>requiredSize</code> is returned.</p> <p><code>stringLength</code>, the maximum number of chars that may be written to <code>string</code>.</p> <p><code>requiredSize</code>, on exit, the required length of the <code>string</code> array.</p> <p><code>info</code>, a number specifying what information is required. The possible values are listed in the table below.</p>
<b>Returns</b>	<p>PICO_OK  PICO_INVALID_HANDLE  PICO_NULL_PARAMETER  PICO_INVALID_INFO  PICO_INFO_UNAVAILABLE  PICO_DRIVER_FUNCTION</p>

<code>info</code>		<b>Example</b>
0	PICO_DRIVER_VERSION - Version number of PicoScope 6000 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 - Model number of device	6403
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

## 3.9.15 ps6000GetValues

```
PICO_STATUS ps6000GetValues
(
    short          handle,
    unsigned long  startIndex,
    unsigned long  * noOfSamples,
    unsigned long  downSampleRatio,
    PS6000_RATIO_MODE downSampleRatioMode,
    unsigned long  segmentIndex,
    short          * 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.

<b>Applicability</b>	<a href="#">Block mode</a> , <a href="#">rapid block mode</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device.</p> <p><code>startIndex</code>, a zero-based index that indicates the start point for data collection. It is measured in sample intervals from the start of the buffer.</p> <p><code>noOfSamples</code>, on entry, the number of samples required. On exit, the actual number retrieved. The number of samples retrieved will not be more than the number requested, and the data retrieved always starts with the first sample captured.</p> <p><code>downSampleRatio</code>, the <a href="#">downsampling</a> factor that will be applied to the raw data.</p> <p><code>downSampleRatioMode</code>, which <a href="#">downsampling</a> mode to use. The available values are: -  <a href="#">PS6000_RATIO_MODE_NONE</a> (<code>downSampleRatio</code> is ignored)  <a href="#">PS6000_RATIO_MODE_AGGREGATE</a>  <a href="#">PS6000_RATIO_MODE_AVERAGE</a>  <a href="#">PS6000_RATIO_MODE_DECIMATE</a>            AGGREGATE, AVERAGE, and DECIMATE are single-bit constants that can be ORed to apply multiple downsampling modes to the same data.</p> <p><code>segmentIndex</code>, the zero-based number of the <a href="#">memory segment</a> where the data is stored.</p> <p><code>overflow</code>, 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.</p>

<u>Returns</u>	
	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
	PICO_RATIO_MODE_NOT_SUPPORTED
	PICO_DRIVER_FUNCTION

### 3.9.15.1 Downsampling modes

Various methods of data reduction, or **downsampling**, are possible with the PicoScope 6000 Series 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 [ps6000GetValues](#). The following modes are available:

PS6000_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.
PS6000_RATIO_MODE_AVERAGE	Reduces every block of $n$ values to a single value representing the average (arithmetic mean) of all the values.
PS6000_RATIO_MODE_DECIMATE	Reduces every block of $n$ values to just the first value in the block, discarding all the other values.

## 3.9.16 ps6000GetValuesAsync

```

PICO_STATUS ps6000GetValuesAsync
(
    short          handle,
    unsigned long  startIndex,
    unsigned long  noOfSamples,
    unsigned long  downSampleRatio,
    PS6000_RATIO_MODE downSampleRatioMode,
    unsigned long  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](#).

<b>Applicability</b>	<a href="#">Streaming mode</a> and <a href="#">block mode</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>startIndex</code>: see <a href="#">ps6000GetValues</a></p> <p><code>noOfSamples</code>: see <a href="#">ps6000GetValues</a></p> <p><code>downSampleRatio</code>: see <a href="#">ps6000GetValues</a></p> <p><code>downSampleRatioMode</code>: see <a href="#">ps6000GetValues</a></p> <p><code>segmentIndex</code>: see <a href="#">ps6000GetValues</a></p> <p><code>lpDataReady</code>, a pointer to the user-supplied function that will be called when the data is ready. This will be a <a href="#">ps6000DataReady</a> function for block-mode data or a <a href="#">ps6000StreamingReady</a> function for streaming-mode data.</p> <p><code>pParameter</code>, a void pointer that will be passed to the callback function. The data type is determined by the application.</p>
<b>Returns</b>	PICO_OK 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

## 3.9.17 ps6000GetValuesBulk

```
PICO_STATUS ps6000GetValuesBulk
(
    short          handle,
    unsigned long  * noOfSamples,
    unsigned long  fromSegmentIndex,
    unsigned long  toSegmentIndex,
    unsigned long  downSampleRatio,
    PS6000_RATIO_MODE downSampleRatioMode,
    short          * overflow
)
```

This function retrieves waveforms captured using [rapid block mode](#). The waveforms must have been collected sequentially and in the same run. This method of collection does not support [downsampling](#).

<b>Applicability</b>	<a href="#">Rapid block mode</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the device</p> <p>* <code>noOfSamples</code>, on entry, the number of samples required; on exit, the actual number retrieved. The number of samples retrieved will not be more than the number requested. The data retrieved always starts with the first sample captured.</p> <p><code>fromSegmentIndex</code>, the first segment from which the waveform should be retrieved</p> <p><code>toSegmentIndex</code>, the last segment from which the waveform should be retrieved</p> <p><code>downSampleRatio</code>: see <a href="#">ps6000GetValues</a></p> <p><code>downSampleRatioMode</code>: see <a href="#">ps6000GetValues</a></p> <p>* <code>overflow</code>, 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 <code>overflow</code> array, with <code>overflow[0]</code> containing the flags for the segment numbered <code>fromSegmentIndex</code> and the last element in the array containing the flags for the segment numbered <code>toSegmentIndex</code>. Each element in the array is a bit field as described under <a href="#">ps6000GetValues</a>.</p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_INVALID_PARAMETER</p> <p>PICO_SEGMENT_OUT_OF_RANGE</p> <p>PICO_NO_SAMPLES_AVAILABLE</p> <p>PICO_STARTINDEX_INVALID</p> <p>PICO_NOT_RESPONDING</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.18 ps6000GetValuesBulkAsync

```
PICO_STATUS ps6000GetValuesBulkAsync
(
    short          handle,
    unsigned long  startIndex,
    unsigned long  * noOfSamples,
    unsigned long  downSampleRatio,
    PS6000_RATIO_MODE downSampleRatioMode,
    unsigned long  fromSegmentIndex,
    unsigned long  toSegmentIndex,
    short          * overflow
)
```

This function retrieves more than one waveform at a time in [rapid block mode](#) after data collection has stopped. The waveforms must have been collected sequentially and in the same run. The data is returned using a [callback](#).

<b>Applicability</b>	<a href="#">Rapid block mode</a>
<b>Arguments</b>	<p>handle, the handle of the device</p> <p>startIndex: see <a href="#">ps6000GetValues</a></p> <p>* noOfSamples: see <a href="#">ps6000GetValues</a></p> <p>downSampleRatio: see <a href="#">ps6000GetValues</a></p> <p>downSampleRatioMode: see <a href="#">ps6000GetValues</a></p> <p>fromSegmentIndex: see <a href="#">ps6000GetValuesBulk</a></p> <p>toSegmentIndex: see <a href="#">ps6000GetValuesBulk</a></p> <p>overflow: see <a href="#">ps6000GetValuesBulk</a></p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_INVALID_PARAMETER</p> <p>PICO_SEGMENT_OUT_OF_RANGE</p> <p>PICO_NO_SAMPLES_AVAILABLE</p> <p>PICO_STARTINDEX_INVALID</p> <p>PICO_NOT_RESPONDING</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.19 ps6000GetValuesOverlapped

```
PICO_STATUS ps6000GetValuesOverlapped
(
    short          handle,
    unsigned long  startIndex,
    unsigned long  * noOfSamples,
    unsigned long  downSampleRatio,
    PS6000_RATIO_MODE downSampleRatioMode,
    unsigned long  segmentIndex,
    short          * overflow
)
```

This function allows you to make a deferred data-collection request, which will later be executed, and the arguments validated, when you call [ps6000RunBlock](#) in block mode. The advantage of this function is that the driver makes contact with the scope only once, when you call [ps6000RunBlock](#), compared with the two contacts that occur when you use the conventional [ps6000RunBlock](#), [ps6000GetValues](#) calling sequence. This slightly reduces the dead time between successive captures in block mode.

After calling [ps6000RunBlock](#), you can optionally use [ps6000GetValues](#) to request further copies of the data. This might be required if you wish to display the data with different data reduction settings.

<b>Applicability</b>	<a href="#">Block mode</a>
<b>Arguments</b>	<p>handle, the handle of the device</p> <p>startIndex: see <a href="#">ps6000GetValues</a></p> <p>* noOfSamples: see <a href="#">ps6000GetValues</a></p> <p>downSampleRatio: see <a href="#">ps6000GetValues</a></p> <p>downSampleRatioMode: see <a href="#">ps6000GetValues</a></p> <p>segmentIndex: see <a href="#">ps6000GetValues</a></p> <p>* overflow: see <a href="#">ps6000GetValuesBulk</a></p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_INVALID_PARAMETER</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.20 ps6000GetValuesOverlappedBulk

```
PICO_STATUS ps6000GetValuesOverlappedBulk
(
    short          handle,
    unsigned long  startIndex,
    unsigned long  * noOfSamples,
    unsigned long  downSampleRatio,
    PS6000_RATIO_MODE downSampleRatioMode,
    unsigned long  fromSegmentIndex,
    unsigned long  toSegmentIndex,
    short          * overflow
)
```

This function allows you to make a deferred data-collection request, which will later be executed, and the arguments validated, when you call [ps6000RunBlock](#) in rapid block mode. The advantage of this method is that the driver makes contact with the scope only once, when you call [ps6000RunBlock](#), compared with the two contacts that occur when you use the conventional [ps6000RunBlock](#), [ps6000GetValues](#) calling sequence. This slightly reduces the dead time between successive captures in rapid block mode.

After calling [ps6000RunBlock](#), you can optionally use [ps6000GetValues](#) to request further copies of the data. This might be required if you wish to display the data with different data reduction settings.

