USB DrDAQ®

Data Logger

Programmer's Guide

usbdrdaqpg.en r9


pico
Technology

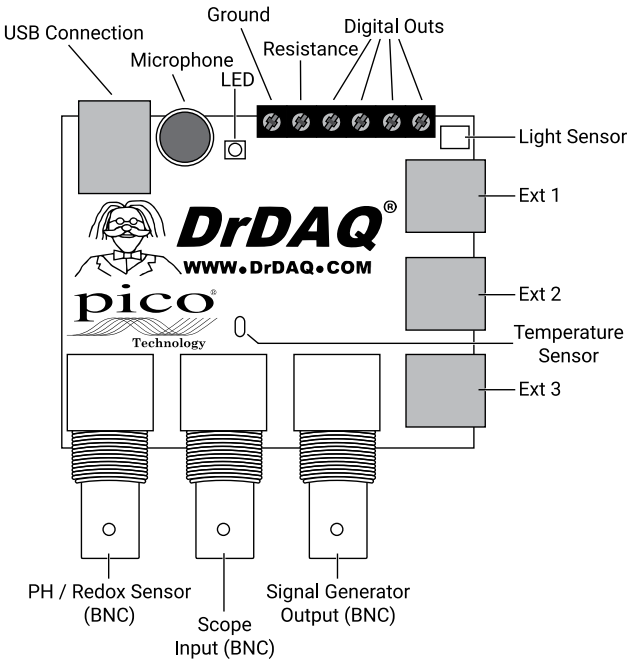
Contents

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 32-bit and 64-bit editions of Microsoft Windows 7, 8 and 10.



1.2 License agreement

Grant of license. The material contained in this release is licensed, not sold. Pico Technology Limited ("Pico") 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 and agree to abide by these conditions.

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

Copyright. The software in this release is for use only with Pico products or with data collected using Pico products. You may copy and distribute the SDK without restriction, as long as you do not remove any Pico Technology copyright statements. The example programs in the SDK may be modified, copied and distributed for the purpose of developing programs to collect data using Pico products.

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

Fitness for purpose. No two applications are the same, so Pico cannot guarantee that its equipment or software is suitable for a given application. It is therefore the user's responsibility to ensure that the product is suitable for the user's application.

Mission-critical applications. Because the software runs on a computer that may be running other software products, and may be subject to interference from these other products, this license specifically excludes usage in "mission-critical" applications, for example life-support systems.

Viruses. This software was continuously monitored for viruses during production. However, the user is responsible for virus checking the software once it is installed.

Support. No software is ever error-free, but if you are dissatisfied with the performance of this software, please contact our technical support staff.

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

1.3 Trademarks

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

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

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

1.4 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. PicoScope can check for updates automatically and advise you if an update is available. You can download the latest versions of the software free of charge from the Pico Technology web site at:

www.picotech.com/downloads

Alternatively, the latest software can be purchased on disk from Pico Technology.

To be kept up-to-date with news of new software releases, click here to join our e-mail mailing list.

2 Writing your own software

2.1 About the driver

USB DrDAQ is supplied with a kernel driver and a library, `UsbDrDaq.dll`, containing routines that you can build into your own programs. The driver is supported by Microsoft Windows 7, 8 and 10 and is supplied in 32-bit and 64-bit versions.

The PicoSDK containing the drivers can be downloaded from www.picotech.com/downloads.

The driver supports up to 64 units at one time.

2.2 Installing the driver

The drivers are supplied with the USB DrDAQ SDK. You can download the latest 32-bit and 64-bit versions of the SDK from our website at:

www.picotech.com/downloads

Click **PicoLog Data Loggers > DrDAQ > Software > PicoSDK**

2.3 Example code

Example code in various programming languages is available from the "picotech" organization on GitHub.

2.4 Connecting the logger

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

To connect the data logger, plug the cable provided into any available USB port on your PC. The first time you connect the unit, some versions of Windows may 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.

2.5 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. The maximum block size is 16,384 samples, shared between all active channels.

`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. If `UsbDrDaqGetValuesF()` is called, floating point numbers will be returned instead of integer values for `UsbDrDaqGetValues()`.

