# ADC-20/ADC-24

## High-Resolution Data Loggers

## Programmer's Guide

adc20.en r1

# Contents


1 Overview ... 1

2 Notices ... 2

- 1 Legal information ... 2
- 2 Trademarks ... 2

3 Getting started ... 3

- 1 Installing the software ... 3
- 2 Connecting the data logger ... 3

4 Concepts ... 4

- 1 Recording methods ... 4
- 2 Windows driver ... 4
- 3 Scaling ... 4

5 Driver functions ... 5

- 1 HRDLCloseUnit – shuts down unit ... 6
- 2 HRDLCollectSingleValueAsync – sample a single value, non-blocking ... 7
- 3 HRDLGetMinMaxAdcCounts – return the maximum and minimum ADC count ... 8
- 4 HRDLGetNumberOfEnabledChannels – return the number of analog channels enabled ... 9
- 5 HRDLGetSingleValue – take one sample for the specified channel ... 10
- 6 HRDLGetSingleValueAsync – retrieves reading after call to HRDLCollectSingleValueAsync() ... 11
- 7 HRDLGetTimesAndValues – return time-stamped samples ... 13
- 8 HRDLGetUnitInfo – returns unit information as character string ... 14
- 9 HRDLGetValues – return samples for each enabled channel ... 16
- 10 HRDLOpenUnit – open a data logger ... 17
- 11 HRDLOpenUnitAsync – open a unit without blocking the calling thread ... 18
- 12 HRDLOpenUnitProgress – check progress of an asynchronous open operation ... 19
- 13 HRDLReady – find out if readings are ready to be collected ... 20
- 14 HRDLRun – start sampling ... 21
- 15 HRDLSetAnalogInChannel – enable or disable an analog channel ... 22
- 16 HRDLSetDigitalIOChannel – set a digital output or input (ADC-24 only) ... 23
- 17 HRDLSetInterval – set the sampling time interval ... 25
- 18 HRDLSetMains – set the mains noise rejection frequency ... 26
- 19 HRDLStop – stop the device when streaming ... 27

6 Sequence of calls and data flow ... 28

- 1 Streaming recording methods ... 28
  - 1 Collecting a block of data ... 28
  - 2 Collecting windowed or streaming data ... 29
- 2 Single-value recording methods ... 30
  - 1 Collecting a single reading, blocking ... 30
  - 2 Collecting a single reading, non-blocking ... 30

7 Glossary ... 31



*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.*

# 1 Overview

The ADC-20 and ADC-24 High-Resolution Data Loggers are multichannel, high-accuracy USB data loggers for use with PCs. They require no external power supply.

We provide 32-bit and 64-bit Windows drivers to allow you to control the data loggers from your own software. These drivers are included in the PicoSDK package, which you can download from www.picotech.com/downloads.

Example code in a variety of programming languages can be downloaded from the "picotech" organization on GitHub.

The hardware and software are compatible with Microsoft Windows 7, 8 and 10.

*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.*

# 2 Notices

## 2.1 Legal information

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

**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 usage in mission-critical applications, such as 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.

## 2.2 Trademarks

**Pico Technology Limited** and **PicoLog** 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.


*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.*

# 3 Getting started

## 3.1 Installing the software

Before you connect the ADC-20 or ADC-24 to your computer for the first time, you must install the driver using PicoSDK. You can download 32-bit and 64-bit versions of PicoSDK from www.picotech.com/downloads.

## 3.2 Connecting the data logger

When you have installed the driver, connect the data logger's USB cable to a spare USB port on your computer and wait until Windows displays the message "Device is ready to use".

Copyright © 2005–2019 Pico Technology Ltd. All rights reserved. 

# 4 Concepts

## 4.1 Recording methods

The ADC-20/ADC-24 driver provides three methods of recording data. All these methods support USB 1.1 and later.

- Streaming. The driver constantly polls the device, and samples are placed in a buffer until retrieved by your application. Precise sample timing is controlled by the unit.
- Single value (blocking). You make a single request for a sample, blocking the calling thread, and when the sample has been received the driver returns the value to your application.
- Single value (non-blocking). You make a single request for a sample without blocking the calling thread, and when the sample has been received the driver returns the value to your application.

## 4.2 Windows driver

The `picohrdl.dll` dynamic link library (DLL) in the `lib` subdirectory of your SDK installation is a driver that allows you to program your ADC-20 or ADC-24 data logger. It is supplied in 32-bit and 64-bit versions. The driver exports the function definitions in standard C format, but this does not limit you to programming in C. You can use the API with any programming language that supports standard C calls. It can also be used with programs like Microsoft Excel. The driver supports Windows 7, 8 and 10.

## 4.3 Scaling

To convert from ADC values to volts, first obtain the minimum and maximum ADC values for the selected channel by calling `HRDLGetMinMaxAdcCounts()` in the driver. Next, scale the ADC value to the voltage range you specified when you called `HRDLSetAnalogInChannel()`. You can calculate the voltage range programmatically by using

**Vmax = 2500 mV / (2^r)**

where **r** is the range constant you supplied to `HRDLSetAnalogInChannel()` (0 for ±2500 mV, 1 for ±1250 mV and so on).