<b>Applicability</b>	<a href="#">Rapid block mode</a>
<b>Arguments</b>	<p>handle, the handle of the device</p> <p>startIndex: see <a href="#">ps6000GetValues</a></p> <p>* noOfSamples: see <a href="#">ps6000GetValues</a></p> <p>downSampleRatio: see <a href="#">ps6000GetValues</a></p> <p>downSampleRatioMode: see <a href="#">ps6000GetValues</a></p> <p>fromSegmentIndex: see <a href="#">ps6000GetValuesBulk</a></p> <p>toSegmentIndex: see <a href="#">ps6000GetValuesBulk</a></p> <p>* overflow, see <a href="#">ps6000GetValuesBulk</a></p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_INVALID_PARAMETER</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.21 ps6000GetValuesTriggerTimeOffsetBulk

```
PICO_STATUS ps6000GetValuesTriggerTimeOffsetBulk
(
    short          handle,
    unsigned long  * timesUpper,
    unsigned long  * timesLower,
    PS6000_TIME_UNITS * timeUnits,
    unsigned long  fromSegmentIndex,
    unsigned long  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 [ps6000GetValuesTriggerTimeOffsetBulk64](#).

<b>Applicability</b>	<a href="#">Rapid block mode</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the device</p> <p>* <code>timesUpper</code>, an array of integers. On exit, the most significant 32 bits of the time offset for each requested segment index. <code>times[0]</code> will hold the <code>fromSegmentIndex</code> time offset and the last <code>times</code> index will hold the <code>toSegmentIndex</code> time offset. The array must be long enough to hold the number of requested times.</p> <p>* <code>timesLower</code>, an array of integers. On exit, the least-significant 32 bits of the time offset for each requested segment index. <code>times[0]</code> will hold the <code>fromSegmentIndex</code> time offset and the last <code>times</code> index will hold the <code>toSegmentIndex</code> time offset. The array size must be long enough to hold the number of requested times.</p> <p>* <code>timeUnits</code>, an array of integers. The array must be long enough to hold the number of requested times. On exit, <code>timeUnits[0]</code> will contain the time unit for <code>fromSegmentIndex</code> and the last element will contain the time unit for <code>toSegmentIndex</code>.</p> <p><code>fromSegmentIndex</code>, the first segment for which the time offset is required</p> <p><code>toSegmentIndex</code>, the last segment for which the time offset is required. If <code>toSegmentIndex</code> is less than <code>fromSegmentIndex</code> then the driver will wrap around from the last segment to the first.</p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_NULL_PARAMETER</p> <p>PICO_DEVICE_SAMPLING</p> <p>PICO_SEGMENT_OUT_OF_RANGE</p> <p>PICO_NO_SAMPLES_AVAILABLE</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.22 ps6000GetValuesTriggerTimeOffsetBulk64

```
PICO_STATUS ps6000GetValuesTriggerTimeOffsetBulk64
(
    short          handle,
    __int64       * times,
    PS6000_TIME_UNITS * timeUnits,
    unsigned long  fromSegmentIndex,
    unsigned long  toSegmentIndex
)
```

This function retrieves the 64-bit time offsets for waveforms captured in [rapid block mode](#).

A 32-bit version of this function, [ps6000GetValuesTriggerTimeOffsetBulk](#), is available for use with programming languages that do not support 64-bit integers.

<b>Applicability</b>	<a href="#">Rapid block mode</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the device</p> <p>* <code>times</code>, an array of integers. On exit, this will hold the time offset for each requested segment index. <code>times[0]</code> will hold the time offset for <code>fromSegmentIndex</code>, and the last <code>times</code> index will hold the time offset for <code>toSegmentIndex</code>. The array must be long enough to hold the number of times requested.</p> <p>* <code>timeUnits</code>, an array of integers long enough to hold the number of requested times. <code>timeUnits[0]</code> will contain the time unit for <code>fromSegmentIndex</code>, and the last element will contain the <code>toSegmentIndex</code>.</p> <p><code>fromSegmentIndex</code>, the first segment for which the time offset is required. The results for this segment will be placed in <code>times[0]</code> and <code>timeUnits[0]</code>.</p> <p><code>toSegmentIndex</code>, the last segment for which the time offset is required. The results for this segment will be placed in the last elements of the <code>times</code> and <code>timeUnits</code> arrays. If <code>toSegmentIndex</code> is less than <code>fromSegmentIndex</code> then the driver will wrap around from the last segment to the first.</p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_NULL_PARAMETER</p> <p>PICO_DEVICE_SAMPLING</p> <p>PICO_SEGMENT_OUT_OF_RANGE</p> <p>PICO_NO_SAMPLES_AVAILABLE</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.23 ps6000IsReady

```
PICO_STATUS ps6000IsReady
(
    short handle,
    short * ready
)
```

This function may be used instead of a callback function to receive data from [ps6000RunBlock](#). To use this method, pass a NULL pointer as the `lpReady` argument to [ps6000RunBlock](#). You must then poll the driver to see if it has finished collecting the requested samples.

<b>Applicability</b>	<a href="#">Block mode</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>ready</code>: output: indicates the state of the collection. If zero, the device is still collecting. If non-zero, the device has finished collecting and <a href="#">ps6000GetValues</a> can be used to retrieve the data.</p>
<b>Returns</b>	

## 3.9.24 ps6000IsTriggerOrPulseWidthQualifierEnabled

```
PICO_STATUS ps6000IsTriggerOrPulseWidthQualifierEnabled
(
    short handle,
    short * triggerEnabled,
    short * pulseWidthQualifierEnabled
)
```

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

<b>Applicability</b>	Call after setting up the trigger, and just before calling either <a href="#">ps6000RunBlock</a> or <a href="#">ps6000RunStreaming</a> .
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>triggerEnabled</code>, on exit, indicates whether the trigger will successfully be set when <a href="#">ps6000RunBlock</a> or <a href="#">ps6000RunStreaming</a> is called. A non-zero value indicates that the trigger is set, zero that the trigger is not set.</p> <p><code>pulseWidthQualifierEnabled</code>, on exit, indicates whether the pulse width qualifier will successfully be set when <a href="#">ps6000RunBlock</a> or <a href="#">ps6000RunStreaming</a> is called. A non-zero value indicates that the pulse width qualifier is set, zero that the pulse width qualifier is not set.</p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_NULL_PARAMETER</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.25 ps6000MemorySegments

```
PICO_STATUS ps6000MemorySegments
(
    short          handle
    unsigned long  nSegments,
    unsigned long  * 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.

<b>Applicability</b>	All modes
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>nSegments</code>, the number of segments required, from 1 to 32,768 for the PicoScope 6402 or from 1 to 1,000,000 for the PicoScope 6403.</p> <p>* <code>nMaxSamples</code>, 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 <code>nMaxSamples</code> divided by the number of channels.</p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_USER_CALLBACK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_TOO_MANY_SEGMENTS</p> <p>PICO_MEMORY</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.26 ps6000NoOfStreamingValues

```
PICO_STATUS ps6000NoOfStreamingValues
(
    short      handle,
    unsigned long * noOfValues
)
```

This function returns the number of samples available after data collection in [streaming mode](#). Call it after calling [ps6000Stop](#).

<b>Applicability</b>	<a href="#">Streaming mode</a>
<b>Arguments</b>	handle, the handle of the required device  * noOfValues, on exit, the number of samples
<b>Returns</b>	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_NOT_USED PICO_BUSY PICO_DRIVER_FUNCTION

## 3.9.27 ps6000OpenUnit

```
PICO_STATUS ps6000OpenUnit
(
    short * handle,
    char * serial
)
```

This function opens a PicoScope 6000 Series scope attached to the computer. The maximum number of units that can be opened depends on the operating system, the kernel driver and the computer.

<b>Applicability</b>	All modes
<b>Arguments</b>	<p>* <i>handle</i>, on exit, the result of the attempt to open a scope:  -1 : if the scope fails to open  0 : if no scope is found  &gt; 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.</p> <p><i>serial</i>, on entry, a null-terminated string containing the serial number of the scope to be opened. If <i>serial</i> is NULL then the function opens the first scope found; otherwise, it tries to open the scope that matches the string.</p>
<b>Returns</b>	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 (if the specified unit was not found) PICO_NOT_RESPONDING PICO_MEMORY_FAIL PICO_ANALOG_BOARD PICO_CONFIG_FAIL_AWG PICO_INITIALISE_FPGA

## 3.9.28 ps6000OpenUnitAsync

```
PICO_STATUS ps6000OpenUnitAsync
(
    short * status
    char * serial
)
```

This function opens a scope without blocking the calling thread. You can find out when it has finished by periodically calling [ps6000OpenUnitProgress](#) until that function returns a non-zero value.

<b>Applicability</b>	All modes
<b>Arguments</b>	<p>* <code>status</code>, 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</p> <p>* <code>serial</code>: see <a href="#">ps6000OpenUnit</a></p>
<b>Returns</b>	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_OPERATION_FAILED

## 3.9.29 ps6000OpenUnitProgress

```
PICO_STATUS ps6000OpenUnitProgress
(
    short * handle,
    short * progressPercent,
    short * complete
)
```

This function checks on the progress of a request made to [ps6000OpenUnitAsync](#) to open a scope.

<b>Applicability</b>	Use after <a href="#">ps6000OpenUnitAsync</a>
<b>Arguments</b>	<ul style="list-style-type: none"> <li>* <code>handle</code>: see <a href="#">ps6000OpenUnit</a>. This handle is valid only if the function returns <code>PICO_OK</code>.</li> <li>* <code>progressPercent</code>, on exit, the percentage progress towards opening the scope. 100% implies that the open operation is complete.</li> <li>* <code>complete</code>, set to 1 when the open operation has finished</li> </ul>
<b>Returns</b>	<ul style="list-style-type: none"> <li><code>PICO_OK</code></li> <li><code>PICO_NULL_PARAMETER</code></li> <li><code>PICO_OPERATION_FAILED</code></li> </ul>

## 3.9.30 ps6000RunBlock

```

PICO_STATUS ps6000RunBlock
(
    short          handle,
    unsigned long  noOfPreTriggerSamples,
    unsigned long  noOfPostTriggerSamples,
    unsigned long  timebase,
    short          oversample,
    long           * timeIndisposedMs,
    unsigned long  segmentIndex,
    ps6000BlockReady 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`.

<b>Applicability</b>	<a href="#">Block mode</a> , <a href="#">rapid block mode</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device.</p> <p><code>noOfPreTriggerSamples</code>, the number of samples to return before the trigger event. If no trigger has been set then this argument is ignored and <code>noOfPostTriggerSamples</code> specifies the maximum number of samples to collect.</p> <p><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: -</p> $\text{noOfPreTriggerSamples} + \text{noOfPostTriggerSamples}$ <p><code>timebase</code>, a number in the range 0 to <math>2^{32}-1</math>. See the <a href="#">guide to calculating timebase values</a>.</p> <p><code>oversample</code>, the <a href="#">oversampling</a> factor, a number in the range 1 to 256.</p> <p><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.</p> <p><code>segmentIndex</code>, zero-based, specifies which <a href="#">memory segment</a> to use.</p> <p><code>lpReady</code>, a pointer to the <a href="#">ps6000BlockReady</a> callback function that the driver will call when the data has been collected. To use the <a href="#">ps6000IsReady</a> polling method instead of a callback function, set this pointer to NULL.</p>

	<p>* <code>pParameter</code>, a void pointer that is passed to the <a href="#">ps6000BlockReady</a> callback function. The callback can use this pointer to return arbitrary data to the application.</p>
<b>Returns</b>	<pre> PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_CHANNEL PICO_INVALID_TRIGGER_CHANNEL PICO_INVALID_CONDITION_CHANNEL PICO_TOO_MANY_SAMPLES PICO_INVALID_TIMEBASE PICO_NOT_RESPONDING PICO_CONFIG_FAIL PICO_INVALID_PARAMETER PICO_NOT_RESPONDING PICO_TRIGGER_ERROR PICO_DRIVER_FUNCTION PICO_EXTERNAL_FREQUENCY_INVALID PICO_FW_FAIL PICO_NOT_ENOUGH_SEGMENTS (in Bulk mode) PICO_TRIGGER_AND_EXTERNAL_CLOCK_CLASH PICO_PWQ_AND_EXTERNAL_CLOCK_CLASH 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 </pre>

## 3.9.31 ps6000RunStreaming

```

PICO_STATUS ps6000RunStreaming
(
    short          handle,
    unsigned long  * sampleInterval,
    PS6000_TIME_UNITS sampleIntervalTimeUnits
    unsigned long  maxPreTriggerSamples,
    unsigned long  maxPostTriggerSamples,
    short          autoStop,
    unsigned long  downSampleRatio,
    PS6000_RATIO_MODE downSampleRatioMode,
    unsigned long  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 [ps6000GetStreamingLatestValues](#) 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.