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() is called, it returns the new readings since the last time it was called. The noOfValues argument passed to UsbDrDaqRun() must be sufficient to ensure that the buffer does not overflow between successive calls to UsbDrDaqGetValues(). For example, if you call UsbDrDaqGetValues() every second and you are collecting 500 samples per second, noOfValues must be at least 500, or preferably 1000, to allow for delays in the operating system. Always call UsbDrDaqStop() after a streaming mode capture to prepare the device for the next capture.

2.6 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 (where *"scaling"* can be replaced with a name of your choice). This file will contain the details of your sensor.

The driver can return values in either integer or floating-point form. If you are calling UsbDrDaqGetValuesF() floating point numbers will be returned. Calling UsbDrDaqGetValues() will return integer values. In both forms the values may require scaling to move the decimal point. For example, the driver returns a pH of 7.65 as 765. You can call UsbDrDaqGetChannelInfo() to find out how many decimal places a channel is using, and also to get a divider that converts the raw value to the scaled value. 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() to get a list of valid scaling codes, then call UsbDrDaqSetScalings() to select one of them. Once selected, UsbDrDaqGetChannelInfo() will return full information about the selected scaling. If you do not use UsbDrDaqSetScalings(), 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 same directory as usbdrdaq.dll on Windows platforms. For Linux platforms, the .DDS files can be placed in the Home directory or in a location that is added to a SCALING_FILE_PATH environment variable. 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 kilohms. 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:

Channel	Resistor value
Sound waveform	1200
Sound level	1300
Voltage	1500
Resistance	1600
pH	1400
Temperature	1100
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 USB DrDAQ API functions

The driver functions supplied with the USB DrDAQ data logger are listed in the following sections.

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 selected.

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

```
Check that the driver version is correct
Open the unit
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 unit
Unload the driver// happens automatically when you terminate the application
```


3.1 UsbDrDaqCloseUnit - close the unit

```
PICO_STATUS UsbDrDaqCloseUnit
(
  int16_t            handle
)
```

This function closes the unit.

Arguments
handle: device identifier returned from UsbDrDaqOpenUnit() or UsbDrDaqOpenUnitProgress().

Returns
PICO_OK
PICO_HANDLE_INVALID

3.2 UsbDrDaqEnableRGBLED - enable or disable RGB mode on the LED

```
PICO_STATUS UsbDrDaqEnableRGBLED
(
  int16_t            handle,
  int16_t            enabled
)
```

This function enables or disables RGB mode on the LED.

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`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`
`PICO_NOT_FOUND`
`PICO_NOT_RESPONDING`

3.3 UsbDrDaqGetChannelInfo - return information about the scaling in use on a channel

```
PICO_STATUS UsbDrDaqGetChannelInfo
(
  int16_t            handle,
  float            * min,
  float            * max,
  int16_t          * places,
  int16_t          * 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. You can obtain a list of available scalings by calling UsbDrDaqGetScalings().

```
// Obtain scaling divider and apply it to sampled data
  float scaled_values[ARRAY_SIZE];

  status = UsbDrDaqGetValues(g_handle, values, &noOfValues, &overflow,
    &triggerIndex);
  status = UsbDrDaqGetChannelInfo(g_handle, &min, &max, &places, &divider,
    channel);

  for (i = 0; i < noOfValues; i++)
     scaled_values[i] = values[i] / (float)divider;
```

Arguments

handle: device identifier returned from UsbDrDaqOpenUnit() or UsbDrDaqOpenUnitProgress().

min: on exit, the minimum value that the channel can take.

max: on exit, the maximum value that the channel can take.

places: on exit, the number of decimal places.

divider: on exit, the number that raw values should be divided by to give scaled numbers.

channel: the channel to return details for. See Channel numbers.

Returns

PICO_OK
PICO_NOT_FOUND
PICO_INVALID_PARAMETER

3.4 UsbDrDaqGetInput - use the general-purpose I/Os as digital inputs

```
PICO_STATUS UsbDrDaqGetInput
(
  int16_t            handle,
  USB_DRDAQ_GPIO     IOChannel,
  int16_t            pullUp,
  int16_t            * value
)
```

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

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`IOChannel`: identifies which of the GPIO inputs to read. See Channel numbers.

`pullUp`: specifies whether or not to connect an internal pull-up resistor to the input:
- `0` = don't connect – input will be in an undefined state when not connected to an external voltage source.
- `1` = connect – input will be `1` if not connected to an external source.

`value`: on exit, indicates the state of the input:
- `0` = logic low
- `1` = logic high

Returns

`PICO_OK`
`PICO_NOT_FOUND`
`PICO_NOT_RESPONDING`
`PICO_INVALID_PARAMETER`

3.5 UsbDrDaqGetPulseCount - return the current pulse count