You can then use Vmax to calculate the scaled voltage, V, with the following formula

**V = (raw_ADC_value / max_ADC_Value) * Vmax**

where raw_ADC_value is the reading from the device and max_ADC_value is the max ADC value for the device obtained from `HRDLGetMinMaxAdcCounts()`.


*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.*

# 5 Driver functions

The following sections describe the functions available to an application using the ADC-20 and ADC-24. All functions are C functions using the standard call naming convention (`__stdcall`) and are exported with both decorated and undecorated names.

Copyright © 2005–2019 Pico Technology Ltd. All rights reserved. 

## 5.1 HRDLCloseUnit – shuts down unit

```
int16_t HRDLCloseUnit
(
    int16_t handle
)
```

Shuts down an ADC-20 or ADC-24 device.

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

**Returns**

1 if a valid handle is passed
0 if not


*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.*

## 5.2 HRDLCollectSingleValueAsync – sample a single value, non-blocking

```
int16_t HRDLCollectSingleValueAsync
(
  int16_t handle,
  int16_t channel,
  int16_t range,
  int16_t conversionTime,
  int16_t singleEnded
)
```

This function starts the unit sampling one value without blocking the calling application's flow. Used in conjunction with `HRDLGetSingleValueAsync()` and `HRDLReady()`.

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

`channel`, Channel number to convert. If the channel is not valid then the function will fail.

`range`, The voltage range to be used. If the range is not valid, the function `HRDLGetSingleValueAsync()` will return 0.

`conversionTime`, The time interval in which the sample should be converted. If the conversion time is invalid, the function `HRDLGetSingleValueAsync()` will fail and return 0.

`singleEnded`, The type of voltage to be measured:
0: differential
<>0: single-ended

**Returns**

1 if a valid handle is passed and the settings are correct
0 if not

Copyright © 2005–2019 Pico Technology Ltd. All rights reserved. 

## 5.3 HRDLGetMinMaxAdcCounts – return the maximum and minimum ADC count

```
int16_t HRDLGetMinMaxAdcCounts
(
  int16_t   handle,
  int32_t * minAdc,
  int32_t * maxAdc,
  int16_t   channel
)
```

This function returns the maximum and minimum ADC count available for the device referenced by handle.

**Arguments**

- `handle`, device identifier returned by `HRDLOpenUnit()`
- `minAdc`, Pointer to an `int32_t`, used to return the minimum ADC count available for the unit referred to by handle
- `maxAdc`, Pointer to an `int32_t`, used to return the maximum ADC count available for the unit referred to by handle
- `channel`, Channel number for which maximum and minimum ADC count are required

**Returns**

| | |
|---|---|
| 1 | if a valid handle is passed |
| 0 | if not |

 *Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.*

## 5.4 HRDLGetNumberOfEnabledChannels – return the number of analog channels enabled

```
int16_t HRDLGetNumberOfEnabledChannels
(
  int16_t   handle,
  int16_t * nEnabledChannels
)
```

This function returns the number of analog channels enabled.

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

`nEnabledChannels`, pointer to an `int16_t` where the number of channels enabled will be written

**Returns**

1 if a valid handle is passed
0 if not

Copyright © 2005–2019 Pico Technology Ltd. All rights reserved. 

## 5.5 HRDLGetSingleValue – take one sample for the specified channel

```
int16_t HRDLGetSingleValue
(
  int16_t   handle,
  int16_t   channel,
  int16_t   range,
  int16_t   conversionTime,
  int16_t   singleEnded,
  int16_t * overflow
  int32_t * value
)
```

This function takes one sample for the specified channel at the selected voltage range and conversion time.

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

`channel`, The channel number to convert.
ADC-20: 1 to 8
ADC-24: 1 to 16

If the channel is not valid then the function will fail and return 0.

`range`, the voltage range to be used. See `HRDLSetAnalogInChannel()` for possible values. If the range is not valid, the function will return 0.

`conversionTime`, The time interval in which the sample should be converted. See `HRDLSetInterval()` for possible values. If the conversion time is invalid, the function will fail and return 0.

`singleEnded`, The type of voltage to be measured.
0: differential
<>0: single-ended

`overflow`, pointer to a bit field that indicates when the voltage on a channel has exceeded the upper or lower limits.
Bit 0: Channel 1
...
Bit 15: Channel 16

`value`, pointer to an `int32_t` where the ADC value will be written.

**Returns**

| | |
|---|---|
| 1 | if a valid handle is passed and settings are correct |
| 0 | if not |

If the function fails, call `HRDLGetUnitInfo()` with `info = HRDL_ERROR (7)` to obtain the error code. If the error code is `HRDL_SETTINGS (5)`, then call `HRDLGetUnitInfo()` again with `info = HRDL_SETTINGS_ERROR (8)` to determine the settings error.


Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.

## 5.6 HRDLGetSingleValueAsync – retrieves reading after call to HRDLCollectSingleValueAsync()

```
int16_t HRDLGetSingleValueAsync
(
  int16_t   handle,
  int32_t * value,
  int16_t * overflow
)
```

This function retrieves the reading when `HRDLCollectSingleValueAsync()` has been called.

