PicoScope 6000 Series
PC Oscilloscopes

Programmer's Guide

ps6000pg.en-1


pico
Technology

Contents

1 Welcome

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

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

2 Introduction

2.1 Software licence conditions

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

Access

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

Usage

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

Copyright

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

Liability

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

Fitness for purpose

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

Mission-critical applications

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

2.2 Trademarks

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

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

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

2.3 Company details

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

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

Email:

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

Web site: www.picotech.com

3 Product information

3.1 System requirements

Using with PicoScope for Windows

To ensure that your PicoScope 6000 Series oscilloscope operates correctly with the PicoScope software, you must have a computer with at least the minimum system requirements to run one of the supported operating systems, as shown in the following table. The performance of the software will increase with more powerful PCs, including those with multi-core processors.

Item	Absolute minimum	Recommended minimum	Recommended full specification
Operating system	Windows XP SP2, Vista or Windows 7 (32-bit versions only)		
Processor	As required by Windows	300 MHz	1 GHz
Memory		256 MB	512 MB
Free disk space (Note 1)		1 GB	2 GB
Ports	USB 1.1 compliant port	USB 2.0 compliant port	

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

Using with custom applications

Drivers are available for Windows XP (SP2), Windows Vista and Windows 7. System specifications for Windows are the same as under "Using with PicoScope for Windows", above.

3.2 Installation instructions

IMPORTANT
Install the Pico 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 SP2**

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**

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.

4 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.

4.1 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 SP2, 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.

4.2 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.

4.3 Voltage ranges

You can set a device input channel to any voltage range from ±50 mV to ±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

4.4 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.

4.5 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.

- **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.

4.5.1 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.

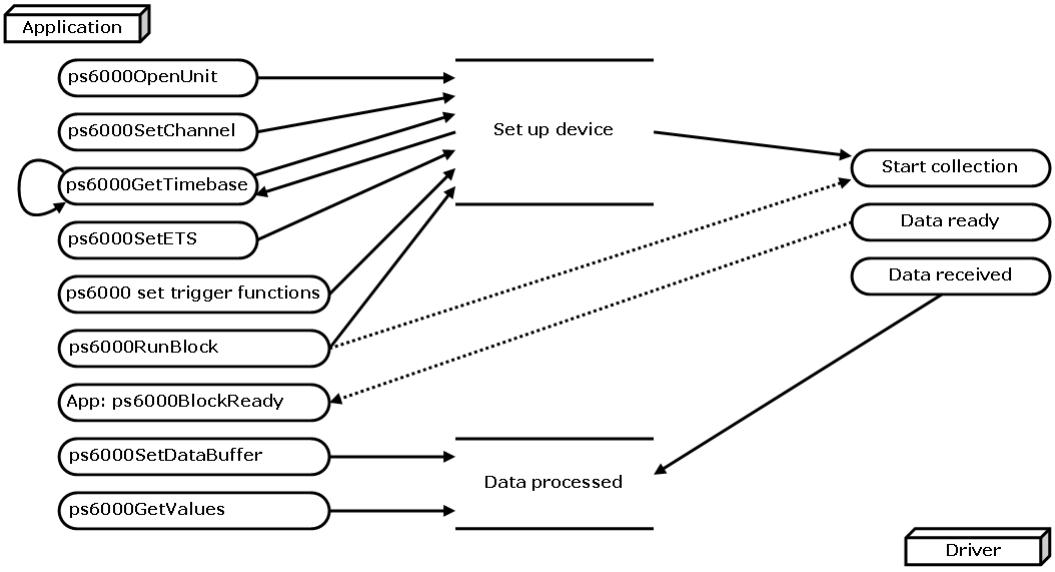
- **Data retention.** The data is lost when a new run is started in the same segment, the settings are changed, or the scope is powered down.

See Using block mode for programming details.

4.5.1.1 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.
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.

4.5.1.2 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.

4.5.2 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.

4.5.2.1 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.

4.5.2.2 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
  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 `toSegementIndex`, 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.

4.5.2.3 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.

4.5.3 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.