<b>Applicability</b>	<a href="#">Streaming mode</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device.</p> <p><code>* sampleInterval</code>, on entry, the requested time interval between samples; on exit, the actual time interval used.</p> <p><code>sampleIntervalTimeUnits</code>, the unit of time used for <code>sampleInterval</code>. Use one of these values:</p> <ul style="list-style-type: none"> <li><a href="#">PS6000_FS</a></li> <li><a href="#">PS6000_PS</a></li> <li><a href="#">PS6000_NS</a></li> <li><a href="#">PS6000_US</a></li> <li><a href="#">PS6000_MS</a></li> <li><a href="#">PS6000_S</a></li> </ul> <p><code>maxPreTriggerSamples</code>, the maximum number of raw samples before a trigger event for each enabled channel. If no trigger condition is set this argument is ignored.</p> <p><code>maxPostTriggerSamples</code>, the maximum number of raw samples after a trigger event for each enabled channel. If no trigger condition is set, this argument states the maximum number of samples to be stored.</p> <p><code>autoStop</code>, a flag that specifies if the streaming should stop when all of <code>maxSamples</code> have been captured.</p> <p><code>downSampleRatio</code>: see <a href="#">ps6000GetValues</a></p> <p><code>downSampleRatioMode</code>: see <a href="#">ps6000GetValues</a></p>

	<p>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 <code>bufferLth</code> value passed to <a href="#">ps6000SetDataBuffer</a>.</p>
<b>Returns</b>	<p>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 PICO_DRIVER_FUNCTION PICO_EXTERNAL_FREQUENCY_INVALID PICO_FW_FAIL PICO_TRIGGER_AND_EXTERNAL_CLOCK_CLASH PICO_PWQ_AND_EXTERNAL_CLOCK_CLASH PICO_MEMORY</p>

## 3.9.32 ps6000SetChannel

```
PICO_STATUS ps6000SetChannel
(
    short          handle,
    PS6000_CHANNEL channel,
    short          enabled,
    PS6000_COUPLING type,
    PS6000_RANGE  range,
    float          analogueOffset,
    PS6000_BANDWIDTH_LIMITER bandwidth
)
```

This function specifies whether an input channel is to be enabled, its input coupling type, voltage range, analog offset and bandwidth limit. Some of the arguments within this function have model—specific values. Please consult the relevant section below according to the model you have.

<b>Applicability</b>	All modes
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>channel</code>, the channel to be configured. The values are:  <a href="#">PS6000_CHANNEL_A</a>: Channel A input  <a href="#">PS6000_CHANNEL_B</a>: Channel B input  <a href="#">PS6000_CHANNEL_C</a>: Channel C input  <a href="#">PS6000_CHANNEL_D</a>: Channel input</p> <p><code>enabled</code>, whether or not to enable the channel. The values are:  TRUE: enable  FALSE: do not enable</p> <p><code>type</code>, the impedance and coupling type. The values supported are:  <b>PicoScope 6402/6403/6404</b>  PS6000_AC, 1 megohm impedance, AC coupling. The channel accepts input frequencies from about 1 hertz up to its maximum -3 dB analog bandwidth.  PS6000_DC_1M, 1 megohm impedance, DC coupling. The scope accepts all input frequencies from zero (DC) up to its maximum -3 dB analog bandwidth.  PS6000_DC_50R, DC coupling, 50 ohm impedance. In this mode the <math>\pm 10</math> volt and <math>\pm 20</math> volt input ranges are not available.  <b>PicoScope 6407</b>  PS6000_DC_50R, DC coupling, 50 ohm impedance.</p> <p><code>range</code>, the input voltage range:  <b>PicoScope 6402/6403/6404</b>  <a href="#">PS6000_50MV</a>: <math>\pm 50</math> mV  <a href="#">PS6000_100MV</a>: <math>\pm 100</math> mV  <a href="#">PS6000_200MV</a>: <math>\pm 200</math> mV  <a href="#">PS6000_500MV</a>: <math>\pm 500</math> mV  <a href="#">PS6000_1V</a>: <math>\pm 1</math> V</p>

	<p><a href="#">PS6000_2V</a>: ±2 V  <a href="#">PS6000_5V</a>: ±5 V  <a href="#">PS6000_10V</a>: ±10 V *  <a href="#">PS6000_20V</a>: ±20 V *  * not available when type = PS6000_DC_50R</p> <p><b>PicoScope 6407</b>  <a href="#">PS6000_100MV</a>: ±100 mV</p> <p>analogueOffset, a voltage to add to the input channel before digitization.</p> <p><b>PicoScope 6402/6403</b>  The allowable range of offsets depends on the input range selected for the channel, as follows:</p> <p>50 mV to 200 mV: <a href="#">MIN_ANALOGUE_OFFSET_50MV_200MV</a> to <a href="#">MAX_ANALOGUE_OFFSET_50MV_200MV</a></p> <p>500 mV to 2 V: <a href="#">MIN_ANALOGUE_OFFSET_500MV_2V</a> to <a href="#">MAX_ANALOGUE_OFFSET_500MV_2V</a></p> <p>5 V to 20 V: <a href="#">MIN_ANALOGUE_OFFSET_5V_20V</a> to <a href="#">MAX_ANALOGUE_OFFSET_5V_20V</a>. (When type = PS6000_DC_50R, the allowable range is reduced to that of the 50 mV to 200 mV input range, i.e. <a href="#">MIN_ANALOGUE_OFFSET_50MV_200MV</a> to <a href="#">MAX_ANALOGUE_OFFSET_50MV_200MV</a>).</p> <p>Allowable range of offsets can also be returned by ps6000GetAnalogueOffset for the device currently connected.</p> <p><b>PicoScope 6404</b>  Allowable range of offsets is returned by ps6000GetAnalogueOffset for the device currently connected.</p> <p><b>PicoScope 6407</b>  analogueOffset, 6407 does not use analogueOffset, therefore this should be set to 0.</p> <p>bandwidth, the bandwidth limiter setting:</p> <p><b>PicoScope 6402/6403</b>  PS6000_BW_FULLL: the connected scope's full specified bandwidth  PS6000_BW_20MHZ: -3 dB bandwidth limited to 20 MHz</p> <p><b>PicoScope 6404</b>  PS6000_BW_FULLL: the scope's full specified bandwidth  PS6000_BW_25MHZ: -3 dB bandwidth limited to 25 MHz</p> <p><b>PicoScope 6407</b>  PS6000_BW_FULLL: the scope's full specified bandwidth</p>
<b>Returns</b>	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

## 3.9.33 ps6000SetDataBuffer

```
PICO_STATUS ps6000SetDataBuffer
(
    short          handle,
    PS6000_CHANNEL channel,
    short          * buffer,
    unsigned long  bufferLth,
    PS6000_RATIO_MODE downSampleRatioMode
)
```

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 [ps6000SetDataBuffers](#) instead.

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

<b>Applicability</b>	<a href="#">Block</a> , <a href="#">rapid block</a> and <a href="#">streaming</a> modes. All <a href="#">downsampling</a> modes except <a href="#">aggregation</a> .
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>channel</code>, the channel you want to use with the buffer. Use one of these values:  <a href="#">PS6000_CHANNEL_A</a>  <a href="#">PS6000_CHANNEL_B</a>  <a href="#">PS6000_CHANNEL_C</a>  <a href="#">PS6000_CHANNEL_D</a></p> <p><code>buffer</code>, the location of the buffer</p> <p><code>bufferLth</code>, the size of the <code>buffer</code> array</p> <p><code>downSampleRatioMode</code>, the <a href="#">downsampling</a> mode. See <a href="#">ps6000GetValues</a> for the available modes, but note that a single call to <a href="#">ps6000SetDataBuffer</a> can only associate one buffer with one downsampling mode. If you intend to call <a href="#">ps6000GetValues</a> with more than one downsampling mode activated, then you must call <a href="#">ps6000SetDataBuffer</a> several times to associate a separate buffer with each downsampling mode.</p>
<b>Returns</b>	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_RATIO_MODE_NOT_SUPPORTED PICO_DRIVER_FUNCTION PICO_INVALID_PARAMETER

## 3.9.34 ps6000SetDataBufferBulk

```

PICO_STATUS ps6000SetDataBufferBulk
(
    short          handle,
    PS6000_CHANNEL channel,
    short          * buffer,
    unsigned long  bufferLth,
    unsigned long  waveform,
    PS6000_RATIO_MODE downSampleRatioMode
)

```

This function allows you to associate a buffer with a specified waveform number and input channel in [rapid block mode](#). The number of waveforms captured is determined by the `nCaptures` argument sent to [ps6000SetNoOfCaptures](#). There is only one buffer for each waveform because the only downsampling mode that requires two buffers, [aggregation](#) mode, is not available in rapid block mode. Call one of the [GetValues](#) functions to retrieve the data after capturing.

<b>Applicability</b>	<a href="#">Rapid block mode</a> without <a href="#">aggregation</a> .
<b>Arguments</b>	<p><code>handle</code>, the handle of the device</p> <p><code>channel</code>, the input channel to use with this buffer</p> <p><code>buffer</code>, an array in which the captured data is stored</p> <p><code>bufferLth</code>, the size of the buffer</p> <p><code>waveform</code>, an index to the waveform number. Range: 0 to <code>nCaptures</code> - 1</p> <p><code>downSampleRatioMode</code>: see <a href="#">ps6000GetValues</a></p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_INVALID_CHANNEL</p> <p>PICO_INVALID_PARAMETER</p> <p>PICO_RATIO_MODE_NOT_SUPPORTED</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.35 ps6000SetDataBuffers

```
PICO_STATUS ps6000SetDataBuffers
(
    short          handle,
    PS6000_CHANNEL channel,
    short          * bufferMax,
    short          * bufferMin,
    unsigned long  bufferLth,
    PS6000_RATIO_MODE downSampleRatioMode
)
```

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 [ps6000SetDataBuffer](#) instead.

<b>Applicability</b>	<a href="#">Block</a> and <a href="#">streaming</a> modes with <a href="#">aggregation</a> .
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device.</p> <p><code>channel</code>, the channel for which you want to set the buffers. Use one of these constants:  <a href="#">PS6000_CHANNEL_A</a>  <a href="#">PS6000_CHANNEL_B</a>  <a href="#">PS6000_CHANNEL_C</a>  <a href="#">PS6000_CHANNEL_D</a></p> <p>* <code>bufferMax</code>, a buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise.</p> <p>* <code>bufferMin</code>, a buffer to receive the minimum aggregated data values. Not used in other downsampling modes.</p> <p><code>bufferLth</code>, the size of the <code>bufferMax</code> and <code>bufferMin</code> arrays.</p> <p><code>downSampleRatioMode</code>: see <a href="#">ps6000GetValues</a></p>
<b>Returns</b>	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_RATIO_MODE_NOT_SUPPORTED PICO_DRIVER_FUNCTION PICO_INVALID_PARAMETER

## 3.9.36 ps6000SetDataBuffersBulk

```

PICO_STATUS ps6000SetDataBuffersBulk
(
    short          handle,
    PS6000_CHANNEL channel,
    short          * bufferMax,
    short          * bufferMin,
    unsigned long  bufferLth,
    unsigned long  waveform,
    PS6000_RATIO_MODE downSampleRatioMode
)

```

This function tells the driver where to find the buffers for [aggregated](#) data for each waveform in [rapid block mode](#). The number of waveforms captured is determined by the `nCaptures` argument sent to [ps6000SetNoOfCaptures](#). Call one of the [GetValues](#) functions to retrieve the data after capture. If you do not need two buffers, because you are not using [aggregate](#) mode, then you can optionally use [ps6000SetDataBufferBulk](#) instead.