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

`value`, pointer to an `int32_t` where the ADC value will be written

`overflow`, pointer to a value that indicates when the voltage on a channel has exceeded the upper or lower limits.
Bit 0: Channel 1
...
Bit 15: Channel 16

**Returns**

1 if a valid handle is passed and the function succeeds
0 if not

**Sample code**

Code extract to get a single value reading without blocking the calling thread:

```
void main()
{
  BOOL    bConversionFinished = FALSE;
  int16_t channelNo;
  int32_t value;
  int16_t handle;

  // Open and initialize the unit
      ...

  // Set the channel parameters
  channelNo = HRDL_ANALOG_IN_CHANNEL_1;
  range = HRDL_2500_MV;
  singleEnded = TRUE;
  bConversionFinished = FALSE;

  while (true)
  {
    PollSingleValue(handle,
                    &bConversionFinished,
                    &value,
                    channelNo,
                    range,
```

Copyright © 2005–2019 Pico Technology Ltd. All rights reserved. 

```
                        singleEnded);

        if (bConversionFinished == TRUE)
        {
          // Do something with the value
          channelNo++;
          // This would be HRDL_ANALOG_IN_CHANNEL_8 for the ADC-20
          if (channelNo > HRDL_ANALOG_IN_CHANNEL_16)
          {
            channelNo = HRDL_ANALOG_IN_CHANNEL_1;
          }
        }
        else
        {
          // Do something else while waiting for the reading from the unit
        }
      }
    }

    void PollSingleValue(int16_t handle,
                         BOOL *bConversionFinished,
                         int32_t *lValue,
                         int16_t channel,
                         int16_t range,
                         int16_t singleEnded)
    {
      static BOOL bStartConversion = FALSE;
      int16_t       overflow;

      // Test to see if the conversion has finished
      if (bStartedConversion)
      {
        if (HRDLReady(handle))
        {
          HRDLGetSingleValueAsync(handle, lValue, &overflow);
          bConversionFinished = TRUE;
          bConversionStarted = FALSE;
        }
      }

      // Test to see if no conversion is in progress
      if (!bStartedConversion)
      {
        // Start the conversion going
        bStartedConversion = HRDLCollectSingleValueAsync(handle,
                                                         channel,
                                                         range,
                                                         conversionTime,
                                                         singleEnded);
        bConversionFinished = TRUE;
      }
    }
```

 Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.

## 5.7 HRDLGetTimesAndValues – return time-stamped samples

```
int32_t HRDLGetTimesAndValues
(
  int16_t   handle,
  int32_t * times,
  int32_t * values,
  int16_t * overflow,
  int32_t   noOfValues
)
```

This function returns the requested number of samples for each enabled channel and the times when the samples were taken, so the `values` array needs to be (number of values) x (number of enabled channels). When one or more of the digital IOs are enabled as inputs, they count as one additional channel. The function informs the user if the voltages for any of the enabled channels have overflowed.

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

`times,` pointer to an `int32_t` where times will be written

`values,` pointer to an `int32_t` where sample values will be written. If more than one channel is active, the samples are interleaved. If digital channels are enabled then they are always the first values. See table below for the order in which data are returned.

`overflow,` pointer to an `int16_t` indicating any inputs that have exceeded their maximum voltage range. Channels with overvoltages are indicated by a high bit, with the LSB indicating channel 1 and the MSB channel 16.

`noOfValues,` the number of samples to collect for each active channel.

**Returns**

A non-zero number if successful indicating the number of values returned,

0 if the call failed or no values available

**Ordering of returned data (example)**

When two analog channels (e.g. 1 and 5) are enabled and a digital channel is set as an input, the data are returned in the following order:

```
Sample No: 0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 . n-3 n-2 n-1
Channel:   DI 1  5  DI 1  5  DI 1  5  DI 1  5  DI 1  5  . DI  1   5
```

where n represents the value returned by the function and DI the digital inputs.

The channels are always ordered from channel 1 up to the maximum channel number (ADC-24: channel 16, ADC-20: channel 8). If one or more digital channels are set as inputs then the first sample contains the digital channels.

**Digital inputs**

The digital channels are represented by a binary bit pattern with 0 representing off, and 1 representing on. Digital input 1 is in bit 0.

Copyright © 2005–2019 Pico Technology Ltd. All rights reserved. 

## 5.8 HRDLGetUnitInfo – returns unit information as character string

```
int16_t HRDLGetUnitInfo
(
  int16_t   handle,
  int8_t  * string,
  int16_t   stringLength,
  int16_t   info
)
```

This function writes information about the ADC-20 or ADC-24 device to a character string. If the logger fails to open, only `info = HRDL_ERROR (7)` is available to explain why the last open unit call failed. When retrieving the driver version, the handle value is ignored.

**Arguments**

`handle,` identifier of the device from which information is required. If an invalid handle is passed, the error code from the last unit that failed to open is returned (as if `info = HRDL_ERROR`), unless `info = HRDL_DRIVER_VERSION` and then the driver version is returned.

