USB TC-08

Thermocouple Data Logger

Programmer's Guide

usbtc08pg r7

Contents

1 Introduction

1.1 Overview

The USB TC-08 is a temperature and voltage logger that can monitor up to eight thermocouples. With the accompanying PicoLog software, the unit can be used with any laptop or PC. If you wish to tailor the product to a particular application, you can write your own programs with the supplied driver. All software runs on Windows 7, 8 and 10.

The driver provides cold junction compensation for thermocouples. PicoLog supports up to 20 USB TC-08 devices and the driver can support as many as 64, subject to the capabilities of your PC.

1.2 About the driver

The USB TC-08 is supplied with driver routines that you can build into your own programs. A selection of code examples is available from repositories under the "picotech" organization in GitHub:

The driver function is supplied as a Windows DLL. The DLL uses the C `stdcall` calling convention and can be used with a number of different programs. It can also be used with programs like Microsoft Excel, where the macro language is a form of Visual Basic for Applications, and with all .NET languages. The DLL is available in 32-bit and 64-bit versions.

1.3 Installing the driver

The driver is installed automatically when you install the PicoLog software. Alternatively, you can download a PicoSDK containing either the 32-bit or the 64-bit driver from www.picotech.com/downloads.

1.4 Further information

For more information on the data logger and how to use it, please refer to the *USB TC-08 User's Guide*.

For information on the PicoLog software supplied with the data logger, refer to the *PicoLog User's Guide*.

1.5 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 Ltd. claims the copyright of, and retains the rights to, all material (software, documents, etc.) contained in this SDK except the example programs. 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 Technology and its agents shall not be liable for any loss, damage or injury, howsoever caused, related to the use of Pico Technology equipment or software, unless excluded by statute.

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

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

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

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

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

1.6 Trademarks

Pico Technology 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, **Excel** and **Visual Basic for Applications** are registered trademarks of Microsoft Corporation in the USA and other countries.

2 Modes of operation

2.1 Introduction

The USB TC-08 is designed for three specific modes of operation to suit a variety of applications. The following modes are supported:

- Streaming mode
- Get Single mode
- Legacy mode

2.2 Streaming mode

Streaming mode is an operational mode in which the USB TC-08 unit samples data and returns it to the computer in an unbroken sequence, using the onboard clock to ensure accurate timing.

The unit can buffer up to two sets of readings at once. To avoid loss of readings, make sure that other applications on the PC - including the one you are writing - do not prevent the driver from collecting readings for more than three sampling intervals.

To allow the driver to sample continuously, call the Windows `Sleep` function in any sampling loops (see example below) to make sure that your application does not use too much processor time.

Hint: Try not to use a `Sleep` call for less than 50 to 100 milliseconds, i.e. `Sleep(50)` to `Sleep(100)`. If you are programming a Windows GUI application, a good alternative to sampling loops is the `WM_TIMER` message.

Example
Fragment of a C application demonstrating how to use streaming mode with the USB TC-08 driver:

```
//================================================
// Setting up and running the unit in streaming mode
//================================================

usb_tc08_set_mains(handle, 0); // use 50 Hz mains noise rejection

for (channel = 0; channel < USBTC08_MAX_CHANNELS + 1; channel++)
{
  // Set each channel up as a type K thermocouple.
  // Channel 0 is the cold junction and will be enabled
  // by setting the third argument to anything other than ' '
  usb_tc08_set_channel(handle, channel, 'K');
}

// Find out how fast the unit can sample in its current setup state
minimum_interval = usb_tc08_get_minimum_interval_ms(handle);

usb_tc08_run(handle, minimum_interval); // sample as fast as possible

// Not required (just illustrates that the application
// can be idle while the driver collects the readings)
Sleep(10000);

// Use a two dimensional array with an array of readings for each channel
// In a real application, this would be a nested loop to regularly poll
// the unit for readings
for (channel = 0; channel < USBTC08_MAX_CHANNELS + 1; channel++)
{
  no_of_readings = usb_tc08_get_temp
  (
    handle,
    reading_buffer[channel],
    times_buffer,
    buffer_length,
    &overflows[channel],
    channel,
    0,  // degrees Celsius units
    0   // do not fill missing readings
  );
}

// Finished polling, now do something with the readings. If overflows[channel]
// is high, one of the readings in reading_buffer[channel] has exceeded the
// input range of the USB TC-08. Stop the unit only when you have completely
// finished streaming.
usb_tc08_stop(handle);
```

Note: You should close down all other applications while you are performing any timing-critical data logging tasks. Check that the Windows scheduler does not have any activities planned during the logging session.

2.3 Get Single mode

Get Single mode is an operational mode in which readings are produced on demand, using the usb_tc08_get_single function. Since the function relies entirely on the timing of the calling application, it is ideal for time intervals greater than 1 minute. If high-speed sampling is required, use Streaming mode.

Note: The function call overhead can be significant, since it takes approximately 900 ms to convert all 9 channels at 100 ms per channel. To reduce this overhead, disable channels that are not required.