<b>Applicability</b>	<a href="#">Rapid block mode</a> with <a href="#">aggregation</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the device</p> <p><code>channel</code>, the input channel to use with the buffer</p> <p>* <code>bufferMax</code>, a buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise</p> <p>* <code>bufferMin</code>, a buffer to receive the minimum data values in <a href="#">aggregate</a> mode. Not used in other <a href="#">downsampling</a> modes.</p> <p><code>bufferLth</code>, the size of the buffer</p> <p><code>waveform</code>, an index to the waveform number between 0 and <code>nCaptures - 1</code></p> <p><code>downSampleRatioMode</code>: see <a href="#">ps6000GetValues</a></p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_INVALID_CHANNEL</p> <p>PICO_INVALID_PARAMETER</p> <p>PICO_RATIO_MODE_NOT_SUPPORTED</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.37 ps6000SetEts

```
PICO_STATUS ps6000SetEts
(
    short          handle,
    PS6000_ETS_MODE mode,
    short          etsCycles,
    short          etsInterleave,
    long           * 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.

<b>Applicability</b>	<a href="#">Block mode</a>
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>mode</code>, the ETS mode. Use one of these values:  <a href="#">PS6000_ETS_OFF</a> - disables ETS  <a href="#">PS6000_ETS_FAST</a> - enables ETS and provides <code>etsCycles</code> of data, which may contain data from previously returned cycles  <a href="#">PS6000_ETS_SLOW</a> - enables ETS and provides fresh data every <code>etsCycles</code>. This mode takes longer to provide each data set, but the data sets are more stable and are guaranteed to contain only new data.</p> <p><code>etscycles</code>, the number of cycles to store: the computer can then select <code>etsInterleave</code> cycles to give the most uniform spread of samples.  Range: between two and five times the value of <code>etsInterleave</code>, and not more than <a href="#">PS6000_MAX_ETS_CYCLES</a></p> <p><code>etsInterleave</code>, the number of waveforms to combine into a single ETS capture  Maximum value: <a href="#">PS6000_MAX_INTERLEAVE</a></p> <p><code>* sampleTimePicoseconds</code>, on exit, the effective sampling interval of the ETS data. For example, if the captured sample time is 20 ns and <code>etsInterleave</code> is 10, then the effective sample time in ETS mode is 2 ns.</p>
<b>Returns</b>	<a href="#">PICO_OK</a> <a href="#">PICO_USER_CALLBACK</a> <a href="#">PICO_INVALID_HANDLE</a> <a href="#">PICO_INVALID_PARAMETER</a> <a href="#">PICO_DRIVER_FUNCTION</a>

## 3.9.38 ps6000SetEtsTimeBuffer

```
PICO_STATUS ps6000SetEtsTimeBuffer
(
    short          handle,
    __int64        * buffer,
    unsigned long  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.

<b>Applicability</b>	<a href="#">ETS mode</a> only.  If your programming language does not support 64-bit data, use the 32-bit version <a href="#">ps6000SetEtsTimeBuffers</a> instead.
<b>Arguments</b>	<code>handle</code> , the handle of the required device  <code>* buffer</code> , an array of 64-bit words, each representing the time in nanoseconds at which the sample was captured  <code>bufferLth</code> , the size of the buffer array
<b>Returns</b>	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_DRIVER_FUNCTION

## 3.9.39 ps6000SetEtsTimeBuffers

```
PICO_STATUS ps6000SetEtsTimeBuffers
(
    short          handle,
    unsigned long * timeUpper,
    unsigned long * timeLower,
    unsigned long  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.

<b>Applicability</b>	<p><a href="#">ETS mode</a> only.</p> <p>If your programming language supports 64-bit data then you can use <a href="#">ps6000SetEtsTimeBuffer</a> instead.</p>
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p>* <code>timeUpper</code>, an array of 32-bit words, each representing the upper 32 bits of the time in nanoseconds at which the sample was captured</p> <p>* <code>timeLower</code>, an array of 32-bit words, each representing the lower 32 bits of the time in nanoseconds at which the sample was captured</p> <p><code>bufferLth</code>, the size of the <code>timeUpper</code> and <code>timeLower</code> arrays</p>
<b>Returns</b>	<p>PICO_OK  PICO_INVALID_HANDLE  PICO_NULL_PARAMETER  PICO_DRIVER_FUNCTION</p>

## 3.9.40 ps6000SetExternalClock

```
PICO_STATUS ps6000SetExternalClock
(
    short          handle,
    PS6000_EXTERNAL_FREQUENCY frequency,
    short          threshold
)
```

This function tells the scope whether or not to use an external clock signal fed into the AUX input. The external clock can be used to synchronise one or more PicoScope 6000 units to an external source.

When the external clock input is enabled, the oscilloscope relies on the clock signal for all of its timing. The driver checks that the clock is running before starting a capture, but if the clock signal stops after the initial check, the oscilloscope will not respond to any further commands until it is powered down and back up again.

Note: if the AUX input is set as an external clock input then it cannot also be used as an external trigger input.

<b>Applicability</b>	All modes						
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>frequency</code>, the external clock frequency. The possible values are:</p> <p><a href="#">PS6000_FREQUENCY_OFF</a>: the scope generates its own clock</p> <p><a href="#">PS6000_FREQUENCY_5MHZ</a>: 5 MHz external clock</p> <p><a href="#">PS6000_FREQUENCY_10MHZ</a>: 10 MHz external clock</p> <p><a href="#">PS6000_FREQUENCY_20MHZ</a>: 20 MHz external clock</p> <p><a href="#">PS6000_FREQUENCY_25MHZ</a>: 25 MHz external clock</p> <p>The external clock signal must be within <math>\pm 5\%</math> of the selected frequency, otherwise this function will report an error.</p> <p><code>threshold</code>, the logic threshold voltage:</p> <table border="1"> <tr> <td>-32,512</td> <td>-1 volt</td> </tr> <tr> <td>0</td> <td>0 volts</td> </tr> <tr> <td>32,512</td> <td>+1 volt</td> </tr> </table>	-32,512	-1 volt	0	0 volts	32,512	+1 volt
-32,512	-1 volt						
0	0 volts						
32,512	+1 volt						
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_USER_CALLBACK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_INVALID_PARAMETER</p> <p>PICO_DRIVER_FUNCTION</p> <p>PICO_EXTERNAL_FREQUENCY_INVALID</p> <p>PICO_FW_FAIL</p> <p>PICO_NOT_RESPONDING</p> <p>PICO_CLOCK_CHANGE_ERROR</p>						

## 3.9.41 ps6000SetNoOfCaptures

```
PICO_STATUS ps6000SetNoOfCaptures
(
    short          handle,
    unsigned long 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.

<b>Applicability</b>	<a href="#">Rapid block mode</a>
<b>Arguments</b>	handle, the handle of the device nCaptures, the number of waveforms to capture in one run
<b>Returns</b>	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

## 3.9.42 ps6000SetPulseWidthQualifier

```
PICO_STATUS ps6000SetPulseWidthQualifier
(
    short                handle,
    PS6000_PWQ_CONDITIONS * conditions,
    short                nConditions,
    PS6000_THRESHOLD_DIRECTION direction,
    unsigned long        lower,
    unsigned long        upper,
    PS6000_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 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.

<b>Applicability</b>	All modes
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>* conditions</code>, an array of <a href="#">PS6000_PWQ_CONDITIONS</a> structures specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there are several elements, the overall trigger condition is the logical OR of all the elements. If <code>conditions</code> is <code>NULL</code> then the pulse-width qualifier is not used.</p> <p><code>nConditions</code>, the number of elements in the <code>conditions</code> array. If <code>nConditions</code> is zero then the pulse-width qualifier is not used. Range: 0 to <a href="#">PS6000_MAX_PULSE_WIDTH_QUALIFIER_COUNT</a>.</p> <p><code>direction</code>, the direction of the signal required for the trigger to fire. See <a href="#">ps6000SetTriggerChannelDirections</a> for the list of possible values. Each channel of the oscilloscope (except the AUX input) has two thresholds for each direction—for example, <a href="#">PS6000_RISING</a> and <a href="#">PS6000_RISING_LOWER</a>—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 <a href="#">PS6000_RISING</a> as the <code>direction</code> argument for both <a href="#">ps6000SetTriggerConditions</a> and <a href="#">ps6000SetPulseWidthQualifier</a> at the same time. There is no such restriction when using window triggers.</p> <p><code>lower</code>, the lower limit of the pulse-width counter</p> <p><code>upper</code>, the upper limit of the pulse-width counter. This parameter is used only when the type is set to <a href="#">PS6000_PW_TYPE_IN_RANGE</a> or <a href="#">PS6000_PW_TYPE_OUT_OF_RANGE</a>.</p>

	<p>type, the pulse-width type, one of these constants:</p> <p><a href="#">PS6000_PW_TYPE_NONE</a>: do not use the pulse width qualifier</p> <p><a href="#">PS6000_PW_TYPE_LESS_THAN</a>: pulse width less than lower</p> <p><a href="#">PS6000_PW_TYPE_GREATER_THAN</a>: pulse width greater than lower</p> <p><a href="#">PS6000_PW_TYPE_IN_RANGE</a>: pulse width between lower and upper</p> <p><a href="#">PS6000_PW_TYPE_OUT_OF_RANGE</a>: pulse width not between lower and upper</p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_USER_CALLBACK</p> <p>PICO_CONDITIONS</p> <p>PICO_PULSE_WIDTH_QUALIFIER</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.42.1 PS6000\_PWQ\_CONDITIONS structure

A structure of this type is passed to [ps6000SetPulseWidthQualifier](#) in the conditions argument to specify the trigger conditions. It is defined as follows:

```
typedef struct tPwqConditions
{
    PS6000_TRIGGER_STATE channelA;
    PS6000_TRIGGER_STATE channelB;
    PS6000_TRIGGER_STATE channelC;
    PS6000_TRIGGER_STATE channelD;
    PS6000_TRIGGER_STATE external;
    PS6000_TRIGGER_STATE aux;
} PS6000_PWQ_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The [ps6000SetPulseWidthQualifier](#) 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.

