PicoScope® 4000 Series (A API)

PC Oscilloscopes

Programmer's Guide

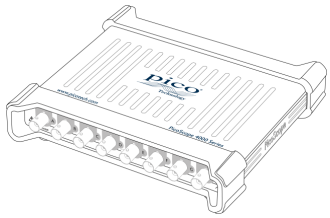

pico
Technology

Contents


pico
Technology

1 Welcome

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



This Programmer's Guide explains how to use the `ps4000a` API, the Application Programming Interface for the PicoScope 4000 Series (A API) oscilloscopes. The `ps4000a` API supports the following models:

- PicoScope 4444 4-channel differential oscilloscope (product web page)
- PicoScope 4824 8-channel oscilloscope (product web page)

Other oscilloscopes in the PicoScope 4000 Series use the ps4000 API. This is documented in the original *PicoScope 4000 Series Programmer's Guide*.

2 Introduction

2.1 License agreement

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

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

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

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

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

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

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

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

Support. If you are dissatisfied with the performance of this software, please contact our technical support staff, who will try to fix the problem within a reasonable time. If you are still dissatisfied, please return the product and software to your supplier within 14 days of purchase for a full refund.

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

2.2 Trademarks

Pico Technology, **PicoScope** and **PicoConnect** 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.

Windows, **Excel** and **Visual Basic for Applications** are registered trademarks or trademarks of Microsoft Corporation in the USA and other countries. **LabVIEW** is a registered trademark of National Instruments Corporation. **MATLAB** is a registered trademark of The MathWorks, Inc.

2.3 System requirements

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

Item	Specification
Operating system	Windows 7, 8 or 10 (32-bit and 64-bit versions) Linux and macOS, 64-bit versions only: see picotech.com/downloads for supported versions
Processor **Memory** **Free disk space**	As required by the operating system
Ports	USB 3.0 or USB 2.0 port(s)

USB

The `ps4000a` driver offers three different methods of recording data, all of which support USB 2.0 and USB 3.0. The fastest transfer rates between the PC and the PicoScope 4000 are achieved using USB 3.0.

2.4 Installation instructions

The PicoSDK installation process varies depending on your operating system. Software and installation instructions are available from picotech.com/downloads.

Windows users

Visit picotech.com/downloads, select your oscilloscope from the list and download the latest PicoSDK installer, choosing either the 32-bit or 64-bit version depending on your operating system and software development environment.

macOS users

If you have already installed PicoScope 6 Beta for macOS, you already have all the drivers installed. If not, visit picotech.com/downloads, select your oscilloscope from the list and download and install the latest version.

Linux users

Visit Linux Software & Drivers for Oscilloscopes and Data Loggers for full instructions.

3 Programming with the ps4000a API

The ps4000a.dll dynamic link library in the lib subdirectory of your SDK installation allows you to program a PicoScope 4000 Series (A API) 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 mode.
- Set up triggering.
- Start capturing data. (See Sampling modes, where programming is discussed in more detail.)
- Wait until the scope unit is ready.
- Stop capturing data.
- Copy data to a buffer.
- Close the scope unit.

Numerous example programs are available on the "picotech" GitHub pages. These show how to use the functions of the driver software in each of the modes available.

3.1 Driver

Microsoft Windows
Your application will communicate with a PicoScope 4000 Series library called ps4000a.dll, which is supplied in 32-bit and 64-bit versions. This DLL is compatible with the PicoScope 4444 and 4824 oscilloscopes. The DLL exports the ps4000a function definitions in stdcall format, which is compatible with a wide range of programming languages.

ps4000a.dll driver depends on another DLL, picoipp.dll (which is supplied in 32-bit and 64-bit versions) and a low-level driver called WinUsb.sys (or CyUsb3.sys on Windows 7). These are installed by PicoSDK and configured when you plug the oscilloscope into each USB port for the first time. Your application does not call these drivers directly.

Linux and macOS
Please see the **Downloads** section of picotech.com for instructions on downloading the drivers for these operating systems. The drivers use the cdecl calling convention. Linux libraries and dependencies are distributed via our package repositories. macOS libraries and dependencies are distributed with PicoScope 6 for macOS.

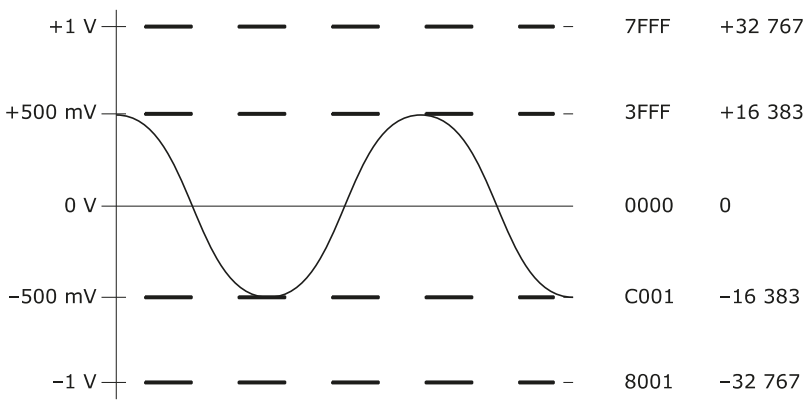
3.2 Voltage ranges

ps4000aSetChannel() allows you to set the voltage range of each input channel of the scope. The allowable voltage ranges are described in the device data sheet. Each sample is normalized to 16 bits, and the minimum and maximum values returned to your application are given by ps4000aMinimumValue() and ps4000aMaximumValue() as follows:

Function	Reading		Voltage
	decimal	hex	
ps4000aMinimumValue()	−32 767	8001	minimum
N/A	0	0000	zero
ps4000aMaximumValue()	+32 767	7FFF	maximum

Example

1. Call ps4000aSetChannel() with `range` set to `PS4000A_1V`.
2. Apply a sine wave input of 500 mV amplitude to the oscilloscope.
3. Capture some data using the desired sampling mode.
4. The data will be encoded as shown opposite.



3.3 Channel selection

You can switch each channel on and off, and set its coupling mode to either AC or DC, using the ps4000aSetChannel() function.

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

3.4 Triggering

PicoScope 4000 Series PC Oscilloscopes can either start collecting data immediately, or be programmed to wait for a trigger event to occur. In both cases you need to use the PicoScope 4000 trigger functions:

- ps4000aSetTriggerChannelConditions() – specifies which channels are included in the trigger logic
- ps4000aSetTriggerChannelDirections() – specifies the edge or threshold to be used for each channel
- ps4000aSetTriggerChannelProperties() – specifies threshold levels, level or window mode, and global trigger timeout
- ps4000aSetTriggerDelay() – defines post-trigger delay (optional)

Alternatively, the above functions can be run in a single operation by calling ps4000aSetSimpleTrigger().

A trigger event can occur when one of the input channels crosses a threshold voltage on either a rising or a falling edge. It is also possible to combine up to four inputs by defining multiple trigger conditions.

The driver supports these triggering methods:

- Simple Edge
- Advanced Edge
- Windowing
- Pulse width
- Logic
- Delay
- Drop-out
- Runt

The pulse width, delay and drop-out triggering methods additionally require the use of the pulse width qualifier functions:

- ps4000aSetPulseWidthQualifierConditions()
- ps4000aSetPulseWidthQualifierProperties()

3.5 Downsampling

The driver can optionally apply a data reduction, or **downsampling**, process before returning data to the application. Downsampling is done by firmware on the device and is generally faster than using the PC's own processor. You instruct the driver to downsample by passing a `downSampleRatioMode` argument to one of the data-retrieval functions such as `ps4000aGetValues()`. You must also pass in an argument called `downSampleRatio`: how many raw samples are to be combined into each processed sample.

Retrieving multiple types of downsampled data
You can optionally retrieve data using more than one downsampling mode with a single call to `ps4000aGetValues()`. Set up a buffer for each downsampling mode by calling `ps4000aSetDataBuffer()`. Then, when calling `ps4000aGetValues()`, set `downSampleRatioMode` to the bitwise OR of the required downsampling modes.

Retrieving both raw and downsampled data
You cannot retrieve raw data and downsampled data in a single operation. If you require both raw and downsampled data, first retrieve the downsampled data as described above and then continue as follows:

1. Call `ps4000aStop()`.
2. Set up a data buffer for each channel using `ps4000aSetDataBuffer()` with the ratio mode set to `PS4000A_RATIO_MODE_NONE`.
3. Call `ps4000aGetValues()` to retrieve the data.

Downsampling modes
The available downsampling modes are:

`PS4000A_RATIO_MODE_NONE (0)`

No downsampling is performed. The `downSampleRatio` parameter is ignored.

`PS4000A_RATIO_MODE_AGGREGATE (1)`

The *aggregate* method generates two buffers of data for every channel, one containing the minimum sample value for every block of `downSampleRatio` raw samples, and the other containing the maximum value.

`PS4000A_RATIO_MODE_DECIMATE (2)`

The *decimate* method returns the first sample in every block of `downSampleRatio` successive samples and discards all the other samples.

`PS4000A_RATIO_MODE_AVERAGE (4)`

The *average* method returns the sum of all the samples in each block of `downSampleRatio` samples, divided by the length of the block.

`PS4000A_RATIO_MODE_DISTRIBUTION (8)`

Reserved for future use.

3.6 Sampling modes

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

- **Block mode.** In this mode, the scope stores data in internal buffer memory and then transfers it to the PC. When the data has been collected it is possible to examine the data, with an optional downsampling factor. The data is lost when a new 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 buffer memory. This enables long periods of slow data collection for chart recorder and data-logging applications. Streaming mode provides fast streaming at up to 160 MS/s with a USB 3.0 connection. Downsampling and triggering are supported in this mode.

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

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

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

Most of the callback functions have a `PICO_STATUS` parameter. The driver sends this value to the callback function to indicate the success or otherwise of the data capture.

Probe callback
The driver can be instructed to signal to your application whenever a probe connection event occurs. It does this using a callback to a function that you define. See Handling PicoConnect probe interactions.

3.6.1 Block mode

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

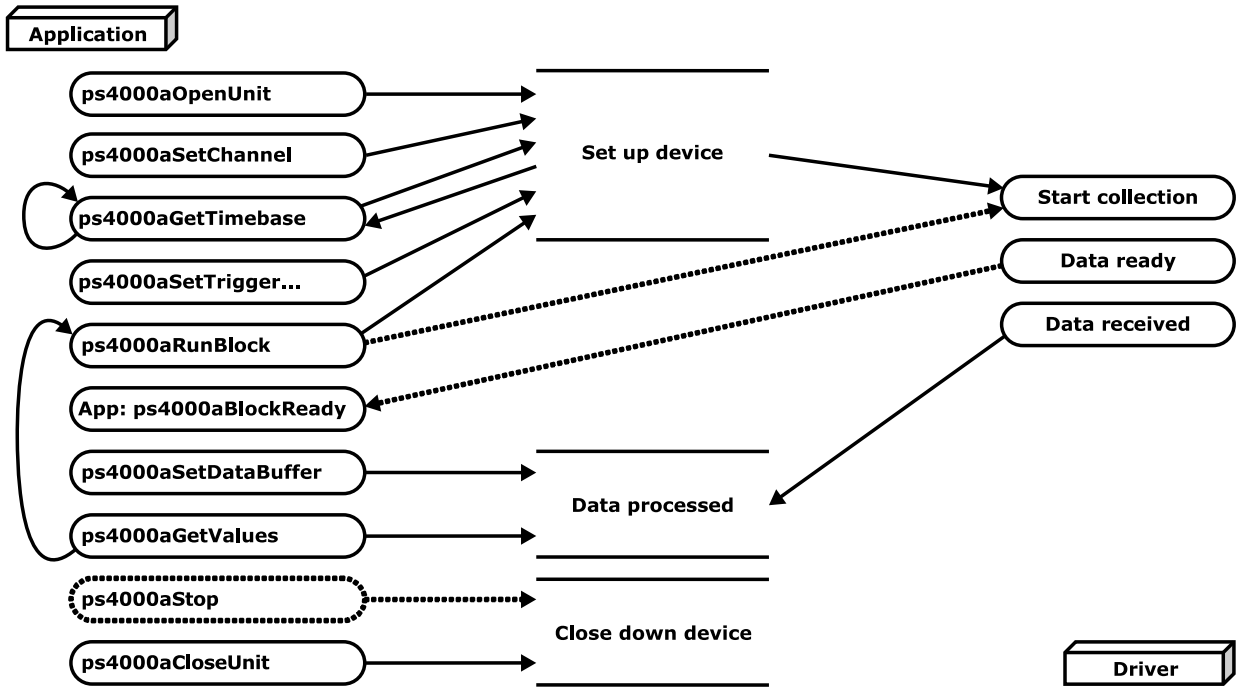
- **Block size.** The maximum number of values depends upon the size of the oscilloscope's memory. The memory buffer is shared between the enabled channels, so if two channels are enabled, each is allocated half the memory. These features are handled transparently by the driver. The block size also depends on the number of memory segments in use (see `ps4000aMemorySegments()`).
- **Sampling rate.** The maximum real-time sampling rate may depend on the number of channels enabled. See the data sheet for your scope model. You specify the sampling rate by passing a timebase number (see Timebases) to `ps4000aRunBlock()`.
- **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 `ps4000aRunBlock()`, `ps4000aStop()` and `ps4000aGetValues()`.
- **Downsampling.** When the data has been collected, you can set an optional downsampling factor and examine the data. Downsampling is the process of reducing the amount of data by combining adjacent samples using one of several algorithms. 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 `ps4000aMemorySegments()`.
- **Data retention.** The data is lost when a new run is started in the same segment, the number of segments is changed, or the scope is powered down.

3.6.1.1 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 `ps4000aOpenUnit()`.
1a. (PicoScope 4444 only) Register your probe interaction callback function using `ps4000aSetProbeInteractionCallback()`.
2. Select channel ranges and AC/DC coupling using `ps4000aSetChannel()`.
3. Using `ps4000aGetTimebase()`, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions `ps4000aSetTriggerChannelConditions()`, `ps4000aSetTriggerChannelDirections()`, `ps4000aSetTriggerChannelProperties()` and `ps4000aSetTriggerDelay()` to set up the trigger if required.
5. Start the oscilloscope running using `ps4000aRunBlock()`.
6. Wait until the oscilloscope is ready using the `ps4000aBlockReady()` callback.
7. Use `ps4000aSetDataBuffer()` to tell the driver where your memory buffer is. For greater efficiency when doing multiple captures, you can call this function outside the loop, after step 4.
8. Transfer the block of data from the oscilloscope using `ps4000aGetValues()`.
9. Display the data.
10. Repeat steps 5 to 9.
11. Stop the oscilloscope using `ps4000aStop()`.
12. Request new views of stored data using different downsampling parameters: see Retrieving stored data.
13. Close the device using `ps4000aCloseUnit()`.

Note that if you use ps4000aGetValues() or ps4000aStop() before the oscilloscope is ready, no capture will be available and the driver will return PICO_NO_SAMPLES_AVAILABLE.



3.6.1.2 Asynchronous calls in block mode

ps4000aGetValues() function may take a long time to complete if a large amount of data is being collected. To avoid hanging the calling thread, it is possible to call ps4000aGetValuesAsync() instead. This immediately returns control to the calling thread, which then has the option of waiting for the data or calling ps4000aStop() to abort the operation.

3.6.2 Rapid block mode

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

Rapid block mode allows you to sample several waveforms at a time with the minimum time between waveforms. On the PicoScope 4824, for example, it reduces the gap from milliseconds to about 2.5 µs.

3.6.2.1 Using rapid block mode

You can use rapid block mode with or without downsampling.

Without downsampling

1. Open the oscilloscope using ps4000aOpenUnit().
1a. (PicoScope 4444 only) Register your probe interaction callback function using ps4000aSetProbeInteractionCallback().
2. Select channel ranges and AC/DC coupling using ps4000aSetChannel().

3. Set the number of memory segments equal to or greater than the number of captures required using ps4000aMemorySegments(). Use ps4000aSetNoOfCaptures() before each run to specify the number of waveforms to capture.
4. Using ps4000aGetTimebase(), select timebases until the required nanoseconds per sample is located. This will indicate the number of samples per channel available for each segment. If you know that the number of samples per segment will not exceed the limit, you can call this function after step 2.
5. Use the trigger setup functions ps4000aSetTriggerChannelConditions(), ps4000aSetTriggerChannelDirections(), ps4000aSetTriggerChannelProperties() and ps4000aSetTriggerDelay() to set up the trigger if required.
6. Start the oscilloscope running using ps4000aRunBlock(). You can call ps4000aGetNoOfCaptures() while capturing is in progress to obtain a count of the number of waveforms captured. Once all the waveforms have been captured, but ready is not complete, call ps4000aGetNoOfProcessedCaptures() to obtain the number of captures processed on the PC.
7. Wait until the oscilloscope is ready using the ps4000aBlockReady() callback.
8. Use ps4000aSetDataBuffer() to tell the driver where your memory buffers are. Call the function once for each channel/segment combination for which you require data. For greater efficiency when doing multiple captures, you can call this function outside the loop, after step 5.
9. Transfer the blocks of data from the oscilloscope using ps4000aGetValuesBulk().
10. Retrieve the time offset for each data segment using ps4000aGetValuesTriggerTimeOffsetBulk64().
11. Display the data.
12. Repeat steps 6 to 11 if necessary.
13. Stop the oscilloscope using ps4000aStop().
14. Close the device using ps4000aCloseUnit().

With downsampling

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

8a. Call ps4000aSetDataBuffers() to set up one pair of buffers for every waveform segment required.
9a. Call ps4000aGetValues() for each pair of buffers.
10a. Retrieve the time offset for each data segment using ps4000aGetTriggerTimeOffset64().

Continue from step 11 above.