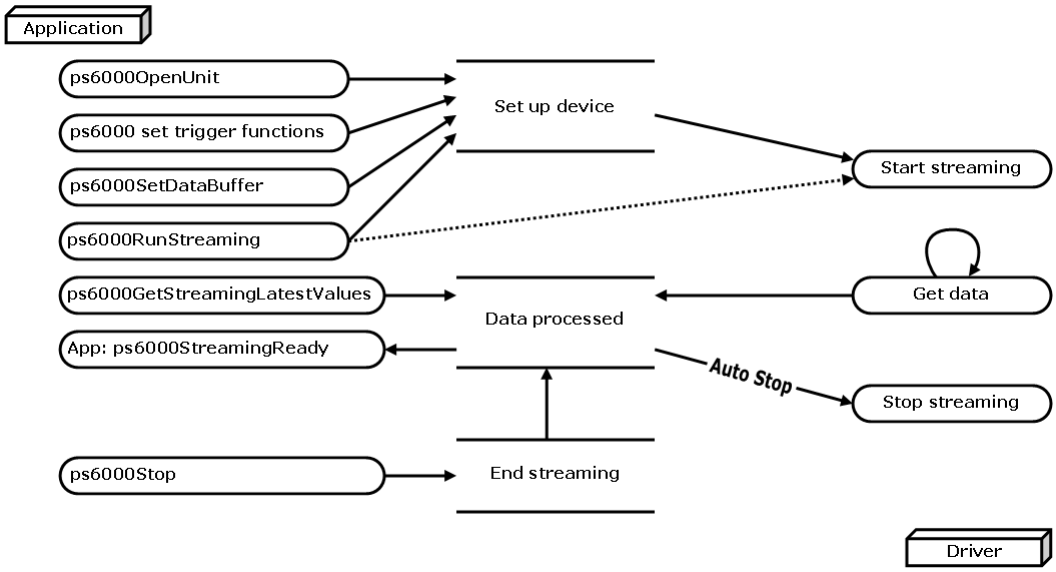
- **Memory segmentation.** The memory can be divided into segments to reduce the latency of data transfers to the PC. However, this increases the risk of losing data if the PC cannot keep up with the device's sampling rate.

See Using streaming mode for programming details.

4.5.3.1 Using streaming mode

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

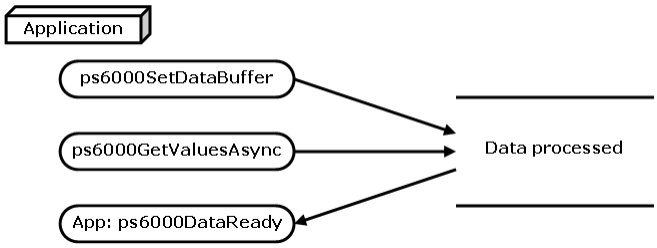
1. Open the oscilloscope using ps6000OpenUnit.
2. Select channels, ranges and AC/DC coupling using ps6000SetChannel.
3. Use the trigger setup functions ps6000SetTriggerChannelConditions, ps6000SetTriggerChannelDirections and ps6000SetTriggerChannelProperties to set up the trigger if required.
4. Call 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.

4.5.4 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.



4.6 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.

Applicability	Available in block mode only. Cannot be used at the same time as downsampling.

4.7 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.

timebase	sample interval formula	sample interval examples
0 to 4	$2^{timebase}$ / 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	(timebase - 4) / 156,250,000	5 => 6.4 ns ... 2^{32}-1 => ~ 6.87 s

Applicability	Use ps6000GetTimebase API call.

4.8 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.

4.9 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.

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

4.9.1 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.

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

4.9.2 ps6000CloseUnit

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

This function shuts down a PicoScope 6000 oscilloscope.

Applicability	All modes
Arguments	`handle`, the handle, returned by ps6000OpenUnit, of the scope device to be closed.

Returns	`PICO_OK`
	`PICO_HANDLE_INVALID`
	`PICO_USER_CALLBACK`
	`PICO_DRIVER_FUNCTION`

4.9.3 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.

Applicability	All modes
Arguments	`handle,` the handle of the device returning the samples. `status,` a `PICO_STATUS` code returned by the driver. `noOfSamples,` the number of samples collected. `overflow,` a set of flags that indicates whether an overvoltage has occurred and on which channels. It is a bit field with bit 0 representing Channel A. `pParameter,` a void pointer passed from ps6000GetValuesAsync. The callback function can write to this location to send any data, such as a status flag, back to the application. The data type is defined by the application programmer.
Returns	nothing

4.9.4 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.

Applicability	All modes
Arguments	`* count,` on exit, the number of PicoScope 6000 units found `* serials,` on exit, a list of serial numbers separated by commas and terminated by a final null. Example: `AQ005/139,VDR61/356, ZOR14/107`. Can be NULL on entry if serial numbers are not required. `* serialLth,` on entry, the length of the char buffer pointed to by `serials;` on exit, the length of the string written to `serials`

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

4.9.5 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.

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

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

4.9.6 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.

Applicability	All modes
Arguments	`handle`, the handle of the required device `noOfUnaggregatedSamples`, the number of unprocessed samples to be downsampled `maxDownSampleRatio`: the maximum possible downsampling ratio `downSampleRatioMode`: the downsampling mode. See ps6000GetValues. `segmentIndex`, the memory segment where the data is stored

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

4.9.7 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.

