USB DrDAQ

Data Logger

Programmer's Guide

Contents
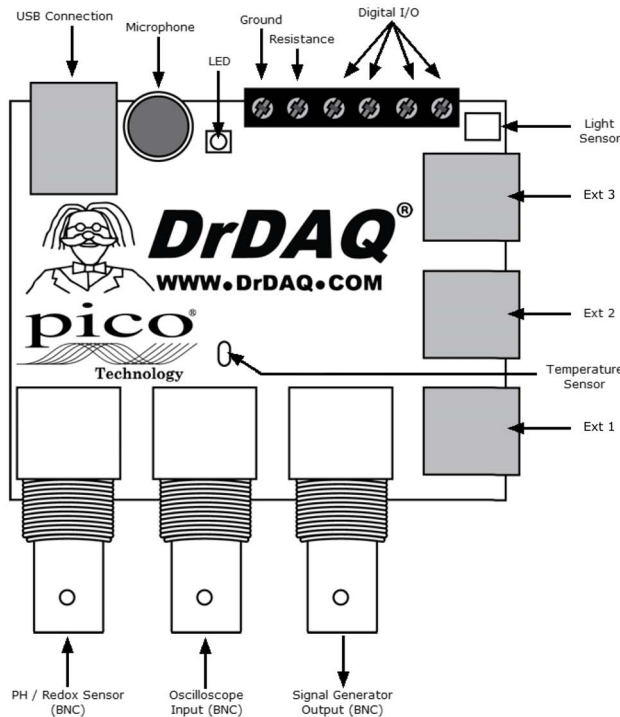
1 Introduction

1.1 Overview

The USB DrDAQ PC Data Logger is a medium-speed, multichannel voltage-input device for sampling data using a PC. This manual explains how to use the Application Programming Interface and drivers to write your own programs to control the unit. You should read it in conjunction with the USB DrDAQ User's Guide.

The Software Development Kit for the USB DrDAQ is compatible with Microsoft Windows XP, Vista and 7 (32-bit and 64-bit editions).

1.2 Safety warning

USB DrDAQ ground is connected directly to the ground of your computer. As with most oscilloscopes and data loggers, this is done in order to minimise interference. You should take care not to connect the ground (screw terminal, outer shell of BNC or exposed metalwork) of USB DrDAQ to anything which may be at some voltage other than ground, as doing so may cause damage to the unit. If in doubt, use a meter to check that there is no significant AC or DC voltage.

For computers that do not have an earth connection (for example, laptops), it must be assumed that USB DrDAQ is not protected by an earth (in the same way a battery multimeter is not protected by an earth).

The scope channel on the USB DrDAQ has a maximum input voltage range of ±10 V. The maximum input voltage for all other inputs is 0 to 5 V. Any voltage in excess of ±30 V may cause permanent damage to the unit.

The unit contains no user serviceable parts: repair or calibration of the unit requires specialised test equipment and must be performed by Pico Technology Limited.

1.3 Legal information

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

Access

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

Usage

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

Copyright

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

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

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

Mission-critical applications

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

Viruses

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

Support

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

Upgrades

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

Trademarks

Pico Technology, PicoScope, PicoLog, DrDAQ and EnviroMon are trademarks of Pico Technology Ltd., registered in the United Kingdom and other countries. Pico Technology acknowledges the following product names as trademarks of their respective owners: Windows, Excel, Visual Basic, LabVIEW, Agilent VEE, HP VEE, Delphi.

1.4 Contact details

Address: Pico Technology
James House
Colmworth Business Park
ST NEOTS
Cambridgeshire
PE19 8YP
United Kingdom
Tel: +44 1480 396395
Fax: +44 1480 396296
Web site: www.picotech.com

2   Getting started

2.1   Software updates

Our software is regularly updated with new features. To check what version of the software you are running, start PicoScope or PicoLog and select the Help | About menu. The latest version of software can be downloaded free of charge from the DrDAQ web site at http://www.drdaq.com. Alternatively, the latest software can be purchased on disk or CD from Pico Technology.

To be kept up-to-date with news of new software releases join our e-mail mailing list. This can be done from the main Pico Technology web site at http://www.picotech.com/.

2.2   USB DrDAQ scaling files (.DDS)

The DrDAQ driver has built-in scaling for each of the built-in and Pico-supplied sensors. You can incorporate scaling for your own sensors by adding a file called `scaling.dds` ⌐4⌐ (where *'scaling'* can be replaced with a name of your choice). This file will contain the details of your sensor.

The values returned by the driver are integers that represent fixed-point decimal numbers. For example, the driver treats pH as a value with two decimal places, so a pH of 7.65 is returned as 765.

You can call the routine UsbDrDaqGetChannelInfo ⌐36⌐ to find out how many decimal places a channel is using, and also to get a divider that converts the integer value to the corresponding real number. For pH, the returned divider is 100, so 765 divided by 100 gives 7.65.

For some sensors, there is more than one possible scaling available. You can call UsbDrDaqGetScalings ⌐24⌐ to get a list of valid scaling codes, then call UsbDrDaqSetScalings ⌐25⌐ to select one of them. Once selected, UsbDrDaqGetChannelInfo ⌐36⌐ will return full information about the selected scaling. If you do not use UsbDrDaqSetScalings ⌐25⌐, the driver will automatically select the first available scaling for each channel.

USB DrDAQ scaling files can be used to supplement the scalings built into the driver. Several DDS files may be used, and these must be placed in the current working directory where the USB DrDAQ software is installed. The total number of sets of scaling data in all the files used must not exceed 99.

Each scaling file may contain more than one set of scaling data. Each scaling must have a unique scaling number, contained in the `[Scale...]` section heading.

A set of typical entries from a `.DDS` file is shown below:

```
[Scale1]
Resistor=1
LongName=CustomTemperature1
ShortName=TempC
Units=C
MinValue=-40
MaxValue=120
OutOfRange=0
Places=1
Method=0
IsFast=Yes
NoOfPoints=32
```

```
Raw1=2.385
Scaled1=-30
...
Raw32=1.32
Scaled32=100

[Scale2]
Resistor=2.2
LongName=CustomTemperature2
ShortName=TempF
Units=F
MinValue=32
MaxValue=160
...
[Scale3]
Resistor=3.3
LongName=CustomLight
ShortName=Light
Units=lux
MinValue=0
MaxValue=20000
...
```

The meanings of the terms in the `.DDS` file are as follows:

`[Scale1]`

A unique number, from 1 to 99, to identify this entry. (Pico-created numbers are from 100 upwards.)