<b>Elements</b>	<p>channelA, channelB, channelC, channelD, aux: the type of condition that should be applied to each channel. Use these constants: -</p> <ul style="list-style-type: none"> <li><a href="#">PS6000_CONDITION_DONT_CARE</a></li> <li><a href="#">PS6000_CONDITION_TRUE</a></li> <li><a href="#">PS6000_CONDITION_FALSE</a></li> </ul> <p>The channels that are set to <a href="#">PS6000_CONDITION_TRUE</a> or <a href="#">PS6000_CONDITION_FALSE</a> must all meet their conditions simultaneously to produce a trigger. Channels set to <a href="#">PS6000_CONDITION_DONT_CARE</a> are ignored.</p> <p>external: not used</p>
-----------------	--

## 3.9.43 ps6000SetSigGenArbitrary

```

PICO_STATUS ps6000SetSigGenArbitrary
(
    short          handle,
    long           offsetVoltage,
    unsigned long  pkToPk,
    unsigned long  startDeltaPhase,
    unsigned long  stopDeltaPhase,
    unsigned long  deltaPhaseIncrement,
    unsigned long  dwellCount,
    short         * arbitraryWaveform,
    long          arbitraryWaveformSize,
    PS6000_SWEEP_TYPE sweepType,
    PS6000_EXTRA_OPERATIONS operation,
    PS6000_INDEX_MODE indexMode,
    unsigned long  shots,
    unsigned long  sweeps,
    PS6000_SIGGEN_TRIG_TYPE triggerType,
    PS6000_SIGGEN_TRIG_SOURCE triggerSource,
    short         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 counter that indicates the present location in the waveform. The top 14 bits of the counter are used as an index into a buffer containing the arbitrary waveform.

The generator steps through the waveform by adding a "delta phase" between 1 and  $2^{32}-1$  to the phase counter every 5 ns. If the delta phase is constant, then the generator produces a waveform at a constant frequency. It is also possible to sweep the frequency by progressively modifying the delta phase. This is done by setting up a "delta phase increment" which is added to the delta phase at specified intervals.

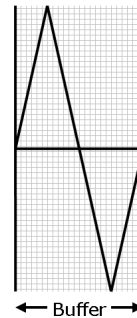
<b>Applicability</b>	All modes
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>offsetVoltage</code>, the voltage offset, in microvolts, to be applied to the waveform</p> <p><code>pkToPk</code>, the peak-to-peak voltage, in microvolts, of the waveform signal</p> <p><code>startDeltaPhase</code>, the initial value added to the phase counter as the generator begins to step through the waveform buffer</p> <p><code>stopDeltaPhase</code>, the final value added to the phase counter before the generator restarts or reverses the sweep</p> <p><code>deltaPhaseIncrement</code>, the amount added to the delta phase value every time the <code>dwellCount</code> period expires. This determines the amount by which the generator sweeps the output frequency in each dwell period.</p>

<b>Arguments</b>	<p><code>dwel1Count</code>, the time, in 5 ns steps, between successive additions of <code>deltaPhaseIncrement</code> to the delta phase counter. This determines the rate at which the generator sweeps the output frequency. Minimum value: <a href="#">PS6000_MIN_DWELL_COUNT</a></p> <p>* <code>arbitraryWaveform</code>, a buffer that holds the waveform pattern as a set of samples equally spaced in time.</p> <p><code>arbitraryWaveformSize</code>, the size of the arbitrary waveform buffer, in samples, from <a href="#">PS6000_MIN_SIG_GEN_BUFFER_SIZE</a> to <a href="#">PS6000_MAX_SIG_GEN_BUFFER_SIZE</a>.</p> <p><code>sweepType</code>, determines whether the <code>startDeltaPhase</code> is swept up to the <code>stopDeltaPhase</code>, or down to it, or repeatedly swept up and down. Use one of these values: -  <a href="#">PS6000_UP</a>  <a href="#">PS6000_DOWN</a>  <a href="#">PS6000_UPDOWN</a>  <a href="#">PS6000_DOWNUP</a></p> <p><code>operation</code>, see <a href="#">ps6000SigGenBuiltIn</a></p> <p><code>indexMode</code>, specifies how the signal will be formed from the arbitrary waveform data. <a href="#">Single, dual and quad index modes</a> are possible. Use one of these constants:  <a href="#">PS6000_SINGLE</a>  <a href="#">PS6000_DUAL</a>  <a href="#">PS6000_QUAD</a></p> <p><code>shots</code>, see <a href="#">ps6000SigGenBuiltIn</a>  <code>sweeps</code>, see <a href="#">ps6000SigGenBuiltIn</a>  <code>triggerType</code>, see <a href="#">ps6000SigGenBuiltIn</a>  <code>triggerSource</code>, see <a href="#">ps6000SigGenBuiltIn</a>  <code>extInThreshold</code>, see <a href="#">ps6000SigGenBuiltIn</a></p>
<b>Returns</b>	PICO_OK PICO_INVALID_HANDLE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING PICO_NOT_RESPONDING PICO_WARNING_AUX_OUTPUT_CONFLICT PICO_WARNING_EXT_THRESHOLD_CONFLICT PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_OFFSET_VOLTAGE PICO_SIGGEN_PK_TO_PK PICO_SIGGEN_OUTPUT_OVER_VOLTAGE PICO_DRIVER_FUNCTION PICO_SIGGEN_WAVEFORM_SETUP_FAILED

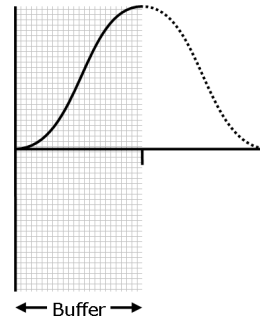
## 3.9.43.1 AWG index modes

The [arbitrary waveform generator](#) supports **single**, **dual** and **quad** index modes to help you make the best use of the waveform buffer.

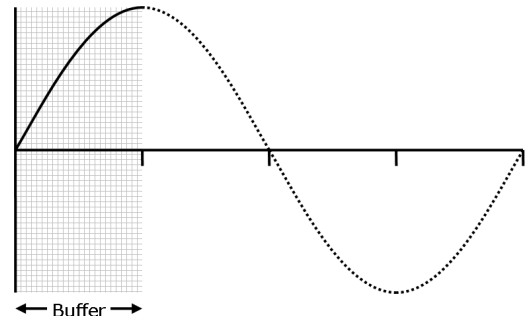
**Single mode.** The generator outputs the raw contents of the buffer repeatedly. This mode is the only one that can generate asymmetrical waveforms. You can also use this mode for symmetrical waveforms, but the dual and quad modes make more efficient use of the buffer memory.



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



**Quad mode.** The generator outputs the contents of the buffer, then on its second pass through the buffer outputs the same data in reverse order. On the third and fourth passes it does the same but with a negative version of the data. This allows you to specify only the first quarter of a waveform with fourfold symmetry, such as a sine wave, and let the generator fill in the other three quarters.



## 3.9.44 ps6000SetSigGenBuiltIn

```

PICO_STATUS ps6000SetSigGenBuiltIn
(
    short          handle,
    long           offsetVoltage,
    unsigned long  pkToPk,
    short          waveType,
    float          startFrequency,
    float          stopFrequency,
    float          increment,
    float          dwellTime,
    PS6000_SWEEP_TYPE sweepType,
    PS6000_EXTRA_OPERATIONS operation,
    unsigned long  shots,
    unsigned long  sweeps,
    PS6000_SIGGEN_TRIG_TYPE triggerType,
    PS6000_SIGGEN_TRIG_SOURCE triggerSource,
    short          extInThreshold
)

```

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

<b>Applicability</b>	All modes
<b>Arguments</b>	
handle, the handle of the required device	
offsetVoltage, the voltage offset, in microvolts, to be applied to the waveform	
pkToPk, the peak-to-peak voltage, in microvolts, of the waveform signal	
waveType, the type of waveform to be generated.	
PS6000_SINE	sine wave
PS6000_SQUARE	square wave
PS6000_TRIANGLE	triangle wave
PS6000_RAMP_UP	rising sawtooth
PS6000_RAMP_DOWN	falling sawtooth
PS6000_SINC	(sin x)/x
PS6000_GAUSSIAN	Gaussian
PS6000_HALF_SINE	half (full-wave rectified) sine
PS6000_DC_VOLTAGE	DC voltage
PS6000_WHITE_NOISE	white noise
startFrequency, the frequency that the signal generator will initially produce. For allowable values see <a href="#">PS6000_SINE_MAX_FREQUENCY</a> 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	
dwellTime, the time for which the sweep stays at each frequency, in seconds	

**Arguments**

`sweepType`, whether the frequency will sweep from `startFrequency` to `stopFrequency`, or in the opposite direction, or repeatedly reverse direction.

Use one of these constants:

[PS6000\\_UP](#)  
[PS6000\\_DOWN](#)  
[PS6000\\_UPDOWN](#)  
[PS6000\\_DOWNUP](#)

`operation`, selects periodic signal, white noise or PRBS:

`PS6000_ES_OFF` (0) produces the waveform specified by `waveType`.  
`PS6000_WHITENOISE` (1) produces white noise and ignores all settings except `offsetVoltage` and `pkTopk`.  
`PS6000_PRBS` (2) produces a [pseudo-random binary sequence \(PRBS\)](#) and ignores all settings except `offsetVoltage` and `pkTopk`.

`shots`, the number of cycles of the waveform to be produced after a trigger event. If non-zero (from 1 to [MAX\\_SWEEPS\\_SHOTS](#)), then `sweeps` must be zero.

`sweeps`, the number of times to sweep the frequency after a trigger event, according to `sweepType`. If non-zero (from 1 to [MAX\\_SWEEPS\\_SHOTS](#)), then `shots` must be zero.

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

`PS6000_SIGGEN_RISING` trigger on rising edge  
`PS6000_SIGGEN_FALLING` trigger on falling edge  
`PS6000_SIGGEN_GATE_HIGH` run while trigger is high  
`PS6000_SIGGEN_GATE_LOW` run while trigger is low

`triggerSource`, the source that will trigger the signal generator:

`PS6000_SIGGEN_NONE` run without waiting for trigger  
`PS6000_SIGGEN_SCOPE_TRIG` use scope trigger  
`PS6000_SIGGEN_AUX_IN` use AUX input  
`PS6000_SIGGEN_SOFT_TRIG` wait for software trigger provided by [ps6000SigGenSoftwareControl](#)  
`PS6000_SIGGEN_TRIGGER_RAW` reserved

If a trigger source other than [P6000\\_SIGGEN\\_NONE](#) is specified, then either `shots` or `sweeps`, but not both, must be non-zero.

`extInThreshold`, not used.

**Returns**

PICO\_OK  
PICO\_INVALID\_HANDLE  
PICO\_SIG\_GEN\_PARAM  
PICO\_SHOTS\_SWEEPS\_WARNING  
PICO\_NOT\_RESPONDING  
PICO\_WARNING\_AUX\_OUTPUT\_CONFLICT  
PICO\_WARNING\_EXT\_THRESHOLD\_CONFLICT  
PICO\_NO\_SIGNAL\_GENERATOR  
PICO\_SIGGEN\_OFFSET\_VOLTAGE  
PICO\_SIGGEN\_PK\_TO\_PK  
PICO\_SIGGEN\_OUTPUT\_OVER\_VOLTAGE  
PICO\_DRIVER\_FUNCTION  
PICO\_SIGGEN\_WAVEFORM\_SETUP\_FAILED  
PICO\_NOT\_RESPONDING

## 3.9.45 ps6000SetSimpleTrigger

```
PICO_STATUS ps6000SetSimpleTrigger
(
    short          handle,
    short          enable,
    PS6000\_CHANNEL source,
    short          threshold,
    THRESHOLD\_DIRECTION direction,
    unsigned long  delay,
    short          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.