```
PICO_STATUS UsbDrDaqGetPulseCount
(
  int16_t            handle,
  USB_DRDAQ_GPIO     IOChannel,
  int16_t          * count
)
```

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

Arguments

handle: device identifier returned from UsbDrDaqOpenUnit() or UsbDrDaqOpenUnitProgress().

IOChannel: which GPIO to use. See Channel numbers.

count: on exit, contains the current count.

Returns

PICO_OK
PICO_NOT_FOUND
PICO_NOT_RESPONDING
PICO_INVALID_PARAMETER

3.6 UsbDrDaqGetScalings - discover the scalings, both built-in and custom, that are available

```
PICO_STATUS UsbDrDaqGetScalings
(
  int16_t            handle
  USB_DRDAQ_INPUTS   channel,
  int16_t          * nScales,
  int16_t          * currentScale,
  int8_t           * names,
  int16_t            namesSize
)
```

This function discovers the scalings, both built-in and custom, that are available for a particular channel. Having chosen a scaling number, you can pass it to `UsbDrDaqSetScalings()` to apply the scaling to the selected channel.

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`channel`: the channel number.

`nScales`: output. The function writes the number of available scales here.

`currentScale`: output. An index to the currently selected scale here.

`names`: output. A string containing the scaling names and indices.

`namesSize`: the size of names.

Returns

`PICO_OK`
`PICO_NOT_FOUND`
`PICO_INVALID_CHANNEL`

3.7 UsbDrDaqGetSingle - get a single value from a specified channel

```
PICO_STATUS UsbDrDaqGetSingle
(
  int16_t            handle,
  USB_DRDAQ_INPUTS   channel,
  int16_t          * value,
  uint16_t         * overflow
)
```

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

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`channel`: which channel to sample. See Channel numbers.

`value`: on exit, the sample value.

> Note: With certain scalings, such as resistance in kilohms, the integer returned in `value` may be unable to represent the full resolution of the device. In this case, use `UsbDrDaqGetSingleF()` instead to obtain a floating-point 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`
`PICO_INVALID_HANDLE`
`PICO_NO_SAMPLES_AVAILABLE`
`PICO_DEVICE_SAMPLING`
`PICO_NULL_PARAMETER`
`PICO_INVALID_PARAMETER`
`PICO_DATA_NOT_AVAILABLE`
`PICO_INVALID_CALL`
`PICO_NOT_RESPONDING`
`PICO_MEMORY`

3.8 UsbDrDaqGetSingleF - get a single floating-point value

```
PICO_STATUS UsbDrDaqGetSingleF
(
  int16_t            handle,
  USB_DRDAQ_INPUTS   channel,
  float            * value,
  uint16_t         * overflow
)
```

This function returns a single floating-point sample value from the specified input channel. In all other respects it is the same as UsbDrDaqGetSingle().

3.9 UsbDrDaqGetTriggerTimeOffsetNs - return jitter-correction parameter

```
PICO_STATUS UsbDrDaqGetTriggerTimeOffsetNs
(
  int16_t            handle,
  int64_t          * time
)
```

This function returns the time between the trigger point and the first post-trigger sample. This is calculated using linear interpolation. The value may be used to reduce the apparent jitter of the signal with respect to the trigger.

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`time`: on exit, trigger time in nanoseconds.

Returns

`PICO_OK`
`PICO_NOT_FOUND`

3.10 UsbDrDaqGetUnitInfo - return information about the unit

```
PICO_STATUS UsbDrDaqGetUnitInfo
(
  int16_t            handle,
  int8_t           * string,
  int16_t            stringLength,
  int16_t          * 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, call the function with `string = NULL` first.

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`string`: location of a buffer 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`: on exit, the length of the information string before it was truncated to `stringLength`. If the string was not truncated, `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`
- `PICO_FIRMWARE_VERSION_1` - firmware number
- `PICO_FIRMWARE_VERSION_2` - same as `PICO_FIRMWARE_VERSION_1`

Returns

`PICO_OK`
`PICO_INVALID_HANDLE`
`PICO_NULL_PARAMETER`
`PICO_INVALID_INFO`
`PICO_INFO_UNAVAILABLE`

3.11 UsbDrDaqGetValues - get sample values after a run

```
PICO_STATUS UsbDrDaqGetValues
(
  int16_t            handle,
  int16_t          * values,
  uint32_t         * noOfValues,
  uint16_t         * overflow,
  uint32_t         * triggerIndex
)
```

This function is used to get values after calling UsbDrDaqRun().