`Resistor=1`

The ID resistor value in kiloohms. In this example "`1`" represents 1k, "`2.2`" represents 2k2 and so on.

For external sensors, this resistor should be fitted in the sensor. You must use one of the following resistors: 1k0, 2k2, 3k3, 5k6, 7k5 or 10k. The resistor must be 1% tolerance or better.

For internal sensors, use the following 'virtual' resistor values:

1	Sound Waveform	1200
2	Sound Level	1300
3	Voltage	1500
4	Resistance	1600
5	pH	1400
6	Temperature	1100
7	Light	1000

`LongName=Temperature`

Used in PicoLog

`ShortName=TempC`

This field is not used by USB DrDAQ running PicoScope or PicoLog.

`Units=C`

Displayed on graphs

```
MinValue=-40
MaxValue=120
```

Note: For PicoScope these values will determine the maximum and minimum values displayed in Scope View. For PicoLog these values determine what Maximum range is displayed in the Graph View (set in the Graph Options dialog).

```
Places=1
```

Number of decimal places. The options are 0, 1, 2 and 3. With `places=1` the value 15.743 would be returned as 157, meaning 15.7. With `places=2`, the same value would be returned as 1574.

```
Method=0
```

This specifies the scaling method. 0 specifies table lookup and 1 specifies linear scaling.

```
Offset=0
Gain=1
```

These are the offset and gain values for linear scaling.

```
OutOfRange=0
```

This specifies what to do if the raw value is outside the range of the table lookup. The options are:
0 - treat as a sensor failure
1 - clip the value to the minimum or maximum table value
2 - extrapolate the value using the nearest two table entries.

```
ScopeRange=1.25V
```

This is used when scaling the oscilloscope channel. It specifies the range of the oscilloscope channel that should be used. Possible values are 10 V, 5 V, 2.5 V, and 1.25 V.

```
NoOfPoints=32
```

This is the number of table lookup points.

```
Raw1=2.385
```

Raw value for the first point in the look up table. The value is in V (volts) and should not be greater than 2.500 V.

```
Scaled1=-30
```

Scaled value for the first point in the look up table. The units are specified by the units parameter.

3 Writing your own software

3.1 About the driver

USB DrDAQ is supplied with a kernel driver, and a DLL containing routines that you can build into your own programs. The driver is supported by the following operating systems:

- Windows XP (SP2 or later)
- Windows Vista
- Windows 7

Once you have installed the software, the installation directory will contain the drivers and a selection of examples of how to use the drivers. It also contains a copy of this help file in PDF format.

The DLL can be used with any programming language or application that can interface with DLLs: for example, C, Delphi, Visual Basic, Excel and LabVIEW. The SDK contains an example for Excel, a C console example that demonstrates all of the facilities in the driver and a LabVIEW interface and examples.

The driver supplied is a 32-bit DLL. However, it will still run on a 64-bit Windows system if you write a 32-bit application and run it under WoW64.

The driver supports up to 64 units at one time.

3.2 Installing the driver

The driver is supplied with the USB DrDAQ SDK. You can download the latest version of the SDK from our website at:

http://www.picotech.com/software.html

Select "USB DrDAQ" as the hardware product and then "SDK" as the software product.

The easiest way to install the USB DrDAQ kernel driver is to install PicoScope or PicoLog. These programs are available free of charge from the Pico Technology website. If you prefer to install the driver manually, then proceed as follows:

1. Go to the directory where you unzipped the USB DrDAQ Series SDK
2. Go to the subdirectory called `system\`
3. Run the program called `dpinst.exe`
4. Plug the device in and follow the instructions in New Hardware Wizard

3.3 Connecting the logger

Before you connect your logger, you must first install the driver 7.

To connect the data logger, plug the cable provided into any available USB port on your PC. The first time you connect the unit, Windows will display a New Hardware Wizard. Follow any instructions in the Wizard and wait for the driver to be installed. The unit is then ready for use.

3.4   Capture modes

Three modes are available for capturing data:

- 🔵 `BM_SINGLE:` collect a single block of data and exit

- 🔵 `BM_WINDOW:` collect a series of overlapping blocks of data

- 🔵 `BM_STREAM:` collect a continuous stream of data

`BM_SINGLE` is useful when you wish to collect data at high speed for a short period: for example, to collect 1000 readings in 50 milliseconds.

`BM_WINDOW` is useful when collecting several blocks of data at low speeds - for example when collecting 10,000 samples over 10 seconds. Collecting a sequence of single blocks like this would take 10 seconds for each block, so displayed data would not be updated frequently. Using windowing, it is possible to ask for a new block more frequently, for example every second, and to receive a block containing 9 seconds of repeat data and 1 second of new data. The block is effectively a 10-second window that advances one second per cycle.

`BM_STREAM` is useful when you need to collect data continuously for long periods. In principle, it could be used to collect data indefinitely. Every time UsbDrDaqGetValues [19] is called, it returns the new readings since the last time it was called. The `noOfValues` argument passed to UsbDrDaqRun [14] must be sufficient to ensure that the buffer does not overflow between successive calls to UsbDrDaqGetValues [19]. For example, if you call UsbDrDaqGetValues [19] every second and you are collecting 500 samples per second, then `noOfValues` must be at least 500, or preferably 1000, to allow for delays in the operating system.