Example
A fragment of a C application demonstrating how to use Get Single mode with the USB TC-08 driver:

```
//=========================================================
// Setting up and converting readings with Get Single mode
//=========================================================

usb_tc08_set_mains(handle, 0); // use 50Hz mains noise rejection

for (channel = 0; channel < USBTC08_MAX_CHANNELS + 1; channel++)
{
  // Set each channel up as a type K thermocouple
  // Channel 0 is the cold junction and will be enabled
  // by setting the third argument to anything other than ' '
  usb_tc08_set_channel(handle, channel, 'K');
}

// Find out the approximate conversion time for a call to
// usb_tc08_get_single

minimum_interval = usb_tc08_get_minimum_interval_ms(handle);
printf("Conversion time: %d\n", minimum_interval);

// Collect 10 readings over approximately 9 minutes
last_time = GetTickCount;

for (i = 0, i < 10, i++)
{
  // do the conversion for all channels
  usb_tc08_get_single
  (
    handle,
    value_array,     // int16_t value_array[9]
    &overflow_flags,
    0                // degrees Celsius units
  );

  // print out the values
  printf("\n\nTime: %d minute(s)", i);

  for (channel = 0; channel < USBTC08_MAX_CHANNELS + 1; channel++)
  {
    // Check for overflows on each channel
    // with a bitwise & comparator
    // Shift the comparison bit to match the channel
    if (overflow_flags &(1 << channel))
    {
      printf("\nChannel %d overflowed", channel);
    }
    else // no overflow
    {
      printf("\nChannel %d: %f", channel, value_array[channel]);
    }
```

```
    }

    if (i < 9)
    {
      while (60000  > (GetTickCount - last_time))  // 60000ms = 1 minute
      {
        Sleep(100); // let other applications run
      }
      last_time = GetTickCount;
    }
  }
```


2.4 Legacy mode

Legacy mode is designed to aid developers who have already written code for the serial version of the TC-08 and are not yet ready to rewrite their code for the Streaming or Get Single modes available with the USB version. The legacy support is going to be phased out and will eventually be removed from the driver altogether. If you want full support in the future, use the Streaming or Get Single modes.

To allow code to be easily developed for both the serial version and the USB version of the TC-08, all the function names have been changed. The `usb_tc08_legacy_set_channel` function no longer supports offset and gain - this is now stored only as calibration information in the unit itself. The legacy routines will be familiar if you previously used the serial TC-08, but the following changes should be made to convert legacy applications:

- Reference the new Header file
- Reference the new Library file
- Place the new DLL in the directory of the application
- Set the mains frequency
- Run and stop the unit
- Store a handle returned from `usb_tc08_open_unit` instead of using the serial port number

Example
The following code is a fragment of a C application demonstrating how to use legacy mode with the USB TC-08 driver:

```
//=========================================================
// Setting up and running the unit in Legacy mode
// This is designed to make it easier to adapt code written
// for the Serial TC08 for use with the USB TC-08
//=========================================================

usb_tc08_set_mains(handle, 0); // Use 50Hz mains noise rejection

for (channel = 1; channel < USBTC08_MAX_CHANNELS + 1; channel++)
{
  // Set each channel up as a type K thermocouple
  // Switch off filtering for all channels
  usb_tc08_legacy_set_channel(handle, channel, 'K', 0);
}

usb_tc08_legacy_run(handle);

last_cycle_no = 0;
no_of_readings = 0;
while (no_of_readings < 50) // Collect 50 readings
{
  usb_tc08_legacy_get_cycle(handle, &this_cycle_no);

  if (last_cycle_no != this_cycle_no)
  {
    last_cycle_no = this_cycle_no;
    no_of_readings++;

    for (channel = 1; channel < USBTC08_MAX_CHANNELS + 1; channel++)
    {
      usb_tc08_legacy_get_temp
      (
        &reading[channel],
        handle,
        channel,
        0
      );
    }
    // Now do something with the readings
    // Check that they have not overflowed by comparing each reading
    // with 2147483647L or LONG_MAX (include limits.h)

    usb_tc08_legacy_get_cold_junction(handle, &cold_junction);
    // Now do something with the cold junction temperature
  }
}

usb_tc08_stop(handle);
```


3 API functions

The API supports the following USB TC-08 functions:

Function	Description
New USB and serial mode	
usb_tc08_open_unit	Opens the USB TC-08 unit and gets a valid USB handle.
usb_tc08_open_unit_async	Opens the unit asynchronously.
usb_tc08_open_unit_progress	Polls the unit's enumeration progress during asynchronous operation.
usb_tc08_close_unit	Closes the handle.
usb_tc08_stop	Stops the unit streaming.
usb_tc08_set_mains	Sets the mains interference rejection filter to either 50 Hz or 60 Hz.
usb_tc08_get_minimum_interval_ms	Returns the minimum sampling interval for the current setup.
usb_tc08_get_unit_info	Retrieves all information on a unit and presents it as a structure.
usb_tc08_get_unit_info2	Retrieves specific information on a unit and presents it as a string.
usb_tc08_get_formatted_info	Retrieves information on a particular unit and presents it in string form.
usb_tc08_get_last_error	Returns the last error for a specified unit or for a call to open a unit.
New USB mode only	
usb_tc08_set_channel	Sets up a USB TC-08 channel.
usb_tc08_run	Starts the USB TC-08 unit streaming.
usb_tc08_get_single	Converts readings from currently set up channels on demand.
usb_tc08_get_temp	In streaming mode, retrieves temperature readings from a specified channel.
usb_tc08_get_temp_deskew	In streaming mode, retrieves temperature readings from a specified channel with time deskewed.
Legacy mode only	
usb_tc08_legacy_run	Starts the USB TC-08 unit running in legacy mode.
usb_tc08_legacy_set_channel	Sets up a USB TC-08 channel.
usb_tc08_legacy_get_temp	Retrieves temperature readings from a specified channel.
usb_tc08_legacy_get_cold_junction	Retrieves a temperature reading for the cold junction.
usb_tc08_legacy_get_driver_version	Returns the driver version.
usb_tc08_legacy_get_version	Returns the hardware version of the USB TC-08 unit.
usb_tc08_legacy_get_cycle	Returns the number of readings taken so far.