> Note: Please refer to USB DrDAQ scaling files (.DDS) because you will need to convert the data values using the divider for that channel.

Arguments

handle: device identifier returned from UsbDrDaqOpenUnit() or UsbDrDaqOpenUnitProgress().

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 1: The order of the channels will be as stated in Channel numbers, regardless of the order used in the UsbDrDaqSetInterval() channels array.

> Note 2: With certain scalings, such as resistance in kilohms, the integers returned in the values array may be unable to represent the full resolution of the device. In this case, use UsbDrDaqGetValuesF() instead to obtain floating-point values.

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
PICO_INVALID_HANDLE
PICO_NO_SAMPLES_AVAILABLE
PICO_DEVICE_SAMPLING
PICO_NULL_PARAMETER
PICO_INVALID_PARAMETER
PICO_TOO_MANY_SAMPLES
PICO_DATA_NOT_AVAILABLE
PICO_INVALID_CALL
PICO_NOT_RESPONDING
PICO_MEMORY

3.12 UsbDrDaqGetValuesF - get floating-point values after a run

```
PICO_STATUS UsbDrDaqGetValuesF
(
  int16_t            handle,
  float            * values,
  uint32_t         * noOfValues,
  uint16_t         * overflow,
  uint32_t         * triggerIndex
)
```

This function is used to get floating-point values after calling UsbDrDaqRun(). In all other respects it is the same as UsbDrDaqGetValues().

Note: Please refer to USB DrDAQ scaling files (.DDS) because you will need to convert the data values using the divider for that channel.

3.13 UsbDrDaqOpenUnit - open and enumerate the unit

```
PICO_STATUS UsbDrDaqOpenUnit
(
  int16_t          * handle
)
```

This function opens and enumerates the unit.

Arguments
`handle`: on exit, a value 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`
`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`

3.14 UsbDrDaqOpenUnitAsync - open the unit without waiting for completion

```
PICO_STATUS UsbDrDaqOpenUnitAsync
(
  int16_t          * 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() 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: on exit, a status flag:

- 0 if there is already an open operation in progress
- 1 if the open operation is initiated

Returns

PICO_OK
PICO_OPEN_OPERATION_IN_PROGRESS
PICO_OPERATION_FAILED

3.15 UsbDrDaqOpenUnitProgress - report progress of UsbDrDaqOpenUnitAsync

```
PICO_STATUS UsbDrDaqOpenUnitProgress
(
  int16_t          * handle,
  int16_t          * progress,
  int16_t          * complete
)
```

This function checks on the progress of UsbDrDaqOpenUnitAsync().

Arguments

`* handle`: on exit, the device identifier 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`: on exit, an estimate of the progress towards opening the data logger. The value is between 0 to 100.

`complete`: on exit, a non-zero value if the operation has completed, otherwise zero.

Returns

`PICO_OK`
`PICO_NULL_PARAMETER`
`PICO_OPERATION_FAILED`

3.16 UsbDrDaqPhTemperatureCompensation - select pH temperature compensation

```
PICO_STATUS UsbDrDaqPhTemperatureCompensation
(
  int16_t            handle,
  uint16_t           enabled
)
```

This function specifies whether or not to use the built-in thermistor to temperature-compensate the pH input. If the thermistor is at the same temperature as the pH sensor, this will produce more accurate pH readings.

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`enabled`: `1` to enable, `0` to disable.

Returns

`PICO_OK`
`PICO_NULL_PARAMETER`
`PICO_OPERATION_FAILED`

3.17 UsbDrDaqPingUnit - check that a device is connected

```
PICO_STATUS UsbDrDaqPingUnit
(
  int16_t          * handle
)
```

This function checks that the specified USB DrDAQ is connected.

Arguments
`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

Returns
`PICO_OK`
`PICO_NOT_RESPONDING`
`PICO_BUSY`
`PICO_DRIVER_FUNCTION`
`PICO_NOT_FOUND`

3.18 UsbDrDaqReady - indicate when UsbDrDaqRun has captured data

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

This function indicates when UsbDrDaqRun() has captured the requested number of samples.

Arguments
handle: device identifier returned from UsbDrDaqOpenUnit() or UsbDrDaqOpenUnitProgress().

ready: nonzero if ready, zero otherwise.

Returns
PICO_OK
PICO_INVALID_HANDLE
PICO_NOT_RESPONDING

3.19 UsbDrDaqRun - tell unit to start capturing data

