



# **USB TC-08**

## **Temperature Logger**

Programmer's Guide



# Contents

1 Introduction .....	1
<b>1 Overview</b> .....	1
<b>2 About the driver</b> .....	1
<b>3 Installing the driver</b> .....	1
<b>4 Further information</b> .....	1
<b>5 Legal information</b> .....	2
<b>6 Company details</b> .....	3
2 Modes of operation .....	4
<b>1 Introduction</b> .....	4
<b>2 Streaming mode</b> .....	5
<b>3 Get Single mode</b> .....	7
<b>4 Legacy mode</b> .....	10
3 API functions .....	12
<b>1 All modes</b> .....	13
<b>1 usb_tc08_open_unit</b> .....	13
<b>2 usb_tc08_open_unit_async</b> .....	15
<b>3 usb_tc08_open_unit_progress</b> .....	17
<b>4 usb_tc08_close_unit</b> .....	18
<b>5 usb_tc08_stop</b> .....	19
<b>6 usb_tc08_set_mains</b> .....	20
<b>7 usb_tc08_get_minimum_interval_ms</b> .....	21
<b>8 usb_tc08_get_unit_info</b> .....	22
<b>9 usb_tc08_get_formatted_info</b> .....	23
<b>10 usb_tc08_get_last_error</b> .....	24
<b>2 New USB mode only</b> .....	26
<b>1 usb_tc08_set_channel</b> .....	26
<b>2 usb_tc08_run</b> .....	27
<b>3 usb_tc08_get_single</b> .....	28
<b>4 usb_tc08_get_temp</b> .....	29
<b>5 usb_tc08_get_temp_deskew</b> .....	31
<b>3 Legacy mode only</b> .....	33
<b>1 usb_tc08_legacy_run</b> .....	33
<b>2 usb_tc08_legacy_set_channel</b> .....	34
<b>3 usb_tc08_legacy_get_temp</b> .....	35
<b>4 usb_tc08_legacy_get_cold_junction</b> .....	36
<b>5 usb_tc08_legacy_get_driver_version</b> .....	37
<b>6 usb_tc08_legacy_get_version</b> .....	38
<b>7 usb_tc08_legacy_get_cycle</b> .....	39
4 Programming examples .....	40
<b>1 C and C++</b> .....	40
<b>2 Delphi</b> .....	40
<b>3 Excel</b> .....	41
<b>4 LabVIEW</b> .....	41
<b>5 Visual Basic</b> .....	41

5 Troubleshooting .....42

6 Glossary .....43

Index .....45

# 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 are technically-minded and want to tailor the product to a particular application, you can write your own programs with the supplied driver. All software runs on Windows XP SP3, Windows Vista, Windows 7 or Windows 8.

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.

Once you have installed the software, the SDK will contain the drivers and a selection of examples of how to use the drivers.

The driver is supplied as a Windows [DLL](#). The DLL uses the C `stdcall` calling convention and can be used with C, Delphi and Visual Basic programs. It can also be used with programs like Microsoft Excel, where the macro language is a form of Visual Basic.

## 1.3 Installing the driver

The driver is installed automatically when you install the PicoLog software. Alternatively, you can download the driver from our website at:

<http://www.picotech.com>.

## 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 licence to the person who installs this software, subject to the conditions listed below.

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

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

**Copyright.** Pico Technology 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 licence 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 28 days of purchase for a full refund.

**Upgrades.** We provide upgrades, free of charge, from our web site at [www.picotech.com](http://www.picotech.com). We reserve the right to charge for updates or replacements sent out on physical media.

**Trademarks.** **Windows**, **Excel** and **Visual Basic** are registered trademarks of Microsoft Corporation in the USA and other countries. **Delphi** is a registered trademark of Borland Software Corporation. **LabView** is a registered trademark of National Instruments Corporation. Pico Technology Limited and PicoLog are internationally registered trademarks.