3.1 All modes

3.1.1 usb_tc08_open_unit

```
int16_t usb_tc08_open_unit(void)
```

This routine returns a valid handle to the USB TC-08 if the driver successfully opens it. If the routine fails, see the error code explanations in the usb_tc08_get_last_error section. If you wish to use more than one USB TC-08, call this routine once for each unit connected to the PC. The function will return 0 if there are no more units found. The driver is thread-safe and will not allow access to a single unit from more than one application. If, therefore, usb_tc08_open_unit does not find a unit, check that other applications are not using the USB TC-08. This includes applications on other user accounts on the same computer, where fast user switching is supported.

Note: The usb_tc08_open_unit function provides a simple way to open USB TC-08 units. However, the function call locks up the calling thread until the attached USB TC-08 unit has been fully enumerated. If a single-threaded application needs to perform concurrent processing, such as displaying a progress bar, use usb_tc08_open_unit_async.

Arguments	None
Returns	`> 0,` The handle of a unit. `0,` No more units were found. `-1,` Unit failed to open. Call usb_tc08_get_last_error with a handle of 0 to obtain the error code.

Tip: Front panel LED

- When a connection to the device has been established, the LED will be green.
- When readings are being captured, the LED will change to alternating red and green.
- The LED will return to green after the function usb_tc08_stop has been called.

Example
The following code is a fragment of a C application which demonstrates how to open multiple units with the USB TC-08 driver. The handles to the open units are stored in an array for later use:

```
//=====================================================
// Opening multiple units
//=====================================================

for (i = 0; (new_handle = usb_tc08_open_unit) > 0; i++)
{
  // store the handle in an array
  handle_array[i] = new_handle;
}
no_of_units = i;

// deal with the error if there is one, if new_handle was zero, then
// there was no error and we reached the last available unit
if (new_handle == -1)
{
  error_code = usb_tc08_get_last_error(0);
  printf("Unit failed to open\nThe error code is %d", error_code);
  // could terminate the application here
}
```



```
  //
  // Start using the open units
  //
```

3.1.2 usb_tc08_open_unit_async

```
int16_t usb_tc08_open_unit_async
(
  void
)
```

This routine begins enumerating USB TC-08 units in the background and provides a return immediately, so the calling thread can continue executing other code.

Note: The driver is thread safe and will not allow access to a single unit from more than one application. If, therefore, `usb_tc08_open_unit_async` does not find a unit, check that other applications are not using the same USB TC-08. This includes applications on other user accounts on the same computer, where fast user switching is supported.

Arguments	None	
Returns	1	The call was successful.
	0	No more units were found.
	-1	An error occurred, call usb_tc08_get_last_error with a handle of 0 to obtain the error code.

Tip: Front panel LED

- When a connection to the device has been established, the LED will be green.
- When readings are being captured, the LED will change to alternating red and green.
- The LED will return to green after the function usb_tc08_stop has been called.

Example
The following code is a fragment of a C application which demonstrates how to open a single unit with the asynchronous open unit functions:

```
//=====================================================
// Opening a unit asynchronously
//=====================================================

// Tell the driver to start enumerating the unit in the background
// (usb_tc08_open_unit_async returns immediately)
result = usb_tc08_open_unit_async;
// handle any error conditions
if (result == -1)
{
  error_code = usb_tc08_get_last_error(0);
  printf("Unit failed to open\nThe error code is %d", error_code);

  // could terminate the application here
}
else if (result == 0)
{
  printf("No USB TC08 units found");

  // could terminate the application here
}

// No errors, so start polling usb_tc08_open_unit_progress
// continuously for its enumeration state
```



```
do
{
  result = usb_tc08_open_unit_progress(&handle, &progress);

  switch(result)
  {
    case USBTC08_PROGRESS_FAIL:   // enum equates to: -1
      error_code = usb_tc08_get_last_error(0);
      printf("Unit failed to open\nThe error code is %d", error_code);

      // could terminate the application here
      break;

    case USBTC08_PROGRESS_PENDING: // enum equates to: 0
      printf("\nThe unit is %d percent enumerated", progress);
      Sleep(500); // wait for approx. half a second
      break;

    case USBTC08_PROGRESS_COMPLETE: // enum equates to: 1

      if (handle > 0)
      {
        printf("\n\nThe unit with handle '%d' opened successfully\n", handle);
      }
      else
      {
        printf("No USB TC-08 units found.\n");
      }
      break;
  }
}
while (result == USBTC08_PROGRESS_PENDING);

//
// Start using the open unit
//
```

3.1.3 usb_tc08_open_unit_progress

```
int16_t usb_tc08_open_unit_progress
(
  int16_t * handle,
  int16_t * progress
)
```

Call this function after usb_tc08_open_unit_async. Repeatedly call it to determine the state of the background enumeration process. For an example of usage, see usb_tc08_open_unit_async.