Applicability	Streaming mode only
Arguments	`handle`, the handle of the required device. `lpPs6000Ready`, a pointer to your ps6000StreamingReady callback function. `pParameter`, a void pointer that will be passed to the ps6000StreamingReady callback function. The callback function may optionally use this pointer to return information to the application.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_INVALID_CALL PICO_BUSY PICO_NOT_RESPONDING PICO_DRIVER_FUNCTION PICO_STARTINDEX_INVALID

4.9.8 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. Next, call one of these functions with the timebase that you have just chosen and verify that the `timeIntervalNanoseconds` argument that the function returns is the value that you require. You may need to iterate this process until you obtain the time interval that you need.

Applicability	All modes
Arguments	`handle,` the handle of the required device. `timebase,` see timebase guide `noSamples,` the number of samples required. This value is used to calculate the most suitable time unit to use. `timeIntervalNanoseconds,` on exit, the time interval between readings at the selected timebase. Use NULL if not required. `oversample,` the amount of oversample required (see Oversampling). Range: 0 to `PS6000_MAX_OVERSAMPLE_8BIT`. `maxSamples,` 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. `segmentIndex,` the index of the memory segment to use.

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

4.9.9 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.

Applicability	All modes
Arguments	`timeIntervalNanoseconds`, a pointer to the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. All other arguments: see ps6000GetTimebase.
Returns	See ps6000GetTimebase.

4.9.10 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.

Applicability	Block mode, rapid block mode
Arguments	`handle`, the handle of the required device `timeUpper`, on exit, the upper 32 bits of the time at which the trigger point occurred `timeLower`, on exit, the lower 32 bits of the time at which the trigger point occurred `timeUnits`, returns the time units in which `timeUpper` and `timeLower` are measured. The allowable values are: - PS6000_FS PS6000_PS PS6000_NS PS6000_US PS6000_MS PS6000_S `segmentIndex`, the number of the memory segment for which the information is required.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_DRIVER_FUNCTION

4.9.11 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.

Applicability	Block mode, rapid block mode
Arguments	`handle`, the handle of the required device `time`, on exit, the time at which the trigger point occurred `timeUnits`, on exit, the time units in which time is measured. The possible values are: - PS6000_FS PS6000_PS PS6000_NS PS6000_US PS6000_MS PS6000_S `segmentIndex`, the number of the memory segment for which the information is required
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_DRIVER_FUNCTION

4.9.12 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.

Applicability	All modes
Arguments	`handle`, the handle of the device from which information is required. If an invalid handle is passed, the error code from the last unit that failed to open is returned. `string`, on exit, the unit information string selected specified by the `info` argument. If `string` is NULL, only `requiredSize` is returned. `stringLength`, the maximum number of chars that may be written to `string`. `requiredSize`, on exit, the required length of the `string` array. `info`, a number specifying what information is required. The possible values are listed in the table below.

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

info		Example
0	PICO_DRIVER_VERSION Version number of PicoScope 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 Variant 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

4.9.13 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.

Applicability	Block mode, rapid block mode

Arguments	`handle`, the handle of the required device.
	
	`startIndex`, a zero-based index that indicates the start point for data collection. It is measured in sample intervals from the start of the buffer.
	
	`noOfSamples`, on entry, the number of samples required. On exit, the actual number retrieved. The number of samples retrieved will not be more than the number requested, and the data retrieved always starts with the first sample captured.
	
	`downSampleRatio`, the downsampling factor that will be applied to the raw data.
	
	`downSampleRatioMode`, which downsampling mode to use. The available values are: -
	`PS6000_RATIO_MODE_NONE` (`downSampleRatio` is ignored)
	`PS6000_RATIO_MODE_AGGREGATE`
	`PS6000_RATIO_MODE_AVERAGE`
	`PS6000_RATIO_MODE_DECIMATE`
	`PS6000_RATIO_MODE_DISTRIBUTION`
	`AGGREGATE`, `AVERAGE`, `DECIMATE` and `DISTRIBUTION` are single-bit constants that can be ORed to apply multiple downsampling modes to the same data.
	
	`segmentIndex`, the zero-based number of the memory segment where the data is stored.
	
	`overflow`, on exit, a set of flags that indicate whether an overvoltage has occurred on any of the channels. It is a bit field with bit 0 denoting Channel A.

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

4.9.13.1 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.
PS6000_RATIO_MODE_DISTRIBUTION	Reduces every block of *n* values to a histogram. Since the values are 8-bit numbers, there are 2^8 = 256 bins in each histogram. A histogram is treated as one "sample" by the data collection function, so the noOfSamples argument specifies the number of 256-bin histograms that will be written to the data buffer.

4.9.14 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.