```
PICO_STATUS UsbDrDaqRun
(
  int16_t            handle,
  uint32_t           no_of_values,
  BLOCK_METHOD       method
)
```

This function tells the unit to start capturing data.

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`no_of_values`: the number of samples that the unit should collect per channel.

At 1 MS/s in `BM_SINGLE` mode, the total for all active channels must not exceed 16 320 samples. At lower sampling rates, the total for all active channels is limited to 1 000 010 samples.

`method`: which method to use to collect the data, from the following list:

`BM_SINGLE`
`BM_WINDOW`
`BM_STREAM`

See Capture modes for details.

Returns

`PICO_OK`
`PICO_INVALID_HANDLE`
`PICO_USER_CALLBACK`
`PICO_INVALID_CHANNEL`
`PICO_TOO_MANY_SAMPLES`
`PICO_INVALID_TIMEBASE`
`PICO_NOT_RESPONDING`
`PICO_CONFIG_FAIL`
`PICO_INVALID_PARAMETER`
`PICO_NOT_RESPONDING`
`PICO_TRIGGER_ERROR`

3.20 UsbDrDaqSetDO - control the digital outputs

```
PICO_STATUS UsbDrDaqSetDO
(
  int16_t            handle,
  USB_DRDAQ_GPIO     IOChannel,
  int16_t            value
)
```

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

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`IOChannel`: identifies the channel. See Channel numbers.

`value`: any non-zero value sets the digital output and zero clears it.

Returns

`PICO_OK`
`PICO_NOT_FOUND`
`PICO_NOT_RESPONDING`
`PICO_INVALID_PARAMETER`

3.21 UsbDrDaqSetInterval - set sampling speed of the unit (integer)

```
PICO_STATUS UsbDrDaqSetInterval
(
  int16_t            handle,
  uint32_t           * us_for_block,
  uint32_t           ideal_no_of_samples,
  USB_DRDAQ_INPUTS   * channels,
  int16_t            no_of_channels
)
```

This function sets the sampling interval of the unit. Sampling of multiple channels is sequential.

The minimum possible sampling interval (`si_min`, in microseconds) depends on the capture mode and number of active channels (`n`) as follows:

- `BM_SINGLE` mode:
  `si_min = n`
- `BM_WINDOW` and `BM_STREAM` modes:
  `si_min = 10*n`

If you wish to know the effective sampling interval (`si`, in microseconds) set by this function, you can calculate it as follows:

```
si = us_for_block / (ideal_no_of_samples * no_of_channels)
```

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`us_for_block`: on entry, the target total time in which to collect `ideal_no_of_samples * no_of_channels`, in microseconds. On exit, the actual total time that was set.

For more accurate setting of total time as a floating-point value, use `UsbDrDaqSetIntervalF()`.

`ideal_no_of_samples`: the number of samples per channel that you want to collect. This number is used only for timing calculations.

At 1 MS/s in `BM_SINGLE` mode, the total for all active channels must not exceed 16 320 samples.
At lower sampling rates, the total for all active channels is limited to 1 000 010 samples.

`channels`: an array of constants identifying the channels from which you wish to capture data. See the list at Channel numbers. If you specify the channels in a different order from that shown in that list, the function will reorder them.

`no_of_channels`: the number of channels in the `channels` array.

Returns

`PICO_OK`
`PICO_INVALID_HANDLE`
`PICO_INVALID_CHANNEL`
`PICO_INVALID_TIMEBASE`
`PICO_NOT_RESPONDING`
`PICO_CONFIG_FAIL`
`PICO_INVALID_PARAMETER`
`PICO_NOT_RESPONDING`

PICO_TRIGGER_ERROR

3.22 UsbDrDaqSetIntervalF - set sampling speed of the unit (floating-point)