Arguments	handle,	(Out) A handle (positive int16_t) to the unit if the enumeration is completed. handle will always be 0 if the enumeration is incomplete.
	progress,	(Out) (Optional - can pass NULL) returns a number from 0 to 100 representing the percentage completion of the enumeration of one unit.
Returns	USBTC08_PROGRESS_FAIL (-1) An error occurred. Call usb_tc08_get_last_error with a handle of 0 to obtain the error code. USBTC08_PROGRESS_PENDING (0) Enumeration has not completed (keep calling usb_tc08_open_unit_progress). USBTC08_PROGRESS_COMPLETE (1) Enumeration has completed and the handle is now valid.	

3.1.4 usb_tc08_close_unit

```
int16_t tc08_close_unit
(
  int16_t handle
)
```

This routine closes the unit for a specified USB handle.

Arguments	handle,	Specifies the USB TC-08 unit.
Returns	0	Use usb_tc08_get_last_error.
	1	Unit closed successfully.

Note: If you successfully open any USB TC-08 units, call usb_tc08_close_unit for each handle before you exit from your program. If you do not, there is a chance that the unit will not reopen until it has been disconnected and reconnected.

3.1.5 usb_tc08_stop

```
int16_t usb_tc08_stop
(
  int16_t handle
)
```

This routine stops the unit from running.

Arguments	handle,	Specifies the USB TC-08 unit.
Returns	0	Invalid parameter.
	1	Unit stopped streaming successfully.

3.1.6 usb_tc08_set_mains

```
int16_t usb_tc08_set_mains
(
  int16_t handle,
  int16_t sixty_hertz
)
```

This routine sets the USB TC-08 to reject either 50 or 60 Hz.

Arguments	handle,	Specifies the USB TC-08 unit.
	sixty_hertz	Specifies whether to reject 50 Hz or 60 Hz. If set to 1, the unit will reject 60 Hz, if set to 0, the unit will reject 50 Hz.
Returns	0	Use usb_tc08_get_last_error.
	1	Mains rejection set correctly.

Note: If the rejection is not set correctly the unit will be more susceptible to mains interference.

3.1.7 usb_tc08_get_minimum_interval_ms

```
int32_t usb_tc08_get_minimum_interval_ms
(
  int16_t handle
)
```

This routine returns the minimum sampling interval (or fastest millisecond interval) that the unit can achieve in its current configuration. The configuration is defined by calling usb_tc08_set_channel for each channel.

Arguments	handle,	Specifies the USB TC-08 unit.
Returns	0,	Use usb_tc08_get_last_error.
	> 0,	Minimum sampling interval for current setup (in milliseconds).

Note: The USB TC-08 can sample, from a single channel, at a rate of 10 samples per second. The absolute minimum sampling interval, with all 8 channels and the cold junction enabled, is 900 ms. You must set up all the channels that you wish to use before calling this routine.

3.1.8 usb_tc08_get_unit_info

```
int16_t usb_tc08_get_unit_info
(
  int16_t        handle,
  USBTC08_INFO * info
)
```

This routine gets information about the USB TC-08 unit and copies it to the USBTC08_INFO structure (see below). If you pass zero to the function as the handle, only the driver version member will be valid, but the function will return 1 (success).

If the programming language you are using does not support structures, use usb_tc08_get_formatted_info.

To query a single item of information, use usb_tc08_get_unit_info2.

Arguments	handle,	Specifies the USB TC-08 unit.
	info,	A pointer to a structure containing unit information.
Returns	0	Use usb_tc08_get_last_error.
	1	Routine was successful.

You must assign the correct value to the size field of your USBTC08_INFO structure before calling this routine. For example, in C, if devinfo is your structure, use this code:

```
devinfo.size = sizeof(USBTC08_INFO);
usb_tc08_get_unit_info(hTC08, &devinfo);
```

USBTC08_INFO

```
typedef struct tUSBTC08Info
{
  int16_t size;
  int8_t  DriverVersion[USBTC08_MAX_VERSION_CHARS];
  int16_t PicoppVersion;
  int16_t HardwareVersion;
  int16_t Variant;
  int8_t  szSerial[USBTC08_MAX_SERIAL_CHARS];
  int8_t  szCalDate[USBTC08_MAX_DATE_CHARS];
} USBTC08_INFO, *LPUSBTC08_INFO;
```

3.1.9 usb_tc08_get_unit_info2

```
int16_t usb_tc08_get_unit_info2
(
  int16_t    handle,
  int8_t   * string,
  int16_t    string_length,
  int16_t    line
)
```

This routine obtains a specified item of device information. To obtain all information in a single call, use usb_tc08_get_unit_info.

Arguments	handle, specifies the USB TC-08 unit. string, pointer to an array to hold the information on exiting the function. string_length, the size of string array. line, identifies the type of information to be returned, as defined in usbtc08.h: USBTC08LINE_DRIVER_VERSION USBTC08LINE_KERNEL_DRIVER_VERSION USBTC08LINE_HARDWARE_VERSION USBTC08LINE_VARIANT_INFO (model number) USBTC08LINE_BATCH_AND_SERIAL (batch and serial number) USBTC08LINE_CAL_DATE (calibration date)
Returns	0, Use usb_tc08_get_last_error 1, Routine was successful

3.1.10 usb_tc08_get_formatted_info

```
int16_t usb_tc08_get_formatted_info
(
  int16_t   handle,
  int8_t  * unit_info,
  int16_t   string_length
)
```

This function is similar to the usb_get_unit_info routine, but the unit information is returned in the form of a formatted character string. The string is separated into the following elements, each appearing on a different line: driver version; hardware version; variant info; serial number; calibration date.