3.6.2.2 Rapid block mode example 1: no aggregation

```
#define MAX_WAVEFORMS 100
#define MAX_SAMPLES 1000
```

Set up the device 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 MAX_WAVEFORMS
ps4000aSetNoOfCaptures(handle, MAX_WAVEFORMS);

pParameter = false;
ps4000aRunBlock
(
  handle,
  0,                 // noOfPreTriggerSamples
  10000,             // noOfPostTriggerSamples
  1,                 // timebase to be used
  &timeIndisposedMs, // calculated duration of capture
  0,                 // segmentIndex
  lpReady,
  &pParameter
);
```

- Get number of captures. Call ps4000aGetNoOfCaptures() to find out the number of captures taken by the device. This is particularly useful if a trigger is being used.

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);

int16_t buffer[PS4000A_MAX_CHANNELS][MAX_WAVEFORMS][MAX_SAMPLES];

for (int32_t i = 0; i < 20; i++)
{
  for (int32_t c = PS4000A_CHANNEL_A; c <= PS4000A_CHANNEL_H; c++)
  {
   ps4000aSetDataBuffer
   (
         handle,
         c,
         buffer[c][i],
         MAX_SAMPLES,
         i,
         PS4000A_RATIO_MODE_NONE
   );
  }
}
```

Comments: buffer has been created as a three-dimensional 16-bit integer array, which will contain 1000 samples as defined by MAX_SAMPLES. There are only 20 buffers set, but it is possible to set up to the number of captures you have requested.

```
ps4000aGetValuesBulk
(
  handle,
  &noOfSamples,
  10,                       // fromSegmentIndex
  19,                       // toSegmentIndex
  1,                        // downSampleRatio
  PS4000A_RATIO_MODE_NONE,  // downSampleRatioMode
  overflow                  // indices 10 to 19 will be populated
)
```

Comments: the number of samples could be up to noOfPreTriggerSamples + noOfPostTriggerSamples, the values set in ps4000aRunBlock(). The samples are always returned from the first sample taken, unlike the ps4000aGetValues() function which allows the sample index to be set. This function does not support downsampling. 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.

```
ps4000aGetValuesTriggerTimeOffsetBulk64
(
  handle,
  times,        // indices 10 to 19 will be populated
  timeUnits,    // indices 10 to 19 will be populated
  10,           // fromSegmentIndex, inclusive
  19            // toSegmentIndex, inclusive
)
```

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

3.6.2.3 Rapid block mode example 2: using aggregation

```
#define MAX_WAVEFORMS 100
#define MAX_SAMPLES 1000
```

Set up the device 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 MAX_WAVEFORMS
ps4000aSetNoOfCaptures(handle, MAX_WAVEFORMS);

pParameter = false;
ps4000aRunBlock
(
  handle,
  0,                 // noOfPreTriggerSamples
  1000000,           // noOfPostTriggerSamples
  1,                 // timebase to be used
  &timeIndisposedMs, // calculated duration of capture
  1,                 // segmentIndex
  lpReady,
  &pParameter
);
```

- Get number of captures. Call ps4000aGetNoOfCaptures() to find out the number of captures taken by the device. This is particularly useful if a trigger is being used.

Comments: the set-up for running the device is exactly the same whether or not you use downsampling when you retrieve the samples.

```
for (int32_t segment = 10; segment < 20; segment++)
{
  for (int32_t c = PS4000A_CHANNEL_A; c <= PS4000A_CHANNEL_H; c++)
  {
   ps4000aSetDataBuffers
   (
         handle,
         c,
         bufferMax[c],
         bufferMin[c]
         MAX_SAMPLES,
         segment,
         downSampleRatioMode     // set to RATIO_MODE_AGGREGATE
   );
  }

  ps4000aGetValues
  (
   handle,
   0,
```

```
      &noOfSamples,                 // set to MAX_SAMPLES on entering
      1000,
      downSampleRatioMode,    // set to RATIO_MODE_AGGREGATE
      segment,
      overflow
     );

     ps4000aGetTriggerTimeOffset64
     (
      handle,
      &time,
      &timeUnits,
      segment
     )
  }
```

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

3.6.3 Streaming mode

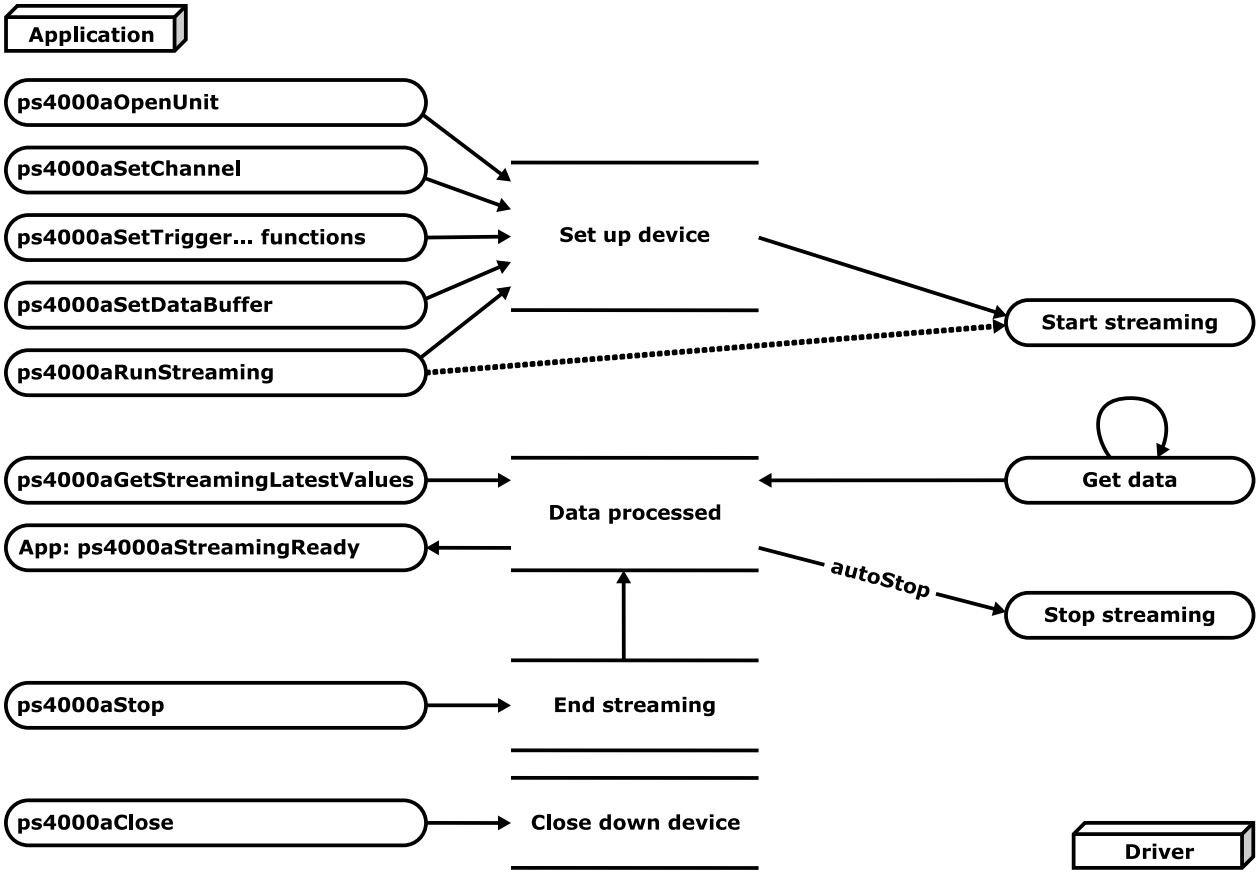
Streaming mode can capture data without the gaps that occur between blocks when using block mode. It can transfer data to the PC at speeds of up to 160 MS/s for the PicoScope 4824, or up to 100 MS/s for the PicoScope 4444, depending on the computer's performance. This makes it suitable for **high-speed data acquisition**, allowing you to capture long data sets limited only by the computer's memory.

Downsampling
The driver returns downsampled readings while the device is streaming. If the downsampling ratio is set to 1, only one buffer is returned per channel. When the downsampling ratio is greater than 1 and aggregation mode is selected, two buffers (maximum and minimum) per channel are returned.

3.6.3.1 Using streaming mode

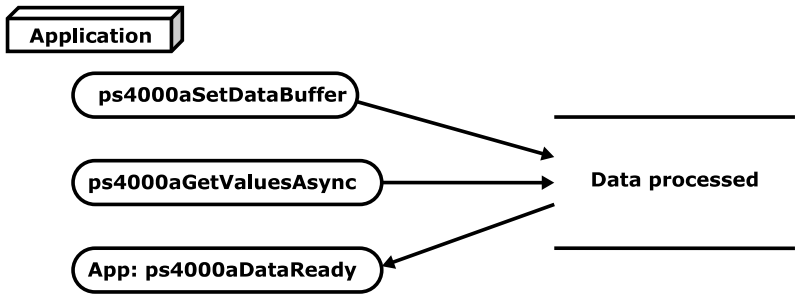
This is the general procedure for reading and displaying data in streaming mode:

1. Open the oscilloscope using `ps4000aOpenUnit()`.
1a. (PicoScope 4444 only) Register your probe interaction callback function using `ps4000aSetProbeInteractionCallback()`.
2. Select channels, ranges and AC/DC coupling using `ps4000aSetChannel()`.
3. Use the trigger setup functions [1] [2] [3] [4] to set up the trigger if required.
4. Call `ps4000aSetDataBuffer()` to tell the driver where your data buffer is.
5. Set up downsampling and start the oscilloscope running using `ps4000aRunStreaming()`.
6. Call `ps4000aGetStreamingLatestValues()` to get data.
7. Process data returned to your application's function. This example is using `autoStop`, so after the driver has received all the data points requested by the application, it stops the device streaming.
8. Call `ps4000aStop()`, even if `autoStop` is enabled.
9. Request new views of stored data using different downsampling parameters: see Retrieving stored data.
10 Close the device using `ps4000aCloseUnit()`.



3.6.4 Retrieving stored data

You can collect data from the ps4000a driver with a different downsampling factor when ps4000aRunBlock() or ps4000aRunStreaming() has already been called and has successfully captured all the data. Use ps4000aGetValuesAsync().



3.7 Timebases

The ps4000a API allows you to select any of 2^{32} different timebases created by dividing the oscilloscope's master sampling clock. The timebases allow slow enough sampling in block mode to overlap the streaming sample intervals, so that you can make a smooth transition between block mode and streaming mode. Calculate the timebase using ps4000aGetTimebase() or refer to the following tables:

PicoScope 4444

Timebase (n)	Sampling interval (t_S) = 2.5 ns x 2^n	Sampling frequency (f_S) = 400 MHz / (n+1)
0 *	2.5 ns	400 MHz
1 *	5 ns	200 MHz
2 *	10 ns	100 MHz
3	20 ns	50 MHz
	= 20 ns x (n–2)	= 50 MHz / (n–2)
4	40 ns	25 MHz
...	...	...
2^{32}–1	~ 11 s	~ 93 mHz

* 12-bit sampling mode only

PicoScope 4824

Timebase (n)	Sampling interval (t_S) = 12.5 ns × (n+1)	Sampling frequency (f_S) = 80 MHz / (n+1)
0	12.5 ns	80 MHz
1	25 ns	40 MHz
...	...	...
2^{32}–1	~54 s	~18.6 mHz

Notes

1. The maximum possible sampling rate may depend on the number of enabled channels and (for flexible-resolution scopes) the selected ADC resolution. Refer to the data sheet for details.

2. In streaming mode, the maximum possible sampling rate may be limited by the speed of the USB interface.

3.8 Combining several oscilloscopes

It is possible to collect data using up to 64 PicoScope 4000 Series PC Oscilloscopes at the same time, depending on the capabilities of the PC. Each oscilloscope must be connected to a separate USB port. ps4000aOpenUnit() 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 ps4000aBlockReady(...)
// Define callback function specific to application

handle1 = ps4000aOpenUnit()
handle2 = ps4000aOpenUnit()

ps4000aSetChannel(handle1) // set up unit 1
ps4000aRunBlock(handle1)

ps4000aSetChannel(handle2) // set up unit 2
ps4000aRunBlock(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

  ps4000aCloseUnit(handle1)
  ps4000aCloseUnit(handle2)
```

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

3.9 Handling PicoConnect probe interactions

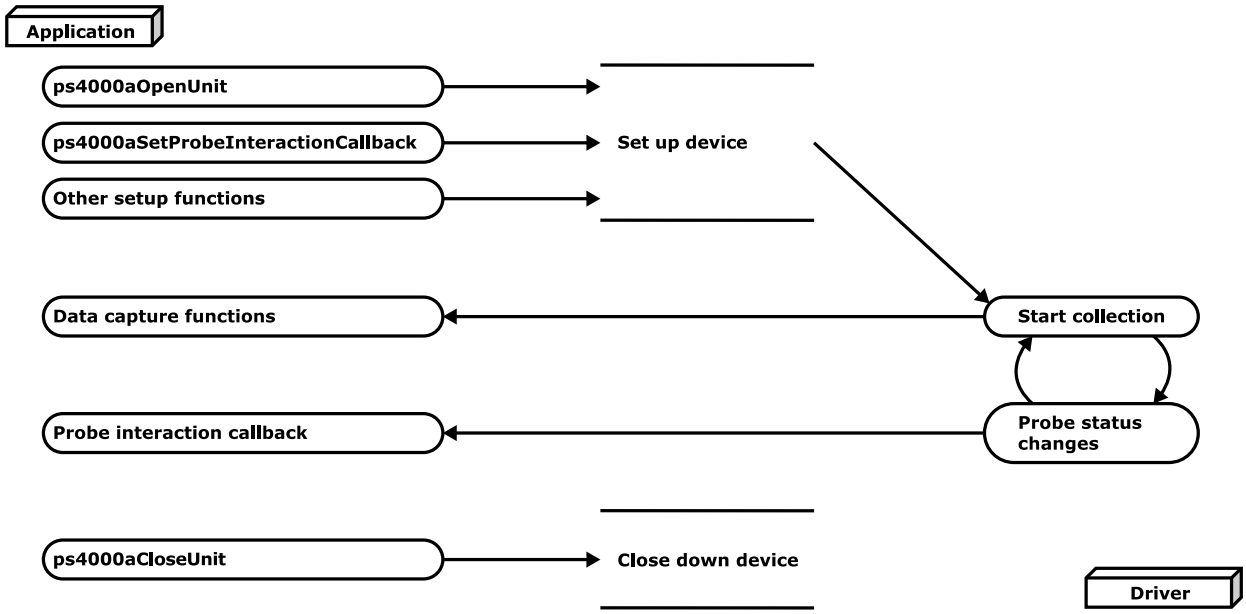
Some devices in the PicoScopes 4000 Series have a PicoConnect™ intelligent probe interface. This interface supplies power to the probe as well as allowing the scope to configure and interrogate the probe. Your application can choose to be alerted whenever a probe is connected or disconnected, or when its status changes.

Probe interactions use a callback mechanism, available in C and similar languages. For languages that do not support callbacks, use the wrapper functions provided.

Applicability	PicoScope 4444 only
Note	In addition to `ps4000aApi.h`, you must also include `PicoConnectProbes.h`. This file contains definitions of enumerated types that describe the PicoConnect probes.

Procedure

1. Define your own function to receive probe interaction callbacks.
2. Call `ps4000aOpenUnit()` to obtain a device handle.
3. Call `ps4000aSetProbeInteractionCallback()` to register your probe interaction callback function.
4. Capture data using the desired sampling mode. See Sampling modes for details.
5. Call `ps4000aCloseUnit()` to release the device handle. The makes the scope device available to other applications.



4 API functions

The ps4000a API exports the following functions for you to use in your own applications. All functions are C functions using the standard call naming convention (__stdcall). They are all exported with both decorated and undecorated names.

4.1 ps4000aChangePowerSource() – handle dual-port USB powering

```
PICO_STATUS ps4000aChangePowerSource
(
  int16_t             handle,
  PICO_STATUS         powerstate
)
```

This function selects the power supply mode.

Whenever the power supply mode is changed, all data and settings in the scope device are lost. You must then reconfigure the device before restarting capture.

PicoScope 4444 only
The PicoScope 4444 can use DC power from either a USB 2.0 or a USB 3.0 port. USB 3.0 might be needed if the probes connected draw enough supply current. If another function returns `PICO_PROBE_POWER_DC_POWER_SUPPLY_REQUIRED` or `PICO_PROBE_NOT_POWERED_WITH_DC_POWER_SUPPLY`, you must call this function to change to the correct power source.

The PicoScope 4444 returns `PICO_POWER_SUPPLY_NOT_CONNECTED` if the DC power supply is not connected.

All USB 3.0 devices
When the device is plugged into a non-USB 3.0 port, it requires a two-stage power-up sequence. You must call this function if any of the following conditions arises:

- USB power is required.
- The power supply is connected or disconnected during use.
- A 2-channel USB 3.0 scope is plugged into a USB 2.0 port (indicated if any function returns the `PICO_USB3_0_DEVICE_NON_USB3_0_PORT` status code).

If you receive the `PICO_USB3_0_DEVICE_NON_USB3_0_PORT` status code from one of the `ps4000aOpenUnit…()` functions (`ps4000aOpenUnit()`, `ps4000aOpenUnitWithResolution()`, `ps4000aOpenUnitAsync()` or `ps4000aOpenUnitProgress()`), you must then call `ps4000aChangePowerSource()` to switch the device into non-USB 3.0-power mode.

Note. The PicoScope 4824 has two power supply options:
1. *To power it from a USB 3.0 port, use the USB 3.0 cable supplied.*
2. *To power it from a non-USB 3.0 port, use a double-headed USB 2.0 cable (available separately) and plug it into two USB 2.0 ports on the host machine.*

Applicability	All modes

Arguments	`handle`, identifier for the scope device.
	