```
PICO_STATUS UsbDrDaqSetIntervalF
(
  int16_t            handle,
  float            * us_for_block,
  uint32_t           ideal_no_of_samples,
  USB_DRDAQ_INPUTS * channels,
  int16_t            no_of_channels
)
```

This function sets the sampling interval of the unit. It works in the same way as UsbDrDaqSetInterval() except that the `us_for_block` argument is a `float` instead of an integer.

3.23 UsbDrDaqSetPWM - configure general-purpose I/Os as pulse-width modulation outputs

```
PICO_STATUS UsbDrDaqSetPWM
(
  int16_t            handle,
  USB_DRDAQ_GPIO     IOChannel,
  uint16_t           period,
  uint8_t            cycle
)
```

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

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`IOChannel`: GPIOs 1 and 2 can be used as PWM outputs. See `UsbDrDaqSetDO()` for values.

`period`: the period of the waveform in microseconds.

`cycle`: duty cycle as a percentage.

Returns

`PICO_OK`
`PICO_NOT_FOUND`
`PICO_NOT_RESPONDING`
`PICO_INVALID_PARAMETER`

3.24 UsbDrDaqSetRGBLED - set color of LED in RGB mode

```
PICO_STATUS UsbDrDaqSetRGBLED
(
  int16_t            handle,
  uint16_t           red,
  uint16_t           green,
  uint16_t           blue
)
```

This function is used to set the color of the LED once RGB mode has been enabled using USBDRDaqEnableRGBLED().

Arguments

handle: device identifier returned from UsbDrDaqOpenUnit() or UsbDrDaqOpenUnitProgress().

red, green, blue: components of the required LED color, in the range 0 to 255.

Returns

PICO_OK
PICO_NOT_FOUND
PICO_NOT_RESPONDING

3.25 UsbDrDaqSetScalings - set the scaling for a channel

```
PICO_STATUS UsbDrDaqSetScalings
(
  int16_t            handle
  USB_DRDAQ_INPUTS   channel,
  int16_t            scalingNumber
)
```

This function sets the scaling for a specified channel. Having set a scaling, you can verify it by calling UsbDrDaqGetChannelInfo().

Arguments

handle: device identifier returned from UsbDrDaqOpenUnit() or UsbDrDaqOpenUnitProgress().

channel: the channel to set. See Channel numbers.

scalingNumber: the number of the required scale, as given by UsbDrDaqGetScalings().

Returns

PICO_OK
PICO_NOT_FOUND
PICO_INVALID_CHANNEL
PICO_INVALID_PARAMETER

3.26 UsbDrDaqSetSigGenArbitrary - control the arbitrary waveform generator

```
PICO_STATUS UsbDrDaqSetSigGenArbitrary
(
  int16_t            handle,
  int32_t            offsetVoltage,
  uint32_t           pkToPk,
  int16_t          * arbitraryWaveform,
  int16_t            arbitraryWaveformSize,
  int32_t            updateRate
)
```

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

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`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`: 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`: the rate at which the AWG steps through the points in the waveform. This value must be in the range 1 to 2,000,000 points per second.

Returns

`PICO_OK`
`PICO_NOT_FOUND`
`PICO_NOT_RESPONDING`
`PICO_INVALID_PARAMETER`

3.27 UsbDrDaqSetSigGenBuiltIn - set the arbitrary waveform generator using standard waveform types

```
PICO_STATUS UsbDrDaqSetSigGenBuiltIn
(
  int16_t            handle,
  int32_t            offsetVoltage,
  uint32_t           pkToPk,
  int16_t            frequency,
  USB_DRDAQ_WAVE     waveType
)
```

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

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`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`
`PICO_NOT_FOUND`
`PICO_NOT_RESPONDING`
`PICO_INVALID_PARAMETER`

3.28 UsbDrDaqSetTrigger - set up the trigger

```
PICO_STATUS UsbDrDaqSetTrigger
(
  int16_t            handle,
  uint16_t           enabled,
  uint16_t           auto_trigger,
  uint16_t           auto_ms,
  uint16_t           channel,
  uint16_t           dir,
  int16_t            threshold,
  uint16_t           hysteresis,
  float              delay
)
```

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

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

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

`auto_trigger`: whether or not to rearm 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. See Channel numbers.

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

`threshold`: trigger threshold (the level at which the trigger will activate) in the currently selected scaling, multiplied to remove any decimal places. The number of decimal places can be found by calling `UsbDrDAQGetChannelInfo()`.

`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`
`PICO_INVALID_HANDLE`
`PICO_USER_CALLBACK`
`PICO_TRIGGER_ERROR`
`PICO_MEMORY_FAIL`

3.29 UsbDrDaqStartPulseCount - use general-purpose I/Os for pulse counting