3.5 USB DrDAQ Routines

3.5.1 Summary

The following table explains each of the driver routines supplied with the USB DrDAQ data logger:

Routine	Description
UsbDrDaqOpenUnit 11	open and enumerate the unit
UsbDrDaqCloseUnit 12	close the unit
UsbDrDaqGetUnitInfo 13	return various items of information about the unit
UsbDrDaqRun 14	tell the unit to start capturing data
UsbDrDaqReady 15	indicate when UsbDrDaqRun 14 has captured data
UsbDrDaqStop 16	abort data collection
UsbDrDaqSetInterval 17	set the sampling speed of the unit
UsbDrDaqSetTrigger 18	set the trigger on the unit
UsbDrDaqGetValues 19	get a number of sample values after a run
UsbDrDaqGetTriggerTimeOffsetNs 21	returns the time between the trigger point and the first post-trigger sample
UsbDrDaqGetSingle 21	get a single value from a specified channel
UsbDrDaqOpenUnitAsync 22	open the unit without waiting for completion
UsbDrDaqOpenUnitProgress 23	report progress of UsbDrDaqOpenUnitAsync 22
UsbDrDaqGetScalings 24	discover the scalings, both built-in and custom, that are available
UsbDrDaqSetScalings 25	sets the scaling for a particular channel
UsbDrDaqSetSigGenBuiltIn 26	sets the arbitrary waveform generator using standard waveform types
UsbDrDaqSetSigGenArbitrary 27	allows full control of the arbitrary waveform generator
UsbDrDaqStopSigGen 28	turns the AWG off
UsbDrDaqSetDO 29	control the digital outputs on the unit
UsbDrDaqSetPWM 30	used to configure the general-purpose I/Os as pulse-width modulation outputs
UsbDrDaqGetInput 31	used to configure the general-purpose I/Os as digital inputs
UsbDrDaqStartPulseCount 32	used to configure the general-purpose I/Os for pulse counting and to start counting
UsbDrDaqGetPulseCount 33	will return the current pulse count
UsbDrDaqEnableRGBLED 34	enables or disables RGB mode on the LED
UsbDrDaqSetRGBLED 35	used to set the colour of the LED once RGB mode has been enabled
UsbDrDaqGetChannelInfo 36	returns a set of information about the currently selected scaling for the specified channel

The driver allows you to do the following:

- Identify and open the logger
- Take a single reading from a particular channel
- Collect a block of samples at fixed time intervals from one or more channels
- Set up a trigger event for a particular channel
- Get information about scalings available for a channel
- Select a scaling for a channel
- Control and read general-purpose I/Os
- Control arbitrary waveform generator

You can specify a sampling interval from 1 microsecond to 1 second. The shortest interval that the driver will accept depends on the [capture mode](#) 8 selected.

The normal calling sequence to collect a block of data is as follows:

```
Check that the driver version is correct
Open the driver
Set trigger mode (if required)
Set sampling mode (channels and time per sample)

While you want to take measurements,
  Run
  While not ready
    Wait
  End while
  ... Get a block of data ...
End While
Close the driver
```

3.5.2   UsbDrDaqOpenUnit

```
PICO_STATUS UsbDrDaqOpenUnit(
  short * handle
)
```

This function opens and enumerates the unit.

Arguments:	handle: the function will write a value here that uniquely identifies the data logger that was opened. Use this as the handle parameter when calling any other UsbDrDaq API function.
Returns:	PICO_OK [37] PICO_OS_NOT_SUPPORTED [37] PICO_OPEN_OPERATION_IN_PROGRESS [37] PICO_EEPROM_CORRUPT [37] PICO_KERNEL_DRIVER_TOO_OLD [37] PICO_FW_FAIL [37] PICO_MAX_UNITS_OPENED [37] PICO_NOT_FOUND [37] PICO_NOT_RESPONDING [37]

3.5.3   UsbDrDaqCloseUnit

```
PICO_STATUS UsbDrDaqCloseUnit(
   short  handle
)
```

This function closes the unit.

Arguments:	handle: handle returned from UsbDrDaqOpenUnit [11] or UsbDrDaqOpenUnitProgress [23]
Returns:	PICO_OK [37] PICO_HANDLE_INVALID [37]

3.5.4 UsbDrDaqGetUnitInfo

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

This function returns a string containing the specified item of information about the unit.

If you want to find out the length of the string before allocating a buffer for it, then call the function with `string = NULL` first.

Arguments:	`handle:` handle returned from UsbDrDaqOpenUnit 11 or UsbDrDaqOpenUnitProgress 23
	`string:` a location where the function writes the requested information, or `NULL` if you are only interested in the value of `requiredSize`
	`stringLength:` the maximum number of characters that the function should write to `string`
	`requiredSize:` a location where the function writes the length of the information string before it was truncated to `stringLength`. If the string was not truncated then `requiredSize` will be less than or equal to `stringLength`.

	`info:` the information that the driver should return. These values are specified in `picoStatus.h`:
	`PICO_DRIVER_VERSION`
	`PICO_USB_VERSION`
	`PICO_HARDWARE_VERSION`
	`PICO_VARIANT_INFO`
	`PICO_BATCH_AND_SERIAL`
	`PICO_CAL_DATE`
	`PICO_KERNEL_DRIVER_VERSION`

Returns:	PICO_OK 37 PICO_INVALID_HANDLE 37 PICO_NULL_PARAMETER 37 PICO_INVALID_INFO 37 PICO_INFO_UNAVAILABLE 37

3.5.5   UsbDrDaqRun

```
PICO_STATUS UsbDrDaqRun(
    short                 handle,
    unsigned long         no_of_values,
    UsbDrDaq_BLOCK_METHOD  method
)
```

This function tells the unit to start capturing data.

Arguments:	`handle:` handle returned from [UsbDrDaqOpenUnit](11) or [UsbDrDaqOpenUnitProgress](23)
	
	`no_of_values:` the number of samples the unit should collect
	
	`method:` which method to use to collect the data, from the following list:
	    `BM_SINGLE`
	    `BM_WINDOW`
	    `BM_STREAM`
	
	    See "[Capture modes](8)" for details.

Returns:	[PICO_OK](37) [PICO_INVALID_HANDLE](37) [PICO_USER_CALLBACK](37) [PICO_INVALID_CHANNEL](37) [PICO_TOO_MANY_SAMPLES](37) [PICO_INVALID_TIMEBASE](37) [PICO_NOT_RESPONDING](37) [PICO_CONFIG_FAIL](37) [PICO_INVALID_PARAMETER](37) [PICO_NOT_RESPONDING](37) [PICO_TRIGGER_ERROR](37)

3.5.6   UsbDrDaqReady

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

This function indicates when UsbDrDaqRun [14] has captured the requested number of samples.