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

Returns	`PICO_OK`
	`PICO_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`

4.9.15 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.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the device `* noOfSamples`, on entry, the number of samples required; on exit, the actual number retrieved. The number of samples retrieved will not be more than the number requested. The data retrieved always starts with the first sample captured. `fromSegmentIndex`, the first segment from which the waveform should be retrieved `toSegmentIndex`, the last segment from which the waveform should be retrieved `downSampleRatio`: see ps6000GetValues `downSampleRatioMode`: see ps6000GetValues `* overflow`, an array of integers equal to or larger than the number of waveforms to be retrieved. Each segment index has a corresponding entry in the `overflow` array, with `overflow[0]` containing the flags for the segment numbered `fromSegmentIndex` and the last element in the array containing the flags for the segment numbered `toSegmentIndex`. Each element in the array is a bit field as described under ps6000GetValues.

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

4.9.16 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.

Applicability	Rapid block mode
Arguments	handle, the handle of the device startIndex: see ps6000GetValues * noOfSamples: see ps6000GetValues downSampleRatio: see ps6000GetValues downSampleRatioMode: see ps6000GetValues fromSegmentIndex: see ps6000GetValuesBulk toSegmentIndex: see ps6000GetValuesBulk overflow: see ps6000GetValuesBulk
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE PICO_STARTINDEX_INVALID PICO_NOT_RESPONDING PICO_DRIVER_FUNCTION

4.9.17 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.

Applicability	Block mode
Arguments	handle, the handle of the device startIndex: see ps6000GetValues * noOfSamples: see ps6000GetValues downSampleRatio: see ps6000GetValues downSampleRatioMode: see ps6000GetValues segmentIndex: see ps6000GetValues * overflow: see ps6000GetValuesBulk
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

4.9.18 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.

Applicability	Rapid block mode
Arguments	handle, the handle of the device startIndex: see ps6000GetValues * noOfSamples: see ps6000GetValues downSampleRatio: see ps6000GetValues downSampleRatioMode: see ps6000GetValues fromSegmentIndex: see ps6000GetValuesBulk toSegmentIndex: see ps6000GetValuesBulk * overflow, see ps6000GetValuesBulk
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION

4.9.19 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.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the device `* timesUpper`, an array of integers. On exit, the most significant 32 bits of the time offset for each requested segment index. `times[0]` will hold the `fromSegmentIndex` time offset and the last `times` index will hold the `toSegmentIndex` time offset. The array must be long enough to hold the number of requested times. `* timesLower`, an array of integers. On exit, the least-significant 32 bits of the time offset for each requested segment index. `times[0]` will hold the `fromSegmentIndex` time offset and the last `times` index will hold the `toSegmentIndex` time offset. The array size must be long enough to hold the number of requested times. `* timeUnits`, an array of integers. The array must be long enough to hold the number of requested times. On exit, `timeUnits[0]` will contain the time unit for `fromSegmentIndex` and the last element will contain the time unit for `toSegmentIndex`. `fromSegmentIndex`, the first segment for which the time offset is required `toSegmentIndex`, the last segment for which the time offset is required. If `toSegmentIndex` is less than `fromSegmentIndex` then the driver will wrap around from the last segment to the first.

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

4.9.20 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.

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

4.9.21 ps6000IsTriggerOrPulseWidthQualifierEnabled

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

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

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

4.9.22 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.

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

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

4.9.23 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.

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

4.9.24 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.

Applicability	All modes
Arguments	* `handle`, on exit, the result of the attempt to open a scope: -1 : if the scope fails to open 0 : if no scope is found > 0 : a number that uniquely identifies the scope If a valid handle is returned, it must be used in all subsequent calls to API functions to identify this scope. `serial`, on entry, a null-terminated string containing the serial number of the scope to be opened. If `serial` is NULL then the function opens the first scope found; otherwise, it tries to open the scope that matches the string.

Returns	`PICO_OK`
	`PICO_OS_NOT_SUPPORTED`
	`PICO_OPEN_OPERATION_IN_PROGRESS`
	`PICO_EEPROM_CORRUPT`
	`PICO_KERNEL_DRIVER_TOO_OLD`
	`PICO_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`

4.9.25 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.

Applicability	All modes
Arguments	* `status`, a status code: 0 if the open operation was disallowed because another open operation is in progress 1 if the open operation was successfully started * `serial`: see ps6000OpenUnit
Returns	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_OPERATION_FAILED

4.9.26 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.

Applicability	Use after ps6000OpenUnitAsync
Arguments	* handle: see ps6000OpenUnit. This handle is valid only if the function returns PICO_OK. * progressPercent, on exit, the percentage progress towards opening the scope. 100% implies that the open operation is complete. * complete, set to 1 when the open operation has finished
Returns	PICO_OK PICO_NULL_PARAMETER PICO_OPERATION_FAILED

4.9.27 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`.