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

## 3.9.46 ps6000SetTriggerChannelConditions

```
PICO_STATUS ps6000SetTriggerChannelConditions
(
    short                handle,
    PS6000_TRIGGER_CONDITIONS * conditions,
    short                nConditions
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is defined by one or more [PS6000\\_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 [ps6000SetSimpleTrigger](#).

<b>Applicability</b>	All modes
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device.</p> <p><code>conditions</code>, an array of <a href="#">PS6000_TRIGGER_CONDITIONS</a> 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.</p> <p><code>nConditions</code>, the number of elements in the <code>conditions</code> array. If <code>nConditions</code> is zero then triggering is switched off.</p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_USER_CALLBACK</p> <p>PICO_CONDITIONS</p> <p>PICO_MEMORY_FAIL</p> <p>PICO_DRIVER_FUNCTION</p>

## 3.9.46.1 PS6000\_TRIGGER\_CONDITIONS structure

A structure of this type is passed to [ps6000SetTriggerChannelConditions](#) in the `conditions` argument to specify the trigger conditions, and is defined as follows: -

```
typedef struct tTriggerConditions
{
    PS6000_TRIGGER_STATE channelA;
    PS6000_TRIGGER_STATE channelB;
    PS6000_TRIGGER_STATE channelC;
    PS6000_TRIGGER_STATE channelD;
    PS6000_TRIGGER_STATE external;
    PS6000_TRIGGER_STATE aux;
    PS6000_TRIGGER_STATE pulseWidthQualifier;
} PS6000_TRIGGER_CONDITIONS
```

Each structure is the logical AND of the states of the scope's inputs. The [ps6000SetTriggerChannelConditions](#) 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.

<b>Elements</b>	<p><code>channelA</code>, <code>channelB</code>, <code>channelC</code>, <code>channelD</code>, <code>aux</code>, <code>pulseWidthQualifier</code>: the type of condition that should be applied to each channel. Use these constants:</p> <p><a href="#">PS6000_CONDITION_DONT_CARE</a>  <a href="#">PS6000_CONDITION_TRUE</a>  <a href="#">PS6000_CONDITION_FALSE</a></p> <p>The channels that are set to <a href="#">PS6000_CONDITION_TRUE</a> or <a href="#">PS6000_CONDITION_FALSE</a> must all meet their conditions simultaneously to produce a trigger. Channels set to <a href="#">PS6000_CONDITION_DONT_CARE</a> are ignored.</p> <p><code>external</code>: not used</p>
-----------------	---

## 3.9.47 ps6000SetTriggerChannelDirections

```
PICO_STATUS ps6000SetTriggerChannelDirections
(
    short                handle,
    PS6000_THRESHOLD_DIRECTION channelA,
    PS6000_THRESHOLD_DIRECTION channelB,
    PS6000_THRESHOLD_DIRECTION channelC,
    PS6000_THRESHOLD_DIRECTION channelD,
    PS6000_THRESHOLD_DIRECTION ext,
    PS6000_THRESHOLD_DIRECTION aux
)
```

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

<b>Applicability</b>	All modes
<b>Arguments</b>	<p>handle, the handle of the required device</p> <p>channelA, channelB, channelC, channelD, aux, the direction in which the signal must pass through the threshold to activate the trigger. See the <a href="#">table</a> below for allowable values. If using a level trigger in conjunction with a pulse-width trigger, see the description of the direction argument to <a href="#">ps6000SetPulseWidthQualifier</a> for more information.</p> <p>ext: not used</p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_USER_CALLBACK</p> <p>PICO_INVALID_PARAMETER</p>

PS6000\_THRESHOLD\_DIRECTION constants

PS6000_ABOVE	for gated triggers: above the upper threshold
PS6000_ABOVE_LOWER	for gated triggers: above the lower threshold
PS6000_BELOW	for gated triggers: below the upper threshold
PS6000_BELOW_LOWER	for gated triggers: below the lower threshold
PS6000_RISING	for threshold triggers: rising edge, using upper threshold
PS6000_RISING_LOWER	for threshold triggers: rising edge, using lower threshold
PS6000_FALLING	for threshold triggers: falling edge, using upper threshold
PS6000_FALLING_LOWER	for threshold triggers: falling edge, using lower threshold
PS6000_RISING_OR_FALLING	for threshold triggers: either edge
PS6000_INSIDE	for window-qualified triggers: inside window
PS6000_OUTSIDE	for window-qualified triggers: outside window
PS6000_ENTER	for window triggers: entering the window
PS6000_EXIT	for window triggers: leaving the window
PS6000_ENTER_OR_EXIT	for window triggers: either entering or leaving the window
PS6000_POSITIVE_RUNT	for window-qualified triggers
PS6000_NEGATIVE_RUNT	for window-qualified triggers
PS6000_NONE	no trigger

## 3.9.48 ps6000SetTriggerChannelProperties

```
PICO_STATUS ps6000SetTriggerChannelProperties
(
    short                handle,
    PS6000_TRIGGER_CHANNEL_PROPERTIES * channelProperties
    short                nChannelProperties
    short                auxOutputEnable,
    long                 autoTriggerMilliseconds
)
```

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

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

## 3.9.48.1 TRIGGER\_CHANNEL\_PROPERTIES structure

A structure of this type is passed to [ps6000SetTriggerChannelProperties](#) in the `channelProperties` argument to specify the trigger mechanism, and is defined as follows: -

```
typedef struct tTriggerChannelProperties
{
    short                thresholdUpper;
    unsigned short      hysteresisUpper;
    short                thresholdLower;
    unsigned short      hysteresisLower;
    PS6000_CHANNEL      channel;
    PS6000_THRESHOLD_MODE thresholdMode;
} PS6000_TRIGGER_CHANNEL_PROPERTIES
```

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

Elements	
	<p><code>thresholdUpper</code>, the upper threshold at which the trigger must fire. This is scaled in 16-bit <a href="#">ADC counts</a> at the currently selected range for that channel.</p>
	<p><code>thresholdUpperHysteresis</code>, the hysteresis by which the trigger must exceed the upper threshold before it will fire. It is scaled in 16-bit counts.</p>
	<p><code>thresholdLower</code>, the lower threshold at which the trigger must fire. This is scaled in 16-bit <a href="#">ADC counts</a> at the currently selected range for that channel.</p>
	<p><code>thresholdLowerHysteresis</code>, the hysteresis by which the trigger must exceed the lower threshold before it will fire. It is scaled in 16-bit counts.</p>
	<p><code>channel</code>, the channel to which the properties apply. This can be one of the four input channels listed under <a href="#">ps6000SetChannel</a>, or <a href="#">PS6000_TRIGGER_AUX</a> for the AUX input.</p>
	<p><code>thresholdMode</code>, either a level or window trigger. Use one of these constants: -</p> <pre>    PS6000_LEVEL     PS6000_WINDOW</pre>

```

3.9.49 ps6000SetTriggerDelay
    PICO_STATUS ps6000SetTriggerDelay
    (
        short      handle,
        unsigned long delay
    )

```

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

<b>Applicability</b>	All modes
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>delay</code>, the time between the trigger occurring and the first sample. For example, if <code>delay=100</code> then the scope would wait 100 sample periods before sampling. At a <a href="#">timebase</a> of 5 GS/s, or 200 ps per sample (<code>timebase = 0</code>), the total delay would then be <math>800 \times 200 \text{ ps} = 160 \text{ ns}</math>. Range: 0 to <a href="#">MAX_DELAY_COUNT</a></p>
<b>Returns</b>	<p>PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_DRIVER_FUNCTION</p>

## 3.9.50 ps6000SetWaveformLimiter

```
PICO_STATUS ps6000SetWaveformLimiter
(
    short          handle,
    unsigned long  nWaveformsPerSecond
)
```

This function sets a limit to the number of waveforms per second transferred over the USB connection in [rapid block mode](#). The driver will wait between captures, if necessary, to obtain the requested waveform rate.

<b>Applicability</b>	<a href="#">Rapid block mode</a>
<b>Arguments</b>	<p>handle, the handle of the required device</p> <p>nWaveformsPerSecond, the maximum number of waveforms per second</p>
<b>Returns</b>	<p>PICO_OK</p> <p>PICO_INVALID_HANDLE</p> <p>PICO_USER_CALLBACK</p> <p>PICO_DRIVER_FUNCTION</p> <p>PICO_INVALID_PARAMETER</p>

## 3.9.51 ps6000SigGenSoftwareControl

```
PICO_STATUS ps6000SigGenSoftwareControl
(
    short    handle,
    short    state
)
```

This function causes a trigger event, or starts and stops gating. It is used when the signal generator is set to [SIGGEN\\_SOFT\\_TRIG](#).

<b>Applicability</b>	Use with <a href="#">ps6000SetSigGenBuiltIn</a> or <a href="#">ps6000SetSigGenArbitrary</a> .
<b>Arguments</b>	<p><code>handle</code>, the handle of the required device</p> <p><code>state</code>, sets the trigger gate high or low when the trigger type is set to either <code>SIGGEN_GATE_HIGH</code> or <code>SIGGEN_GATE_LOW</code>. Ignored for other trigger types.</p>
<b>Returns</b>	<p><code>PICO_OK</code></p> <p><code>PICO_INVALID_HANDLE</code></p> <p><code>PICO_NO_SIGNAL_GENERATOR</code></p> <p><code>PICO_SIGGEN_TRIGGER_SOURCE</code></p> <p><code>PICO_DRIVER_FUNCTION</code></p> <p><code>PICO_NOT_RESPONDING</code></p>

## 3.9.52 ps6000Stop

```
PICO_STATUS ps6000Stop
(
    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.

<b>Applicability</b>	All modes
<b>Arguments</b>	<code>handle</code> , the handle of the required device.
<b>Returns</b>	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_DRIVER_FUNCTION

## 3.9.53 ps6000StreamingReady

```

typedef void (CALLBACK *ps6000StreamingReady)
(
    short          handle,
    unsigned long  noOfSamples,
    unsigned long  startIndex,
    short          overflow,
    unsigned long  triggerAt,
    short          triggered,
    short          autoStop,
    void           * pParameter
)

```

This [callback](#) function is part of your application. You register it with the driver using [ps6000GetStreamingLatestValues](#), and the driver calls it back when streaming-mode data is ready. You can then download the data using the [ps6000GetValuesAsync](#) function.

<b>Applicability</b>	<a href="#">Streaming mode</a> only
<b>Arguments</b>	<p><code>handle</code>, the handle of the device returning the samples.</p> <p><code>noOfSamples</code>, the number of samples to collect.</p> <p><code>startIndex</code>, an index to the first valid sample in the buffer. This is the buffer that was previously passed to <a href="#">ps6000SetDataBuffer</a>.</p> <p><code>overflow</code>, returns a set of flags that indicate whether an overvoltage has occurred on any of the channels. It is a bit pattern with bit 0 denoting Channel A.</p> <p><code>triggerAt</code>, an index to the buffer indicating the location of the trigger point. This parameter is valid only when <code>triggered</code> is non-zero.</p> <p><code>triggered</code>, a flag indicating whether a trigger occurred. If non-zero, a trigger occurred at the location indicated by <code>triggerAt</code>.</p> <p><code>autoStop</code>, the flag that was set in the call to <a href="#">ps6000RunStreaming</a>.</p> <p><code>pParameter</code>, a void pointer passed from <a href="#">ps6000GetStreamingLatestValues</a>. The callback function can write to this location to send any data, such as a status flag, back to the application.</p>
<b>Returns</b>	nothing

## 3.10 Programming examples

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

- [C](#)
- [Visual Basic](#)
- [Excel](#)
- [LabView](#)

### 3.10.1 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: -

- `ps6000con.c`

and:

- `ps6000bc.lib` (Borland 32-bit applications) or
- `ps6000.lib` (Microsoft Visual C 32-bit applications)

The following files must be in the compilation directory:

- `ps6000Api.h`
- `picoStatus.h`

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

- `ps6000.dll`

### 3.10.2 Visual Basic

The `Examples/ps6000/` subdirectory of your PicoScope installation contains the following files:

- `ps6000.vbp` - project file
- `ps6000.bas` - procedure prototypes
- `ps6000.frm` - form and program

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

### 3.10.3 Excel

1. Load the spreadsheet `ps6000.xls`
2. Select **Tools | Macro**
3. Select **GetData**
4. Select **Run**

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

### 3.10.4 LabView

The SDK contains a library of VIs that can be used to control the PicoScope 6000 and some simple examples of using these VIs in [streaming mode](#), [block mode](#) and [rapid block mode](#).