	`powerstate`, the required state of the unit.
	USB 3.0 devices
	Set to one of:
	`PICO_POWER_SUPPLY_CONNECTED`
	– to use power from the external power supply
	`PICO_POWER_SUPPLY_NOT_CONNECTED`
	– to use power from the USB port
	`PICO_USB3_0_DEVICE_NON_USB3_0_PORT`
	– to use power from a non-USB 3.0 port

	USB 2.0 devices Set to one of: PICO_PROBE_POWER_DC_POWER_SUPPLY_REQUIRED – to use external DC power PICO_PROBE_NOT_POWERED_WITH_DC_POWER_SUPPLY – to use USB power
Returns	PICO_OK PICO_POWER_SUPPLY_REQUEST_INVALID PICO_INVALID_PARAMETER PICO_NOT_RESPONDING PICO_INVALID_HANDLE PICO_PROBE_POWER_DC_POWER_SUPPLY_REQUIRED PICO_PROBE_NOT_POWERED_WITH_DC_POWER_SUPPLY PICO_DRIVER_FUNCTION PICO_FPGA_FAIL PICO_INTERNAL_ERROR PICO_MEMORY PICO_NOT_RESPONDING PICO_PROBE_CONFIG_FAILURE PICO_RESOURCE_ERROR PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_RESPONDING

4.2 ps4000aCloseUnit() – close a scope device

```
PICO_STATUS ps4000aCloseUnit
(
  int16_t       handle
)
```

This function disconnects the PicoScope device from the ps4000a driver. Once disconnected, the device can then be opened or enumerated by this or another application.

Applicability	All modes
Arguments	handle, identifier for the scope device.
Returns	PICO_OK PICO_HANDLE_INVALID PICO_DRIVER_FUNCTION

4.3 ps4000aCurrentPowerSource() – read current power source

```
PICO_STATUS ps4000aCurrentPowerSource
(
  int16_t       handle
)
```

This function returns the current power state of the device.

PicoScope 4824: there is no need to call this function as the device has only one possible state. Normally returns PICO_OK.

PicoScope 4444: returns PICO_POWER_SUPPLY_NOT_CONNECTED if device is USB-powered; returns PICO_POWER_SUPPLY_CONNECTED if DC power supply is connected.

Applicability	PicoScope 4444 only
Arguments	handle, identifier for the scope device.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_NOT_RESPONDING PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING

4.4 ps4000aEnumerateUnits() – find out how many units are connected

```
PICO_STATUS ps4000aEnumerateUnits
(
  int16_t       * count,
  int8_t        * serials,
  int16_t       * serialLth
)
```

This function counts the number of PicoScope 4000 Series (A API) units connected to the computer, and returns a list of serial numbers as a string. Note that this function will only detect devices that are not yet being controlled by an application.

Applicability	All modes
Arguments	* count, on exit, the number of scopes 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` `* serialLth`, on entry, the length of the `int8_t` buffer pointed to by `serials`; on exit, the length of the string written to `serials`.

Returns	`PICO_OK`
	`PICO_BUSY`
	`PICO_NULL_PARAMETER`
	`PICO_FW_FAIL`
	`PICO_CONFIG_FAIL`
	`PICO_MEMORY_FAIL`
	`PICO_ANALOG_BOARD`
	`PICO_CONFIG_FAIL_AWG`
	`PICO_INITIALISE_FPGA`
	`PICO_INTERNAL_ERROR`
	`PICO_TIMEOUT`
	`PICO_RESOURCE_ERROR`
	`PICO_DEVICE_NOT_FUNCTIONING`

4.5 ps4000aFlashLed() – flash the front-panel LED

```
PICO_STATUS ps4000aFlashLed
(
  int16_t       handle,
  int16_t       start
)
```

This function flashes the LED on the front of the scope without blocking the calling thread. Calls to ps4000aRunStreaming() and ps4000aRunBlock() cancel any flashing started by this function.

Applicability	All modes
Arguments	handle, identifier for 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_MEMORY PICO_INTERNAL_ERROR PICO_POWER_SUPPLY_UNDERVOLTAGE PICO_NOT_RESPONDING PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING

4.6 ps4000aGetAnalogueOffset() – find the allowable analog offset range

```
PICO_STATUS ps4000aGetAnalogueOffset
(
  int16_t                         handle,
  PICO_CONNECT_PROBE_RANGE        range,
  PS4000A_COUPLING                coupling,
  float                           * maximumVoltage,
  float                           * minimumVoltage
)
```

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

Applicability	All modes
Arguments	handle, identifier for the scope device. range, the voltage range to be used when gathering the min and max information. coupling, the type of AC/DC coupling used. * maximumVoltage, on exit, the maximum voltage allowed for the range. Pointer may be NULL if not required. * minimumVoltage, on exit, the minimum voltage allowed for the range. Pointer may be NULL if not required. If both maximumVoltage and minimumVoltage are NULL, the driver returns PICO_NULL_PARAMETER.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_INVALID_VOLTAGE_RANGE PICO_NULL_PARAMETER PICO_MEMORY PICO_INTERNAL_ERROR

4.7 ps4000aGetChannelInformation() – find out if extra ranges available

```
PICO_STATUS ps4000aGetChannelInformation
(
  int16_t                   handle,
  PS4000A_CHANNEL_INFO      info,
  int32_t                   probe,
  int32_t                   * ranges,
  int32_t                   * length,
  int32_t                   channels
)
```

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

Applicability	Reserved for future expansion
Arguments	`handle`, identifier for the scope device. `info`, the type of information required. The only value supported is: `PS4000A_CI_RANGES`, returns the extra ranges available `probe`, not used, must be set to 0. `* ranges`, on exit, an array populated with available ranges for the given value of `info`. May be `NULL`. See `ps4000aSetChannel()` for possible values. `* length`, on entry: the length of the `ranges` array; on exit: the number of elements written to `ranges` or, if `ranges` is `NULL`, the number of elements that would have been written. `channels`, the channel for which the information is required. See `ps4000aSetChannel()` for possible values.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`

4.8 ps4000aGetCommonModeOverflow() – find out which channels have overflowed

```
PICO_STATUS ps4000aGetCommonModeOverflow
(
  int16_t       handle,
  uint16_t      * overflow
)
```

On each channel of a differential oscilloscope, both the positive and negative differential input voltages must remain within the specified limits to avoid measurement errors. These limits are independent of the differential voltage limit, which is the maximum voltage difference allowed between the two inputs.

This function queries whether any channel has exceeded the common mode voltage limit.

Applicability	PicoScope 4444 only
Arguments	`handle`, identifier for the scope device. `overflow`, a set of flags that indicate whether a common-mode overflow has occurred on any of the channels. It is a bit pattern with bit 0 denoting Channel A.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`
	`PICO_DRIVER_FUNCTION`
	`PICO_NOT_SUPPORTED_BY_THIS_DEVICE`
	`PICO_BUSY`
	`PICO_MEMORY_FAIL`
	`PICO_INTERNAL_ERROR`
	`PICO_TIMEOUT`
	`PICO_RESOURCE_ERROR`
	`PICO_DEVICE_NOT_FUNCTIONING`

4.9 ps4000aGetDeviceResolution() – query the ADC resolution

```
PICO_STATUS ps4000aGetDeviceResolution
(
  int16_t                         handle,
  PS4000A_DEVICE_RESOLUTION       * resolution
)
```

This function retrieves the ADC resolution that is in use on the specified device.

Applicability	PicoScope 4444 only
Arguments	`handle,` the handle of the required device `* resolution,` returns the resolution of the device. Values are defined by PS4000A_DEVICE_RESOLUTION.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_NULL_PARAMETER

4.10 ps4000aGetMaxDownSampleRatio() – find out downsampling ratio for data

```
PICO_STATUS ps4000aGetMaxDownSampleRatio
(
  int16_t                   handle,
  uint32_t                  noOfUnaggregatedSamples,
  uint32_t                  * maxDownSampleRatio,
  PS4000A_RATIO_MODE        downSampleRatioMode,
  uint32_t                  segmentIndex
)
```

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

Applicability	All modes
Arguments	`handle`, identifier for the scope device. `noOfUnaggregatedSamples`, the number of raw samples to be used to calculate the maximum downsampling ratio. `* maxDownSampleRatio`, on exit, the maximum possible downsampling ratio. `downSampleRatioMode`, see Downsampling. `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`
	`PICO_DRIVER_FUNCTION`
	`PICO_NOT_USED`
	`PICO_BUSY`

4.11 ps4000aGetMaxSegments() – get maximum number of memory segments

```
PICO_STATUS ps4000aGetMaxSegments
(
  int16_t       handle,
  uint32_t      * maxSegments
)
```

This function retrieves the maximum number of memory segments allowed by the device.

Applicability	All modes
Arguments	`handle`, identifier for the scope device. `* maxSegments`, on exit, the maximum possible number of memory segments. This information can also be found in the data sheet for the device.

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

4.12 ps4000aGetNoOfCaptures() – get number of rapid block captures

```
PICO_STATUS ps4000aGetNoOfCaptures
(
  int16_t       handle,
  uint32_t      * nCaptures
)
```

This function gets the number of captures collected in one run of rapid block mode. You can call `ps4000aGetNoOfCaptures` during device capture, after collection has completed or after interrupting waveform collection by calling `ps4000aStop()`.

Applicability	Rapid block mode
Arguments	`handle`, identifier for the scope device. `* nCaptures`, on exit, the number of waveforms captured.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`
	`PICO_DRIVER_FUNCTION`
	`PICO_NO_SAMPLES_AVAILABLE`
	`PICO_NOT_USED_IN_THIS_CAPTURE_MODE`
	`PICO_MEMORY_FAIL`
	`PICO_INTERNAL_ERROR`
	`PICO_TIMEOUT`
	`PICO_RESOURCE_ERROR`
	`PICO_DEVICE_NOT_FUNCTIONING`
	`PICO_NOT_RESPONDING`

4.13 ps4000aGetNoOfProcessedCaptures() – get number of downsampled rapid block captures

```
PICO_STATUS ps4000aGetNoOfProcessedCaptures
(
  int16_t       handle,
  uint32_t      * nProcessedCaptures
)
```

This function gets the number of captures collected and processed in one run of rapid block mode. It enables your application to start processing captured data while the driver is still transferring later captures from the device to the computer.

The function returns the number of captures the driver has processed since you called `ps4000aRunBlock()`. It is for use in rapid block mode, alongside `ps4000aGetValuesOverlappedBulk()`, when the driver is set to transfer data from the device automatically as soon as the `ps4000aRunBlock()` function is called. You can call `ps4000aGetNoOfProcessedCaptures()` during device capture, after collection has completed or after interrupting waveform collection by calling `ps4000aStop()`.

The returned value (`nProcessedCaptures`) can then be used to iterate through the number of segments using `ps4000aGetValues()`, or in a single call to `ps4000aGetValuesBulk()`, where it is used to calculate the `toSegmentIndex` parameter.

When capture is stopped
If `nProcessedCaptures` = 0, you will also need to call `ps4000aGetNoOfCaptures()`, in order to determine how many waveform segments were captured, before calling `ps4000aGetValues()` or `ps4000aGetValuesBulk()`.

Applicability	Rapid block mode
Arguments	`handle`, identifier for the scope device. `* nProcessedCaptures`, on exit, the number of waveforms captured and processed.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`
	`PICO_DRIVER_FUNCTION`
	`PICO_NULL_PARAMETER`
	`PICO_NOT_USED_IN_THIS_CAPTURE_MODE`

4.14 ps4000aGetStreamingLatestValues() – get streaming data while scope is running

```
PICO_STATUS ps4000aGetStreamingLatestValues
(
  int16_t                  handle,
  ps4000aStreamingReady    lpPs4000Ready,
  void                     * pParameter
)
```

This function is used to collect the next block of values while streaming is running. You must call ps4000aRunStreaming() beforehand to set up streaming.

Applicability	Streaming mode only
Arguments	handle, identifier for the scope device. lpPs4000Ready, a pointer to your ps4000aStreamingReady() callback function that will return the latest downsampled values. pParameter, a void pointer that will be passed to the ps4000aStreamingReady() callback function.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_INVALID_CALL PICO_BUSY PICO_NOT_RESPONDING PICO_DRIVER_FUNCTION PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_NOT_RESPONDING PICO_POWER_SUPPLY_UNDERVOLTAGE PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_STREAMING_FAILED

4.15 ps4000aGetTimebase() – find out what timebases are available

```
PICO_STATUS ps4000aGetTimebase
(
  int16_t            handle,
  uint32_t           timebase,
  int32_t            noSamples,
  int32_t            * timeIntervalNanoseconds,
  int32_t            * maxSamples
  uint32_t           segmentIndex
)
```

This function discovers which timebases are available on the oscilloscope. You should set up the channels using ps4000aSetChannel() first.

Applicability	All modes
Arguments	handle, identifier for the scope device. timebase, a code between 0 and $2^{32}-1$ that specifies the sampling interval (see Timebases). noSamples, the number of samples required. * timeIntervalNanoseconds, on exit, the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. * maxSamples, on exit, the maximum number of samples available. The scope allocates a certain amount of memory for internal overheads and this may vary depending on the number of segments, number of channels enabled, and the timebase chosen. If this pointer is null, nothing will be written here. segmentIndex, the number 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_DRIVER_FUNCTION PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_TIMEBASE

4.16 ps4000aGetTimebase2() – find out what timebases are available

```
PICO_STATUS ps4000aGetTimebase2
(
  int16_t             handle,
  uint32_t            timebase,
  int32_t             noSamples,
  float               * timeIntervalNanoseconds,
  int32_t             * maxSamples,
  uint32_t            segmentIndex
)
```

This function differs from ps4000aGetTimebase() only in the type of the timeIntervalNanoseconds argument.

Applicability	All modes
Arguments	handle, timebase, noSamples, see ps4000aGetTimebase(). * timeIntervalNanoseconds, on exit, the time interval between readings at the selected timebase. If a null pointer is passed, nothing will be written here. maxSamples, segmentIndex, see ps4000aGetTimebase().
Returns	See ps4000aGetTimebase().

4.17 ps4000aGetTriggerTimeOffset() – read trigger timing adjustments (32-bit)

```
PICO_STATUS ps4000aGetTriggerTimeOffset
(
  int16_t                   handle,
  uint32_t                  * timeUpper,
  uint32_t                  * timeLower,
  PS4000A_TIME_UNITS        * timeUnits,
  uint32_t                  segmentIndex
)
```

This function gets the trigger time offset for waveforms in block mode or rapid block mode. The trigger time offset is an adjustment value used for correcting jitter in the waveform, and is intended mainly for applications that wish to display the waveform with reduced jitter. The offset is zero if the waveform crosses the threshold at the trigger sampling instant, or a positive or negative value if jitter correction is required. The value should be added to the nominal trigger time to get the corrected trigger time.

Call this function after data has been captured or when data has been retrieved from a previous capture.

This function is provided for use in programming environments that do not support 64-bit integers. Another version of this function, ps4000aGetTriggerTimeOffset64(), is available that returns the time as a single 64-bit value.

Applicability	Block mode and rapid block mode
Arguments	handle, identifier for the scope 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, on exit, the time units in which * timeUpper and * timeLower are measured. The allowable values are: PS4000A_FS PS4000A_PS PS4000A_NS PS4000A_US PS4000A_MS PS4000A_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 PICO_NOT_USED_IN_THIS_CAPTURE_MODE PICO_TRIGGER_ERROR PICO_FW_FAIL PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_RESPONDING PICO_MEMORY_FAIL PICO_INTERNAL_ERROR

4.18 ps4000aGetTriggerTimeOffset64() – read trigger timing adjustments (64-bit)

```
PICO_STATUS ps4000aGetTriggerTimeOffset64
(
  int16_t                  handle,
  int64_t                  * time,
  PS4000A_TIME_UNITS       * timeUnits,
  uint32_t                 segmentIndex
)
```

This function gets the trigger time offset for a waveform. It is equivalent to ps4000aGetTriggerTimeOffset() except that the time offset is returned as a single 64-bit value instead of two 32-bit values.

Applicability	Block mode and rapid block mode
Arguments	handle, identifier for the scope device. * time, on exit, the time at which the trigger point occurred. * timeUnits, on exit, the time units in which time is measured. See ps4000aGetTriggerTimeOffset(). 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 PICO_NOT_USED_IN_THIS_CAPTURE_MODE PICO_TRIGGER_ERROR PICO_FW_FAIL PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_RESPONDING PICO_MEMORY_FAIL PICO_INTERNAL_ERROR

4.19 ps4000aGetUnitInfo() – read information about scope device

```
PICO_STATUS ps4000aGetUnitInfo
(
  int16_t       handle,
  int8_t        * string,
  int16_t       stringLength,
  int16_t       * requiredSize,
  PICO_INFO     info
)
```

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

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

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

`PICO_INFO` constant	**Example**
`0: PICO_DRIVER_VERSION,` version number of `ps4000a` DLL	`1.0.4.56`
`1: PICO_USB_VERSION,` type of USB connection to device: 1.1, 2.0 or 3.0	`3.0`
`2: PICO_HARDWARE_VERSION,` hardware version of device	`1`
`3: PICO_VARIANT_INFO,` variant number of device	`4824`
`4: PICO_BATCH_AND_SERIAL,` batch and serial number of device	`KJ087/0006`
`5: PICO_CAL_DATE,` calibration date of device	`11Nov13`
`6: PICO_KERNEL_VERSION,` version of kernel driver	`1.0`
`7: PICO_DIGITAL_HARDWARE_VERSION,` version of digital board	`1`
`8: PICO_ANALOGUE_HARDWARE_VERSION,` version of analog board	`1`
`9: PICO_FIRMWARE_VERSION_1`	`1.4.0.0`
`10: PICO_FIRMWARE_VERSION_2`	`0.9.15.0`

4.20 ps4000aGetValues() – retrieve block-mode data

```
PICO_STATUS ps4000aGetValues
(
  int16_t                  handle,
  uint32_t                 startIndex,
  uint32_t                 * noOfSamples,
  uint32_t                 downSampleRatio,
  PS4000A_RATIO_MODE       downSampleRatioMode,
  uint32_t                 segmentIndex,
  int16_t                  * overflow
)
```

This function retrieves block-mode 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, and store it in a user buffer previously passed to `ps4000aSetDataBuffer()` or `ps4000aSetDataBuffers()`. It blocks the calling function while retrieving data.

If multiple channels are enabled, a single call to this function is sufficient to retrieve data for all channels.