Arguments	handle,	Specifies the USB TC-08 unit.
	unit_info,	A string where the unit info is to be placed.
	string_length,	Length of the string to be copied. Should be at least 256 (USBTC08_MAX_INFO_CHARS) characters long.
Returns	0,	Too many bytes to copy. Will copy as many full lines as possible.
	1,	Routine was successful.

3.1.11 usb_tc08_get_last_error

```
int16_t usb_tc08_get_last_error
(
  int16_t handle
)
```

This function returns the last error for the specified device.

Note: If an invalid handle is passed to a function, the function will fail. The error code, however, cannot be associated with a device so usb_tc08_get_last_error will not retain an error code in this instance. usb_tc08_get_last_error will also fail if the invalid handle is passed to it.

Arguments	handle,	Specifies the USB TC-08 device. If zero, returns the error associated with the last call to usb_tc08_open_unit or usb_tc08_open_unit_async.
Returns	-1, >= 0,	Invalid handle. See below.

The error codes, also found in the C header file, are as follows:

User/Developer error codes:

Error code	Error	Further information
0	USBTC08_ERROR_OK	No error occurred.
1	USBTC08_ERROR_OS_NOT_SUPPORTED	The driver does not support the current operating system.
2	USBTC08_ERROR_NO_CHANNELS_SET	A call to usb_tc08_set_channel is required.
3	USBTC08_ERROR_INVALID_PARAMETER	One or more of the function arguments were invalid.
4	USBTC08_ERROR_VARIANT_NOT_SUPPORTED	The hardware version is not supported. Download the latest driver.
5	USBTC08_ERROR_INCORRECT_MODE	An incompatible mix of legacy and non-legacy functions was called (or usb_tc08_get_single was called while in streaming mode).
6	USBTC08_ERROR_ENUMERATION_INCOMPLETE	usb_tc08_open_unit_async was called again while a background enumeration was already in progress.

Note: For more details on error codes, see troubleshooting.

Reserved Pico error codes

Error code	Error	Further information
7	USBTC08_ERROR_NOT_RESPONDING	Cannot get a reply from a USB TC-08.
8	USBTC08_ERROR_FW_FAIL	Unable to download firmware.
9	USBTC08_ERROR_CONFIG_FAIL	Missing or corrupted EEPROM.
10	USBTC08_ERROR_NOT_FOUND	Cannot find enumerated device.
11	USBTC08_ERROR_THREAD_FAIL	A threading function failed.
12	USBTC08_ERROR_PIPE_INFO_FAIL	Can not get USB pipe information.
13	USBTC08_ERROR_NOT_CALIBRATED	No calibration date was found.
14	USBTC08_ERROR_PICOPP_TOO_OLD	An old picopp.sys driver was found on the system.
15	USBTC08_ERROR_COMMUNICATION	The PC has lost communication with the device.

Note: These reserved error code values are only meaningful to Pico Technology Technical Support staff, but they are supplied to allow developers to display warnings in their applications. For more details on error codes, see troubleshooting.

3.2 New USB mode only

3.2.1 usb_tc08_set_channel

```
int16_t usb_tc08_set_channel
(
  int16_t handle,
  int16_t channel,
  int8_t  tc_type
)
```

Call this routine once for each channel that you want to use. You can do this any time after calling usb_tc08_open_unit. By default, all channels are disabled.

Arguments	handle,	Specifies the USB TC-08 unit.
	channel,	Specifies which channel you want to set the details for: this should be between 0 and 8 (0 denotes the cold junction).
	tc_type,	Specifies what type of thermocouple is connected to this channel. Set to one of the following characters: B, E, J, K, N, R, S, T. Use a space (' ') to disable the channel. Voltage readings can be obtained by passing X as the character.
Returns	0,	Use usb_tc08_get_last_error.
	1,	Routine was successful.

Note: The CJC is always enabled automatically if a thermocouple is being used. When no channels are active as thermocouples, the CJC can be optionally enabled or disabled.

3.2.2 usb_tc08_run

```
int32_t usb_tc08_run
(
  int16_t  handle,
  int32_t  interval
)
```

This routine starts the unit running with a sampling interval, specified in milliseconds. This routine should be called after usb_tc08_set_channel has been called.

Arguments	handle,	Specifies the USB TC-08 unit.
	interval,	Specifies the requested sampling period. You can use usb_tc08_get_minimum_interval_ms to obtain the smallest sampling period permitted with the current setup.
Returns	0,	Use usb_tc08_get_last_error.
	interval,	Actual interval allowed by the driver.

3.2.3 usb_tc08_get_single

```
int16_t usb_tc08_get_single
(
  int16_t   handle,
  float   * temp,
  int16_t * overflow_flags,
  int16_t   units
)
```

You must set up the channels before calling this function. You must **not** have put the unit into Streaming mode with usb_tc08_run, as this will cause usb_tc08_get_single to fail. The function will convert all readings on demand. For more details and an example see the Get Single mode section.