Applicability	Block mode, rapid block mode
Arguments	`handle`, the handle of the required device. `noOfPreTriggerSamples`, the number of samples to return before the trigger event. If no trigger has been set then this argument is ignored and `noOfPostTriggerSamples` specifies the maximum number of samples to collect. `noOfPostTriggerSamples`, 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: - `noOfPreTriggerSamples + noOfPostTriggerSamples` `timebase`, a number in the range 0 to 2^{32}-1. See the guide to calculating timebase values. `oversample`, the oversampling factor, a number in the range 1 to 256. `* timeIndisposedMs`, 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. `segmentIndex`, zero-based, specifies which memory segment to use. `lpReady`, a pointer to the ps6000BlockReady callback function that the driver will call when the data has been collected.

	* pParameter, a void pointer that is passed to the ps6000BlockReady callback function. The callback can use this pointer to return arbitrary data to the application.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_CHANNEL PICO_INVALID_TRIGGER_CHANNEL PICO_INVALID_CONDITION_CHANNEL PICO_TOO_MANY_SAMPLES PICO_INVALID_TIMEBASE PICO_NOT_RESPONDING PICO_CONFIG_FAIL PICO_INVALID_PARAMETER PICO_NOT_RESPONDING PICO_TRIGGER_ERROR 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

4.9.28 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.

Applicability	Streaming mode

Arguments	`handle`, the handle of the required device.
	
	`* sampleInterval`, on entry, the requested time interval between samples; on exit, the actual time interval used.
	
	`sampleIntervalTimeUnits`, the unit of time used for `sampleInterval`. Use one of these values:
	`PS6000_FS`
	`PS6000_PS`
	`PS6000_NS`
	`PS6000_US`
	`PS6000_MS`
	`PS6000_S`
	
	`maxPreTriggerSamples`, the maximum number of raw samples before a trigger event for each enabled channel. If no trigger condition is set this argument is ignored.
	
	`maxPostTriggerSamples`, the maximum number of raw samples after a trigger event for each enabled channel. If no trigger condition is set, this argument states the maximum number of samples to be stored.
	
	`autoStop`, a flag that specifies if the streaming should stop when all of `maxSamples` have been captured.
	
	`downSampleRatio`: see ps6000GetValues
	`downSampleRatioMode`: see ps6000GetValues

	overviewBufferSize, the size of the overview buffers. These are temporary buffers used for storing the data before returning it to the application. The size is the same as the bufferLth value passed to ps6000SetDataBuffer.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_NULL_PARAMETER PICO_INVALID_PARAMETER PICO_STREAMING_FAILED PICO_NOT_RESPONDING PICO_TRIGGER_ERROR PICO_INVALID_SAMPLE_INTERVAL PICO_INVALID_BUFFER PICO_DRIVER_FUNCTION PICO_EXTERNAL_FREQUENCY_INVALID PICO_FW_FAIL PICO_TRIGGER_AND_EXTERNAL_CLOCK_CLASH PICO_PWQ_AND_EXTERNAL_CLOCK_CLASH PICO_MEMORY

4.9.29 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.

Applicability	All modes
Arguments	`handle`, the handle of the required device `channel`, the channel to be configured. The values are: PS6000_CHANNEL_A: Channel A input PS6000_CHANNEL_B: Channel B input PS6000_CHANNEL_C: Channel C input PS6000_CHANNEL_D: Channel D input `enabled`, whether or not to enable the channel. The values are: TRUE: enable FALSE: do not enable `type`, the impedance and coupling type. The values are: 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 ±10 volt and ±20 volt input ranges are not available. `range`, the input voltage range: PS6000_50MV: ±50 mV PS6000_100MV: ±100 mV PS6000_200MV: ±200 mV PS6000_500MV: ±500 mV PS6000_1V: ±1 V PS6000_2V: ±2 V PS6000_5V: ±5 V PS6000_10V: ±10 V * PS6000_20V: ±20 V * * not available when `type` = PS6000_DC_50R `analogueOffset`, a voltage to add to the input channel before digitization. The allowable range of offsets depends on the input range selected for the channel, as follows: 50 mV to 200 mV: MIN_ANALOGUE_OFFSET_50MV_200MV to MAX_ANALOGUE_OFFSET_50MV_200MV

	500 mV to 2 V: MIN_ANALOGUE_OFFSET_500MV_2V to MAX_ANALOGUE_OFFSET_500MV_2V 5 V to 20 V: MIN_ANALOGUE_OFFSET_5V_20V to MAX_ANALOGUE_OFFSET_5V_20V. (When type = PS6000_DC_50R, the allowable range is reduced to that of the 50 mV to 200 mV input range, i.e. MIN_ANALOGUE_OFFSET_50MV_200MV to MAX_ANALOGUE_OFFSET_50MV_200MV). bandwidth, the bandwidth limiter setting: PS6000_BW_FULL: the scope's full specified bandwidth PS6000_BW_20MHZ: -3 dB bandwidth limited to 20 MHz
Returns	PICO_OK PICO_USER_CALLBACK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_INVALID_VOLTAGE_RANGE PICO_INVALID_COUPLING PICO_INVALID_ANALOGUE_OFFSET PICO_DRIVER_FUNCTION

4.9.30 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.