Note that if you are using block mode and call this function before the oscilloscope is ready, no capture will be available and the driver will return `PICO_NO_SAMPLES_AVAILABLE`.

Applicability	Block mode and rapid block mode
Arguments	handle, identifier for the scope device. startIndex, a zero-based index that indicates the start point for data collection. It is measured in sample intervals from the start of the buffer. * noOfSamples, on entry, the number of samples requested; on exit, the number of samples actually retrieved. downSampleRatio, the downsampling factor that will be applied to the raw data. Multiple downsampling modes can be bitwise-ORed together, but the downSampleRatio must be the same for all modes. downSampleRatioMode, whether to use downsampling to reduce the amount of data. See Downsampling. 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 pattern, with bit 0 corresponding to 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_DRIVER_FUNCTION PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_NOT_RESPONDING PICO_POWER_SUPPLY_UNDERVOLTAGE PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_BUFFERS_NOT_SET PICO_INVALID_PARAMETER PICO_INVALID_SAMPLERATIO PICO_ETS_NOT_RUNNING PICO_MEMORY_FAIL PICO_INTERNAL_ERROR PICO_RESOURCE_ERROR

4.21 ps4000aGetValuesAsync() – retrieve block or streaming data

```
PICO_STATUS ps4000aGetValuesAsync
(
  int16_t                 handle,
  uint32_t                startIndex,
  uint32_t                noOfSamples,
  uint32_t                downSampleRatio,
  PS4000A_RATIO_MODE      downSampleRatioMode,
  uint32_t                segmentIndex,
  void                    * lpDataReady,
  void                    * pParameter
)
```

This function returns data, either with or without downsampling, starting at the specified sample number. It can be used in block mode to retrieve data from the device, using a callback so as not to block the calling function. It can also be used in streaming mode to retrieve data from the driver, but in this case it blocks the calling function.

Applicability	Block mode and streaming mode
Arguments	handle, identifier for the scope device. startIndex, see ps4000aGetValues() noOfSamples, see ps4000aGetValues() downSampleRatio, see ps4000aGetValues() downSampleRatioMode, see ps4000aGetValues() segmentIndex, see ps4000aGetValues() * lpDataReady, the ps4000aStreamingReady() function that is called when the data is ready pParameter, a void pointer that will be passed to the ps4000aStreamingReady() callback function. The data type depends on the design of the callback function, which is determined by the application programmer.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NO_SAMPLES_AVAILABLE PICO_DEVICE_SAMPLING – streaming only PICO_NULL_PARAMETER PICO_STARTINDEX_INVALID PICO_SEGMENT_OUT_OF_RANGE PICO_INVALID_PARAMETER PICO_DATA_NOT_AVAILABLE PICO_INVALID_SAMPLERATIO PICO_INVALID_CALL PICO_DRIVER_FUNCTION PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_NOT_RESPONDING PICO_POWER_SUPPLY_UNDERVOLTAGE PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_BUFFERS_NOT_SET PICO_INTERNAL_ERROR PICO_MEMORY

4.22 ps4000aGetValuesBulk() – retrieve more than one waveform at a time

```
PICO_STATUS ps4000aGetValuesBulk
(
  int16_t                  handle,
  uint32_t                 * noOfSamples,
  uint32_t                 fromSegmentIndex,
  uint32_t                 toSegmentIndex,
  unit32_t                 downSampleRatio,
  PS4000A_RATIO_MODE       downSampleRatioMode,
  int16_t                  * overflow
)
```

This function allows more than one waveform to be retrieved at a time in rapid block mode. The waveforms must have been collected sequentially and in the same run.

If multiple channels are enabled, a single call to this function is sufficient to retrieve data for all channels.

Applicability	Rapid block mode
Arguments	`handle`, identifier for the scope 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 waveforms should be retrieved `toSegmentIndex`, the last segment from which waveforms should be retrieved `downSampleRatio`, see Downsampling `downSampleRatioMode`, see Downsampling `* overflow`, an array of at least as many integers as the number of waveforms to be retrieved. Each segment index has a separate `overflow` element, with `overflow[0]` containing the `fromSegmentIndex` and the last index the `toSegmentIndex`. Each element in the array is a bit field as described under `ps4000aGetValues()`.
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 PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_NOT_RESPONDING PICO_POWER_SUPPLY_UNDERVOLTAGE PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_NO_CAPTURES_AVAILABLE PICO_NOT_USED_IN_THIS_CAPTURE_MODE PICO_CAPTURING_DATA PICO_INVALID_SAMPLERATIO

4.23 ps4000aGetValuesOverlapped() – retrieve data in overlapping blocks

```
PICO_STATUS ps4000aGetValuesOverlapped
(
  int16_t                  handle,
  uint32_t                 startIndex,
  uint32_t                 * noOfSamples,
  uint32_t                 downSampleRatio,
  PS4000A_RATIO_MODE       downSampleRatioMode,
  uint32_t                 segmentIndex,
  int16_t                  * overflow
)
```

This function allows you to make a deferred data-collection request in block mode. The request will be executed, and the arguments validated, when you call ps4000aRunBlock(). The advantage of this function is that the driver makes contact with the scope only once, when you call ps4000aRunBlock(), compared with the two contacts that occur when you use the conventional ps4000aRunBlock(), ps4000aGetValues() calling sequence. This slightly reduces the dead time between successive captures in block mode.

After calling ps4000aRunBlock(), you can optionally use ps4000aGetValues() to request further copies of the data. This might be required if you wish to display the data with different data reduction settings.

If multiple channels are enabled, a single call to this function is sufficient to retrieve data for all channels.

Applicability	Block mode
Arguments	handle, startIndex, * noOfSamples, downSampleRatio, downSampleRatioMode, segmentIndex: see ps4000aGetValues() * overflow: see ps4000aGetValuesBulk()
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_NOT_RESPONDING PICO_POWER_SUPPLY_UNDERVOLTAGE

4.23.1 Using the GetValuesOverlapped functions

This procedure is similar to that described in Using block mode, with differences shown in *italics*:

1. Open the oscilloscope using `ps4000aOpenUnit()`.
2. Select channel ranges and AC/DC coupling using `ps4000aSetChannel()`.
3. Using `ps4000aGetTimebase()`, select timebases until the required nanoseconds per sample is located.
4. Use the trigger setup functions `ps4000aSetTriggerChannelDirections()` and `ps4000aSetTriggerChannelProperties()` to set up the trigger if required.

4a. Use `ps4000aSetDataBuffer()` to tell the driver where your memory buffer is.

4b. Set up the transfer of the block of data from the oscilloscope using `ps4000aGetValuesOverlapped()`.

5. Start the oscilloscope running using `ps4000aRunBlock()`.
6. Wait until the oscilloscope is ready using the `ps4000aBlockReady()` callback (or poll using `ps4000aIsReady()`).

7. (not needed)

8. (not needed)

9. Display the data.
10. Repeat steps 5 to 9 if needed.
11. Stop the oscilloscope using `ps4000aStop()`.
12. Request new views of stored data using different downsampling parameters: see Retrieving stored data.
13. Close the device using `ps4000aCloseUnit()`.

A similar procedure can be used with rapid block mode using `ps4000aGetValuesOverlappedBulk()`.

4.24 ps4000aGetValuesOverlappedBulk() – retrieve overlapping data from multiple segments

```
PICO_STATUS ps4000aGetValuesOverlappedBulk
(
  int16_t                  handle,
  uint32_t                 startIndex,
  uint32_t                 * noOfSamples,
  uint32_t                 downSampleRatio,
  PS4000A_RATIO_MODE       downSampleRatioMode,
  uint32_t                 fromSegmentIndex,
  uint32_t                 toSegmentIndex,
  int16_t                  * overflow
)
```

This function requests data from multiple segments in rapid block mode. It is similar to calling ps4000aGetValuesOverlapped() multiple times, but more efficient.

Applicability	Rapid block mode
Arguments	handle, startIndex, * noOfSamples, downSampleRatio, downSampleRatioMode: see ps4000aGetValues() fromSegmentIndex, toSegmentIndex, * overflow, see ps4000aGetValuesBulk()
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_INVALID_PARAMETER PICO_DRIVER_FUNCTION PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_NOT_RESPONDING PICO_POWER_SUPPLY_UNDERVOLTAGE

4.25 ps4000aGetValuesTriggerTimeOffsetBulk() – get trigger timing adjustments (multiple)

```
PICO_STATUS ps4000aGetValuesTriggerTimeOffsetBulk
(
  int16_t                  handle,
  uint32_t                 * timesUpper,
  uint32_t                 * timesLower,
  PS4000A_TIME_UNITS       * timeUnits,
  uint32_t                 fromSegmentIndex,
  uint32_t                 toSegmentIndex
)
```

This function retrieves the trigger time offset for multiple waveforms obtained in block mode or rapid block mode. It is a more efficient alternative to calling `ps4000aGetTriggerTimeOffset()` once for each waveform required. See `ps4000aGetTriggerTimeOffset()` for an explanation of trigger time offsets.

This function is provided for use in programming environments that do not support 64-bit integers. If your programming environment does support 64-bit integers, it is easier to use `ps4000aGetValuesTriggerTimeOffsetBulk64()`.

Applicability	Rapid block mode
Arguments	`handle`, identifier for the scope device. `* timesUpper`, an array of integers. On exit, the most significant 32 bits of the time offset for each requested segment index. `times[0]` will hold the `fromSegmentIndex` time offset and the last `times` index will hold the `toSegmentIndex` time offset. The array must be long enough to hold the number of requested times. `* timesLower`, an array of integers. On exit, the least significant 32 bits of the time offset for each requested segment index. `times[0]` will hold the `fromSegmentIndex` time offset and the last `times` index will hold the `toSegmentIndex` time offset. The array size must be long enough to hold the number of requested times. `* timeUnits`, an array of integers. The array must be long enough to hold the number of requested times. On exit, `timeUnits[0]` will contain the time unit for `fromSegmentIndex` and the last element will contain the time unit for `toSegmentIndex`. Refer to `ps4000aGetTriggerTimeOffset()` for specific figures. `fromSegmentIndex`, the first segment for which the time offset is required. `toSegmentIndex`, the last segment for which the time offset is required. If `toSegmentIndex` is less than `fromSegmentIndex` then the driver will wrap around from the last segment to the first.

Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_NOT_USED_IN_THIS_CAPTURE_MODE PICO_NOT_RESPONDING PICO_NULL_PARAMETER PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE PICO_DRIVER_FUNCTION

4.26 ps4000aGetValuesTriggerTimeOffsetBulk64() – get trigger timing adjustments (multiple)

```
PICO_STATUS ps4000aGetValuesTriggerTimeOffsetBulk64
(
  int16_t                  handle,
  int64_t                  * times,
  PS4000A_TIME_UNITS       * timeUnits,
  uint32_t                 fromSegmentIndex,
  uint32_t                 toSegmentIndex
)
```

This function is equivalent to ps4000aGetValuesTriggerTimeOffsetBulk() but retrieves the trigger time offsets as 64-bit values instead of pairs of 32-bit values.

Applicability	Rapid block mode
Arguments	handle, identifier for the scope device. * times, an array of integers. On exit, this will hold the time offset for each requested segment index. times[0] will hold the time offset for fromSegmentIndex, and the last times index will hold the time offset for toSegmentIndex. The array must be long enough to hold the number of times requested. * timeUnits, an array of integers long enough to hold the number of requested times. timeUnits[0] will contain the time unit for fromSegmentIndex, and the last element will contain the toSegmentIndex. Refer to ps4000aGetTriggerTimeOffset64() for specific figures. fromSegmentIndex, the first segment for which the time offset is required. The results for this segment will be placed in times[0] and timeUnits[0]. toSegmentIndex, the last segment for which the time offset is required. The results for this segment will be placed in the last elements of the times and timeUnits arrays. If toSegmentIndex is less than fromSegmentIndex, then the driver will wrap around from the last segment to the first.
Returns	PICO_OK PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_INVALID_HANDLE PICO_NOT_USED_IN_THIS_CAPTURE_MODE PICO_NOT_RESPONDING PICO_NULL_PARAMETER PICO_DEVICE_SAMPLING PICO_SEGMENT_OUT_OF_RANGE PICO_NO_SAMPLES_AVAILABLE PICO_DRIVER_FUNCTION

4.27 ps4000aIsLedFlashing() – read status of LED

```
PICO_STATUS ps4000aIsLedFlashing
(
  int16_t       handle,
  int16_t       * status
)
```

This function reports whether or not the LED is flashing.

Applicability	All modes
Arguments	`handle`, identifier for the scope device. `status`, returns a flag indicating the status of the LED: <> 0 : flashing 0 : not flashing

Returns	`PICO_OK`
	`PICO_HANDLE_INVALID`
	`PICO_NULL_PARAMETER`
	`PICO_DRIVER_FUNCTION`
	`PICO_NOT_SUPPORTED_BY_THIS_DEVICE`
	`PICO_NOT_USED`

4.28 ps4000aIsReady() – poll the driver in block mode

```
PICO_STATUS ps4000aIsReady
(
  int16_t       handle,
  int16_t       * ready
)
```

This function may be used instead of a callback function to receive data from ps4000aRunBlock(). To use this method, pass a NULL pointer as the lpReady argument to ps4000aRunBlock(). You must then poll the driver to see if it has finished collecting the requested samples.

Applicability	Block mode
Arguments	handle, identifier for the scope device. ready, on exit, indicates the state of the collection. If zero, the device is still collecting. If non-zero, the device has finished collecting and ps4000aGetValues() can be used to retrieve the data.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_NULL_PARAMETER PICO_NO_SAMPLES_AVAILABLE PICO_CANCELLED PICO_NOT_RESPONDING

4.29 ps4000aIsTriggerOrPulseWidthQualifierEnabled() – find out whether trigger is enabled

```
PICO_STATUS ps4000aIsTriggerOrPulseWidthQualifierEnabled
(
  int16_t       handle,
  int16_t       * triggerEnabled,
  int16_t       * pulseWidthQualifierEnabled
)
```

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

Applicability	Call after setting up the trigger, and just before calling either ps4000aRunBlock() or ps4000aRunStreaming().
Arguments	handle, identifier for the scope device. * triggerEnabled, on exit, indicates whether the trigger will successfully be set when ps4000aRunBlock() or ps4000aRunStreaming() is called. A non-zero value indicates that the trigger is set, otherwise the trigger is not set. * pulseWidthQualifierEnabled, on exit, indicates whether the pulse width qualifier will successfully be set when ps4000aRunBlock() or ps4000aRunStreaming() is called. A non-zero value indicates that the pulse width qualifier is set, otherwise the pulse width qualifier is not set.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_NULL_PARAMETER PICO_DRIVER_FUNCTION

4.30 ps4000aMaximumValue() – get maximum allowed sample value

```
PICO_STATUS ps4000aMaximumValue
(
  int16_t       handle,
  int16_t       * value
)
```

This function returns the maximum possible sample value in the current operating mode.

Applicability	All modes
Arguments	`handle`, identifier for the scope device. `* value`, on exit, the maximum value.

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

4.31 ps4000aMemorySegments() – divide scope memory into segments

```
PICO_STATUS ps4000aMemorySegments
(
  int16_t       handle,
  uint32_t      nSegments,
  int32_t       * nMaxSamples
)
```

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

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

Applicability	Block mode, rapid block mode
Arguments	`handle`, identifier for the scope device. `nSegments`, the number of segments to be used, from 1 to the number returned by `ps4000aGetMaxSegments()`. `* nMaxSamples`, on exit, the number of samples that are available in each segment. This is the total number over all channels, so if more than one channel is in use, the number of samples available to each channel is `nMaxSamples` divided by 2 (for 2 channels) or 4 (for 3 or 4 channels) or 8 (for 5 to 8 channels).

Returns	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_TOO_MANY_SEGMENTS`
	`PICO_DRIVER_FUNCTION`
	`PICO_MEMORY_FAIL`

4.32 ps4000aMinimumValue() – get minimum allowed sample value

```
PICO_STATUS ps4000aMinimumValue
(
  int16_t       handle,
  int16_t       * value
)
```

This function returns the minimum possible sample value in the current operating mode.

Applicability	All modes
Arguments	`handle`, identifier for the scope device. `* value`, on exit, the minimum value.

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

4.33 ps4000aNoOfStreamingValues() – get number of samples in streaming mode

```
PICO_STATUS ps4000aNoOfStreamingValues
(
  int16_t       handle,
  uint32_t      * noOfValues
)
```

This function returns the number of raw samples available after data collection in streaming mode. Call it after ps4000aStop().

Applicability	Streaming mode.
Arguments	handle, identifier for the scope 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

4.34 ps4000aOpenUnit() – open a scope device

```
PICO_STATUS ps4000aOpenUnit
(
  int16_t       * handle,
  int8_t        * serial
)
```

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

PicoScope 4824 only: If the function returns PICO_USB3_0_DEVICE_NON_USB3_0_PORT, the application must call ps4000aChangePowerSource() to complete the two-stage power-up sequence for a USB 2.0 port (or USB 3.0 port with USB 2.0 cable). Returns PICO_OK if connected to a USB 3.0 port.

PicoScope 4444 only: If the function returns PICO_POWER_SUPPLY_NOT_CONNECTED, the application must call ps4000aChangePowerSource() to complete the two-stage power-up sequence for a USB 2.0 or USB 3.0 port. Returns PICO_POWER_SUPPLY_CONNECTED if a power supply is connected.

PicoScope 4444 only: This function opens the device with the lowest available resolution. To open the device with a different resolution, use ps4000aOpenUnitWithResolution().