## 1.6 Company details

You can obtain technical assistance from Pico Technology at the following address:

**Address:** Pico Technology  
James House  
Colmworth Business Park  
Eaton Socon  
St Neots  
Cambridgeshire  
PE19 8YP  
United Kingdom

Phone: +44 (0) 1480 396 395

Fax: +44 (0) 1480 396 296

**Email:**  
Technical Support: [support@picotech.com](mailto:support@picotech.com)  
Sales: [sales@picotech.com](mailto:sales@picotech.com)

**Web site:** [www.picotech.com](http://www.picotech.com)

## 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 another application on the PC - including the one you are writing - does 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**

The following code is a 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 50Hz mains noise rejection

for (channel = 0; channel < 9; 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 < 9; channel++)
{
    no_of_readings = usb_tc08_get_temp( handle,
        &reading_buffer[channel],
        &times_buffer[channel],
        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, then one of the readings
// in reading_buffer[channel] has exceeded the input range
// of the USB TC-08

// only stop the unit when we've 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 360 ms to convert all 9 channels at 40 ms per channel. To reduce this overhead, disable channels that are not required.

**Example**

The following code is 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 < 9; 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,      // short value_array[9]
                        &overflow_flags,
                        0);                // degrees Celsius units

    // print out the values
    printf("\n\nTime: %d minute(s)", i);
    for (c = 0; c < 9; c++)
    {
        // check for overflows on each channel
        // with a bitwise & comparator
        // shift the comparison bit to match the channel
        if (overflow_flags & (1 << c))
        {
            printf("\nChannel %d overflowed", c);
        }
        else // no overflow
        {
            printf("\nChannel %d: %f", c, value_array[c]);
        }
    }
}

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 TC08
//=====

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

for (channel = 1; channel < 9; 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 < 9; 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
<b>New USB and serial mode</b>	
<a href="#">usb_tc08_open_unit</a>	Opens the USB TC-08 unit and gets a valid USB handle.
<a href="#">usb_tc08_open_unit_async</a>	Opens the unit asynchronously.
<a href="#">usb_tc08_open_unit_progress</a>	Polls the unit's enumeration progress during asynchronous operation.
<a href="#">usb_tc08_close_unit</a>	Closes the handle.
<a href="#">usb_tc08_stop</a>	Stops the unit streaming.
<a href="#">usb_tc08_set_mains</a>	Sets the mains interference rejection filter to either 50 Hz or 60 Hz.
<a href="#">usb_tc08_get_minimum_interval_ms</a>	Returns the minimum sampling interval for the current setup.
<a href="#">usb_tc08_get_unit_info</a>	Retrieves information on a particular unit and presents it as a structure.
<a href="#">usb_tc08_get_formatted_info</a>	Retrieves information on a particular unit and presents it in string form.
<a href="#">usb_tc08_get_last_error</a>	Returns the last error for a specified unit or for a call to open a unit.
<b>New USB mode only</b>	
<a href="#">usb_tc08_set_channel</a>	Sets up a USB TC-08 channel.
<a href="#">usb_tc08_run</a>	Starts the USB TC-08 unit streaming.
<a href="#">usb_tc08_get_single</a>	Converts readings from currently set up channels on demand.
<a href="#">usb_tc08_get_temp</a>	In streaming mode, retrieves temperature readings from a specified channel.
<a href="#">usb_tc08_get_temp_deskew</a>	In streaming mode, retrieves temperature readings from a specified channel with time deskewed.
<b>Legacy mode only</b>	
<a href="#">usb_tc08_legacy_run</a>	Starts the USB TC-08 unit running in legacy mode.
<a href="#">usb_tc08_legacy_set_channel</a>	Sets up a USB TC-08 channel.
<a href="#">usb_tc08_legacy_get_temp</a>	Retrieves temperature readings from a specified channel.
<a href="#">usb_tc08_legacy_get_cold_junction</a>	Retrieves a temperature reading for the cold junction.
<a href="#">usb_tc08_legacy_get_driver_version</a>	Returns the driver version.
<a href="#">usb_tc08_legacy_get_version</a>	Returns the hardware version of the USB TC-08 unit.
<a href="#">usb_tc08_legacy_get_cycle</a>	Returns the number of readings taken so far.