`string,` pointer to the `int8_t` string buffer in the calling function where the unit information string (selected with `info`) will be stored. If a null pointer is passed, no information will be written.

`stringLength,` Length of the `int8_t` string buffer. If the string is not long enough to accept all of the information, only the first `stringLength` characters are returned.

`info,` Enumerated type (listed below) specifying what information is required from the driver.

**Returns**

The length of the string written to the `int8_t` string buffer, `string,` by the function.

If one of the parameters is out of range, or a null pointer is passed for `string,` the function will return zero.


Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.

**Values of `info`**

| Value of `info` | Description | Example |
|---|---|---|
| HRDL_DRIVER_VERSION (0) | The version of picohrdl.dll | 1.0.0.1 |
| HRDL_USB_VERSION (1) | The type of USB to which the unit is connected | 1.1 |
| HRDL_HARDWARE_VERSION (2) | The hardware version of the HRDL attached | 1 |
| HRDL_VARIANT_INFO (3) | Information about the type of HRDL attached | 24 |
| HRDL_BATCH_AND_SERIAL (4) | Batch and serial numbers of the unit | CMY02/116 |
| HRDL_CAL_DATE (5) | Calibration date of the unit | 09Sep05 |
| HRDL_KERNEL_DRIVER_VERSION (6) | Kernel driver version | |
| HRDL_ERROR (7) | One of the error codes listed in **Error codes** below | 4 |
| HRDL_SETTINGS (8) | One of the error codes listed in **Settings Error Codes** below | |

**Error codes (when `info = HRDL_ERROR`)**

| Error code | Description |
|---|---|
| HRDL_OK (0) | The unit is functioning correctly |
| HRDL_KERNEL_DRIVER (1) | The picopp.sys file is to old to support this product |
| HRDL_NOT_FOUND (2) | No data logger could be found |
| HRDL_CONFIG_FAIL (3) | Unable to download firmware |
| HRDL_ERROR_OS_NOT_SUPPORTED (4) | The operating system is not supported by this device |
| HRDL_MAX_DEVICES (5) | The maximum number of units allowed are already open |

**Settings Error Codes (when `info = HRDL_SETTINGS`)**

| Settings Error Code | Description |
|---|---|
| SE_CONVERSION_TIME_OUT_OF_RANGE (0) | The conversion time parameter is out of range |
| SE_SAMPLEINTERVAL_OUT_OF_RANGE (1) | The sample time interval is out of range |
| SE_CONVERSION_TIME_TOO_SLOW (2) | The conversion time chosen is not fast enough to convert all channels within the sample interval |
| SE_CHANNEL_NOT_AVAILABLE (3) | The channel being set is valid but not currently available |
| SE_INVALID_CHANNEL (4) | The channel being set is not valid for this device |
| SE_INVALID_VOLTAGE_RANGE (5) | The voltage range being set for this device is not valid |
| SE_INVALID_PARAMETER (6) | One or more parameters are invalid |
| SE_CONVERSION_IN_PROGRESS (7) | A conversion is in progress for a single asynchronous operation |
| SE_COMMUNICATION_FAILED (8) | The PC has lost communication with the device |
| SE_OK (9) | All settings have been completed successfully |

*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.* 

## 5.9 HRDLGetValues – return samples for each enabled channel

```
int32_t HRDLGetValues
(
  int16_t   handle,
  int32_t * values,
  int16_t * overflow,
  int32_t   noOfValues
)
```

This function returns the requested number of samples for each enabled channel, so the size of the `values` array needs to be (number of values) x (number of enabled channels). When one or more of the digital IOs are enabled as inputs, they count as one additional channel. The function informs the user if the voltages of any of the enabled channels have overflowed.

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

`values,` pointer to an `int32_t` where the sample values are written. If more than one channel is active, the samples are interleaved. If digital channels are enabled then they are always the first value. See table below for the order in which data are returned.

`overflow,` pointer to an `int16_t` indicating any inputs that have exceeded their maximum voltage range. Channels with overvoltages are indicated by a high bit, with the LSB indicating channel 1 and the MSB channel 16.

`noOfValues,` the number of samples to collect for each active channel.

**Returns**

A non-zero number if successful indicating the number of values returned, or

0 if the call failed or no values available

**Ordering of returned data (example)**

When two analog channels (such as 1 and 5) are enabled and a digital channel is set as an input, the data are returned in the following order.

```
Sample No: 0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 . n-3 n-2 n-1
Channel:   DI 1  5  DI 1  5  DI 1  5  DI 1  5  DI 1  5  . DI  1   5
```

where `n` represents the total number of values returned by the function and `DI` the digital inputs.

The channels are always ordered from channel 1 up to the maximum channel number (ADC-24: channel 16, ADC-20: channel 8). If one or more digital channels are set as inputs, and digital inputs are enabled by calling `HRDLSetDigitalIOChannel()` with `enabledDigitalIn=1,` the first sample in each group contains the digital channels.

**Digital inputs**

The digital channels are represented by a binary bit pattern with 0 representing off and 1 representing on. Digital input 1 is in bit 0.


Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.

## 5.10 HRDLOpenUnit – open a data logger

```
int16_t HRDLOpenUnit
(
  void
)
```

This function opens an ADC-20 or ADC-24 device. The API driver can support up to 20 units.

**Arguments**

None

**Returns**

| | |
|---|---|
| −1 | if the unit fails to open |
| 0 | if no unit is found |
| ≥ 1 | handle to the device opened |

*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.* 

## 5.11 HRDLOpenUnitAsync – open a unit without blocking the calling thread

```
int16_t HRDLOpenUnitAsync
(
  void
)
```

Opens an ADC-20 or ADC-24 device without blocking the calling thread.

**Arguments**

None

**Returns**

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

 *Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.*

## 5.12 HRDLOpenUnitProgress – check progress of an asynchronous open operation

```
int16_t HRDLOpenUnitProgress
(
  int16_t * handle,
  int16_t * progress
)
```

Checks the progress of an asynchronous open operation.

**Arguments**

`handle,` pointer to an `int16_t` where the unit handle is to be written:

- −1: if the unit fails to open
- 0: if no unit is found
- >0: a handle to the device opened (this handle is not valid unless the function returns true)

`progress,` pointer to an `int16_t` to which the percentage progress is to be written. 100% implies that the open operation is complete.

**Returns**

| | |
|---|---|
| 0 | if open operation is still in progress |
| 1 | if the open operation is complete |

Copyright © 2005–2019 Pico Technology Ltd. All rights reserved. 

## 5.13 HRDLReady – find out if readings are ready to be collected

```
int16_t HRDLReady
(
  int16_t handle
)
```

This function indicates when the readings are ready to be retrieved from the driver.

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

**Returns**

0 if not ready, or failed
1 if ready


*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.*

## 5.14 HRDLRun – start sampling

```
int16_t HRDLRun
(
  int16_t handle,
  int32_t nValues,
  int16_t method
)
```

This function starts the device sampling and storing the samples in the driver's buffer. See Streaming recording methods for help on using this function.

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

`nValues`, number of samples to collect for each active channel.

`method`, sampling method. This should be one of the values listed below.

**Returns**

0 if failed,
1 if successful

**Sampling methods**

| Value of `method` | Description |
|---|---|
| `BM_BLOCK (0)` | Collect a single block and stop |
| `BM_WINDOW (1)` | Collect a sequence of overlapping blocks |
| `BM_STREAM (2)` | Collect a continuous stream of data |

Copyright © 2005–2019 Pico Technology Ltd. All rights reserved. 

## 5.15 HRDLSetAnalogInChannel – enable or disable an analog channel

```
int16_t HRDLSetAnalogInChannel
(
   int16_t handle,
   int16_t channel,
   int16_t enabled,
   int16_t range,
   int16_t singleEnded
)
```

This function enables or disables the selected analog channel. If you wish to enable an odd-numbered channel in differential mode, you must first make sure that its corresponding even-numbered channel is disabled. (For example, to set channel 1 to differential mode, first ensure that channel 2 is disabled).

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

`channel`, the channel that will be enabled or disabled.
ADC-20: 1 to 8
ADC-24: 1 to 16

`enabled`, sets the channel active or dormant:
0: dormant
<> 0: active

`range`, The voltage range to be used during sampling. Applies only to selected channel. See **Voltage ranges** below.

`singleEnded`, non-zero to measure a single-ended voltage. Zero for a differential voltage.

**Returns**

0 if failed
1 if successful

If the function fails, call `HRDLGetUnitInfo()` with `info = HRDL_SETTINGS_ERROR (8)` to obtain the specific settings error.

**Voltage ranges**

| Value of `range` | Voltage range | Availability |
|---|---|---|
| `HRDL_2500_MV (0)` | ±2500 mV | ADC-20 and ADC-24 |
| `HRDL_1250_MV (1)` | ±1250 mV | ADC-20 and ADC-24 |
| `HRDL_625_MV (2)` | ±625 mV | ADC-24 only |
| `HRDL_313_MV (3)` | ±312.5 mV | ADC-24 only |
| `HRDL_156_MV (4)` | ±156.25 mV | ADC-24 only |
| `HRDL_78_MV (5)` | ±78.125 mV | ADC-24 only |
| `HRDL_39_MV (6)` | ±39.0625 mV | ADC-24 only |


Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.

## 5.16 HRDLSetDigitalIOChannel – set a digital output or input (ADC-24 only)

```
int16_t HRDLSetDigitalIOChannel
(
  int16_t handle,
  int16_t directionOut,
  int16_t digitalOutPinState,
  int16_t enabledDigitalIn
)
```

Configures the digital input/output channels of the ADC-24. If the direction is "output" then the pin can be driven high or low. While the device is sampling, the direction cannot be changed but the state can.

**Applicability**

ADC-24 only

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

`directionOut`, the directions of the digital IO pins. Add up the `HRDL_DIGITAL_IO_CHANNEL` constants (see below) for the pins that you want to be outputs. Any pins not configured as outputs will become inputs.