Arguments	handle,	Specifies the USB TC-08 unit.
	temp,	Pointer to an array of length [9]. There are 9 channels on the USB TC-08 (8 + cold junction) and the readings are always placed in the array subscript corresponding to the channel number. Channels which are not enabled are filled with QNaN values.
	overflow_flags,	Pointer to a variable containing a set of bit flags that are set high when an overflow occurs on a particular channel. An overflow occurs when the input signal is higher than the measuring range of the USB TC-08. The lowest significant bit (bit 0) represents channel 0 (the cold junction channel) and bit 8 represents channel 8 (the last thermocouple channel). Bitwise comparisons should be performed to determine the overflow state of each channel.
	units,	Specifies the temperature units for returned data: 0: USBTC08_UNITS_CENTIGRADE 1: USBTC08_UNITS_FAHRENHEIT 2: USBTC08_UNITS_KELVIN 3: USBTC08_UNITS_RANKINE
Returns	0,	An error occurred. Use usb_tc08_get_last_error to get the code.
	1,	The function succeeded.

3.2.4 usb_tc08_get_temp

```
int32_t usb_tc08_get_temp
(
  int16_t   handle,
  float   * temp_buffer,
  int32_t * times_ms_buffer,
  int32_t   buffer_length,
  int16_t * overflow,
  int16_t   channel,
  int16_t   units,
  int16_t   fill_missing
)
```

Once you open the driver and set up some channels, you can call the usb_tc08_run routine. The driver will then begin to continually take readings from the USB TC-08. Use the usb_tc08_get_temp routine to retrieve readings from the driver's buffer periodically. You must call the function at least once per minute to avoid losing data (the driver's buffer is circular, so the oldest readings will be overwritten first).

Streaming mode relies on the driver to buffer readings without interruption. If the driver does not get enough share of the PC's processor time (the most frequent cause of which is too many applications running at the same time), readings will be dropped and the sample buffer will be padded with QNaN floating integers.

Warning: The padding of the buffer is also dependent on the performance of the PC and under very heavy processor loading, padding may not always be accurate.

Arguments	
handle,	Specifies the USB TC-08 unit.
temp_buffer,	Pointer to a location where the readings are to be placed.
times_ms_buffer,	Returns the time that the first channel was converted (optional).
buffer_length,	Length of data buffers.
overflow,	Pointer to a variable that will be assigned a value of 1 if an overflow occurred on any of the readings copied into temp_buffer, or 0 if an overflow did not occur. An overflow occurs when the input signal is higher than the measuring range of the USB TC-08.
channel,	Specifies the channel to read the temperature from.
units,	Specifies the temperature units for returned data: 0: USBTC08_UNITS_CENTIGRADE 1: USBTC08_UNITS_FAHRENHEIT 2: USBTC08_UNITS_KELVIN 3: USBTC08_UNITS_RANKINE Voltages are always returned in millivolts.
fill_missing,	Choose whether or not to replace QNaN values with the last known value: 0 - Use QNaNs to represent missing readings 1 - Fill missing readings with previous good reading.
Returns	
-1,	An error occurred. Use usb_tc08_get_last_error to get the code.
0,	Currently no readings to collect.
>0,	Number of readings copied into array (there may still be more readings in the driver's internal buffer).

3.2.5 usb_tc08_get_temp_deskew

```
int32_t usb_tc08_get_temp_deskew
(
  int16_t   handle,
  float   * temp_buffer,
  int32_t * times,
  int32_t   buffer_length,
  int16_t * overflow,
  int16_t   channel,
  int16_t   units,
  int16_t   fill_missing
)
```

Same as usb_tc08_get_temp but the times take account of small differences caused by the order in which channels are converted. Note: Unless there is a specific reason to use the usb_tc08_get_temp_deskew routine, use usb_tc08_get_temp instead.

Arguments	
handle,	Specifies the USB TC-08 unit.
temp_buffer,	Pointer to a buffer where the temperatures are to be placed.
times_ms_buffer,	Returns the exact time that this channel was converted (optional).
buffer_length,	Specifies the length of the sample buffer.
overflow,	Pointer to a variable that will be assigned a value of 1 if an overflow occurred on any of the readings copied into temp_buffer, or 0 if an overflow did not occur. An overflow occurs when the input signal is higher than the measuring range of the USB TC-08.
channel,	Specifies the channel to read the temperature from.
units,	Specifies the temperature units in which the data are returned: 0: USBTC08_UNITS_CENTIGRADE 1: USBTC08_UNITS_FAHRENHEIT 2: USBTC08_UNITS_KELVIN 3: USBTC08_UNITS_RANKINE Voltages are always returned in millivolts.
fill_missing,	Choose whether or not to replace QNaN values (missing readings) with the last known value: 0 - Use QNaNs to represent missing readings 1 - Fill missing readings (no QNaNs).

Returns		
	-1,	An error occurred. Use usb_tc08_get_last_error to get the code.
	0,	Currently no readings to collect.
	>0,	Number of readings copied into array (there may still be more readings in the driver's internal buffer).

3.3 Legacy mode only

3.3.1 usb_tc08_legacy_run

```
int16_t usb_tc08_legacy_run
(
  int16_t handle
)
```

This routine starts the sampling thread and forces the specified unit to run in legacy mode.

Arguments	handle,	Specifies the USB TC-08 unit.
Returns	0, 1,	Use usb_tc08_get_last_error. Legacy run successful.