The LabVIEW library (`PicoScope6000.llb`) can be placed in the `user.lib` sub-directory to make the VIs available on the 'User Libraries' palette. You must also copy `ps6000.dll` and `ps6000wrap.dll` to the folder containing your LabView project.

The library contains the following VIs:

- `PicoErrorHandler.vi` - takes an error cluster and, if an error has occurred, displays a message box indicating the source of the error and the status code returned by the driver
- `PicoScope6000AdvancedTriggerSettings.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 `PicoScope6000Settings.vi`.

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

```
ps6000SetTriggerChannelConditions
ps6000SetTriggerChannelDirections
ps6000SetTriggerChannelProperties
ps6000SetPulseWidthQualifier
ps6000SetTriggerDelay
```

- `PicoScope6000AWG.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 `ps6000SetSigGenBuiltIn` and `ps6000SetSigGenArbitrary`.

- `PicoScope6000Close.vi` - closes the oscilloscope

Should be called before exiting an application.

- `PicoScope6000GetBlock.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 `PicoScope6000Settings.vi`. The VI outputs data arrays in two clusters (max and min). If not using aggregation, 'Min Buffers' is not used.

- `PicoScope6000GetRapidBlock.vi` - collects a set of data blocks or captures from the oscilloscope in [rapid block mode](#)

This VI is similar to `PicoScope6000GetBlock.vi`. It outputs two-dimensional arrays for each channel that contain data from all the requested number of captures.

- `PicoScope6000GetStreamingValues.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 `PicoScope6000Settings.vi` and streaming has been started by calling `PicoScope6000StartStreaming.vi`. The VI outputs the number of samples available and the start index of these samples in the array output by `PicoScope6000StartStreaming.vi`.

- `PicoScope6000Open.vi` - opens a PicoScope 6000 and returns a handle to the device
- `PicoScope6000Settings.vi` - sets up the oscilloscope

The inputs are clusters for setting up channels and simple triggers. Advanced triggers can be set up using `PicoScope6000AdvancedTriggerSettings.vi`.

- `PicoScope6000StartStreaming.vi` - starts the oscilloscope [streaming](#)

It outputs arrays that will contain samples once `PicoScope6000GetStreamingValues.vi` has returned.

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

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

### 3.11 Driver status codes

Every function in the ps6000 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 PicoScope 6000 Series SDK. Not all codes apply to the PicoScope 6000 Series SDK.

#### Code (hex) / Symbol and meaning

00	PICO_OK	- The PicoScope 6000 is functioning correctly.
01	PICO_MAX_UNITS_OPENED	- An attempt has been made to open more than PS6000_MAX_UNITS.
02	PICO_MEMORY_FAIL	- Not enough memory could be allocated on the host machine.
03	PICO_NOT_FOUND	- No PicoScope 6000 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 6000 is not responding to commands from the PC.
08	PICO_CONFIG_FAIL	- The configuration information in the PicoScope 6000 has become corrupt or is missing.
09	PICO_KERNEL_DRIVER_TOO_OLD	- The <code>picopp.sys</code> 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 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 [ps6000BlockReady](#) 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 [ps6000Stop](#), or use [ps6000GetStreamingLatestValues](#).
- 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.
- 2D PICO\_MEMORY - Driver cannot allocate memory.
- 2E PICO\_SIG\_GEN\_PARAM - Incorrect parameter passed to signal generator.
- 34 PICO\_WARNING\_AUX\_OUTPUT\_CONFLICT - AUX cannot be used as input and output at the same time.
- 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.
- 45 PICO\_INVALID\_COUPLING - An invalid coupling type was specified in [ps6000SetChannel](#).
- 46 PICO\_BUFFERS\_NOT\_SET - An attempt was made to get data before a [data buffer](#) was defined.
- 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 [ps6000SetTriggerChannelProperties](#).
- 4A PICO\_INTERFACE\_NOT\_CONNECTED - The driver was unable to contact the oscilloscope.

- 4D PICO\_SIGGEN\_WAVEFORM\_SETUP\_FAILED - A problem occurred in [ps6000SetSigGenBuiltIn](#) or [ps6000SetSigGenArbitrary](#).
- 4E PICO\_FPGA\_FAIL
- 4F PICO\_POWER\_MANAGER
- 50 PICO\_INVALID\_ANALOGUE\_OFFSET - An impossible analogue offset value was specified in [ps6000SetChannel](#).
- 51 PICO\_PLL\_LOCK\_FAILED - Unable to configure the PicoScope 6000.
- 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  $\pm 5\%$  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 configure 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.
- 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.

### 3.12 Enumerated types and constants

Here are the enumerated types used in the PicoScope 6000 Series SDK, as defined in the file `ps6000Api.h`. We recommend that you refer to these constants by name unless your programming language allows only numerical values.

```
#define PS6000_MAX_OVERSAMPLE_8BIT 256

#define PS6000_MAX_VALUE 32512
#define PS6000_MIN_VALUE -32512

#define MAX_PULSE_WIDTH_QUALIFIER_COUNT 16777215L

#define MAX_SIG_GEN_BUFFER_SIZE 16384
#define MIN_SIG_GEN_BUFFER_SIZE 10
#define MIN_DWELL_COUNT 10
#define MAX_SWEEPS_SHOTS ((1 << 30) - 1)

#define MAX_WAVEFORMS_PER_SECOND 1000000

#define MAX_ANALOGUE_OFFSET_50MV_200MV 0.500f
#define MIN_ANALOGUE_OFFSET_50MV_200MV -0.500f
#define MAX_ANALOGUE_OFFSET_500MV_2V 2.500f
#define MIN_ANALOGUE_OFFSET_500MV_2V -2.500f
#define MAX_ANALOGUE_OFFSET_5V_20V 20.f
#define MIN_ANALOGUE_OFFSET_5V_20V -20.f

#define PS6000_MAX_ETS_CYCLES 250
#define PS6000_MAX_INTERLEAVE 50

typedef enum enPS6000ExternalFrequency
{
    PS6000_FREQUENCY_OFF,
    PS6000_FREQUENCY_5MHZ,
    PS6000_FREQUENCY_10MHZ,
    PS6000_FREQUENCY_20MHZ,
    PS6000_FREQUENCY_25MHZ,
    PS6000_MAX_FREQUENCIES
} PS6000_EXTERNAL_FREQUENCY;

typedef enum enPS6000BandwidthLimiter
{
    PS6000_BW_FULL,
    PS6000_BW_20MHZ,
    PS6000_BW_25MHZ,
} PS6000_BANDWIDTH_LIMITER;

typedef enum enPS6000Channel
{
    PS6000_CHANNEL_A,
    PS6000_CHANNEL_B,
    PS6000_CHANNEL_C,
    PS6000_CHANNEL_D,
    PS6000_EXTERNAL,
    PS6000_MAX_CHANNELS = PS6000_EXTERNAL,
    PS6000_TRIGGER_AUX,
    PS6000_MAX_TRIGGER_SOURCES
} PS6000_CHANNEL;

typedef enum enPS6000ChannelBufferIndex
{
    PS6000_CHANNEL_A_MAX,
    PS6000_CHANNEL_A_MIN,
    PS6000_CHANNEL_B_MAX,
    PS6000_CHANNEL_B_MIN,
    PS6000_CHANNEL_C_MAX,
    PS6000_CHANNEL_C_MIN,
    PS6000_CHANNEL_D_MAX,
    PS6000_CHANNEL_D_MIN,
    PS6000_MAX_CHANNEL_BUFFERS
} PS6000_CHANNEL_BUFFER_INDEX;

typedef enum enPS6000Range
{
    PS6000_10MV,
    PS6000_20MV,
    PS6000_50MV,
    PS6000_100MV,
```

```

    PS6000_200MV,
    PS6000_500MV,
    PS6000_1V,
    PS6000_2V,
    PS6000_5V,
    PS6000_10V,
    PS6000_20V,
    PS6000_50V,
    PS6000_MAX_RANGES
} PS6000_RANGE;

typedef enum enPS6000Coupling
{
    PS6000_AC,
    PS6000_DC_1M,
    PS6000_DC_50R
} PS6000_COUPLING;

typedef enum enPS6000EtsMode
{
    PS6000_ETS_OFF,
    PS6000_ETS_FAST,
    PS6000_ETS_SLOW,
    PS6000_ETS_MODES_MAX
} PS6000_ETS_MODE;

typedef enum enPS6000TimeUnits
{
    PS6000_FS,
    PS6000_PS,
    PS6000_NS,
    PS6000_US,
    PS6000_MS,
    PS6000_S,
    PS6000_MAX_TIME_UNITS,
} PS6000_TIME_UNITS;

typedef enum enPS6000SweepType
{
    PS6000_UP,
    PS6000_DOWN,
    PS6000_UPDOWN,
    PS6000_DOWNUP,
    PS6000_MAX_SWEEP_TYPES
} PS6000_SWEEP_TYPE;

typedef enum enPS6000WaveType
{
    PS6000_SINE,
    PS6000_SQUARE,
    PS6000_TRIANGLE,
    PS6000_RAMP_UP,
    PS6000_RAMP_DOWN,
    PS6000_SINC,
    PS6000_GAUSSIAN,
    PS6000_HALF_SINE,
    PS6000_DC_VOLTAGE,
    PS6000_WHITE_NOISE,
    PS6000_MAX_WAVE_TYPES
} PS6000_WAVE_TYPE;

typedef enum enPS6000ExtraOperations
{
    PS6000_ES_OFF,
    PS6000_WHITENOISE,
    PS6000_PRBS // Pseudo-Random Bit Stream
} PS6000_EXTRA_OPERATIONS;

#define PS6000_SINE_MAX_FREQUENCY 20000000.f
#define PS6000_SQUARE_MAX_FREQUENCY 20000000.f
#define PS6000_TRIANGLE_MAX_FREQUENCY 20000000.f
#define PS6000_SINC_MAX_FREQUENCY 20000000.f
#define PS6000_RAMP_MAX_FREQUENCY 20000000.f
#define PS6000_HALF_SINE_MAX_FREQUENCY 20000000.f
#define PS6000_GAUSSIAN_MAX_FREQUENCY 20000000.f
#define PS6000_MIN_FREQUENCY 0.03f

typedef enum enPS6000SigGenTrigType
{
    PS6000_SIGGEN_RISING,
    PS6000_SIGGEN_FALLING,