`digitalOutPinState`, the states of the digital outputs. Add up the `HRDL_DIGITAL_IO_CHANNEL` constants (see below) for the pins that you want to be high. Any pins not defined as high will be driven low.

`enabledDigitalIn`, Sets the digital input as active. Use a combination of `HRDL_DIGITAL_IO_CHANNEL` constants (see below). The ordering of returned data is described under `HRDLGetValues()`.

**Returns**

0 if failed,
1 if successful

If the function fails, call `HRDLGetUnitInfo()` with `info = HRDL_SETTINGS_ERROR (8)` to obtain the specific setting error.

**Pin values for `directionOut` and `digitalOutPinState`:**

| Pin value | Description |
|---|---|
| `HRDL_DIGITAL_IO_CHANNEL_1 (1)` | IO Pin 1 |
| `HRDL_DIGITAL_IO_CHANNEL_2 (2)` | IO Pin 2 |
| `HRDL_DIGITAL_IO_CHANNEL_3 (4)` | IO Pin 3 |
| `HRDL_DIGITAL_IO_CHANNEL_4 (8)` | IO Pin 4 |

Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.

**Examples:**

- To configure digital channels 1 and 2 as inputs and digital channels 3 and 4 as outputs:
  `directionOut = HRDL_DIGITAL_IO_CHANNEL_3 (4) + HRDL_DIGITAL_IO_CHANNEL_4 (8) = 12`
- To drive digital channel 4 high and digital channel 3 low:
  `digitalOutPinState = HRDL_DIGITAL_IO_CHANNEL_4 (8) = 8`
- To drive only digital channel 3 high:
  `digitalOutPinState = HRDL_DIGITAL_IO_CHANNEL_3 (4) = 4`
- To drive both digital channels 3 and 4 high:
  `digitalOutPinState = HRDL_DIGITAL_IO_CHANNEL_3 (4) + HRDL_DIGITAL_IO_CHANNEL_4 (8) = 12`

**Example bit patterns for `directionOut` parameter:**

| Decimal | Bit Pattern | Digital Channel 4 | Digital Channel 3 | Digital Channel 2 | Digital Channel 1 |
|---|---|---|---|---|---|
| 1 | 0001 | Input | Input | Input | Output |
| 10 | 1010 | Output | Input | Output | Input |
| 12 | 1100 | Output | Output | Input | Input |
| 13 | 1101 | Output | Output | Input | Output |

The above is a selection of the 16 different options available for the `directionOut` parameter. When a digital channel has been configured as an output, it can then be driven high or low with the `digitalOutputPinState` parameter, again using bit patterns to represent the different digital channels.

The default setting for the digital channels is "output, low".


*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.*

## 5.17 HRDLSetInterval – set the sampling time interval

```
int16_t HRDLSetInterval
(
  int16_t handle,
  int32_t sampleInterval_ms,
  int16_t conversionTime
)
```

This sets the sampling time interval. The number of channels active must be able to convert within the specified interval.

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

`sampleInterval_ms`, time interval in milliseconds within which all conversions must take place before the next set of conversions starts.

`conversionTime`, the amount of time given to one channel's conversion. This must be one of the constants below.

**Returns**

0 if failed
1 if successful

If the function fails, call `HRDLGetUnitInfo()` with `info = HRDL_SETTINGS_ERRORS` for the specific settings error.

**Conversion times**

| Value of `conversionTime` | Conversion time |
|---|---|
| `HRDL_60MS (0)` | 60 ms |
| `HRDL_100MS (1)` | 100 ms |
| `HRDL_180MS (2)` | 180 ms |
| `HRDL_340MS (3)` | 340 ms |
| `HRDL_660MS (4)` | 660 ms |

Copyright © 2005–2019 Pico Technology Ltd. All rights reserved. 

## 5.18 HRDLSetMains – set the mains noise rejection frequency

```
int16_t HRDLSetMains
(
  int16_t handle,
  int16_t sixtyHertz
)
```

This function configures the mains noise rejection setting. Rejection takes effect the next time sampling occurs.

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

`sixtyHertz`, specifies whether 50 Hz or 60 Hz noise rejection is applied:
0: reject 50 Hz
<> 0: reject 60 Hz

**Returns**

0 if failed
1 if successful


*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.*

## 5.19 HRDLStop – stop the device when streaming

```
void HRDLStop
(
  int16_t handle
)
```

This function stops the device from sampling data.

When running the device in windowed or streaming mode, you will need to call this function to end data collection. This is particularly important in streaming mode, to ensure that the scope is ready for the next capture.