Arguments:	`handle:` handle returned from UsbDrDaqOpenUnit [11] or UsbDrDaqOpenUnitProgress [23] `ready: TRUE` if ready, `FALSE` otherwise
Returns:	PICO_OK [37] PICO_INVALID_HANDLE [37] PICO_NOT_RESPONDING [37]

3.5.7   UsbDrDaqStop

```
PICO_STATUS UsbDrDaqStop(
  short handle
)
```

This function aborts data collection.

Arguments:	handle: handle returned from UsbDrDaqOpenUnit 11 or UsbDrDaqOpenUnitProgress 23
Returns:	PICO_OK 37 PICO_INVALID_HANDLE 37

3.5.8 UsbDrDaqSetInterval

```
PICO_STATUS UsbDrDaqSetInterval(
    short          handle,
    unsigned long * us_for_block,
    unsigned long  ideal_no_of_samples,
    short        * channels,
    short          no_of_channels
)
```

This function sets the sampling rate of the unit.

The fastest possible sampling interval is 1 microsecond, when the number of samples is 8129 divided by the number of channels active and the capture mode⌐8⌐ is BM_SINGLE. Under all other conditions, the fastest possible sampling interval is 10 microseconds.

Arguments:	`handle:` handle returned from UsbDrDaqOpenUnit⌐11⌐ or UsbDrDaqOpenUnitProgress⌐23⌐
	`us_for_block:` target total time in which to collect `ideal_no_of_samples`, in microseconds
	`ideal_no_of_samples:` the number of samples that you want to collect. This number is used only for timing calculations.
	`channels:` an array of constants identifying the channels from which you wish to capture data. `USB_DRDAQ_CHANNEL_EXT1`, `USB_DRDAQ_CHANNEL_EXT2`, `USB_DRDAQ_CHANNEL_EXT3`, `USB_DRDAQ_CHANNEL_SCOPE`, `USB_DRDAQ_CHANNEL_PH`, `USB_DRDAQ_CHANNEL_RES`, `USB_DRDAQ_CHANNEL_LIGHT`, `USB_DRDAQ_CHANNEL_TEMP`, `USB_DRDAQ_CHANNEL_MIC_WAVE`, `USB_DRDAQ_CHANNEL_MIC_LEVEL`.
	Note: If you specify the channels in a different order from that shown here, the function will re-order them.
	`no_of_channels:` the number of channels in the `channels` array.
Returns:	PICO_OK⌐37⌐
	PICO_INVALID_HANDLE⌐37⌐
	PICO_INVALID_CHANNEL⌐37⌐
	PICO_INVALID_TIMEBASE⌐37⌐
	PICO_NOT_RESPONDING⌐37⌐
	PICO_CONFIG_FAIL⌐37⌐
	PICO_INVALID_PARAMETER⌐37⌐
	PICO_NOT_RESPONDING⌐37⌐
	PICO_TRIGGER_ERROR⌐37⌐

3.5.9   UsbDrDaqSetTrigger

```
PICO_STATUS UsbDrDaqSetTrigger(
    short           handle,
    unsigned short  enabled,
    unsigned short  auto_trigger,
    unsigned short  auto_ms,
    unsigned short  channel,
    unsigned short  dir,
    unsigned short  threshold,
    unsigned short  hysteresis,
    float           delay
)
```

This function sets up the trigger, which controls when the unit starts capturing data.

Arguments:	`handle:` handle returned from UsbDrDaqOpenUnit [11] or UsbDrDaqOpenUnitProgress [23]

	`enabled:` whether to enable or disable the trigger:
	    0: disable the trigger
	    1: enable the trigger

	`auto_trigger:` whether or not to re-arm the trigger automatically after each trigger event:
	    0: do not auto-trigger
	    1: auto-trigger

	`auto_ms:` time in milliseconds after which the unit will auto-trigger if the trigger condition is not met
	`channel:` which channel to trigger on.

	`dir:` which edge to trigger on:
	    0: rising edge
	    1: falling edge

	`threshold:` trigger threshold (the level at which the trigger will activate) in ADC counts
	`hysteresis:` trigger hysteresis in ADC counts. This is the difference between the upper and lower thresholds. The signal must then pass through both thresholds in the same direction in order to activate the trigger, so that there are fewer unwanted trigger events caused by noise. The minimum value allowed is 1.
	`delay:` delay between the trigger event and the start of the block as a percentage of the block size. 0% means that the trigger event is the first data value in the block, and -50% means that the trigger event is in the middle of the block.
Returns:	PICO_OK [37] PICO_INVALID_HANDLE [37] PICO_USER_CALLBACK [37] PICO_TRIGGER_ERROR [37] PICO_MEMORY_FAIL [37]

3.5.10 UsbDrDaqGetValues

```
PICO_STATUS UsbDrDaqGetValues(
    short             handle,
    short           * values,
    unsigned long   * noOfValues,
    unsigned short  * overflow,
    unsigned long   * triggerIndex
)
```

This function is used to get values after calling UsbDrDaqRun 14 .

Arguments:	handle: handle returned from UsbDrDaqOpenUnit 11 or UsbDrDaqOpenUnitProgress 23
	values: an array of sample values returned by the function. The size of this buffer must be the number of enabled channels multiplied by the number of samples to be collected.
	Note: The order of the channels will be as stated in `UsbDrDaqApi.h`, regardless of the order used in the UsbDrDaqSetInterval 17 `channels` array.
	noOfValues: on entry, the number of sample values per channel that the function should collect. On exit, the number of samples per channel that were actually written to the buffer.
	overflow: on exit, a bit field indicating which, if any, input channels overflowed the input range of the device. A bit set to 1 indicates an overflow. The least significant bit corresponds to channel 1. May be `NULL` if an overflow warning is not required.
	triggerIndex: on exit, a number indicating when the trigger event occurred. The number is a zero-based index to the `values` array, or 0xFFFFFFFF if the information is not available. On entry, the pointer may be `NULL` if a trigger index is not required.
Returns:	PICO_OK 37 PICO_INVALID_HANDLE 37 PICO_NO_SAMPLES_AVAILABLE 37 PICO_DEVICE_SAMPLING 37 PICO_NULL_PARAMETER 37 PICO_INVALID_PARAMETER 37 PICO_TOO_MANY_SAMPLES 37 PICO_DATA_NOT_AVAILABLE 37 PICO_INVALID_CALL 37 PICO_NOT_RESPONDING 37 PICO_MEMORY 37

3.5.11 UsbDrDaqGetTriggerTimeOffsetNs

```
PICO_STATUS UsbDrDaqGetTriggerTimeOffsetNs(
  short    handle
  _int64 * time
)
```

This function returns the time between the trigger point and the first post-trigger sample. This is calculated using linear interpolation.