3.3.2 usb_tc08_legacy_set_channel

```
int16_t usb_tc08_legacy_set_channel
(
  int16_t handle,
  int16_t channel,
  int8_t  tc_type,
  int16_t filter_factor,
  int16_t offset,
  int16_t slope
)
```

Call this routine once for each channel that you would like to take readings from. You can do this any time after calling usb_tc08_open_unit.

Arguments	handle,	Specifies the USB TC-08 unit.
	channel,	Specifies which channel you want to set the details for: This should be between 0 and 8.
	tc_type,	Specifies what type of thermocouple is connected to this channel. Set to one of the following characters: B, E, J, K, N, R, S, T. Use a space (' ') to disable the channel.
	filter_factor,	Specifies the size of the median filter. Each time the driver takes a reading from this channel, it updates the filtered value by adding a reading to a median filter. The filter factor can be set to any value between 0 and 255 (0 or 1 turns filtering off). These parameters are provided for backward compatibility with the serial TC-08 and have no effect when used with the USB TC-08.
	offset and slope,	The parameters offset and slope are not used and therefore should be set to 0.
Returns	0,	Use usb_tc08_get_last_error.
	1,	Legacy set channel successful.

Note: Do not call this function unless you are operating the USB TC-08 in legacy mode, having called usb_tc08_legacy_run.

3.3.3 usb_tc08_legacy_get_temp

```
int16_t usb_tc08_legacy_get_temp
(
  int32_t  * temp,
  int16_t    handle,
  uint16_t   channel,
  uint16_t   filtered
)
```

Once you open the driver and define some channels, you can call the usb_tc08_legacy_run routine. The driver will then constantly take readings from the USB TC-08. Temperatures are returned in hundredths of a degree Celsius and voltages are returned in microvolts.

Arguments	
temp,	Pointer to a location where readings are to be placed. Each reading should be compared with 2147483647L or LONG_MAX (include limits.h) to check for overflows. An overflow occurs when the input signal is higher than the measuring range of the USB TC-08.
handle,	Specifies the USB TC-08 unit.
channel,	Specifies from which channel to read temperature. Should be 0 for CJC, 1 for Channel 1, 2 for Channel 2 and so on.
filtered,	Specifies whether or not to filter the data. The readings are median filtered if set to 1. 0 causes the unfiltered temperature to be stored in temp. The filter has a depth defined by the filter factor, set during a usb_legacy_set_channel call.
Returns	0, Use usb_tc08_get_last_error. 1, Temperature retrieval successful.

Note: Do not call this function unless you are operating the USB TC-08 in legacy mode, having called usb_tc08_legacy_run.

3.3.4 usb_tc08_legacy_get_cold_junction

```
int16_t usb_tc08_legacy_get_cold_junction
(
  int32_t * temp,
  int16_t   handle
)
```

This routine retrieves a cold junction temperature reading. This can also be achieved by passing `channel = 0` to `usb_tc08_get_temp`. Temperatures are returned in hundredths of a degree Celsius. Normally, you do not need to worry about the cold junction temperature, as the driver automatically uses it to compensate thermocouples. You can, however, use it as an indication of ambient temperature.

Arguments	`temp,`	Pointer to a location where the temperature reading is to be stored.
	`handle,`	Specifies the USB TC-08 unit.
Returns	`0,`	Use `usb_tc08_get_last_error`.
	`1,`	Retrieval of cold junction temperature reading successful.

3.3.5 usb_tc08_legacy_get_driver_version

```
int16_t usb_tc08_legacy_get_driver_version
(
  void
)
```

This routine returns the driver version. This is useful when you need to find out if the latest driver is being used.

Arguments	None
Returns	Driver version number

3.3.6 usb_tc08_legacy_get_version

```
int16_t usb_tc08_legacy_get_version
(
  int16_t * version,
  int16_t   handle
)
```

This routine sets the `version` variable to match the version of the USB TC-08 currently being used.

Arguments	`version,`	Pointer to a location where the version number is to be stored.
	`handle,`	Specifies the USB TC-08 unit.
Returns	`0,`	Invalid handle.
	`1,`	Retrieval of version number successful.

3.3.7 usb_tc08_legacy_get_cycle

```
int16_t usb_tc08_legacy_get_cycle
(
  int32_t * cycle,
  int16_t   handle
)
```

This routine gives the number of complete cycles of readings taken from a particular USB TC-08. Calling usb_tc08_legacy_get_temp causes the most recent reading for the specified channel to be returned immediately. If you wish to record values only when the driver has taken a new reading, you can use this routine to find out how many complete cycles of readings the driver has taken. Then you can call usb_tc08_legacy_get_temp, but only when the cycle has been incremented.

Arguments	cycle,	Pointer to a location where the cycle number is to be stored.
	handle,	Specifies the USB TC-08 unit.
Returns	0,	Use usb_tc08_get_last_error.
	1,	Legacy get cycle successful.

Tip: Do not test for an exact cycle number; instead, test for a different cycle number as your application may have missed readings. See the Legacy mode section for an example.

Note: Do not call this function unless you are operating the USB TC-08 in legacy mode, having called usb_tc08_legacy_run.

4 Troubleshooting

The following table lists each of the error codes described in the usb_tc08_get_last_error topic, and divides them into categories so that you know what to do in the event of a particular error occurring.