Applicability	All devices
Arguments	handle, on exit, an identifier for the device: −1 : if the unit fails to open, 0 : if no unit is found or > 0 : if successful (value is handle of the device opened) handle must be used in all subsequent calls to API functions to identify this scope device. * serial, on entry, an empty string, a serial number string or NULL; on exit, a null-terminated string containing the device's serial number. If serial is NULL, 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 PICO_NOT_RESPONDING PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_RESOURCE_ERROR PICO_MEMORY_FAIL PICO_HARDWARE_VERSION_NOT_SUPPORTED PICO_MEMORY_FAIL PICO_INTERNAL_ERROR PICO_POWER_SUPPLY_NOT_CONNECTED PICO_TIMEOUT PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_USED PICO_FPGA_FAIL

4.35 ps4000aOpenUnitAsync() – open a scope device without waiting

```
PICO_STATUS ps4000aOpenUnitAsync
(
  int16_t       * status,
  int8_t        * serial
)
```

This function opens a scope device without blocking the calling thread. You can find out when it has finished by periodically calling ps4000aOpenUnitProgress() until that function returns a non-zero value.

Applicability	All devices
Arguments	* status, on exit, indicates: 0 if there is already an open operation in progress 1 if the open operation is initiated * serial, on exit, a null-terminated string containing the device's serial number.
Returns	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_OPERATION_FAILED PICO_OS_NOT_SUPPORTED PICO_EEPROM_CORRUPT PICO_KERNEL_DRIVER_TOO_OLD PICO_FW_FAIL PICO_MAX_UNITS_OPENED PICO_NOT_FOUND PICO_NOT_RESPONDING PICO_RESOURCE_ERROR PICO_MEMORY_FAIL PICO_HARDWARE_VERSION_NOT_SUPPORTED PICO_MEMORY_FAIL PICO_INTERNAL_ERROR PICO_POWER_SUPPLY_NOT_CONNECTED PICO_TIMEOUT PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_USED PICO_FPGA_FAIL

4.36 ps4000aOpenUnitAsyncWithResolution() – open a flexible-resolution scope

```
PICO_STATUS ps4000aOpenUnitAsyncWithResolution
(
  int16_t                       * status,
  int8_t                        * serial,
  PS4000A_DEVICE_RESOLUTION     resolution
)
```

This function is similar to ps4000aOpenUnitAsync() but also sets the ADC resolution for scope devices that have flexible resolution.

Applicability	All devices
Arguments	`* status`, `* serial`, see ps4000aOpenUnitAsync(). `resolution`, see ps4000aOpenUnitWithResolution(). If the device has fixed ADC resolution, this argument is ignored.
Returns	PICO_OK PICO_OPEN_OPERATION_IN_PROGRESS PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_OPERATION_FAILED PICO_OS_NOT_SUPPORTED PICO_EEPROM_CORRUPT PICO_KERNEL_DRIVER_TOO_OLD PICO_FW_FAIL PICO_MAX_UNITS_OPENED PICO_NOT_FOUND PICO_NOT_RESPONDING PICO_RESOURCE_ERROR PICO_MEMORY_FAIL PICO_HARDWARE_VERSION_NOT_SUPPORTED PICO_MEMORY_FAIL PICO_INTERNAL_ERROR PICO_POWER_SUPPLY_NOT_CONNECTED PICO_TIMEOUT PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_USED PICO_FPGA_FAIL

4.37 ps4000aOpenUnitProgress() – check progress of OpenUnit() call

```
PICO_STATUS ps4000aOpenUnitProgress
(
int16_t         * handle,
int16_t         * progressPercent,
int16_t         * complete
)
```

This function checks on the progress of ps4000aOpenUnitAsync(). For status codes related to USB 2.0 powering, see ps4000aOpenUnit().

PicoScope 4444: returns PICO_POWER_SUPPLY_NOT_CONNECTED on completion if no power supply is connected; returns PICO_OK if a power supply is connected.

PicoScope 4824: returns PICO_USB3_0_DEVICE_NON_USB3_0_PORT if connected to a USB 2.0 port, or to any type of port through a USB 2.0 cable. Returns PICO_OK if connected to a USB 3.0 port.

Applicability	Use after ps4000aOpenUnitAsync()
Arguments	* handle, on exit, the device identifier. –1 if the unit fails to open, 0 if no unit is found or a non-zero handle to the device. **This handle is valid only if the function returns PICO_OK**. * progressPercent, on exit, the percentage progress. 100% implies that the open operation is complete. * complete, on exit, set to 1 when the open operation has finished
Returns	PICO_OK PICO_NULL_PARAMETER PICO_OPERATION_FAILED PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_OPEN_OPERATION_IN_PROGRESS PICO_OS_NOT_SUPPORTED PICO_EEPROM_CORRUPT PICO_KERNEL_DRIVER_TOO_OLD PICO_FW_FAIL PICO_MAX_UNITS_OPENED PICO_NOT_FOUND PICO_NOT_RESPONDING PICO_RESOURCE_ERROR PICO_MEMORY_FAIL PICO_HARDWARE_VERSION_NOT_SUPPORTED PICO_MEMORY_FAIL PICO_INTERNAL_ERROR PICO_POWER_SUPPLY_NOT_CONNECTED PICO_TIMEOUT PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_USED PICO_FPGA_FAIL

4.38 ps4000aOpenUnitWithResolution() – open a flexible-resolution scope

```
PICO_STATUS ps4000aOpenUnitWithResolution
(
  int16_t                       * handle,
  int8_t                        * serial,
  PS4000A_DEVICE_RESOLUTION     resolution
)
```

This function is similar to ps4000aOpenUnit() but additionally sets the hardware ADC resolution of a flexible-resolution device.

Applicability	All devices
Arguments	handle, see ps4000aOpenUnit() * serial, see ps4000aOpenUnit() resolution, an enumerated value of type PS4000A_DEVICE_RESOLUTION indicating the number of bits of ADC resolution required from the scope device. If the device has fixed ADC resolution, this argument is ignored.
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 PICO_NOT_RESPONDING PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_RESOURCE_ERROR PICO_MEMORY_FAIL PICO_HARDWARE_VERSION_NOT_SUPPORTED PICO_MEMORY_FAIL PICO_INTERNAL_ERROR PICO_POWER_SUPPLY_NOT_CONNECTED PICO_TIMEOUT PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_USED PICO_FPGA_FAIL

4.39 ps4000aPingUnit() – check that unit is responding

```
PICO_STATUS ps4000aPingUnit
(
  int16_t       handle
)
```

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

Applicability	All modes
Arguments	`handle`, identifier for the scope device.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_DRIVER_FUNCTION`
	`PICO_BUSY`
	`PICO_NOT_RESPONDING`
	`PICO_INTERNAL_ERROR`
	`PICO_TIMEOUT`
	`PICO_RESOURCE_ERROR`
	`PICO_DEVICE_NOT_FUNCTIONING`
	`PICO_USB3_0_DEVICE_NON_USB3_0_PORT`
	`PICO_POWER_SUPPLY_UNDERVOLTAGE`
	`PICO_POWER_SUPPLY_CONNECTED`
	`PICO_POWER_SUPPLY_NOT_CONNECTED`

4.40 ps4000aQueryOutputEdgeDetect() – query special trigger mode

```
PICO_STATUS ps4000aQueryOutputEdgeDetect
(
  int16_t       handle,
  int16_t       * state
)
```

This function obtains the state of the edge-detect flag, which is described in ps4000aSetOutputEdgeDetect().

Applicability	Level and window trigger types
Arguments	handle, identifier for the scope device. state, on exit, the value of the edge-detect flag: 0 : do not wait for a signal transition <> 0 : wait for a signal transition (default)
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_NULL_PARAMETER PICO_NOT_SUPPORTED_BY_THIS_DEVICE

4.41 ps4000aRunBlock() – start block mode

```
PICO_STATUS ps4000aRunBlock
(
  int16_t                  handle,
  int32_t                  noOfPreTriggerSamples,
  int32_t                  noOfPostTriggerSamples,
  uint32_t                 timebase,
  int32_t                  * timeIndisposedMs,
  uint32_t                 segmentIndex,
  ps4000aBlockReady        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 memory depth of the segment referred to by `segmentIndex`.

Applicability	Block mode and rapid block mode
Arguments	`handle`, identifier for the scope device. `noOfPreTriggerSamples`, the number of samples to return before the trigger event. If no trigger has been set, then this argument is added to `noOfPostTriggerSamples` to give the maximum number of data points (samples) to collect. `noOfPostTriggerSamples`, the number of samples to return after the trigger event. If no trigger event has been set, then this argument is added to `noOfPreTriggerSamples` to give the maximum number of data points to collect. If a trigger condition has been set, this specifies the number of data points to collect after a trigger has fired, and the number of data points to be collected is: `noOfPreTriggerSamples + noOfPostTriggerSamples` `timebase`, a number in the range 0 to $2^{32}-1$. See the guide to calculating timebase values. In ETS mode this argument is ignored and the driver chooses the timebase automatically. `* 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 `ps4000aBlockReady()` callback that the driver will call when the data has been collected. To use the `ps4000aIsReady()` polling method instead of a callback function, set this pointer to NULL. `* pParameter`, a void pointer that is passed to the `ps4000aBlockReady()` callback function. The callback can use the pointer to return arbitrary data to your 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_NOT_USED_IN_THIS_CAPTURE_MODE PICO_TRIGGER_WITHIN_PRE_NOT_ALLOWED_WITH_DELAY PICO_INVALID_NUMBER_CHANNELS_FOR_RESOLUTION PICO_NOT_ENOUGH_SEGMENTS PICO_NO_TRIGGER_ENABLED_FOR_TRIGGER_IN_PRE_TRIG PICO_MEMORY_FAIL PICO_INTERNAL_ERROR PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_POWER_SUPPLY_UNDERVOLTAGE PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_WARNING_PROBE_CHANNEL_OUT_OF_SYNC

4.42 ps4000aRunStreaming() – start streaming mode

```
PICO_STATUS ps4000aRunStreaming
(
  int16_t                  handle,
  uint32_t                 * sampleInterval,
  PS4000A_TIME_UNITS       sampleIntervalTimeUnits,
  uint32_t                 maxPreTriggerSamples,
  uint32_t                 maxPostTriggerSamples,
  int16_t                  autoStop,
  uint32_t                 downSampleRatio,
  PS4000A_RATIO_MODE       downSampleRatioMode,
  uint32_t                 overviewBufferSize
)
```

This function tells the oscilloscope to start collecting data in streaming mode. When data has been collected from the device it is downsampled and the values returned to the application. Call `ps4000aGetStreamingLatestValues()` to retrieve the data. See Using streaming mode for a step-by-step guide to this process.

This function always starts collecting data immediately, regardless of the trigger settings. Whether a trigger is set or not, the total number of samples stored in the driver is always `maxPreTriggerSamples + maxPostTriggerSamples`. If `autoStop` is false, the scope will collect data continuously, using the buffer as a first-in first-out (FIFO) memory.

Applicability	Streaming mode only
Arguments	`handle`, identifier for the scope device. `* sampleInterval`, on entry, the requested time interval between data points on entry; on exit, the actual time interval assigned. `sampleIntervalTimeUnits`, the unit of time that the `sampleInterval` is set to. See `ps4000aGetTriggerTimeOffset()` for values. `maxPreTriggerSamples`, the maximum number of raw samples before a trigger event for each enabled channel. `maxPostTriggerSamples`, the maximum number of raw samples after a trigger event for each enabled channel. `autoStop`, a flag to specify if the streaming should stop when all of `maxPreTriggerSamples + maxPostTriggerSamples` have been taken. `downSampleRatio`, the number of raw values to each downsampled value. `downSampleRatioMode`, the type of data reduction to use. `overviewBufferSize`, the size of the overview buffers (the buffers passed by the application to the driver). The size must be less than or equal to the `bufferLth` value passed to `ps4000aSetDataBuffer()`.

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_USB3_0_DEVICE_NON_USB3_0_PORT PICO_POWER_SUPPLY_UNDERVOLTAGE PICO_POWER_SUPPLY_CONNECTED PICO_POWER_SUPPLY_NOT_CONNECTED PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_USED_IN_THIS_CAPTURE_MODE PICO_INVALID_NUMBER_CHANNELS_FOR_RESOLUTION PICO_INTERNAL_ERROR PICO_MEMORY PICO_WARNING_PROBE_CHANNEL_OUT_OF_SYNC

4.43 ps4000aSetBandwidthFilter() – enable the bandwidth limiter

```
PICO_STATUS ps4000aSetBandwidthFilter
(
  int16_t                         handle,
  PS4000A_CHANNEL                 channel,
  PS4000A_BANDWIDTH_LIMITER       bandwidth
)
```

This function sets up the bandwidth limiter filter, if one is available on the selected device.

Applicability	PicoScope 4444 only
Arguments	`handle`, identifier for the scope device. `channel`, an enumerated type in the following range: `PS4000A_CHANNEL_A` … `PS4000A_CHANNEL_D` `bandwidth`, the required cutoff frequency of the filter. See `ps4000aApi.h` for allowable values.

Returns	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_NOT_USED` (if the device does not have a bandwidth limiter)
	`PICO_BUSY`
	`PICO_ARGUMENT_OUT_OF_RANGE`
	`PICO_INVALID_BANDWIDTH`

4.44 ps4000aSetCalibrationPins() – set up the CAL output pins

```
PICO_STATUS ps4000aSetCalibrationPins
(
  int16_t                   handle,
  PS4000A_PIN_STATES        pinStates,
  PS4000A_WAVE_TYPE         waveType,
  double                    frequency,
  uint32_t                  amplitude,
  uint32_t                  offset
)
```

This function sets up the CAL pins on the back of the PicoScope 4444 differential oscilloscope. These pins can generate test signals for use when compensating scope probes.

Applicability	PicoScope 4444 only

Arguments	`handle`, identifier for the scope device.
	
	`pinStates`, the desired state of the CAL pins:
	
	`PS4000A_CAL_PINS_OFF (0)` 0 volts on both pins
	`PS4000A_GND_SIGNAL (1)` 0 volts on **CAL –** pin, test signal on **CAL +** pin
	`PS4000A_SIGNAL_SIGNAL (2)` same test signal on both pins
	
	`waveType`, as defined in `ps4000aApi.h`. Only the following types are allowed:
	`PS4000A_SINE`
	`PS4000A_SQUARE`
	`PS4000A_DC_VOLTAGE`
	
	`frequency`, the signal repetition frequency in hertz. Range [100, 10 000] for `PS4000A_SQUARE`, [100, 100 000] for `PS4000A_SINE`. Value ignored for `PS4000A_DC_VOLTAGE`.
	
	`amplitude`, the signal amplitude in microvolts. Range [0, 8 000 000]. Value ignored for `PS4000A_DC_VOLTAGE`.
	
	`offset`, the signal offset in microvolts. Range [–4 000 000, +4 000 000]. If `offset` is zero, the signal range is [0 V, `amplitude`]. If the total of `offset ± amplitude` exceeds the range [–4 000 000, +4 000 000], the output will be clipped.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_DRIVER_FUNCTION`
	`PICO_NOT_SUPPORTED_BY_THIS_DEVICE`
	`PICO_CAL_PINS_WAVETYPE`
	`PICO_TIMEOUT`
	`PICO_RESOURCE_ERROR`
	`PICO_DEVICE_NOT_FUNCTIONING`
	`PICO_NOT_RESPONDING`

4.45 ps4000aSetChannel() – set up input channels

```
PICO_STATUS ps4000aSetChannel
(
  int16_t                         handle,
  PS4000A_CHANNEL                 channel,
  int16_t                         enabled,
  PS4000A_COUPLING                type,
  PICO_CONNECT_PROBE_RANGE        range,
  float                           analogOffset
)
```

This function sets up the characteristics of the specified input channel.

Applicability	All modes
Arguments	`handle`, identifier for the scope device. `channel`, the channel to be configured. The allowable values are: `PS4000A_CHANNEL_A` … `PS4000A_CHANNEL_D` (PicoScope 4444) `PS4000A_CHANNEL_A` … `PS4000A_CHANNEL_H` (PicoScope 4824) `enabled`, specifies if the channel is active (`TRUE`) or inactive (`FALSE`). `type`, specifies the coupling mode: DC (`TRUE`) or AC (`FALSE`). `range`, specifies the measuring range. This is defined differently depending on the oscilloscope. PicoScope 4444: the measuring ranges are defined in `PicoConnectProbes.h`. Refer to the `PICO_CONNECT_PROBE_RANGE` enumeration (`ps4000aProbeInteractions()`) for the list, which is specific to each probe. PicoScope 4824: Measuring ranges 0 to 13, defined `ps4000aApi.h`, are shown in the table below. `analogOffset`, an offset, in volts, to be added to the input signal before it reaches the input amplifier and digitizer. See the device data sheet for the allowable range.

Returns	`PICO_OK`
	`PICO_USER_CALLBACK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_CHANNEL`
	`PICO_INVALID_VOLTAGE_RANGE`
	`PICO_DRIVER_FUNCTION`
	`PICO_INVALID_COUPLING`
	`PICO_INVALID_ANALOGUE_OFFSET`
	`PICO_WARNING_PROBE_CHANNEL_OUT_OF_SYNC`
	Indicates that the channel configuration is not applicable to the PicoConnect probe in use. Check the most recent probe notification (received via callback) and apply a range appropriate to your probe.
	
	`PICO_PROBE_NOT_POWERED_WITH_DC_POWER_SUPPLY`
	`PICO_PROBE_POWER_DC_POWER_SUPPLY_REQUIRED`

range		Voltage range
0	PICO_X1_PROBE_10MV	±10 mV
1	PICO_X1_PROBE_20MV	±20 mV
2	PICO_X1_PROBE_50MV	±50 mV
3	PICO_X1_PROBE_100MV	±100 mV
4	PICO_X1_PROBE_200MV	±200 mV
5	PICO_X1_PROBE_500MV	±500 mV
6	PICO_X1_PROBE_1V	±1 V
7	PICO_X1_PROBE_2V	±2 V
8	PICO_X1_PROBE_5V	±5 V
9	PICO_X1_PROBE_10V	±10 V
10	PICO_X1_PROBE_20V	±20 V
11	PICO_X1_PROBE_50V	±50 V
12	PICO_X1_PROBE_100V	±100 V
13	PICO_X1_PROBE_200V	±200 V

4.46 ps4000aSetDataBuffer() – register data buffer with driver