Arguments:	`handle:` handle returned from [UsbDrDaqOpenUnit] 11 or [UsbDrDaqOpenUnitProgress] 23 `time:` a location where the driver writes the trigger time in nanoseconds.
Returns:	[PICO_OK] 37 [PICO_NOT_FOUND] 37

3.5.12   UsbDrDaqGetSingle

```
PICO_STATUS UsbDrDaqGetSingle(
   short               handle,
   USB_DrDAQ_INPUTS   channel,
   unsigned short    * value
   unsigned short    * overflow
)
```

This function returns a single sample value from the specified input channel.

Arguments:	`handle:` handle returned from UsbDrDaqOpenUnit [11] or UsbDrDaqOpenUnitProgress [23] `channel:` which channel to sample `value:` a location where the function will write the sample value `overflow:` on exit, a bit field indicating which, if any, input channels overflowed the input range of the device. A bit set to 1 indicates an overflow. The least significant bit corresponds to channel 1. May be `NULL` if an overflow warning is not required.
Returns:	PICO_OK [37] PICO_INVALID_HANDLE [37] PICO_NO_SAMPLES_AVAILABLE [37] PICO_DEVICE_SAMPLING [37] PICO_NULL_PARAMETER [37] PICO_INVALID_PARAMETER [37] PICO_DATA_NOT_AVAILABLE [37] PICO_INVALID_CALL [37] PICO_NOT_RESPONDING [37] PICO_MEMORY [37]

3.5.13   UsbDrDaqOpenUnitAsync

```
PICO_STATUS UsbDrDaqOpenUnitAsync(
  short * status
)
```

This function opens a USB DrDAQ data logger without waiting for the operation to finish. You can find out when it has finished by periodically calling UsbDrDaqOpenUnitProgress 23 until that function returns a non-zero value and a valid data logger handle.

The driver can support up to 64 data loggers.

Arguments:	status: the function writes a status flag to this location:
	    0 if there is already an open operation in progress
	    1 if the open operation is initiated

Returns:	PICO_OK 37 PICO_OPEN_OPERATION_IN_PROGRESS 37 PICO_OPERATION_FAILED 37

3.5.14 UsbDrDaqOpenUnitProgress

```
PICO_STATUS UsbDrDaqOpenUnitProgress(
  short * handle,
  short * progress,
  short * complete
)
```

This function checks on the progress of [UsbDrDaqOpenUnitAsync](22).

Arguments:	`handle:` a pointer to where the function should store the handle of the opened data logger, if the operation was successful. Use this as the `handle` parameter when calling any other USB DrDAQ API function.
	0: if no unit is found or the unit fails to open
	<>0: handle of unit (valid only if function returns `PICO_OK`)
	`progress:` a location where the function writes an estimate of the progress towards opening the data logger. The value is between 0 to 100.
	`complete:` a location where the function will write a non-zero value if the operation has completed
Returns:	[PICO_OK](37) [PICO_NULL_PARAMETER](37) [PICO_OPERATION_FAILED](37)

3.5.15   UsbDrDaqGetScalings

```
PICO_STATUS UsbDrDaqGetScalings(
    short                handle
    USB_DRDAQ_INPUTS   channel,
    short              * nScales,
    short              * currentScale,
    char               * names,
    short                namesSize
```

This function discovers the scalings, both built-in and custom, that are available for a particular channel.

Arguments:	handle: handle returned from UsbDrDaqOpenUnit [11] or UsbDrDaqOpenUnitProgress [23] channel: the channel of interest nScales: the function will write the number of available scales here currentScale: The function will write an index to the currently selected scale here names: The function will write a string containing the scaling names and indices namesSize: the size of names
Returns:	PICO_OK [37] PICO_NOT_FOUND [37] PICO_INVALID_CHANNEL [37]

3.5.16 UsbDrDaqSetScalings

```
PICO_STATUS UsbDrDaqSetScalings(
  short              handle
  USB_DRDAQ_INPUTS  channel,
  short              scalingNumber
)
```

This function sets the scaling for a particular channel.

Arguments:	`handle:` handle returned from <u>UsbDrDaqOpenUnit</u> [11] or <u>UsbDrDaqOpenUnitProgress</u> [23]
	`channel:` the channel of interest
	`scalingNumber:` the number of the required scale, as given by <u>UsbDrDaqGetScalings</u> [24]
Returns:	<u>PICO_OK</u> [37]
	<u>PICO_NOT_FOUND</u> [37]
	<u>PICO_INVALID_CHANNEL</u> [37]
	<u>PICO_INVALID_PARAMETER</u> [37]

3.5.17   UsbDrDaqSetSigGenBuiltIn

```
PICO_STATUS UsbDrDaqSetSigGenBuiltIn(
    short            handle,
    long             offsetVoltage,
    unsigned long    pkToPk,
    short            frequency,
    USB_DRDAQ_WAVE   waveType

)
```

This function sets the arbitrary waveform generator using standard waveform types.

Arguments:	`handle:` handle returned from UsbDrDaqOpenUnit[11] or UsbDrDaqOpenUnitProgress[23]
	
	`offsetVoltage:` The offset voltage in microvolts. The offset voltage must be in the range -1.5 V to 1.5 V.
	
	`pkToPk:` The peak-to-peak voltage in microvolts. The maximum allowed is 3 V.
	
	`frequency:` Frequency in hertz. The maximum allowed frequency is 20 kHz.
	