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

4.9.31 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.

Applicability	Rapid block mode without aggregation.
Arguments	`handle`, the handle of the device `channel`, the input channel to use with this buffer `buffer`, an array in which the captured data is stored `bufferLth`, the size of the buffer `waveform`, an index to the waveform number. Range: `0` to `nCaptures - 1` `downSampleRatioMode`: see ps6000GetValues

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

4.9.32 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.

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

4.9.33 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.

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

4.9.34 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.

Applicability	Block mode
Arguments	`handle`, the handle of the required device `mode`, the ETS mode. Use one of these values: `PS6000_ETS_OFF`: disables ETS `PS6000_ETS_FAST`: enables ETS and provides `ets_cycles` of data, which may contain data from previously returned cycles `PS6000_ETS_SLOW`: enables ETS and provides fresh data every `ets_cycles`. This mode takes longer to provide each data set, but the data sets are more stable and are guaranteed to contain only new data. `ets_cycles`, the number of cycles to store: the computer can then select `ets_interleave` cycles to give the most uniform spread of samples. Range: between two and five times the value of `ets_interleave`, and not more than `PS6000_MAX_ETS_CYCLES` `ets_interleave`, the number of waveforms to combine into a single ETS capture Maximum value: `PS6000_MAX_INTERLEAVE` `sampleTimePicoseconds`, on exit, the effective sampling interval of the ETS data. For example, if the captured sample time is 20 ns and `ets_interleave` is 10, then the effective sample time in ETS mode is 2 ns.

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

4.9.35 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.

Applicability	ETS mode only. If your programming language does not support 64-bit data, use the 32-bit version ps6000SetEtsTimeBuffers instead.
Arguments	`handle,` the handle of the required device `* buffer,` an array of 64-bit words, each representing the time in nanoseconds at which the sample was captured `bufferLth,` the size of the buffer array

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

4.9.36 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.

Applicability	ETS mode only. If your programming language supports 64-bit data then you can use ps6000SetEtsTimeBuffer instead.
Arguments	`handle`, the handle of the required device `* timeUpper`, an array of 32-bit words, each representing the upper 32 bits of the time in nanoseconds at which the sample was captured `* timeLower`, an array of 32-bit words, each representing the lower 32 bits of the time in nanoseconds at which the sample was captured `bufferLth`, the size of the `timeUpper` and `timeLower` arrays

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

4.9.37 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.

Applicability	All modes
Arguments	`handle`, the handle of the required device `frequency`, the external clock frequency. The possible values are: PS6000_FREQUENCY_OFF: the scope generates its own clock PS6000_FREQUENCY_5MHZ: 5 MHz external clock PS6000_FREQUENCY_10MHZ: 10 MHz external clock PS6000_FREQUENCY_20MHZ: 20 MHz external clock PS6000_FREQUENCY_25MHZ: 25 MHz external clock The external clock signal must be within ±5% of the selected frequency, otherwise this function will report an error. `threshold`, the logic threshold voltage. -32,512 corresponds to -1 volt, 0 to 0 volts and 32,512 to +1 volt.
Returns	PICO_OK PICO_USER_CALLBACK PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION PICO_EXTERNAL_FREQUENCY_INVALID PICO_FW_FAIL PICO_NOT_RESPONDING PICO_CLOCK_CHANGE_ERROR

4.9.38 ps6000SetNoOfCaptures

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

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

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

4.9.39 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.

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

	type, the pulse-width type, one of these constants: PS6000_PW_TYPE_NONE: do not use the pulse width qualifier PS6000_PW_TYPE_LESS_THAN: pulse width less than lower PS6000_PW_TYPE_GREATER_THAN: pulse width greater than lower PS6000_PW_TYPE_IN_RANGE: pulse width between lower and upper PS6000_PW_TYPE_OUT_OF_RANGE: pulse width not between lower and upper
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_PULSE_WIDTH_QUALIFIER PICO_DRIVER_FUNCTION

4.9.39.1 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.

Elements	`channelA, channelB, channelC, channelD, aux:` the type of condition that should be applied to each channel. Use these constants: - PS6000_CONDITION_DONT_CARE PS6000_CONDITION_TRUE PS6000_CONDITION_FALSE The channels that are set to PS6000_CONDITION_TRUE or PS6000_CONDITION_FALSE must all meet their conditions simultaneously to produce a trigger. Channels set to PS6000_CONDITION_DONT_CARE are ignored. `external:` not used

4.9.40 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,
  short                    whiteNoise,
  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 13 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.