```
PICO_STATUS ps4000aSetDataBuffer
(
  int16_t                   handle,
  PS4000A_CHANNEL           channel,
  int16_t                   * buffer,
  int32_t                   bufferLth,
  uint32_t                  segmentIndex,
  PS4000A_RATIO_MODE        mode
)
```

This function registers your data buffer, for non-aggregated data, with the ps4000a driver. You need to allocate the buffer before calling this function.

Applicability	All sampling modes. Non-aggregated data only. For aggregated data, use ps4000aSetDataBuffers().
Arguments	handle, identifier for the scope device. channel, the channel for which you want to set the buffers. Use one of these values: PS4000A_CHANNEL_A … PS4000A_CHANNEL_D (PicoScope 4444) PS4000A_CHANNEL_A … PS4000A_CHANNEL_H (PicoScope 4824) * buffer, a buffer to receive the data values. Each value is a 16-bit ADC count scaled according to the selected voltage range. bufferLth, the size of the buffer array. segmentIndex, the number of the memory segment to be retrieved. mode, the type of data reduction to use. See Downsampling for options.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_DRIVER_FUNCTION PICO_RATIO_MODE_NOT_SUPPORTED PICO_INVALID_PARAMETER

4.47 ps4000aSetDataBuffers() – register min/max data buffers with driver

```
PICO_STATUS ps4000aSetDataBuffers
(
  int16_t                   handle,
  PS4000A_CHANNEL           channel,
  int16_t                   * bufferMax,
  int16_t                   * bufferMin,
  int32_t                   bufferLth,
  uint32_t                  segmentIndex,
  PS4000A_RATIO_MODE        mode
)
```

This function registers your data buffers, for receiving aggregated data, with the ps4000a driver. You need to allocate memory for the buffers before calling this function.

Applicability	All sampling modes. All downsampling modes. For non-aggregated data, the simpler ps4000aSetDataBuffer() can be used instead.
Arguments	handle, identifier for the scope device. channel, the channel for which you want to set the buffers, in the following range: PS4000A_CHANNEL_A … PS4000A_CHANNEL_D (PicoScope 4444) PS4000A_CHANNEL_A … PS4000A_CHANNEL_H (PicoScope 4824) * bufferMax, a user-allocated buffer to receive the maximum data values in aggregation mode, or the non-aggregated values otherwise. Each value is a 16-bit ADC count scaled according to the selected voltage range. * bufferMin, a user-allocated buffer to receive the minimum data values in aggregation mode. Not normally used in other modes, but you can direct the driver to write non-aggregated values to this buffer by setting bufferMax to NULL. To enable aggregation, the downsampling ratio and mode must be set appropriately when calling one of the ps4000aGetValues...() functions. bufferLth, specifies the size of the bufferMax and bufferMin arrays. segmentIndex, the number of the memory segment to be retrieved. mode, the type of downsampling to use. See Downsampling.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_INVALID_CHANNEL PICO_DRIVER_FUNCTION PICO_RATIO_MODE_NOT_SUPPORTED PICO_INVALID_PARAMETER

4.48 ps4000aSetDeviceResolution() – set up a flexible-resolution scope

```
PICO_STATUS ps4000aSetDeviceResolution
(
  int16_t                       handle,
  PS4000A_DEVICE_RESOLUTION     resolution
  )
```

This function sets the ADC resolution. Increasing the resolution affects other properties such as the maximum sampling rate and analog bandwidth. When the resolution is changed, any data captured that has not been saved will be lost. If ps4000aSetChannel() is not called, ps4000aRunBlock() and ps4000aRunStreaming() may fail.

Applicability	PicoScope 4444 only
Arguments	handle, identifier for the scope device. resolution, determines the resolution of the device when opened. This is chosen from the available values of PS4000A_DEVICE_RESOLUTION. If resolution is out of range the device will return PICO_INVALID_DEVICE_RESOLUTION.
Returns	PICO_OK PICO_INVALID_DEVICE_RESOLUTION PICO_OS_NOT_SUPPORTED PICO_OPEN_OPERATION_IN_PROGRESS PICO_EEPROM_CORRUPT PICO_KERNEL_DRIVER_TOO_OLD PICO_FPGA_FAIL PICO_MEMORY_CLOCK_FREQUENCY PICO_FW_FAIL PICO_MAX_UNITS_OPENED PICO_NOT_FOUND (if the specified unit was not found) PICO_NOT_RESPONDING PICO_MEMORY_FAIL PICO_ANALOG_BOARD PICO_CONFIG_FAIL_AWG PICO_INITIALISE_FPGA PICO_POWER_SUPPLY_NOT_CONNECTED PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_POWER_SUPPLY_UNDERVOLTAGE PICO_POWER_SUPPLY_CONNECTED PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING

4.49 ps4000aSetEts() – set up equivalent-time sampling (ETS)

```
PICO_STATUS ps4000aSetEts
(
  int16_t             handle,
  PS4000A_ETS_MODE    mode,
  int16_t             etsCycles,
  int16_t             etsInterleave,
  int32_t             * sampleTimePicoseconds
)
```

This function is reserved for future use.

Applicability	Not implemented
Arguments	`handle`, identifier for the scope device. `mode`, `ets_cycles`, `ets_interleave`, `* sampleTimePicoseconds`, not used.

Returns	`PICO_ETS_NOT_SUPPORTED`
	`PICO_DRIVER_FUNCTION`
	`PICO_INVALID_HANDLE`

4.50 ps4000aSetEtsTimeBuffer() – set up 64-bit buffer for ETS time data

```
PICO_STATUS ps4000aSetEtsTimeBuffer
(
  int16_t       handle,
  int64_t       * buffer,
  int32_t       bufferLth
)
```

Reserved for future use.

Applicability	Not implemented
Arguments	`handle`, identifier for the scope device. `* buffer`, `bufferLth`, not used.

Returns	`PICO_ETS_NOT_SUPPORTED`
	`PICO_DRIVER_FUNCTION`
	`PICO_INVALID_HANDLE`

4.51 ps4000aSetEtsTimeBuffers() – set up 32-bit buffers for ETS time data

```
PICO_STATUS ps4000aSetEtsTimeBuffers
(
  int16_t       handle,
  uint32_t      * timeUpper,
  uint32_t      * timeLower,
  int32_t       bufferLth
)
```

This function is reserved for future use.

Applicability	Not implemented
Arguments	`handle`, identifier for the scope device. `* timeUpper`, `* timeLower`, `bufferLth`, not used.

Returns	`PICO_ETS_NOT_SUPPORTED`
	`PICO_DRIVER_FUNCTION`
	`PICO_INVALID_HANDLE`

4.52 ps4000aSetNoOfCaptures() – set number of rapid block captures

```
PICO_STATUS ps4000aSetNoOfCaptures
(
  int16_t       handle,
  uint32_t      nCaptures
)
```

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

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

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_INVALID_PARAMETER`
	`PICO_DRIVER_FUNCTION`
	`PICO_MEMORY_FAIL`
	`PICO_INTERNAL_ERROR`

4.53 ps4000aSetOutputEdgeDetect() – set special trigger mode

```
PICO_STATUS ps4000aSetOutputEdgeDetect
(
  int16_t       handle,
  int16_t       state
)
```

This function tells the device whether or not to wait for an edge on the trigger input when one of the 'level' or 'window' trigger types is in use. By default the device waits for an edge on the trigger input before firing the trigger. If you switch off edge detect mode, the device will trigger continually for as long as the trigger input remains in the specified state.

You can query the state of this flag by calling ps4000aQueryOutputEdgeDetect().

Applicability	Level and window trigger types
Arguments	handle, identifier for the scope device. state, a flag that specifies the trigger behavior: 0 : do not wait for a signal transition <> 0 : wait for a signal transition (default)
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION

4.54 ps4000aSetProbeInteractionCallback() – register callback function for PicoConnect events

```
PICO_STATUS ps4000aSetProbeInteractionCallback
(
  int16_t                         handle,
  ps4000aProbeInteractions        callback
)
```

This function registers your ps4000aProbeInteractions() callback function with the ps4000a driver. The driver will then call your function whenever a PicoConnect™ probe is plugged into, or unplugged from, a PicoScope 4444 device, or if the power consumption of the connected probes exceeds the power available. See Handling PicoConnect probe interactions for more information on this process.

You should call this function as soon as the device has been successfully opened and before any call to ps4000aSetChannel().

Applicability	PicoScope 4444 only
Arguments	handle, identifier for the scope device. callback, a pointer to your callback function.
Returns	PICO_OK

4.55 ps4000aSetPulseWidthQualifierConditions() – set up pulse width triggering

```
PICO_STATUS ps4000aSetPulseWidthQualifierConditions
(
  int16_t handle,
  PS4000A_CONDITION               * conditions,
  int16_t nConditions,
  PS4000A_CONDITIONS_INFO         info
)
```

This function sets up the conditions for pulse width qualification, which is used with either threshold triggering, level triggering or window triggering to produce time-qualified triggers. Each call to this function creates a pulse width qualifier equal to the logical AND of the elements of the `conditions` array. Calling this function multiple times creates the logical OR of multiple AND operations. This AND-OR logic allows you to create any possible Boolean function of the scope's inputs.

To cease ORing pulse width qualifier conditions and start again with a new set, call with `info = PS4000A_CLEAR`.

Other settings of the pulse width qualifier are configured by calling `ps4000aSetPulseWidthQualifierProperties()`.

Note: The oscilloscope contains a single pulse-width counter. It is possible to include multiple channels in a pulse-width qualifier but the same pulse-width counter will apply to all of them. The counter starts when your selected trigger condition occurs, and the scope then triggers if the trigger condition ends after a time that satisfies the pulse-width condition.

Applicability	All modes

Arguments	`handle`, identifier for the scope device.
	
	`* conditions`: see `ps4000aSetTriggerChannelConditions()`
	`nConditions`: see `ps4000aSetTriggerChannelConditions()`
	`info`: see `ps4000aSetTriggerChannelConditions()`

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_CONDITIONS`
	`PICO_PULSE_WIDTH_QUALIFIER`
	`PICO_DRIVER_FUNCTION`
	`PICO_INVALID_CONDITION_INFO`
	`PICO_INVALID_PARAMETER`
	`PICO_DUPLICATE_CONDITION_SOURCE`
	`PICO_MEMORY_FAIL`
	`PICO_INTERNAL_ERROR`
	`PICO_TOO_MANY_CHANNELS_IN_USE`

4.56 ps4000aSetPulseWidthQualifierProperties() – set up pulse width triggering

```
PICO_STATUS ps4000aSetPulseWidthQualifierProperties
(
  int16_t                         handle,
  PS4000A_THRESHOLD_DIRECTION     direction,
  uint32_t                        lower,
  uint32_t                        upper,
  PS4000A_PULSE_WIDTH_TYPE        type
)
```

This function configures the general properties of the pulse width qualifier.

Applicability	All modes
Arguments	`handle`, identifier for the scope device. `direction`, the direction of the signal required for the trigger to fire. See `PS4000A_DIRECTION` for allowable values. This is also the direction that resets and starts the counter. `lower`, the lower limit of the pulse width counter, in samples. `upper`, the upper limit of the pulse width counter, in samples. This parameter is used only when the type is set to `PW_TYPE_IN_RANGE` or `PW_TYPE_OUT_OF_RANGE`. `type`, the pulse width type, one of these constants: `PW_TYPE_NONE` (do not use the pulse width qualifier) `PW_TYPE_LESS_THAN` (pulse width less than `lower`) `PW_TYPE_GREATER_THAN` (pulse width greater than `lower`) `PW_TYPE_IN_RANGE` (pulse width between `lower` and `upper`) `PW_TYPE_OUT_OF_RANGE` (pulse width not between `lower` and `upper`)

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_CONDITIONS`
	`PICO_PULSE_WIDTH_QUALIFIER`
	`PICO_DRIVER_FUNCTION`
	`PICO_MEMORY_FAIL`
	`PICO_INTERNAL_ERROR`

4.57 ps4000aSetSigGenArbitrary() – set up arbitrary waveform generator

```
PICO_STATUS ps4000aSetSigGenArbitrary
(
  int16_t                       handle,
  int32_t                       offsetVoltage,          // see note 1
  uint32_t                      pkToPk,                 // see note 1
  uint32_t                      startDeltaPhase,
  uint32_t                      stopDeltaPhase,
  uint32_t                      deltaPhaseIncrement,
  uint32_t                      dwellCount,
  int16_t                       * arbitraryWaveform,    // see note 1
  int32_t                       arbitraryWaveformSize,  // see note 1
  PS4000A_SWEEP_TYPE            sweepType,
  PS4000A_EXTRA_OPERATIONS      operation,              // see note 1
  PS4000A_INDEX_MODE            indexMode,
  uint32_t                      shots,
  uint32_t                      sweeps,
  PS4000A_SIGGEN_TRIG_TYPE      triggerType,
  PS4000A_SIGGEN_TRIG_SOURCE    triggerSource,
  int16_t                       extInThreshold
)
```

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

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

Note 1: in general, this function can be called with new arguments while waiting for a trigger; the exceptions are the arguments noted above, which must be unchanged on subsequent calls, otherwise the function will return `PICO_BUSY`.

Note 2: call this function before starting data acquisition, even if the signal generator will be triggered during data collection.

Note 3: for more information about using this function, read the article Triggering a PicoScope signal generator using the PicoScope API functions.

Applicability	All modes. PicoScope 4824 only.
Arguments	
`handle`, identifier for the scope 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. Call `ps4000aSigGenFrequencyToPhase()` to calculate this.	

stopDeltaPhase, the final value added to the phase counter before the generator restarts or reverses the sweep. If required, call ps4000aSigGenFrequencyToPhase() to calculate it. When frequency sweeping is not required, set equal to startDeltaPhase.

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

dwellCount, the time, in multiples of *dacPeriod,* between successive additions of deltaPhaseIncrement to the delta phase counter. This determines the rate at which the generator sweeps the output frequency. Minimum allowable values are as follows:

PicoScope 4824: MIN_DWELL_COUNT

* arbitraryWaveform, a buffer that holds the waveform pattern as a set of samples equally spaced in time. Call ps4000aSigGenArbitraryMinMaxValues() to obtain the range of allowable values, or use these constants:

PicoScope 4824: [–32768, 32767]

arbitraryWaveformSize, the size of the arbitrary waveform buffer, in samples. Call ps4000aSigGenArbitraryMinMaxValues() to obtain the range of allowable values, or use these constants:

PicoScope 4824: PS4000A_MIN_SIG_GEN_BUFFER_SIZE (10)
PS4000A_MAX_SIG_GEN_BUFFER_SIZE (16384)

sweepType, determines whether the startDeltaPhase is swept up to the stopDeltaPhase, or down to it, or repeatedly up and down. Use one of the following values: UP, DOWN, UPDOWN, DOWNUP.

operation, configures the white noise/PRBS (pseudo-random binary sequence) generator:

PS4000A_ES_OFF:	White noise/PRBS output disabled. The waveform is defined by the other arguments.
PS4000A_WHITENOISE:	The signal generator produces white noise and ignores all settings except offsetVoltage and pkTopk.
PS4000A_PRBS:	The signal generator produces a PRBS.

indexMode, specifies how the signal will be formed from the arbitrary waveform data. SINGLE, DUAL and QUAD index modes are possible (see AWG index modes).

shots, the number of cycles of the waveform to be produced after a trigger event. If this is set to a non-zero value [1, MAX_SWEEPS_SHOTS], then sweeps must be set to zero.

sweeps, the number of times to sweep the frequency after a trigger event, according to sweepType. If this is set to a non-zero value [1, MAX_SWEEPS_SHOTS], then shots must be set to zero.

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

PS4000A_SIGGEN_RISING:	rising edge
PS4000A_SIGGEN_FALLING:	falling edge
PS4000A_SIGGEN_GATE_HIGH:	high level
PS4000A_SIGGEN_GATE_LOW:	low level

triggerSource, the source that will trigger the signal generator:

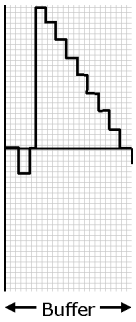
PS4000A_SIGGEN_NONE:	no trigger (free-running)
PS4000A_SIGGEN_SCOPE_TRIG:	the selected oscilloscope channel (see ps4000aSetSimpleTrigger())
PS4000A_SIGGEN_SOFT_TRIG:	a software trigger (see ps4000aSigGenSoftwareControl())

If a trigger source other than PS4000A_SIGGEN_NONE is specified, then either shots or sweeps, but not both, must be set to a non-zero value. extInThreshold, not used	
Returns	PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_NO_SIGNAL_GENERATOR PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_MEMORY_FAIL PICO_INTERNAL_ERROR PICO_SIG_GEN_PARAM PICO_NULL_PARAMETER PICO_SIGGEN_OFFSET_VOLTAGE PICO_SIGGEN_PK_TO_PK PICO_SIGGEN_OUTPUT_OVER_VOLTAGE PICO_SHOTS_SWEEPS_WARNING PICO_BUSY PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_RESPONDING

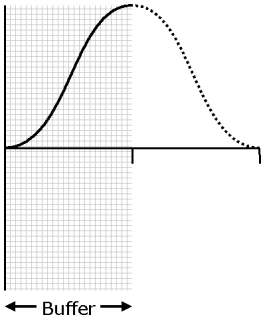
4.57.1 AWG index modes

The arbitrary waveform generator supports SINGLE, DUAL and QUAD index modes to make the best use of the waveform buffer.

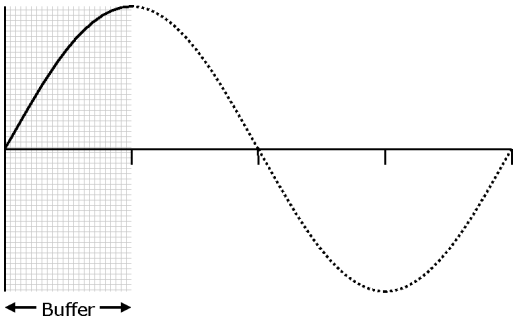
SINGLE **mode.** The generator outputs the raw contents of the buffer repeatedly. This mode is the only one that can generate asymmetrical waveforms. You can also use this mode for symmetrical waveforms, but the dual and quad modes make more efficient use of the buffer memory.



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 as in dual mode. 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.



4.57.2 Calculating deltaPhase

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

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

where:

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

You can call ps4000aSigGenFrequencyToPhase() to calculate *deltaPhase*.

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

Parameter	PicoScope 4824
dacFrequency	80 MHz
dacPeriod (= 1/dacFrequency)	12.5 ns
phaseAccumulatorSize	4 294 967 296 (2^{32})
awgBufferSize	16 384 (2^{14})

4.58 ps4000aSetSigGenBuiltIn() – set up function generator