```
PICO_STATUS UsbDrDaqStartPulseCount
(
  int16_t            handle,
  USB_DRDAQ_GPIO     IOChannel,
  int16_t            direction
)
```

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

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

`IOChannel`: specifies the GPIO channel to use, either GPIO 1 or GPIO 2. See Channel numbers.

`direction`: the direction of the edges to count (0: rising, 1: falling).

Returns

`PICO_OK`
`PICO_NOT_FOUND`
`PICO_NOT_RESPONDING`
`PICO_INVALID_PARAMETER`

3.30 UsbDrDaqStop - abort data collection

```
PICO_STATUS UsbDrDaqStop
(
  int16_t            handle
)
```

This function aborts data collection. It is normally used in streaming mode to stop capturing data. It can also be used to interrupt a block mode capture, but in this case all the data will be invalid.

Arguments

`handle`: device identifier returned from `UsbDrDaqOpenUnit()` or `UsbDrDaqOpenUnitProgress()`.

Returns

`PICO_OK`
`PICO_INVALID_HANDLE`

3.31 UsbDrDaqStopSigGen - turn AWG off

```
PICO_STATUS UsbDrDaqStopSigGen
(
  int16_t            handle
)
```

This function turns the AWG off.

Arguments
handle: device identifier returned from UsbDrDaqOpenUnit() or UsbDrDaqOpenUnitProgress().

Returns
PICO_OK
PICO_NOT_FOUND
PICO_NOT_RESPONDING

3.32 Channel numbers

Use the following values for the channel argument in those API functions that deal with a specified input channel or channels:

```
typedef enum enUsbDrDaqInputs
{
  USB_DRDAQ_CHANNEL_EXT1 = 1,  //Ext. sensor 1            1
  USB_DRDAQ_CHANNEL_EXT2,      //Ext. sensor 2            2
  USB_DRDAQ_CHANNEL_EXT3,      //Ext. sensor 3            3
  USB_DRDAQ_CHANNEL_SCOPE,     //Scope channel            4
  USB_DRDAQ_CHANNEL_PH,        //pH                       5
  USB_DRDAQ_CHANNEL_RES,       //Resistance               6
  USB_DRDAQ_CHANNEL_LIGHT,     //Light                    7
  USB_DRDAQ_CHANNEL_TEMP,      //Thermistor               8
  USB_DRDAQ_CHANNEL_MIC_WAVE,  //Microphone waveform      9
  USB_DRDAQ_CHANNEL_MIC_LEVEL, //Microphone level        10
  USB_DRDAQ_MAX_CHANNELS = USB_DRDAQ_CHANNEL_MIC_LEVEL
} USB_DRDAQ_INPUTS;
```

Use the following values for the IOChannel argument in the API functions that deal with a specified GPIO channel:

```
typedef enum enUsbDrDaqDO
{
   USB_DRDAQ_GPIO_1 = 1,        //                       1
   USB_DRDAQ_GPIO_2,            //                       2
   USB_DRDAQ_GPIO_3,            //                       3
   USB_DRDAQ_GPIO_4             //                       4
} USB_DRDAQ_GPIO;
```

Source: usbDrDaqApi.h 2013-01-22

3.33 PICO_STATUS values

Every function in the USB DrDAQ API returns a status code from the list of PICO_STATUS values in PicoStatus.h.

4 Glossary

Driver. A program that controls a piece of hardware. The driver for the USB DrDAQ is supplied in the form of 32-bit and 64-bit versions of a Windows DLL, UsbDrDaq.dll. This is used by your application to control the data logger.

Sampling interval. The time interval between samples as the USB DrDAQ acquires data. The sampling interval can be set to any value returned by UsbDrDaqSetInterval() and UsbDrDaqSetIntervalF().

USB 2.0. Universal Serial Bus. This is a standard port that enables you to connect external devices to PCs. A full-speed USB 2.0 port operates at up to 480 megabits per second. The PicoLog 1000 Series is also compatible with any USB port from USB 1.1 upwards.

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.


pico
Technology

Index

S

T

U

W

UK 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

USA regional office:

Pico Technology
320 N Glenwood Blvd
Tyler
Texas 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.china@picotech.com

usbdrdaqpg.en r9 2019-11-18