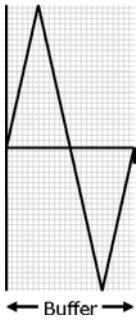
Applicability	All modes
Arguments	`handle`, the handle of the required device `offsetVoltage`, the voltage offset, in microvolts, to be applied to the waveform `pkToPk`, the peak-to-peak voltage, in microvolts, of the waveform signal `startDeltaPhase`, the initial value added to the phase counter as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

Arguments	dwellCount, the time, in 5 ns steps, between successive additions of deltaPhaseIncrement to the delta phase counter. This determines the rate at which the generator sweeps the output frequency. Minimum value: PS6000_MIN_DWELL_COUNT * arbitraryWaveform, a buffer that holds the waveform pattern as a set of samples equally spaced in time. arbitraryWaveformSize, the size of the arbitrary waveform buffer, in samples, from PS6000_MIN_SIG_GEN_BUFFER_SIZE to PS6000_MAX_SIG_GEN_BUFFER_SIZE. sweepType, determines whether the startDeltaPhase is swept up to the stopDeltaPhase, or down to it, or repeatedly swept up and down. Use one of these values: - PS6000_UP PS6000_DOWN PS6000_UPDOWN PS6000_DOWNUP whiteNoise. If TRUE, the signal generator produces white noise and ignores all settings except pkToPk and offsetVoltage. If FALSE, the generator produces the arbitrary waveform. indexMode, specifies how the signal will be formed from the arbitrary waveform data. Single, dual and quad index modes are possible. Use one of these constants: PS6000_SINGLE PS6000_DUAL PS6000_QUAD shots, see ps6000SigGenBuiltIn sweeps, see ps6000SigGenBuiltIn triggerType, see ps6000SigGenBuiltIn triggerSource, see ps6000SigGenBuiltIn extInThreshold, see ps6000SigGenBuiltIn
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

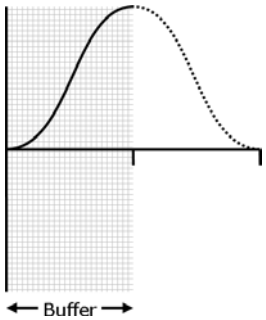
4.9.40.1 AWG index modes

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

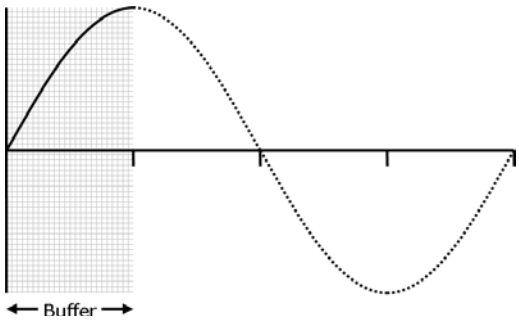
Single mode. The generator outputs the raw contents of the buffer repeatedly. This mode is the only one that can generate asymmetrical waveforms. You can also use this mode for symmetrical waveforms, but the dual and quad modes make more efficient use of the buffer memory.



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.



4.9.41 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,
  short                     whiteNoise,
  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.

Applicability	All modes
Arguments	`handle`, the handle of the required device `offsetVoltage`, the voltage offset, in microvolts, to be applied to the waveform `pkToPk`, the peak-to-peak voltage, in microvolts, of the waveform signal `waveType`, the type of waveform to be generated. `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 `PS6000_SINE_MAX_FREQUENCY` and related values. `stopFrequency`, the frequency at which the sweep reverses direction or returns to the initial frequency `increment`, the amount of frequency increase or decrease in sweep mode

	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 whiteNoise. If TRUE, the signal generator produces white noise and ignores all settings except offsetVoltage and pkTopk. If FALSE, the signal generator produces the waveform specified by waveType. shots, the number of cycles of the waveform to be produced after a trigger event. If non-zero (from 1 to PS6000_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 PS6000_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

4.9.42 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.

Applicability	All modes
Arguments	`handle`, the handle of the required device. `conditions`, an array of PS6000_TRIGGER_CONDITIONS structures specifying the conditions that should be applied to each channel. In the simplest case, the array consists of a single element. When there is more than one element, the overall trigger condition is the logical OR of all the elements. `nConditions`, the number of elements in the `conditions` array. If `nConditions` is zero then triggering is switched off.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_CONDITIONS PICO_MEMORY_FAIL PICO_DRIVER_FUNCTION

4.9.42.1 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.

Elements	`channelA, channelB, channelC, channelD, aux, pulseWidthQualifier:` the type of condition that should be applied to each channel. Use these constants: PS6000_CONDITION_DONT_CARE PS6000_CONDITION_TRUE PS6000_CONDITION_FALSE The channels that are set to PS6000_CONDITION_TRUE or PS6000_CONDITION_FALSE must all meet their conditions simultaneously to produce a trigger. Channels set to PS6000_CONDITION_DONT_CARE are ignored. `external:` not used

4.9.43 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.

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

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

4.9.44 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.

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

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

4.9.45 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.