Error code	Description	Category
1	USBTC08_ERROR_OS_NOT_SUPPORTED	P*
2	USBTC08_ERROR_NO_CHANNELS_SET	P
3	USBTC08_ERROR_INVALID_PARAMETER	P
4	USBTC08_ERROR_VARIANT_NOT_SUPPORTED	P*
5	USBTC08_ERROR_INCORRECT_MODE	P
6	USBTC08_ERROR_ENUMERATION_INCOMPLETE	S
7	USBTC08_ERROR_NOT_RESPONDING	S
8	USBTC08_ERROR_FW_FAIL	S
9	USBTC08_ERROR_CONFIG_FAIL	S
10	USBTC08_ERROR_NOT_FOUND	S
11	USBTC08_ERROR_THREAD_FAIL	S
12	USBTC08_ERROR_PIPE_INFO_FAIL	S
13	USBTC08_ERROR_NOT_CALIBRATED	S
14	USBTC08_ERROR_PICOPP_TOO_OLD	S

Key	
S	Errors in this category indicate that a fault has occurred with the USB TC-08 unit or your PC. Try disconnecting the USB TC-08 and then reconnecting it. If this does not work, restart your PC. If this does not work, do the following: 1. Uninstall the software and restart the PC 2. Reinstall the software and restart the PC 3. If this does not work, download the latest version of the software from https://www.picotech.com/downloads and install this, then restart the PC 4. If this does not work, contact support@picotech.com
P	Errors in this category are handled internally by software applications, and only developers need to be aware of their meanings.
P*	Errors in this category are user-dependent and developers should make sure that their application provides suitable error messages for users to read in the event of an error occurring.

5 Glossary

Cold junction compensation (CJC). A method of compensating for ambient temperature variations in thermocouple circuits.

Common mode range. The voltage range, relative to the ground of the data logger, within which both inputs of a differential measurement must lie in order to achieve an accurate measurement.

DLL. Dynamic Link Library. Files with this file extension contain a collection of Windows functions designed to perform a specific class of operations.

Input impedance. The resistance measured between the input terminals of a circuit.

NFR. Noise-Free Resolution. The effective number of bits of resolution that can be considered noise-free.

Overvoltage protection. The maximum input voltage that can be applied without damaging the unit.

QNaN. Quiet Not a Number. In the context of the USB TC-08, QNaNs are numbers created artificially to fill in gaps in sampling. These gaps are interruptions caused by lack of available PC or laptop processor time, normally caused by too many applications being open simultaneously. QNaNs are defined in the **IEEE 754-1985** ISO standard and are indeterminate, meaning that a comparison between two QNaNs always returns false.

In C/C++, the `int32_t _isnan(double)` function in the `float.h` header file can be used to identify QNaN float representations, cast to a `double` first. QNaNs will not cause an error if arithmetic operations are performed on them - however, the results will remain indeterminate.

Resolution. A value in bits, related to the number of increments of an analog input signal that can be detected by a digital measurement system. A high-resolution measurement system detects smaller signal increments than a low-resolution measurement system.

Thermocouple. A device consisting of two dissimilar metals joined together. The thermoelectric voltage developed between the two junctions is proportional to the temperature difference between the junctions.

Type B thermocouple. Type B thermocouples are made from platinum and rhodium and are suitable for high temperature measurements of up to 1820 °C. Unusually, due to the shape of their temperature / voltage curve, type B thermocouples give the same output at 0 °C as at 42 °C.

Type E thermocouple. Type E thermocouples are made from chromel and constantan. They have a high output (68 µV/°C), making them well suited to low-temperature (cryogenic) use. They are non-magnetic.

Type J thermocouple. Type J thermocouples are made from iron and constantan. They measure temperatures in the range –210 to +1200 °C. The main application is with old equipment that can not accept more modern thermocouples. J types should not be used above 760 °C, as an abrupt magnetic transformation will cause permanent decalibration.

Type K thermocouple. Type K thermocouples are low-cost, general-purpose thermocouples, made from chromel and alumel, operating in the –270 °C to +1370 °C temperature range. Sensitivity is about 41 µV/°C.

Type N thermocouple. Type N thermocouples are made from nicrosil and nisil. The high stability and resistance to high-temperature oxidation of these thermocouples make them suitable for measuring high temperatures. They are less expensive than platinum types B,R, and S and were designed to be an improved type K.

Type R thermocouple. Type R thermocouples are made from platinum and rhodium, and are suitable for high-temperature measurements of up to 1760 °C. Low sensitivity (10 µV/°C) and high cost make them unsuitable for general purpose use.

Type S thermocouple. Type S thermocouples are made from platinum and rhodium, and are suitable for high-temperature measurements of up to 1760 °C. Low sensitivity (10 µV/°C) and high cost make these thermocouples unsuitable for general purpose use. Due to their high stability, type S thermocouples are used as the standard of calibration for the melting point of gold.

Type T thermocouple. Type T thermocouples are made from copper and constantan, are highly accurate, and operate in the –270 °C to +400 °C temperature range.

USB. Universal Serial Bus. This is a standard port that enables you to connect external devices to PCs. A typical USB 1.1 port supports a data transfer rate of 12 megabits per second, and is much faster than a COM port.

Index

UK headquarters:

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

USA regional office:

Pico Technology
320 N Glenwood Blvd
Tyler
Texas 75702
United States

Tel: +1 800 591 2796
Fax: +1 620 272 0981

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

sales@picotech.com
support@picotech.com

pico.china@picotech.com

www.picotech.com

usbtc08pg r7 2017-09-20