```
PICO_STATUS ps4000aSetSigGenBuiltIn
(
  int16_t                       handle,
  int32_t                       offsetVoltage,     // see note 1
  uint32_t                      pkToPk,            // see note 1
  PS4000A_WAVE_TYPE             waveType,          // see note 1
  double                        startFrequency,
  double                        stopFrequency,
  double                        increment,
  double                        dwellTime,
  PS4000A_SWEEP_TYPE            sweepType,
  PS4000A_EXTRA_OPERATIONS      operation,         // see note 1
  uint32_t                      shots,
  uint32_t                      sweeps,
  PS4000A_SIGGEN_TRIG_TYPE      triggerType,
  PS4000A_SIGGEN_TRIG_SOURCE    triggerSource,
  int16_t                       extInThreshold
)
```

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

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

Note 2: call this function before starting data acquisition, even if the signal generator will be triggered during data collection.

Note 3: for more information about using this function, read the article Triggering a PicoScope signal generator using the PicoScope API functions.

Applicability	All modes. PicoScope 4824 only.
Arguments	`handle`, identifier for the scope device. `offsetVoltage`, the voltage offset, in microvolts, to be applied to the waveform. `pkToPk`, the peak-to-peak voltage, in microvolts, of the waveform signal.

	`waveType`, the type of waveform to be generated by the oscilloscope: `PS4000A_SINE` sine wave `PS4000A_SQUARE` square wave `PS4000A_TRIANGLE` triangle wave `PS4000A_RAMP_UP` rising sawtooth `PS4000A_RAMP_DOWN` falling sawtooth `PS4000A_SINC` sin(x)/x `PS4000A_GAUSSIAN` normal distribution `PS4000A_HALF_SINE` full-wave rectified sinusoid `PS4000A_DC_VOLTAGE` DC voltage `PS4000A_WHITE_NOISE` random values `startFrequency`, the frequency in hertz at which the signal generator should begin. Range: `MIN_SIG_GEN_FREQ` to `MAX_SIG_GEN_FREQ`. `stopFrequency`, the frequency in hertz at which the sweep should reverse direction or return to the start frequency. Range: `MIN_SIG_GEN_FREQ` to `MAX_SIG_GEN_FREQ`. `increment`, the amount in hertz by which the frequency rises or falls every `dwellTime` seconds in sweep mode. `dwellTime`, the time in seconds between frequency changes in sweep mode. `sweepType`, `operation`, `shots`, `sweeps`, `triggerType`, `triggerSource`, `extInThreshold`: see `ps4000aSetSigGenArbitrary()`

Returns	`PICO_INVALID_HANDLE`
	`PICO_DRIVER_FUNCTION`
	`PICO_NO_SIGNAL_GENERATOR`
	`PICO_USB3_0_DEVICE_NON_USB3_0_PORT`
	`PICO_MEMORY_FAIL`
	`PICO_INTERNAL_ERROR`
	`PICO_SIG_GEN_PARAM`
	`PICO_SIGGEN_OFFSET_VOLTAGE`
	`PICO_SIGGEN_PK_TO_PK`
	`PICO_SIGGEN_OUTPUT_OVER_VOLTAGE`
	`PICO_SHOTS_SWEEPS_WARNING`
	`PICO_BUSY`
	`PICO_TIMEOUT`
	`PICO_RESOURCE_ERROR`
	`PICO_DEVICE_NOT_FUNCTIONING`
	`PICO_NOT_RESPONDING`

4.59 ps4000aSetSigGenPropertiesArbitrary() – set up arbitrary waveform generator

```
PICO_STATUS ps4000aSetSigGenPropertiesArbitrary
(
  int16_t                         handle,
  uint32_t                        startDeltaPhase,
  uint32_t                        stopDeltaPhase,
  uint32_t                        deltaPhaseIncrement,
  uint32_t                        dwellCount,
  PS4000A_SWEEP_TYPE              sweepType,
  uint32_t                        shots,
  uint32_t                        sweeps,
  PS4000A_SIGGEN_TRIG_TYPE        triggerType,
  PS4000A_SIGGEN_TRIG_SOURCE      triggerSource,
  int16_t                         extInThreshold
)
```

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

Applicability	All modes. PicoScope 4824 only.
Arguments	See ps4000SetSigGenArbitrary()
Returns	PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_RESPONDING PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_SIGGEN_PK_TO_PK PICO_SIGGEN_OFFSET_VOLTAGE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING

4.60 ps4000aSetSigGenPropertiesBuiltIn() – set up function generator

```
PICO_STATUS ps4000aSetSigGenPropertiesBuiltIn
(
  int16_t                         handle,
  double                          startFrequency,
  double                          stopFrequency,
  double                          increment,
  double                          dwellTime,
  PS4000A_SWEEP_TYPE              sweepType,
  uint32_t                        shots,
  uint32_t                        sweeps,
  PS4000A_SIGGEN_TRIG_TYPE        triggerType,
  PS4000A_SIGGEN_TRIG_SOURCE      triggerSource,
  int16_t                         extInThreshold
)
```

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

Applicability	All modes. PicoScope 4824 only.
Arguments	See ps4000SetSigGenBuiltIn()
Returns	PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_RESPONDING PICO_USB3_0_DEVICE_NON_USB3_0_PORT PICO_SIGGEN_PK_TO_PK PICO_SIGGEN_OFFSET_VOLTAGE PICO_SIG_GEN_PARAM PICO_SHOTS_SWEEPS_WARNING

4.61 ps4000aSetSimpleTrigger() – set up level triggers only

```
PICO_STATUS ps4000aSetSimpleTrigger
(
  int16_t                         handle,
  int16_t                         enable,
  PS4000A_CHANNEL                 source,
  int16_t                         threshold,
  PS4000A_THRESHOLD_DIRECTION     direction,
  uint32_t                        delay,
  int16_t                         autoTrigger_ms
)
```

This function simplifies arming the trigger. It supports only the LEVEL trigger types and does not allow more than one channel to have a trigger applied to it. Any previous pulse width qualifier is canceled. The trigger threshold includes a small, fixed amount of hysteresis.

Applicability	All modes
Arguments	handle, identifier for the scope device. enabled, zero to disable the trigger, any non-zero value to set the trigger. source, the channel on which to trigger. See ps4000aSetChannel(). threshold, the ADC count at which the trigger will fire. direction, the direction in which the signal must move to cause a trigger. The following directions are supported: ABOVE, BELOW, RISING, FALLING and RISING_OR_FALLING. delay, the time, in sample periods, between the trigger occurring and the first sample being taken. autoTrigger_ms, the number of milliseconds the device will wait if no trigger occurs. If 0, the device will wait indefinitely.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION PICO_INVALID_TRIGGER_CHANNEL PICO_INVALID_CHANNEL PICO_INVALID_PARAMETER PICO_MEMORY_FAIL PICO_INTERNAL_ERROR

4.62 ps4000aSetTriggerChannelConditions() – specify which channels to trigger on

```
PICO_STATUS ps4000aSetTriggerChannelConditions
(
  int16_t                   handle,
  PS4000A_CONDITION         * conditions,
  int16_t                   nConditions,
  PS4000A_CONDITIONS_INFO   info
)
```

This function sets up trigger conditions on the scope's inputs. The trigger is set up by defining an array of one or more PS4000A_CONDITION structures that are then ANDed together. The function can be called multiple times, in which case the trigger logic is ORed with that defined by previous calls. This AND-OR logic allows you to create any possible Boolean function of up to four of the scope's inputs.

To cease ORing trigger channel conditions and start again with a new set, call with `info = PS4000A_CLEAR`.

You can also call ps4000aSetPulseWidthQualifierConditions() to add timing conditions to the trigger.

Applicability	All modes
Arguments	`handle`, identifier for the scope device. `* conditions`, an array of PS4000A_CONDITION 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 AND of all the elements. `nConditions`, the number of elements in the `conditions` array, or zero to switch off triggering. `info`, determines whether the function clears previous conditions: `PS4000A_CLEAR`, clears previous conditions `PS4000A_ADD`, adds the specified conditions (ORing them with previously set conditions, if any) You can combine both actions by passing `(PS4000A_CONDITIONS_INFO) (PS4000A_CLEAR | PS4000A_ADD)`.

Returns	`PICO_OK`
	`PICO_INVALID_HANDLE`
	`PICO_USER_CALLBACK`
	`PICO_CONDITIONS`
	`PICO_MEMORY_FAIL`
	`PICO_TOO_MANY_CHANNELS_IN_USE` (if you attempt to create a function of more than four inputs)
	`PICO_INVALID_CONDITION_INFO`
	`PICO_INVALID_PARAMETER`
	`PICO_DUPLICATE_CONDITION_SOURCE`
	`PICO_INTERNAL_ERROR`

4.62.1 PS4000A_CONDITION structure

A structure of this type is passed to ps4000aSetPulseWidthQualifierConditions() and ps4000SetTriggerChannelConditions() in the conditions argument to specify the trigger conditions, and is defined as follows: -

```
typedef struct tPS4000ACondition
{
  PS4000A_CHANNEL           source;
  PS4000A_TRIGGER_STATE     condition;
} PS4000A_CONDITION;
```

Elements	source, the input to the trigger or pulse width qualifier. See ps4000aSetChannel() for values. condition, the type of condition that should be applied to each channel. Use any these constants: CONDITION_DONT_CARE CONDITION_TRUE CONDITION_FALSE The channels that are set to CONDITION_TRUE or CONDITION_FALSE must all meet their conditions simultaneously to produce a trigger. Channels set to CONDITION_DONT_CARE are ignored.

4.63 ps4000aSetTriggerChannelDirections() – set up signal polarities for triggering

```
PICO_STATUS ps4000aSetTriggerChannelDirections
(
  int16_t                  handle,
  PS4000A_DIRECTION        * directions,
  int16_t                  nDirections
)
```

This function sets the direction of the trigger for the specified channels.

Applicability	All modes.
Arguments	`handle`, identifier for the scope device. `* directions`, on entry, an array of structures containing trigger directions. See PS4000A_DIRECTION for allowable values. `nDirections`, the length of the `directions` array.

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

4.63.1 PS4000A_DIRECTION structure

A structure of this type is passed to ps4000aSetTriggerChannelDirections() in the directions argument to specify the trigger direction for a specified source, and is defined as follows: -

```
typedef struct tPS4000ADirection
{
  PS4000A_CHANNEL                channel;
  PS4000A_THRESHOLD_DIRECTION    direction;
} PS4000A_DIRECTION;
```

Elements

channel, the channel being configured. See ps4000aSetChannel for allowable values.

direction, the trigger direction that should be applied to each channel. Use one of these constants:

Constant	Type	Direction
PS4000A_ABOVE	gated	above the upper threshold
PS4000A_ABOVE_LOWER	gated	above the lower threshold
PS4000A_BELOW	gated	below the upper threshold
PS4000A_BELOW_LOWER	gated	below the lower threshold
PS4000A_RISING	threshold	rising edge, using upper threshold
PS4000A_RISING_LOWER	threshold	rising edge, using lower threshold
PS4000A_FALLING	threshold	falling edge, using upper threshold
PS4000A_FALLING_LOWER	threshold	falling edge, using lower threshold
PS4000A_RISING_OR_FALLING	threshold	either edge
PS4000A_INSIDE	window-qualified	inside window
PS4000A_OUTSIDE	window-qualified	outside window
PS4000A_ENTER	window	entering the window
PS4000A_EXIT	window	leaving the window
PS4000A_ENTER_OR_EXIT	window	either entering or leaving the window
PS4000A_POSITIVE_RUNT	window-qualified	entering and leaving from below
PS4000A_NEGATIVE_RUNT	window-qualified	entering and leaving from above
PS4000A_NONE	none	none

4.64 ps4000aSetTriggerChannelProperties() – set up trigger thresholds

```
PICO_STATUS ps4000aSetTriggerChannelProperties
(
  int16_t                                handle,
  PS4000A_TRIGGER_CHANNEL_PROPERTIES     * channelProperties,
  int16_t                                nChannelProperties,
  int16_t                                auxOutputEnable,
  int32_t                                autoTriggerMilliseconds
)
```

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

Applicability	All modes
Arguments	`handle`, identifier for the scope device. `* channelProperties`, an array of PS4000A_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 number of elements in 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_INTERNAL_ERROR`

4.64.1 PS4000A_TRIGGER_CHANNEL_PROPERTIES structure

A structure of this type is passed to ps4000aSetTriggerChannelProperties in the channelProperties argument to specify the trigger mechanism, and is defined as follows:

```
typedef struct tPS4000ATriggerChannelProperties
{
  int16_t                 thresholdUpper;
  uint16_t                thresholdUpperHysteresis;
  int16_t                 thresholdLower;
  uint16_t                thresholdLowerHysteresis;
  PS4000A_CHANNEL         channel;
  PS4000A_THRESHOLD_MODE  thresholdMode;
} PS4000A_TRIGGER_CHANNEL_PROPERTIES
```

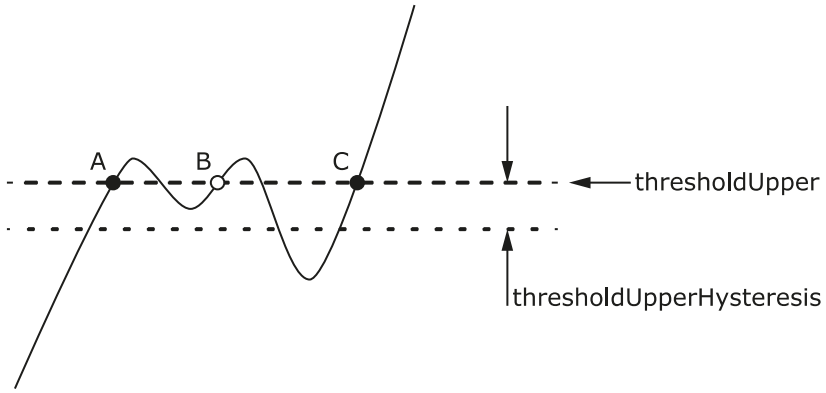
The structure is byte-aligned. In C++, for example, you should specify this using the #pragma pack() instruction.

Upper and lower thresholds
The digital triggering hardware in your PicoScope has two independent trigger thresholds called *upper* and *lower*. For some trigger types you can freely choose which threshold to use. The table in ps4000aSetTriggerChannelDirections() shows which thresholds are available for use with which trigger types. Dual thresholds are used for pulse-width triggering, when one threshold applies to the level trigger and the other to the pulse-width qualifier; and for window triggering, when the two thresholds define the upper and lower limits of the window.

Each threshold has its own trigger and hysteresis settings.

Hysteresis
Each trigger threshold (*upper* and *lower*) has an accompanying parameter called *hysteresis*. This defines an additional threshold, called the *hysteresis threshold*, at a small offset from the main threshold. The trigger fires when the signal crosses the hysteresis threshold and then the main threshold. It will not fire again until the signal has crossed the both the hysteresis threshold and main threshold again. The double-threshold mechanism prevents low-amplitude noise on the signal from causing unwanted trigger events.

For a rising-edge trigger the hysteresis threshold is below the main threshold. After one trigger event, the signal must fall below the hysteresis threshold and then rise above it before the trigger is enabled for the next event. Conversely, for a falling-edge trigger, the hysteresis threshold is always above the main threshold. After a trigger event, the signal must rise above the hysteresis threshold and then fall below it before the trigger is enabled for the next event.



Hysteresis — The trigger fires at **A** as the signal rises past both thresholds. It does not fire at **B** because the signal has not passed the hysteresis threshold. The trigger fires again at **C** after the signal has dipped below the hysteresis threshold and risen past both thresholds.

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. See `ps4000aSetChannel()` for possible values.
	
	`thresholdMode`, either a level or window trigger. Use one of these constants:
	`PS4000A_LEVEL`
	`PS4000A_WINDOW`

4.65 ps4000aSetTriggerDelay() – set up post-trigger delay

```
PICO_STATUS ps4000aSetTriggerDelay
(
  int16_t       handle,
  uint32_t      delay
)
```

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

Applicability	All modes (but `delay` is ignored in streaming mode)
Arguments	`handle`, identifier for the scope device. `delay`, the time between the trigger occurring and the first sample, in sample periods. For example, if `delay = 100`, the scope would wait 100 sample periods before sampling. Example: with the PicoScope 4824, at a timebase of 80 MS/s, or 12.5 ns per sample `(timebase = 0)` the total delay would be: 100 x 12.5 ns = 1.25 µs

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

4.66 ps4000aSigGenArbitraryMinMaxValues() – get AWG sample value limits

```
PICO_STATUS ps4000aSigGenArbitraryMinMaxValues
(
  int16_t       handle,
  int16_t       * minArbitraryWaveformValue,
  int16_t       * maxArbitraryWaveformValue,
  uint32_t      * minArbitraryWaveformSize,
  uint32_t      * maxArbitraryWaveformSize
)
```

This function returns the range of possible sample values and waveform buffer sizes that can be supplied to ps4000aSetSigGenArbitrary() for setting up the arbitrary waveform generator (AWG). These values may vary between models.

Applicability	PicoScope 4824 only.
Arguments	handle, identifier for the scope device. minArbitraryWaveformValue, on exit, the lowest sample value allowed in the arbitraryWaveform buffer supplied to ps4000aSetSigGenArbitrary(). maxArbitraryWaveformValue, on exit, the highest sample value allowed in the arbitraryWaveform buffer supplied to ps4000aSetSigGenArbitrary(). minArbitraryWaveformSize, on exit, the minimum value allowed for the arbitraryWaveformSize argument supplied to ps4000aSetSigGenArbitrary(). maxArbitraryWaveformSize, on exit, the maximum value allowed for the arbitraryWaveformSize argument supplied to ps4000aSetSigGenArbitrary().
Returns	PICO_OK PICO_NOT_SUPPORTED_BY_THIS_DEVICE, if the device does not have an arbitrary waveform generator. PICO_NULL_PARAMETER, if all the parameter pointers are NULL. PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION

4.67 ps4000aSigGenFrequencyToPhase() – get phase increment for signal generator