Applicability	All modes
Arguments	`handle`, the handle of the required device `delay`, the time between the trigger occurring and the first sample. For example, if `delay`=100 then the scope would wait 100 sample periods before sampling. At a timebase of 5 GS/s, or 200 ps per sample (`timebase` = 0), the total delay would then be 800 x 200 ps = 160 ns. Range: 0 to `MAX_DELAY_COUNT`

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

4.9.46 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.

Applicability	Rapid block mode
Arguments	`handle`, the handle of the required device `nWaveformsPerSecond`, the maximum number of waveforms per second
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_DRIVER_FUNCTION PICO_INVALID_PARAMETER

4.9.47 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.

Applicability	Use with ps6000SetSigGenBuiltIn or ps6000SetSigGenArbitrary.
Arguments	handle, the handle of the required device state, sets the trigger gate high or low when the trigger type is set to either SIGGEN_GATE_HIGH or SIGGEN_GATE_LOW. Ignored for other trigger types.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SIGNAL_GENERATOR PICO_SIGGEN_TRIGGER_SOURCE PICO_DRIVER_FUNCTION PICO_NOT_RESPONDING

4.9.48 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.

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

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

4.9.49 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.

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

4.10 Programming examples

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

- C
- Visual Basic
- Excel
- LabView

4.10.1 C

There are two **C** example programs: one is a simple GUI application, and the other is a more comprehensive console mode program that demonstrates all of the facilities of the driver.

The GUI example program is a generic Windows application - that is, it does not use Borland AppExpert or Microsoft AppWizard. To compile the program, create a new project for an Application containing the following files from the `Examples/ps6000/` subdirectory of your PicoScope installation: -

- `ps6000.c`
- `ps6000.rc`

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:

- `resource.h`
- `ps6000.h`

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

- `ps6000.dll`

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

- `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`

4.10.2 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 .

4.10.3 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.

4.10.4 LabView

The PS6000.vi example in the Examples/ps6000/ subdirectory of your PicoScope installation shows how to access the driver functions using LabVIEW. It was tested using version 6.1 of LabVIEW for Windows. To use the example, copy all the .vi files to your LabVIEW directory:

You will also need:

- ps6000.dll

from the installation directory.

4.11 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.

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 picopp.sys file is too old to be used with the device driver
0A	PICO_EEPROM_CORRUPT The EEPROM has become corrupt, so the device will use a default setting
0B	PICO_OS_NOT_SUPPORTED The operating system on the PC is not supported by this driver
0C	PICO_INVALID_HANDLE There is no device with the handle value passed
0D	PICO_INVALID_PARAMETER A parameter value is not valid
0E	PICO_INVALID_TIMEBASE The 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 variant

1B	PICO_AUTO_TRIGGER_TIME_TOO_SHORT The auto trigger time is less than the time it will take to collect the data
1C	PICO_BUFFER_STALL The collection of data has stalled as unread data would be overwritten
1D	PICO_TOO_MANY_SAMPLES Number of samples requested is more than available in the current memory segment
1E	PICO_TOO_MANY_SEGMENTS Not possible to create number of segments requested
1F	PICO_PULSE_WIDTH_QUALIFIER A null pointer has been passed in the trigger function or one of the parameters is out of range
20	PICO_DELAY One or more of the hold-off parameters are out of range
21	PICO_SOURCE_DETAILS One or more of the source details are incorrect
22	PICO_CONDITIONS One or more of the conditions are incorrect
23	PICO_USER_CALLBACK The driver's thread is currently in the ps6000...Ready callback function and therefore the action cannot be carried out
24	PICO_DEVICE_SAMPLING An attempt is being made to get stored data while streaming. Either stop streaming by calling 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
36	PICO_DELAY_NULL NULL pointer passed as delay parameter
37	PICO_INVALID_BUFFER The buffers for overview data have not been set while streaming
3A	PICO_CANCELLED A block collection has been cancelled
3B	PICO_SEGMENT_NOT_USED The segment index is not currently being used
3C	PICO_INVALID_CALL The wrong GetValues 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_DRIVE_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 ±5% of the stated value
57	PICO_CLOCK_CHANGE_ERROR The FPGA could not lock the clock signal
58	PICO_TRIGGER_AND_EXTERNAL_CLOCK_CLASH You are trying to configure the AUX input as both a trigger and a reference clock
59	PICO_PWQ_AND_EXTERNAL_CLOCK_CLASH You are trying to congfigure the AUX input as both a pulse width qualifier and a reference clock
103	PICO_GET_DATA_ACTIVE reserved for future use

4.12 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_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;

#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;
```

4.13 Numeric data types

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

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

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

Driver. A program that controls a piece of hardware. The driver for the PicoScope 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.

USB 2.0. Universal Serial Bus. This is a standard 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

B

C

D

E

F

H

I

L

M

N

O

P

R

S

T

U

V

W


pico
Technology

Pico Technology

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

ps6000pg.en-1

10.11.09