	`waveType:` An enumerated data type that has the following values, corresponding to standard waveforms:
	
	`USB_DRDAQ_SINE,`
	`USB_DRDAQ_SQUARE,`
	`USB_DRDAQ_TRIANGLE,`
	`USB_DRDAQ_RAMP_UP,`
	`USB_DRDAQ_RAMP_DOWN,`
	`USB_DRDAQ_DC`

Returns:	PICO_OK[37] PICO_NOT_FOUND[37] PICO_NOT_RESPONDING[37] PICO_INVALID_PARAMETER[37]

3.5.18 UsbDrDaqSetSigGenArbitrary

```
PICO_STATUS UsbDrDaqSetSigGenArbitrary(
short              handle,
long               offsetVoltage,
unsigned long      pkToPk,
short            * arbitraryWaveform,
short              arbitraryWaveformSize,
long               updateRate

)
```

This function allows full control of the arbitrary waveform generator by allowing an arbitrary waveform to be passed to the driver.

Arguments:	handle: handle returned from UsbDrDaqOpenUnit 11 or UsbDrDaqOpenUnitProgress 23 offsetVoltage: The offset voltage in microvolts. The offset voltage must be in the range -1.5 V to 1.5 V. pkToPk: The peak-to-peak voltage in microvolts. The maximum allowed is 3 V. arbitraryWaveform: A pointer to an array containing the waveform. The waveform values must be in the range -1000 to 1000. arbitraryWaveformSize: The number of points in the waveform. updateRate: This is the rate at which the AWG steps through the points in the waveform. This value must be in the range 1 to 2000000 points per second.
Returns:	PICO_OK 37 PICO_NOT_FOUND 37 PICO_NOT_RESPONDING 37 PICO_INVALID_PARAMETER 37

3.5.19 UsbDrDaqStopSigGen

```
PICO_STATUS UsbDrDaqStopSigGen(
  short         handle,

)
```

This function turns the AWG off.

Arguments:	`handle:` handle returned from UsbDrDaqOpenUnit [11] or UsbDrDaqOpenUnitProgress [23]
Returns:	PICO_OK [37] PICO_NOT_FOUND [37] PICO_NOT_RESPONDING [37]

3.5.20 UsbDrDaqSetDO

```
PICO_STATUS UsbDrDaqSetDO(
    short              handle,
    USB_DRDAQ_GPIO     IOChannel,
    short              value

    )
```

This function is used to configure the general-purpose I/Os as digital outputs.

Arguments:	`handle:` handle returned from UsbDrDaqOpenUnit [11] or UsbDrDaqOpenUnitProgress [23] `IOChannel:` One of the following: `USB_DRDAQ_GPIO_1 = 1,` `USB_DRDAQ_GPIO_2,` `USB_DRDAQ_GPIO_3,` `USB_DRDAQ_GPIO_4` `value:` Used to set or clear the digital output. Any non-zero value will set the output and zero will clear it.
Returns:	PICO_OK [37] PICO_NOT_FOUND [37] PICO_NOT_RESPONDING [37] PICO_INVALID_PARAMETER [37]

3.5.21   UsbDrDaqSetPWM

```
PICO_STATUS UsbDrDaqSetPWM(
    short              handle,
    USB_DRDAQ_GPIO     IOChannel,
    unsigned short     period,
    unsigned char      cycle

)
```

This function is used to configure the general-purpose I/Os as pulse-width modulation outputs.

Arguments:	`handle:` handle returned from <u>UsbDrDaqOpenUnit</u> 11 or <u>UsbDrDaqOpenUnitProgress</u> 23 `IOChannel:` GPIOs 1 and 2 can be used as PWM outputs. `period:` The period of the waveform in microseconds. `cycle:` Duty cycle as a percentage
Returns:	<u>PICO_OK</u> 37 <u>PICO_NOT_FOUND</u> 37 <u>PICO_NOT_RESPONDING</u> 37 <u>PICO_INVALID_PARAMETER</u> 37

3.5.22 UsbDrDaqGetInput

```
PICO_STATUS UsbDrDaqGetInput(
    short               handle,
    USB_DRDAQ_GPIO      IOChannel,
    short               pullUp,
    short             * value

)
```

This function is used to configure the general-purpose I/Os as digital inputs.

Arguments:	handle: handle returned from UsbDrDaqOpenUnit [11] or UsbDrDaqOpenUnitProgress [23] IOChannel: All GPIOs can be used for digital inputs. pullUp: Used to specify whether pull-up resistor is used. value: A location where the driver writes a value indicating the state of the input (0 or 1)
Returns:	PICO_OK [37] PICO_NOT_FOUND [37] PICO_NOT_RESPONDING [37] PICO_INVALID_PARAMETER [37]

3.5.23   UsbDrDaqStartPulseCount

```
PICO_STATUS UsbDrDaqStartPulseCount(
  short               handle,
  USB_DRDAQ_GPIO      IOChannel,
  short               direction

  )
```

This function is used to configure the general-purpose I/Os for pulse counting and to start counting.

Arguments:	`handle:` handle returned from UsbDrDaqOpenUnit [11] or UsbDrDaqOpenUnitProgress [23]
	`IOChannel:` GPIOs 1 and 2 can be used as pulse-counting inputs.
	`direction:` The direction of the edges to count (0: rising, 1: falling)
Returns:	PICO_OK [37] PICO_NOT_FOUND [37] PICO_NOT_RESPONDING [37] PICO_INVALID_PARAMETER [37]

3.5.24   UsbDrDaqGetPulseCount

```
PICO_STATUS UsbDrDaqGetPulseCount(
  short              handle,
  USB_DRDAQ_GPIO     IOChannel,
  short            * count

)
```

This function will return the current pulse count. It should be called after pulse counting has been started using UsbDrDaqStartPulseCount[32].

Arguments:	handle: handle returned from UsbDrDaqOpenUnit[11] or UsbDrDaqOpenUnitProgress[23] IOChannel: GPIO of interest. count: A location where the driver will write the current count.
Returns:	PICO_OK[37] PICO_NOT_FOUND[37] PICO_NOT_RESPONDING[37] PICO_INVALID_PARAMETER[37]

3.5.25   UsbDrDaqEnableRGBLED

```
PICO_STATUS UsbDrDaqEnableRGBLED(
    short              handle,
    short              enabled
)
```

This function enables or disables RGB mode on the LED.