### 3.1 All modes

#### 3.1.1 `usb_tc08_open_unit`

```
short 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](#).

<b>Arguments</b>	None
<b>Returns</b>	<p>&gt; 0    The handle of a unit.</p> <p>0      No more units were found.</p> <p>-1     Unit failed to open. Call <a href="#">usb_tc08_get_last_error</a> with a handle of 0 to obtain the error code.</p>

#### 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
        short 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.

<b>Arguments</b>	None	
<b>Returns</b>	1	The call was successful.
	0	No more units were found.
	-1	An error occurred, call <a href="#">usb_tc08_get_last_error</a> with a handle of 0 to obtain the error code.

**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
            printf("\n\nThe unit with handle '%d', opened successfully",
            handle);
            break;
    }
}
while (result == USBTC08_PROGRESS_PENDING);

//
// Start using the open unit
//

```

```

3.1.3  usb_tc08_open_unit_progress
        short usb_tc08_open_unit_progress
        (
            short * handle,
            short * 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\(\)](#).

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

```
3.1.4  usb_tc08_close_unit
        short tc08_close_unit
        (
            short handle
        )
```

This routine closes the unit for a specified USB handle.

<b>Arguments</b>	<code>handle</code> , Specifies the USB TC-08 unit.
<b>Returns</b>	0 Use <a href="#">usb_tc08_get_last_error</a> . 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
        short usb_tc08_stop
        (
            short handle
        )
```

This routine stops the unit from running.

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

```

3.1.6  usb_tc08_set_mains
        short usb_tc08_set_mains
        (
            short handle,
            short sixty_hertz
        )

```

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

<b>Arguments</b>	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.
<b>Returns</b>	0	Use <a href="#">usb_tc08_get_last_error</a> .
	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
        long usb_tc08_get_minimum_interval_ms
        (
            short 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\(\)](#).

<b>Arguments</b>	handle,	Specifies the USB TC-08 unit.
<b>Returns</b>	0 > 0	Use <a href="#">usb_tc08_get_last_error</a> . 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
        short usb_tc08_get_unit_info
        (
            short          handle,
            USBTC08_INFO * info
        )

```

This routine gets the unit information and copies it to the [USBTC08\\_INFO](#) structure, declared in the `usbtc08.h` header file. 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).

<b>Arguments</b>	<code>handle</code> ,	Specifies the USB TC-08 unit.
	<code>info</code> ,	A pointer to a structure containing unit information.
<b>Returns</b>	0	Use <a href="#">usb_tc08_get_last_error</a> .
	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);

```

```

3.1.9  usb_tc08_get_formatted_info
        short usb_tc08_get_formatted_info
        (
            short    handle,
            char    * unit_info,
            short    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.

<b>Arguments</b>	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.
<b>Returns</b>	0	Too many bytes to copy. Will copy as many full lines as possible.
	1	Routine was successful.

3.1.10 `usb_tc08_get_last_error`

```
short usb_tc08_get_last_error
(
    short 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.

<b>Arguments</b>	<code>handle,</code>	Specifies the USB TC-08 device. If zero, returns the error associated with the last call to <a href="#">usb_tc08_open_unit()</a> or <a href="#">usb_tc08_open_unit_async()</a> .
<b>Returns</b>	-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	<code>USBTC08_ERROR_OK</code>	No error occurred.
1	<code>USBTC08_ERROR_OS_NOT_SUPPORTED</code>	The driver supports Windows XP SP3, Windows Vista, Windows 7 and Windows 8.
2	<code>USBTC08_ERROR_NO_CHANNELS_SET</code>	A call to <a href="#">usb_tc08_set_channel()</a> is required.
3	<code>USBTC08_ERROR_INVALID_PARAMETER</code>	One or more of the function arguments were invalid.
4	<code>USBTC08_ERROR_VARIANT_NOT_SUPPORTED</code>	The hardware version is not supported. Download the latest driver.
5	<code>USBTC08_ERROR_INCORRECT_MODE</code>	An incompatible mix of legacy and non-legacy functions was called (or <a href="#">usb_tc08_get_single()</a> was called while in <a href="#">streaming mode</a> .)
6	<code>USBTC08_ERROR_ENUMERATION_INCOMPLETE</code>	<a href="#">usb_tc08_open_unit_async()</a> was called again while a background enumeration was already in progress.