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

**Arguments**

`handle`, device identifier returned by `HRDLOpenUnit()`

*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.* 

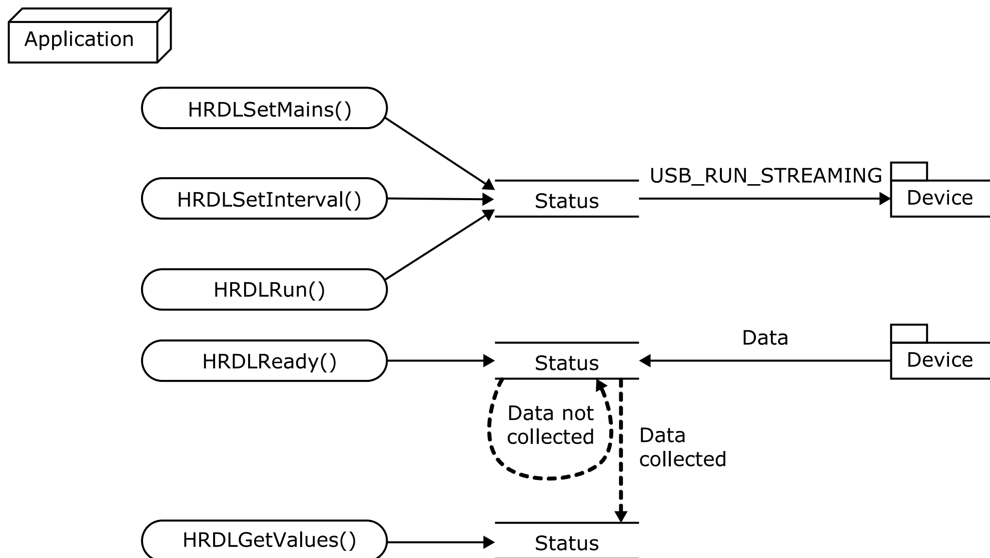
# 6 Sequence of calls and data flow

The C sample program `picohrdlCon.c` demonstrates the use of all the functions of the API driver, and includes examples showing each mode of operation. It is available from the *picohrdl* subdirectory in the picotech/picosdk-c-examples repository on GitHub.

## 6.1 Streaming recording methods

### 6.1.1 Collecting a block of data

- This method collects a single block of data and then stops.
- Open the data logger with one of the `HRDLOpenUnit()` calls
- Set mains noise rejection with `HRDLSetMains()`
- Set the analog or/and digital channels
- Set the sample interval with `HRDLSetInterval()`
- Start the unit collecting samples by calling `HRDLRun()` with `method = BM_BLOCK`
- Loop
- Repeat Loop until ready (`HRDLReady()`)
- Collect data with `HRDLGetValues()`
- Repeat from "Start the unit" until you have finished collecting data.
- Close the connection to the unit with `HRDLCloseUnit()`



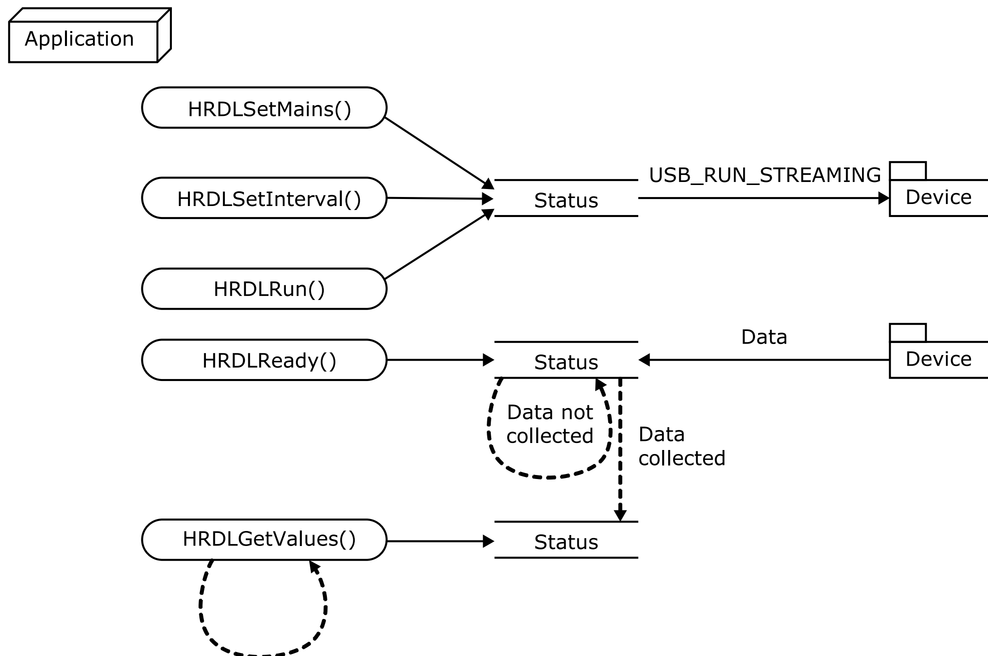


Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.

### 6.1.2 Collecting windowed or streaming data

This method causes the device to start sampling. Samples are stored in the driver's buffer. In windowed mode, the buffer will always contain the requested number of samples, but generally only a subset of these are new data. In streaming mode, new data are returned continuously.