Arguments:	handle: handle returned from UsbDrDaqOpenUnit[11] or UsbDrDaqOpenUnitProgress[23] enabled: If non-zero, RGB mode is enabled. If zero RGB mode is disabled and the LED returns to normal operation (flashing when sampling).
Returns:	PICO_OK[37] PICO_NOT_FOUND[37] PICO_NOT_RESPONDING[37]

3.5.26   UsbDrDaqSetRGBLED

```
PICO_STATUS UsbDrDaqSetRGBLED(

    short  handle,
    unsigned short    red,
    unsigned short    green,
    unsigned short    blue

)
```

This function is used to set the colour of the LED once RGB mode has been enabled using <u>USBDRDaqEnableRGBLED</u> 34 .

Arguments:	`handle:` handle returned from <u>UsbDrDaqOpenUnit</u> 11 or <u>UsbDrDaqOpenUnitProgress</u> 23 `red, green, blue:` Components of the requred LED colour, in the range 0 to 255.
Returns:	<u>PICO_OK</u> 37 <u>PICO_NOT_FOUND</u> 37 <u>PICO_NOT_RESPONDING</u> 37

3.5.27   UsbDrDaqGetChannelInfo

```
PICO_STATUS UsbDrDaqGetChannelInfo(
  short               handle,
  float             * min,
  float             * max,
  short             * places,
  short             * divider,
  USB_DRDAQ_INPUTS    channel
)
```

This procedure returns a set of information about the currently selected scaling for the specified channel. If a parameter is not required, you can pass a null pointer to the routine.

Arguments:	handle: handle returned from UsbDrDaqOpenUnit 11 or UsbDrDaqOpenUnitProgress 23
	min: the minimum value that the channel can take
	max: the maximum value that the channel can take
	places: the number of decimal places
	divider: the number that values should be divided by, to give real numbers
	channel: the channel to return details for
Returns:	PICO_OK 37
	PICO_NOT_FOUND 37
	PICO_INVALID_PARAMETER 37

3.5.28 PICO_STATUS values

Every function in the USB DrDAQ API returns a status code from the following list of `PICO_STATUS` values:

Code (hex)	Enum	Description
00	PICO_OK	The Data Logger is functioning correctly
01	PICO_MAX_UNITS_OPENED	An attempt has been made to open more than `UsbDrDaq_MAX_UNITS`
02	PICO_MEMORY_FAIL	Not enough memory could be allocated on the host machine
03	PICO_NOT_FOUND	No USB DrDAQ device could be found
04	PICO_FW_FAIL	Unable to download firmware
05	PICO_OPEN_OPERATION_IN_PROGRESS	A request to open a device is in progress
06	PICO_OPERATION_FAILED	The operation was unsuccessful
07	PICO_NOT_RESPONDING	The device is not responding to commands from the PC
08	PICO_CONFIG_FAIL	The configuration information in the device has become corrupt or is missing
09	PICO_KERNEL_DRIVER_TOO_OLD	The `picopp.sys` file is too old to be used with the device driver
0A	PICO_EEPROM_CORRUPT	The EEPROM has become corrupt, so the device will use a default setting
0B	PICO_OS_NOT_SUPPORTED	The operating system on the PC is not supported by this driver
0C	PICO_INVALID_HANDLE	There is no device with the handle value passed
0D	PICO_INVALID_PARAMETER	A parameter value is not valid
0E	PICO_INVALID_TIMEBASE	The time base is not supported or is invalid
0F	PICO_INVALID_VOLTAGE_RANGE	The voltage range is not supported or is invalid
10	PICO_INVALID_CHANNEL	The channel number is not valid on this device or no channels have been set
11	PICO_INVALID_TRIGGER_CHANNEL	The channel set for a trigger is not available on this device
12	PICO_INVALID_CONDITION_CHANNEL	The channel set for a condition is not available on this device
13	PICO_NO_SIGNAL_GENERATOR	The device does not have a signal generator
14	PICO_STREAMING_FAILED	Streaming has failed to start or has stopped without user request
15	PICO_BLOCK_MODE_FAILED	Block failed to start - a parameter may have been set wrongly
16	PICO_NULL_PARAMETER	A parameter that was required is `NULL`
18	PICO_DATA_NOT_AVAILABLE	No data is available from a run block call
19	PICO_STRING_BUFFER_TOO_SMALL	The buffer passed for the information was too small
1A	PICO_ETS_NOT_SUPPORTED	ETS is not supported on this device
1B	PICO_AUTO_TRIGGER_TIME_TOO_SHORT	The auto trigger time is less than the time it will take to collect the data
1C	PICO_BUFFER_STALL	The collection of data has stalled as unread data would be overwritten
1D	PICO_TOO_MANY_SAMPLES	The number of samples requested is more than available in the current memory segment
1E	PICO_TOO_MANY_SEGMENTS	Not possible to create number of segments requested
1F	PICO_PULSE_WIDTH_QUALIFIER	A null pointer has been passed in the trigger function or one of the parameters is out of range

20	`PICO_DELAY`	One or more of the hold-off parameters are out of range
21	`PICO_SOURCE_DETAILS`	One or more of the source details are incorrect
22	`PICO_CONDITIONS`	One or more of the conditions are incorrect
23	`PICO_USER_CALLBACK`	The driver's thread is currently in the [UsbDrDaqReady] 15 callback function and therefore the action cannot be carried out
24	`PICO_DEVICE_SAMPLING`	An attempt is being made to get stored data while streaming. Either stop streaming by calling [UsbDrDaqStop] 16
25	`PICO_NO_SAMPLES_AVAILABLE`	…because a run has not been completed
26	`PICO_SEGMENT_OUT_OF_RANGE`	The memory index is out of range
27	`PICO_BUSY`	Data cannot be returned yet
28	`PICO_STARTINDEX_INVALID`	The start time to get stored data is out of range
29	`PICO_INVALID_INFO`	The information number requested is not a valid number
2A	`PICO_INFO_UNAVAILABLE`	The handle is invalid so no information is available about the device. Only `PICO_DRIVER_VERSION` is available.
2B	`PICO_INVALID_SAMPLE_INTERVAL`	The sample interval selected for streaming is out of range
2C	`PICO_TRIGGER_ERROR`	Not used
2D	`PICO_MEMORY`	Driver cannot allocate memory
36	`PICO_DELAY_NULL`	`NULL` pointer passed as delay parameter
37	`PICO_INVALID_BUFFER`	The buffers for overview data have not been set while streaming
3A	`PICO_CANCELLED`	A block collection has been cancelled
3B	`PICO_SEGMENT_NOT_USED`	The segment index is not currently being used
3C	`PICO_INVALID_CALL`	The wrong [GetValues] 19 function has been called for the collection mode in use
3F	`PICO_NOT_USED`	The function is not available
41	`PICO_INVALID_STATE`	Device is in an invalid state
43	`PICO_DRIVER_FUNCTION`	You called a driver function while another driver function was still being processed