Note: For more details on error codes, see [troubleshooting](#).

**Reserved Pico error codes**

<b>Error code</b>	<b>Error</b>	<b>Further information</b>
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 <code>picopp.sys</code> 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 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`

```
short usb_tc08_set_channel
(
    short handle,
    short channel,
    char 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.

<b>Arguments</b>	<code>handle,</code>	Specifies the USB TC-08 unit.
	<code>channel,</code>	Specifies which channel you want to set the details for: this should be between 0 and 8 (0 denotes the cold junction.)
	<code>tc_type,</code>	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.
<b>Returns</b>	0	Use <a href="#">usb_tc08_get_last_error()</a> .
	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
        long usb_tc08_run
        (
            short handle,
            long 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.

<b>Arguments</b>	<code>handle,</code>	Specifies the USB TC-08 unit.
	<code>interval,</code>	Specifies the requested sampling period. You can use <a href="#">usb_tc08_get_minimum_interval_ms()</a> to obtain the smallest sampling period permitted with the current setup.
<b>Returns</b>	<code>0</code>	Use <a href="#">usb_tc08_get_last_error()</a> .
	<code>interval</code>	Actual interval allowed by the driver.

```

3.2.3  usb_tc08_get_single
        short usb_tc08_get_single
        (
            short    handle,
            float *  temp,
            short *  overflow_flags,
            short    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.

<b>Arguments</b>	<code>handle,</code>	Specifies the USB TC-08 unit.
	<code>temp,</code>	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 <a href="#">QNaN</a> values.
	<code>overflow_flags,</code>	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.
	<code>units,</code>	Specifies the temperature units for returned data:  0: USBTC08_UNITS_CENTIGRADE 1: USBTC08_UNITS_FAHRENHEIT 2: USBTC08_UNITS_KELVIN 3: USBTC08_UNITS_RANKINE
<b>Returns</b>	0	An error occurred. Use <a href="#">usb_tc08_get_last_error()</a> to get the code.
	1	The function succeeded.



```

3.2.4  usb_tc08_get_temp
        long usb_tc08_get_temp
        (
            short    handle,
            float *  temp_buffer,
            long *   times_ms_buffer,
            long     buffer_length,
            short *  overflow,
            short    channel,
            short    units,
            short    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		
	<code>handle,</code>	Specifies the USB TC-08 unit.
	<code>temp_buffer,</code>	Pointer to a location where the readings are to be placed.
	<code>times_ms_buffer,</code>	Returns the time that the first channel was converted (optional).
	<code>buffer_length,</code>	Length of data buffers.
	<code>overflow,</code>	Pointer to a variable that will be assigned a value of 1 if an overflow occurred on any of the readings copied into <code>temp_buffer</code> , 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.
	<code>channel,</code>	Specifies the channel to read the temperature from.
	<code>units,</code>	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

<b>Arguments</b>	<code>fill_missing,</code>	Choose whether or not to replace <a href="#">QNaN</a> values with the last known value:  0 - Use <a href="#">QNaNs</a> to represent missing readings 1 - Fill missing readings with previous good reading.
<b>Returns</b>	-1  0  >0	An error occurred. Use <a href="#">usb_tc08_get_last_error</a> to get the code. Currently no readings to collect. 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
        long usb_tc08_get_temp_deskew
        (
            short  handle,
            float  * temp,
            long   * times,
            long   buffer_length,
            short  * overflow,
            short  channel,
            short  units,
            short  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,	Pointer to a location where the temperature is 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 <a href="#">QNaN</a> values (missing readings) with the last known value:  0 - Use <a href="#">QNaNs</a> to represent missing readings 1 - Fill missing readings (no <a href="#">QNaNs</a> ).

<b>Returns</b>	-1 0 >0	An error occurred. Use <a href="#">usb_tc08_get_last_error()</a> to get the code. Currently no readings to collect. 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`

```
short usb_tc08_legacy_run
(
    short handle
)
```

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

<b>Arguments</b>	<code>handle,</code>	Specifies the USB TC-08 unit.
<b>Returns</b>	0	Use <a href="#">usb_tc08_get_last_error()</a> .
	1	Legacy run successful.