- Open the data logger with one of the `HRDLOpenUnit()` calls
- Set mains noise rejection with `HRDLSetMains()`
- Set the analog or/and digital channels
- Set the sample interval with `HRDLSetInterval()`
- Start the unit collecting samples by calling `HRDLRun()` with `method = BM_WINDOW` or `BM_STREAM`
- Loop
- Repeat Loop until ready (`HRDLReady()`)
- Collect data whenever you want with `HRDLGetValues()`
- Call `HRDLStop()` to end data collection
- Close the connection to the unit with `HRDLCloseUnit()`



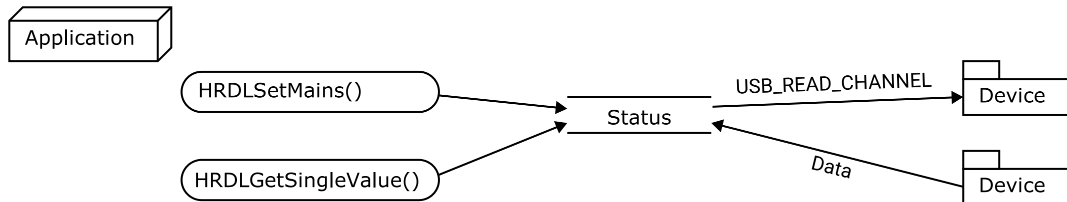

*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.*

# 6.2 Single-value recording methods

## 6.2.1 Collecting a single reading, blocking

This method collects a single reading and blocks the calling thread.

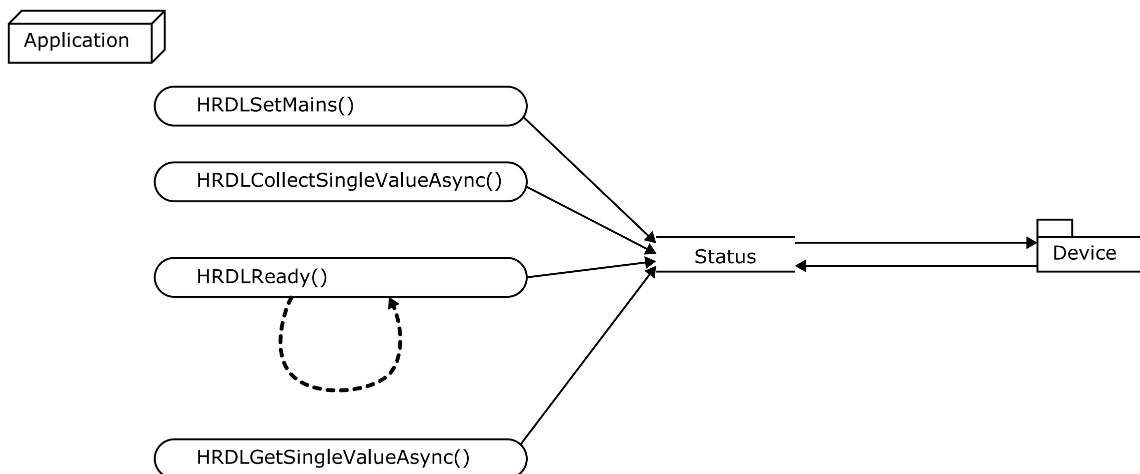
- Open the data logger with one of the `HRDLOpenUnit()` calls
- Set mains noise rejection with `HRDLSetMains()`
- Get a single reading (one channel only at a time) with `HRDLGetSingleValue()`
- Close the connection to the unit with `HRDLCloseUnit()`




## 6.2.2 Collecting a single reading, non-blocking

This method collects a single reading without blocking the calling thread.

- Open the data logger with one of the `HRDLOpenUnit()` calls
- Set mains noise rejection with `HRDLSetMains()`
- Start the conversion for a single reading with `HRDLCollectSingleValueAsync()`
- Wait until the reading is ready (`HRDLReady()`)
- Get the reading from the driver with `HRDLGetSingleValueAsync()`
- Close the connection to the unit with `HRDLCloseUnit()`




 Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.

# 7 Glossary

**Asynchronous.** In asynchronous data collection, your application requests data from the driver, and the driver immediately returns without blocking the application. The application must then poll a status function until the data is ready.

**Data logger.** A measuring instrument that monitors one or more analog signals, samples them at pre-programmed intervals, then accurately converts the samples to digital data and stores them in memory. The ADC-20 and ADC-24 use your PC for storage and display.

**DLL.** Dynamic Link Library. A DLL is a file containing a collection of Windows functions designed to perform a specific class of operations.

**Driver.** A driver is a computer program that acts as an interface, generally between a hardware component and a computer system, the hardware in this case being the data logger.

**LSB.** Least significant bit. In a binary word, the least significant bit has the value 1.

**MSB.** Most significant bit. In an *n*-bit binary word, the most significant bit has the value $2^{(n-1)}$.

**Noise rejection.** The ability of the data logger to attenuate noise in a given frequency range. The ADC-20/ADC-24 can be programmed to reject noise at either 50 hertz or 60 hertz. The noise rejection ratio is defined as:

**NRR(dB) = 20 log10 (Vin/Vmeas)**

where NRR(dB) is the noise rejection ratio in decibels, Vin is the noise voltage at the input, and Vmeas is the noise voltage that appears in the measurement.

**USB.** 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 compatible with any USB port from USB 1.1 upwards.

*Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.* 

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 of America

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

adc20.en r1 2019-06-11
Copyright © 2005–2019 Pico Technology Ltd. All rights reserved.