3.6 Programming support

3.6.1 Introduction

We supply examples for the following programming languages:

3.6.2 C and C++

C

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

`usbdrdaqcon.c`
and
`usbdrdaq.lib` (Microsoft Visual C 32-bit applications).
or
`usbdrdaqbc.lib` (Borland 32-bit applications)

The following files must be in the compilation directory:
`usbdrdaqapi.h`
`picostatus.h`

The following file must be in the same directory as the executable, or in the search path.
`usbdrdaq.dll`

C++

C++ programs can access all versions of the driver. If `ubsdrdaqapi.h` is included in a C++ program, the `PREF1` macro expands to `extern "C"`: this disables name-decoration, and enables C++ routines to make calls to the driver routines using C headers.

3.6.3 Excel

The easiest way to transfer data to Excel is to use PicoLog. However, you can also write an Excel macro which calls `ubsdrdaq.dll` to read in a set of data values. The Excel macro language is similar to Visual Basic.

The example `ubsdrdaq.xls` reads in 20 values from Channels 1 and 2, one per second, and assigns them to cells A1..B20.

Use Excel Version 7 or higher.

Note that it is usually necessary to copy the DLL file to the `\windows\system` directory.

3.6.4   LabVIEW

The SDK contains a library of VIs that can be used to control the USB DrDAQ and two examples of using these VIs.

`DrDAQBlockExample.vi` is a simple block mode example that demonstrates using block mode on a single channel with a trigger.

`DrDAQStreamingExample.vi` demonstrates streaming mode acquisition on all channels with triggering. It also demonstrates use of the general-purpose I/Os, the arbitrary waveform generator, channel scaling and the RGB LED.

The LabVIEW library (UsbDrDaq.llb) can be placed in the user.lib subdirectory to make the VIs available on the 'User Libraries' palette. You should also copy UsbDrDaq.dll to the folder containing your LabView project.

The library contains the following VIs:

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

`PicoStatus.vi` - checks the status value returned by calls to the driver If the driver returns an error, the status member of the error cluster is set to 'true' and the error code and source are set.

`UsbDrDaqChannelScaling.vi` – is used to discover available scales for a given channel, the scale that is currently being used and to change the scale.

`UsbDrDaqClose.vi` – Closes the USB DrDAQ.

`UsbDrDaqGetBlock.vi` – is used to get a block of data from the USB DrDAQ. The method can be either single or windowed and the VI returns the trigger index and the values. The acquisition settings must be specified first using `UsbDrDaqSettings.vi`.

`UsbDrDaqGetStreamingData.vi` – is used to get samples once streaming has been started (using `UsbDrDaqStartStreaming.vi`). The size of the buffer created by this VI must be specified. This should be large enough to contain all the samples returned when the VI is called. The VI returns the number of values collected, the trigger index and the values themselves.

`UsbDrDaqGPIO.vi` – is used to control the general-purpose I/Os. GPIOs 1 & 2 can be used as digital inputs, digital outputs, PWM outputs and pulse-counting inputs. GPIOs 3 & 4 can be used as digital inputs and digital outputs.

`UsbDrDaqLEDControl.vi` – can be used to enable and control the RGB LED.

`UsbDrDaqOpen.vi` – Opens a USB DrDAQ and returns a handle to the device.

`UsbDrDaqSettings.vi` – is used to set up data acquisition and the trigger and should be called before starting streaming or block-mode collection.

`UsbDrDaqSigGen.vi` – is used to control the signal generator. There are a set of standard waveforms that can be selected and "arbitrary waveform" can also be selected. When selecting an arbitrary waveform, an array of values and the number of values should be specified. The update rate can also be selected. Selecting "none" under waveform stops the signal generator.

`UsbDrDaqStartStreaming.vi` – starts the device streaming data after the USB DrDAQ has been set up using `UsbDrDaqSettings.vi`. The "number of values" input is used by the driver to create its buffer. Once this VI has been called, values can be obtained from the driver using `UsbDrDaqGetStreamingData.vi`.

4   Glossary

Analog bandwidth. The input frequency at which the measured signal amplitude is 3 decibels below the true signal amplitude.

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.

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

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

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

MS. Megasamples (1,000,000 samples).

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

PicoScope software. A software product that accompanies all PicoScope oscilloscopes. It turns your PC into an oscilloscope, spectrum analyzer and multimeter.

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

USB 2.0. Universal Serial Bus. This is a standard port used to connect external devices to PCs. The port supports a data transfer rate of up to 480 megabits per second, so is much faster than the RS-232 COM ports found on older PCs.

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 stated in the Specifications table in the User's Guide.

Index

6

A

B

C

D

E

G

I

L

N

O

P

R

Repair   1
Running a unit   14

S

Safety warning   1
Sampling interval, setting   17
Scaling   25
Scaling files   4
Signal Generator   26, 28
Software updates   4
Stopping a unit   16
Streaming   8

T

Trigger, setting   18

U

Unit information, obtaining   13

W

Windows XP/Vista/Windows 7
    support   7
WoW64   7

Pico Technology

James House
Colmworth Business Park
ST. NEOTS
Cambridgeshire
PE19 8YP
United Kingdom
Tel: +44 (0) 1480 396 395
Fax: +44 (0) 1480 396 296
<%WEBADDRESS%>