```
PICO_STATUS ps4000aSigGenFrequencyToPhase
(
  int16_t                  handle,
  double                   frequency,
  PS4000A_INDEX_MODE       indexMode,
  uint32_t                 bufferLength,
  uint32_t                 * phase
)
```

This function converts a frequency to a phase count for use with the arbitrary waveform generator (AWG). The value returned depends on the length of the buffer, the index mode passed and the device model. The phase count can then be sent to the driver through ps4000aSetSigGenArbitrary() or ps4000aSetSigGenPropertiesArbitrary().

Applicability	PicoScope 4824 only.
Arguments	handle, identifier for the scope device. frequency, the required AWG output frequency. indexMode, see AWG index modes. bufferLength, the number of samples in the AWG buffer. phase, on exit, the deltaPhase argument to be sent to the AWG setup function
Returns	PICO_OK PICO_NOT_SUPPORTED_BY_THIS_DEVICE, if the device does not have an AWG. PICO_SIGGEN_FREQUENCY_OUT_OF_RANGE, if the frequency is out of range. PICO_NULL_PARAMETER, if phase is a NULL pointer. PICO_SIG_GEN_PARAM, if indexMode or bufferLength is out of range. PICO_INVALID_HANDLE PICO_DRIVER_FUNCTION

4.68 ps4000aSigGenSoftwareControl() – trigger the signal generator

```
PICO_STATUS ps4000aSigGenSoftwareControl
(
  int16_t       handle,
  int16_t       state
)
```

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

Applicability	Use with ps4000aSetSigGenBuiltIn() or ps4000aSetSigGenArbitrary().
Arguments	handle, identifier for the scope 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_MEMORY_FAIL PICO_INTERNAL_ERROR PICO_TIMEOUT PICO_RESOURCE_ERROR PICO_DEVICE_NOT_FUNCTIONING PICO_NOT_RESPONDING

4.69 ps4000aStop() – stop data capture

```
PICO_STATUS ps4000aStop
(
  int16_t       handle
)
```

This function stops the scope device from sampling data.

When running the device in streaming mode, always call this function after the end of a capture to ensure that the scope is ready for the next capture.

When running the device in block mode, rapid block mode or ETS mode, you can call this function to interrupt data capture.

Note that if you are using block mode and call this function before the oscilloscope is ready, no capture will be available and the driver will return PICO_NO_SAMPLES_AVAILABLE.

Applicability	All modes
Arguments	handle, identifier for the scope device.
Returns	PICO_OK PICO_INVALID_HANDLE PICO_USER_CALLBACK PICO_DRIVER_FUNCTION

4.70 Callback functions

Callback functions are functions that you create as part of your application to receive information from the ps4000a driver. After you register a callback function with the driver, the driver will call the function when a relevant event occurs.

4.70.1 ps4000aBlockReady() – receive notification when block-mode data ready

```
typedef void (PREF4 *ps4000aBlockReady)
(
  int16_t             handle,
  PICO_STATUS         status,
  void                * pParameter
)
```

This callback function receives a notification when block-mode data is ready.

If you wish to use this feature, you must create this function as part of your application. You register it with the ps4000a driver using ps4000aRunBlock(), and the driver calls it back when a capture is complete. This callback function may check that data is available or detect that an error has occurred, but should not attempt to retrieve captured data by calling other functions.

After the callback function has returned, you can download the data using ps4000aGetValues().

Applicability	Block mode only
Arguments	handle, identifier for the scope device. status, indicates whether an error occurred during collection of the data. pParameter, a void pointer passed from ps4000aRunBlock(). The callback function can write to this location to send any data, such as a status flag, back to your application.
Returns	nothing

4.70.2 ps4000aDataReady() – indicate when post-collection data ready

```
typedef void (PREF4 *ps4000aDataReady)
(
  int16_t             handle,
  PICO_STATUS         status,
  uint32_t            noOfSamples,
  int16_t             overflow,
  void                * pParameter
)
```

This callback function receives a notification when post-collection data is ready after a call to ps4000aGetValuesAsync().

If you wish to use this feature, you must create this function as part of your application. You register it with the ps4000a driver using ps4000aGetValuesAsync(), and the driver calls it back when data is ready. You can then download the data using the ps4000aGetValues() function.

Applicability	All modes
Arguments	handle, identifier for the scope device. status, indicates success or failure. noOfSamples, the number of samples collected. overflow, 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. pParameter, a void pointer passed from ps4000aGetValuesAsync(). 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

4.70.3 ps4000aProbeInteractions() – callback for PicoConnect probe events

```
typedef void (PREF4 *ps4000aProbeInteractions)
(
  int16_t                                handle,
  PICO_STATUS                            status,
  PS4000A_USER_PROBE_INTERACTIONS        * probes,
  uint32_t                               nProbes
)
```

This callback function handles notifications of probe changes on scope devices that support PicoConnect™ probes.

If you wish to use this feature, you must create this function as part of your application. You register it with the ps4000a driver using ps4000aSetProbeInteractionCallback(), and the driver calls it back whenever a PicoConnect probe generates an error. See Handling PicoConnect probe interactions for more information on this process.

Applicability	PicoScope 4444 only
Arguments	handle, identifier for the scope device. status, indicates success or failure. If multiple errors have occurred, the most general error is returned here. Probe-specific errors are returned in the status field of the relevant elements of the probes array. probes, on entry, pointer to an array of PS4000A_USER_PROBE_INTERACTIONS structures. nProbes, the number of elements in the probes array.
Returns	nothing

4.70.3.1 PS4000A_USER_PROBE_INTERACTIONS structure

A structure of this type is passed to the ps4000aProbeInteractions() callback function. It is defined as follows:

```
typedef struct tPS4000AUserProbeInteractions
{
  uint16_t                              connected;

  PS4000A_CHANNEL                       channel;
  uint16_t                              enabled;

  PicoConnectProbe                      probeName;

  uint8_t                               requiresPower_;
  uint8_t                               isPowered_;

  PICO_STATUS                           status_;

  PICO_CONNECT_PROBE_RANGE              probeOff;

  PICO_CONNECT_PROBE_RANGE              rangeFirst_;
```

```
    PICO_CONNECT_PROBE_RANGE            rangeLast_;
    PICO_CONNECT_PROBE_RANGE            rangeCurrent_;

    PS4000A_COUPLING                    couplingFirst_;
    PS4000A_COUPLING                    couplingLast_;
    PS4000A_COUPLING                    couplingCurrent_;

    PS4000A_BANDWIDTH_LIMITER_FLAGS     filterFlags_;
    PS4000A_BANDWIDTH_LIMITER_FLAGS     filterCurrent_;

    PS4000A_BANDWIDTH_LIMITER           defaultFilter_;
  } PS4000A_USER_PROBE_INTERACTIONS;
```

Elements

connected, indicates whether the probe is connected or not. The driver saves information on disconnected probes in case they are reconnected, in which case it reapplies the previous settings.

channel, the scope channel to which the probe is connected.

enabled, indicates whether the probe is switched on or off.

probeName, identifies the type of probe from the PICO_CONNECT_PROBE enumerated list.

requiresPower_, indicates whether the probe draws power from the scope.

isPowered_, indicates whether the probe is receiving power.

status_, a status code indicating success or failure. See PicoStatus.h for definitions.

probeOff, the range in use when the probe was last switched off.

rangeFirst_, the first applicable range in the PICO_CONNECT_PROBE_RANGE enumerated list.

rangeLast_, the last applicable range in the PICO_CONNECT_PROBE_RANGE enumerated list.

rangeCurrent_, the range currently in use.

couplingFirst_, the first applicable coupling type in the PS4000A_COUPLING list.

couplingLast_, the last applicable coupling type in the PS4000A_COUPLING list.

couplingCurrent_, the coupling type currently in use.

filterFlags_, a bit field indicating which bandwidth limiter options are available.

filterCurrent_, the bandwidth limiter option currently selected.

defaultFilter_, the default bandwidth limiter option for this type of probe.

4.70.4 ps4000aStreamingReady() – indicate when streaming-mode data ready

```
typedef void (PREF4 *ps4000aStreamingReady)
(
  int16_t       handle,
  int32_t       noOfSamples,
  uint32_t      startIndex,
  int16_t       overflow,
  uint32_t      triggerAt,
  int16_t       triggered,
  int16_t       autoStop,
  void          * pParameter
)
```

This callback function receives a notification when streaming-mode data is ready.

If you wish to use this feature, you must create this function as part of your application. You register it with the ps4000a driver using ps4000aGetStreamingLatestValues(), and the driver calls it back when streaming-mode data is ready.

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

You can then download the data using the ps4000aGetValuesAsync() function.

Applicability	Streaming mode only
Arguments	handle, identifier for the scope device. 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 ps4000aSetDataBuffer(). overflow, returns a set of flags that indicate whether an overvoltage has occurred on any of the channels. It is a bit pattern with bit 0 denoting Channel A. triggerAt, an index to the buffer indicating the location of the trigger point relative to startIndex. The trigger point is therefore at startIndex + triggerAt. 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 ps4000aRunStreaming(). pParameter, a void pointer passed from ps4000aGetStreamingLatestValues(). The callback function can write to this location to send any data, such as a status flag, back to the application.
Returns	nothing

4.71 Wrapper functions

The software development kit (SDK) for your PicoScope device contains wrapper dynamic link library (DLL) files in the `lib` subdirectory of your SDK installation for 32-bit and 64-bit systems. The wrapper functions provided by the wrapper DLLs are for use with programming languages such as MathWorks MATLAB, National Instruments LabVIEW and Microsoft Excel VBA that do not support features of the C programming language such as callback functions.

The source code contained in the wrapper project contains a description of the functions and the input and output parameters.

4.71.1 Streaming mode

Below we explain the sequence of calls required to capture data in streaming mode using the wrapper API functions.

The `ps4000aWrap.dll` wrapper DLL has a callback function for streaming data collection that copies data from the driver buffer specified to a temporary application buffer of the same size. To do this, you must register the driver and application buffers with the wrapper and specify the corresponding channel(s) as being enabled. You should process the data in the temporary application buffer accordingly, for example by copying the data into a large array.

Procedure:
1. Open the oscilloscope using `ps4000aOpenUnit()`.

1a. Inform the wrapper of the number of channels on the device by calling `setChannelCount()`.

2. Select channels, ranges and AC/DC coupling using `ps4000aSetChannel()`.

2a. Inform the wrapper which channels have been enabled by calling `setEnabledChannels()`.

3. Use the appropriate trigger setup functions. For programming languages that do not support structures, use the wrapper's advanced trigger setup functions.

4. Call `ps4000aSetDataBuffer()` (or for aggregated data collection `ps4000aSetDataBuffers()`) to tell the driver where your data buffer(s) is(are).

4a. Register the data buffer(s) with the wrapper and set the application buffer(s) into which the data will be copied. Call `setAppAndDriverBuffers()` (or `setMaxMinAppAndDriverBuffers()` for aggregated data collection).

5. Start the oscilloscope running using `ps4000aRunStreaming()`.

6. Loop and call `GetStreamingLatestValues()` and `IsReady()` to get data and flag when the wrapper is ready for data to be retrieved.

6a. Call the wrapper's `AvailableData()` function to obtain information on the number of samples collected and the start index in the buffer.

6b. Call the wrapper's `IsTriggerReady()` function for information on whether a trigger has occurred and the trigger index relative to the start index in the buffer.

7. Process data returned to your application data buffers.

8. Call `AutoStopped()` if the `autoStop` parameter has been set to `TRUE` in the call to `ps4000aRunStreaming()`.

9. Repeat steps 6 to 8 until AutoStopped() returns true or you wish to stop data collection.

10. Call ps4000aStop(), even if the autoStop parameter was set to TRUE.

11. To disconnect a device, call ps4000aCloseUnit().

4.71.2 Advanced triggers

Use the following functions to set up advanced triggers. ps4000aWrap.c contains the descriptions of the functions.

- setTriggerConditions()
- setTriggerDirections()
- setTriggerProperties()
- setPulseWidthQualifierConditions()

4.71.3 Probe interactions

Applicability	PicoScope 4444 only

Use the following functions to set up probe interaction handling. ps4000aWrap.c contains the descriptions of the functions.

- setProbeInteractionCallback()
- hasProbeStateChanged()
- clearProbeStateChanged()
- getUserProbeInteractionsInfo()
- getNumberOfProbes()
- getUserProbeTypeInfo()
- getUserProbeRangeInfo()
- getUserProbeCouplingInfo()
- getUserProbeBandwidthInfo()

The process to use the probe interaction functions is as follows:

1. Call setProbeInteractionCallback() after opening a connection to the device (ensure any power status codes are processed) and before calling ps4000aSetChannel().

2. Poll hasProbeStateChanged().

3. Retrieve the initial probe information after a short delay of a few milliseconds:

    a. If your programming language supports structs call getUserProbeInteractionsInfo(), otherwise

    b. Call the following functions:

        i. getNumberOfProbes() to obtain the number of probes and status code from the callback function

        ii. getUserProbeTypeInfo() to retrieve information about the type of probe, channel connected on and power for the probe number specified

        iii. getUserProbeRangeInfo() to retrieve information on the probe range for the probe number specified

iv. `getUserProbeCouplingInfo()` to retrieve information on the probe coupling for the probe number specified

v. `getUserProbeBandwidthInfo()` to retrieve information on the probe bandwidth limiter options for the probe number specified

vi. `clearProbeStateChanged()` – to reset the flag that indicates if there has been a change to the probe status

4. Repeat step 3 to obtain the actual probe information.

5. For subsequent queries to check if the probe status has changed, either call the `hasProbeStateChanged()` function once or poll it for a defined period of time to check if there have been any changes.

The probe number is zero-based.

5 Reference

5.1 Driver status codes

Every function in the ps4000a.dll driver returns a status code from the list of PICO_STATUS values defined in the PicoStatus.h header file supplied with the SDK. See the header file for more information.

5.2 Enumerated types and constants

Enumerated types and constants are defined in the files ps4000aApi.h and PicoConnectProbes.h, which are included in the PicoSDK. We recommend that you refer to these constants by name unless your programming environment forces you to use numeric values.

5.3 Numeric data types

Here is a list of the sizes and ranges of the numeric data types used in the ps4000a API.

Type	Bits	Signed or unsigned?
int8_t	8	signed
int16_t	16	signed
uint16_t	16	unsigned
enum	32	enumerated
int32_t	32	signed
uint32_t	32	unsigned
float	32	signed (IEEE 754)
double	64	signed (IEEE 754)
int64_t	64	signed
uint64_t	64	unsigned

5.4 Glossary

ADC. Analog-to-digital converter. The electronic component in a PC oscilloscope that converts analog signals from the inputs into digital data suitable for transmission to the PC.

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

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

Callback. A mechanism that the ps4000a 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.

Coupling mode. This mode selects either AC or DC coupling in the oscilloscope's input path. Use AC mode for small signals that may be superimposed on a DC level. Use DC mode for measuring absolute voltage levels. Set the coupling mode using ps4000aSetChannel().

Differential oscilloscope. An oscilloscope that measures the difference between two input voltages on each channel. Conventional oscilloscopes are *single-ended*, meaning that they measure the difference between one input voltage and a common ground on each channel.

GS/s. Gigasamples (billions of samples) per second.

Maximum sampling rate. A figure indicating the maximum number of samples the oscilloscope can acquire per second. The higher the sampling rate of the oscilloscope, the more accurate the representation of the high-frequency details in a fast signal.

MS/s. Megasamples (millions of samples) per second.

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

PicoConnect™. A range of probes compatible with devices such as the PicoScope 4444 differential oscilloscope. PicoConnect probe types can be identified by the `ps4000a` driver, allowing an application to configure itself automatically when a probe is plugged is or unplugged. Some probes offer additional functions such as software-controlled range setting.

PicoScope 4000 Series. A range of high-resolution PC Oscilloscopes from Pico Technology. The range includes two-channel and four-channel models, with or without a built-in function generator and arbitrary waveform generator.

Streaming mode. A sampling mode in which the oscilloscope samples data and returns it to the computer in an unbroken stream. This mode allows the capture of data sets whose size is not limited by the size of the scope's memory buffer, at sampling rates up to 160 million samples per second.

Timebase. The sampling rate that the scope uses to acquire data. The timebase can be set to any value returned by the `ps4000aGetTimebase()` or `ps4000aGetTimebase2()` functions.

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

USB 2.0. Universal Serial Bus (High Speed). The maximum signaling rate is 480 megabits per second.

USB 3.0. Universal Serial Bus (SuperSpeed). The maximum signaling rate is 5 gigabits per second. Also known as **USB 3.1 Gen 1**.

Vertical resolution. A value, in bits, indicating the precision with which the oscilloscope converts input voltages to digital values.

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

Index

H

I

L

M

O

P

R

S

T

U

V

W

UK global headquarters:

Pico Technology
James House
Colmworth Business Park
St. Neots
Cambridgeshire
PE19 8YP
United Kingdom

Tel: +44 (0) 1480 396 395

sales@picotech.com
support@picotech.com

www.picotech.com

North America regional office:

Pico Technology
320 N Glenwood Blvd
Tyler
TX 75702
United States

Tel: +1 800 591 2796

sales@picotech.com
support@picotech.com

Asia-Pacific regional office:

Pico Technology
Room 2252, 22/F, Centro
568 Hengfeng Road
Zhabei District
Shanghai 200070
PR China

Tel: +86 21 2226-5152

pico.asia-pacific@picotech.com

ps4000apg.en-6