```

```

    PS6000_SIGGEN_GATE_HIGH,
    PS6000_SIGGEN_GATE_LOW
} PS6000_SIGGEN_TRIG_TYPE;

typedef enum enPS6000SigGenTrigSource
{
    PS6000_SIGGEN_NONE,
    PS6000_SIGGEN_SCOPE_TRIG,
    PS6000_SIGGEN_AUX_IN,
    PS6000_SIGGEN_EXT_IN,
    PS6000_SIGGEN_SOFT_TRIG,
    PS6000_SIGGEN_TRIGGER_RAW
} PS6000_SIGGEN_TRIG_SOURCE;

typedef enum enPS6000IndexMode
{
    PS6000_SINGLE,
    PS6000_DUAL,
    PS6000_QUAD,
    PS6000_MAX_INDEX_MODES
} PS6000_INDEX_MODE;

typedef enum enPS6000ThresholdMode
{
    PS6000_LEVEL,
    PS6000_WINDOW
} PS6000_THRESHOLD_MODE;

typedef enum enPS6000ThresholdDirection
{
    PS6000_ABOVE,
    PS6000_BELOW,
    PS6000_RISING,
    PS6000_FALLING,
    PS6000_RISING_OR_FALLING,
    PS6000_ABOVE_LOWER,
    PS6000_BELOW_LOWER,
    PS6000_RISING_LOWER,
    PS6000_FALLING_LOWER,

    // Windowing using both thresholds
    PS6000_INSIDE = PS6000_ABOVE,
    PS6000_OUTSIDE = PS6000_BELOW,
    PS6000_ENTER = PS6000_RISING,
    PS6000_EXIT = PS6000_FALLING,
    PS6000_ENTER_OR_EXIT = PS6000_RISING_OR_FALLING,
    PS6000_POSITIVE_RUNT = 9,
    PS6000_NEGATIVE_RUNT,

    // no trigger set
    PS6000_NONE = PS6000_RISING
} PS6000_THRESHOLD_DIRECTION;

typedef enum enPS6000TriggerState
{
    PS6000_CONDITION_DONT_CARE,
    PS6000_CONDITION_TRUE,
    PS6000_CONDITION_FALSE,
    PS6000_CONDITION_MAX
} PS6000_TRIGGER_STATE;

typedef enum enPS6000RatioMode
{
    PS6000_RATIO_MODE_NONE,
    PS6000_RATIO_MODE_AGGREGATE = 1,
    PS6000_RATIO_MODE_AVERAGE = 2,
    PS6000_RATIO_MODE_DECIMATE = 4,
    PS6000_RATIO_MODE_DISTRIBUTION = 8
} PS6000_RATIO_MODE;

typedef enum enPS6000PulseWidthType
{
    PS6000_PW_TYPE_NONE,
    PS6000_PW_TYPE_LESS_THAN,
    PS6000_PW_TYPE_GREATER_THAN,
    PS6000_PW_TYPE_IN_RANGE,
    PS6000_PW_TYPE_OUT_OF_RANGE
} PS6000_PULSE_WIDTH_TYPE;

```

### 3.13 Numeric data types

Here is a list of the sizes and ranges of the numeric data types used in the PicoScope 6000 Series API.

<b>Type</b>	<b>Bits</b>	<b>Signed or unsigned?</b>
short	16	signed
enum	32	enumerated
int	32	signed
long	32	signed
unsigned long	32	unsigned
float	32	signed (IEEE 754)
__int64	64	unsigned

## 4 Glossary

**Callback.** A mechanism that the PicoScope 6000 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.** A Windows program that displays the current hardware configuration of your computer. On Windows XP, Vista, or 7, right-click 'My Computer,' choose 'Properties', then click the 'Hardware' tab and the 'Device Manager' button.

**Driver.** A program that controls a piece of hardware. The driver for the PicoScope 6000 Series oscilloscopes is supplied in the form of a 32-bit Windows DLL, `ps6000.dll`. This is used by the PicoScope software, and by user-designed applications, to control the oscilloscopes.

**PC Oscilloscope.** A virtual instrument formed by connecting a PicoScope 6000 Series oscilloscope to a computer running the PicoScope software.

**PicoScope 6000 Series.** A range of PC Oscilloscopes from Pico Technology. The common features include 5 GS/s sampling and 8-bit resolution. The scopes are available with a range of buffer sizes up to 1 GS.

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

**PRBS (pseudo-random binary sequence).** A very long, fixed, repeating sequence of binary digits that appears random when analyzed over a time shorter than the repeat period. The waveform swings between two values: logic high (binary 1) and logic low (binary 0).

**USB (Universal Serial Bus).** A serial port used to connect external devices to PCs. USB 2.0 ports supports data transfer rates of up to 480 megabits per second.



# Index

## A

- AC coupling 55
- Aggregation 16, 36
- Analog offset 26, 55
- API function calls 20
- Arbitrary waveform generator 69
  - index modes 71
- Averaging 36

## B

- Bandwidth limiter 55
- Block mode 6, 7, 8
  - asynchronous call 8
  - callback 21
  - polling status 44
  - running 51
- Buffers
  - overrun 5

## C

- C programming 85
- Callback function
  - block mode 21
  - for data 23
  - streaming mode 84
- Channels
  - enabling 55
  - settings 55
- Clock, external 64
- Closing units 22
- Company information 2
- Constants 91
- Contact details 2
- Coupling type, setting 55

## D

- Data acquisition 16
- Data buffers
  - declaring 57
  - declaring, aggregation mode 59
  - declaring, rapid block mode 58
  - setting up 60
- DC coupling 55
- Decimation 36
- Disk space 3

- Distribution 36
- Downsampling 35
  - maximum ratio 27
  - modes 36
- Driver 5
  - status codes 88

## E

- Enabling channels 55
- Enumerated types 91
- Enumerating oscilloscopes 24
- ETS
  - overview 14
  - setting time buffers 62, 63
  - setting up 61
  - using 14
- Excel macros 85
- External clock 64

## F

- Function calls 20
- Functions
  - ps6000BlockReady 21
  - ps6000CloseUnit 22
  - ps6000DataReady 23
  - ps6000EnumerateUnits 24
  - ps6000FlashLed 25
  - ps6000GetAnalogueOffset 26
  - ps6000GetMaxDownSampleRatio 27
  - ps6000GetNoOfCaptures 28
  - ps6000GetStreamingLatestValues 29
  - ps6000GetTimebase 30
  - ps6000GetTimebase2 31
  - ps6000GetTriggerTimeOffset 32
  - ps6000GetTriggerTimeOffset64 33
  - ps6000GetUnitInfo 34
  - ps6000GetValues 35
  - ps6000GetValuesAsync 37
  - ps6000GetValuesBulk 38
  - ps6000GetValuesBulkAsync 39
  - ps6000GetValuesOverlapped 40
  - ps6000GetValuesOverlappedBulk 41
  - ps6000GetValuesTriggerTimeOffsetBulk 42
  - ps6000GetValuesTriggerTimeOffsetBulk64 43
  - ps6000IsReady 44
  - ps6000IsTriggerOrPulseWidthQualifierEnabled 45
  - ps6000MemorySegments 46
  - ps6000NoOfStreamingValues 47
  - ps6000OpenUnit 48
  - ps6000OpenUnitAsync 49

## Functions

- ps6000OpenUnitProgress 50
- ps6000RunBlock 51
- ps6000RunStreaming 53
- ps6000SetChannel 55
- ps6000SetDataBuffer 57
- ps6000SetDataBufferBulk 58
- ps6000SetDataBuffers 59
- ps6000SetDataBuffersBulk 60
- ps6000SetEts 61
- ps6000SetEtsTimeBuffer 62
- ps6000SetEtsTimeBuffers 63
- ps6000SetExternalClock 64
- ps6000SetNoOfCaptures 65
- ps6000SetPulseWidthQualifier 66
- ps6000SetSigGenArbitrary 69
- ps6000SetSigGenBuiltIn 72
- ps6000SetSimpleTrigger 74
- ps6000SetTriggerChannelConditions 75
- ps6000SetTriggerChannelDirections 77
- ps6000SetTriggerChannelProperties 78
- ps6000SetTriggerDelay 80
- ps6000SetWaveformLimiter 81
- ps6000SigGenSoftwareControl 82
- ps6000Stop 83
- ps6000StreamingReady 84

## H

- Hysteresis 79

## I

- Information, reading from units 34
- Input range, selecting 55
- Installation 4

## L

- LabView 86
- LED
  - flashing 25

## M

- Macros in Excel 85
- Memory in scope 7
- Memory segments 46
- Microsoft Windows 3
- Multi-unit operation 19

## N

- Numeric data types 94

## O

- One-shot signals 14
- Opening a unit 48
  - checking progress 50
  - without blocking 49
- Operating system 3
- Oversampling 17

## P

- Pico Technical Support 2
- PICO\_STATUS enum type 88
- picopp.inf 5
- picopp.sys 5
- PicoScope 6000 Series 1
- PicoScope software 4, 5, 88
- Processor 3
- Programming
  - C 85
  - Excel 85
  - LabView 86
  - Visual Basic 85
- PS6000\_CONDITION\_ constants 68, 76
- PS6000\_LEVEL constant 79
- PS6000\_LOST\_DATA constant 5
- PS6000\_MAX\_VALUE constant 5
- PS6000\_MIN\_VALUE constant 5
- PS6000\_PWQ\_CONDITIONS structure 68
- PS6000\_TIME\_UNITS constant 32
- PS6000\_TRIGGER\_CHANNEL\_PROPERTIES structure 79
- PS6000\_TRIGGER\_CONDITIONS structure 76
- PS6000\_WINDOW constant 79
- Pulse-width qualifier 66
  - conditions 68
  - requesting status 45

## R

- Rapid block mode 9
  - setting number of captures 65
- Resolution, vertical 17
- Retrieving data 35, 37
  - block mode, deferred 40
  - rapid block mode 38
  - rapid block mode with callback 39
  - rapid block mode, deferred 41

Retrieving data 35, 37  
  stored 17  
  streaming mode 29  
Retrieving times  
  rapid block mode 42, 43

## S

Sampling rate  
  maximum 7  
Scaling 5  
Serial numbers 24  
Signal generator 8  
  arbitrary waveforms 69  
  built-in waveforms 72  
  software trigger 82  
Software licence conditions 1  
Status codes 88  
Stopping sampling 83  
Streaming mode 6, 16  
  callback 84  
  getting number of samples 47  
  retrieving data 29  
  running 53  
  using 16  
Synchronising units 19  
System memory 3  
System requirements 3

## T

Technical support 2  
Threshold voltage 6  
Time buffers  
  setting for ETS 62, 63  
Timebase 18  
  calculating 30, 31  
Trademarks 2  
Trigger 6  
  channel properties 78  
  conditions 75, 76  
  delay 80  
  directions 77  
  pulse-width qualifier 66  
  pulse-width qualifier conditions 68  
  requesting status 45  
  setting up 74  
  time offset 32, 33

## U

USB 3, 5  
  changing ports 4

hub 19

## V

Vertical resolution 17  
Visual Basic programming 85  
Voltage ranges 5  
  selecting 55

## W

Waveform limiter 81







## 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](http://www.picotech.com)

ps6000pg.en-5

2.9.11

Copyright © 2009-2011 Pico Technology Ltd. All rights reserved.