```

3.3.2  usb_tc08_legacy_set_channel
        short usb_tc08_legacy_set_channel
        (
            short handle,
            short channel,
            char  tc_type,
            short filter_factor,
            short offset,
            short 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\(\)](#).

<b>Arguments</b>	<code>handle,</code>	Specifies the USB TC-08 unit.
	<code>channel,</code>	Specifies which channel you want to set the details for: This should be between 0 and 8.
	<code>tc_type,</code>	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.
	<code>filter_factor,</code>	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.
	<code>offset and slope,</code>	The parameters <code>offset</code> and <code>slope</code> are not used and therefore should be set to 0.
<b>Returns</b>	0	Use <a href="#">usb_tc08_get_last_error()</a> .
	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
        short usb_tc08_legacy_get_temp
        (
            long * temp,
            short handle,
            short channel,
            short 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.

<b>Arguments</b>	<code>temp,</code>	Pointer to a location where readings are to be placed. Each reading should be compared with 2147483647L or <code>LONG_MAX</code> (include <code>limits.h</code> ) to check for overflows. An overflow occurs when the input signal is higher than the measuring range of the USB TC-08.
	<code>handle,</code>	Specifies the USB TC-08 unit.
	<code>channel,</code>	Specifies from which channel to read temperature. Should be 0 for CJC, 1 for Channel 1, 2 for Channel 2 and so on.
	<code>filtered,</code>	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 <code>temp</code> . The filter has a depth defined by the filter factor, set during a <a href="#">usb_legacy_set_channel()</a> call.
<b>Returns</b>	0	Use <a href="#">usb_tc08_get_last_error()</a> .
	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
        short usb_tc08_legacy_get_cold_junction
        (
            long * temp,
            short 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.

<b>Arguments</b>	<code>temp,</code>	Pointer to a location where the temperature reading is to be stored.
	<code>handle,</code>	Specifies the USB TC-08 unit.
<b>Returns</b>	0	Use <a href="#">usb_tc08_get_last_error()</a> .
	1	Retrieval of cold junction temperature reading successful.



```
3.3.5  usb_tc08_legacy_get_driver_version
        short 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.

<b>Arguments</b>	None
<b>Returns</b>	Driver version number

```
3.3.6  usb_tc08_legacy_get_version
        short usb_tc08_legacy_get_version
        (
            short * version,
            short  handle
        )
```

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

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

```

3.3.7  usb_tc08_legacy_get_cycle
        short usb_tc08_legacy_get_cycle
        (
            long * cycle,
            short 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.

<b>Arguments</b>	<code>cycle,</code>	Pointer to a location where the cycle number is to be stored.
	<code>handle,</code>	Specifies the USB TC-08 unit.
<b>Returns</b>	0,	Use <a href="#">usb_tc08_get_last_error()</a> .
	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 Programming examples

The USB TC-08 SDK contains examples for the following programming languages:

- [C and C++](#)
- [Delphi](#)
- [Excel](#)
- [LabVIEW](#)
- [Visual Basic](#)

### 4.1 C and C++

#### C

The C example program is a generic Windows application: it does not use Borland AppExpert or Microsoft AppWizard. To compile the program, create a new project containing the following files:

- `usb_tc08.c`
- `usb_tc08.rc`
- either `usbtc08bc.lib` (Borland 32-bit applications)
- or `usbtc08.lib` (Microsoft Visual C 32-bit applications)

The following files must be in the same directory:

- `usb_tc08.rch`
- `usbtc08.h`
- `usbtc08.dll` (all 32-bit applications)

#### C++

C++ programs can access all versions of the driver. If `usbtc08.h` is included in a C++ program, the `PREF1` macro expands to `extern "C"`: this disables name-decoration, and enables C++ routines to make calls to the driver routines using C headers.

### 4.2 Delphi

The SDK contains a simple program `usbtc08.dpr` which opens the drivers and reads temperatures from the three channels. You will need the following files to build a complete program.

```
USBTC08FM.dfm
USBTC08FM.PAS
usbtc08.inc
```

The file `usbtc08.inc` contains procedure prototypes for the driver routines. You can include this file in your application.

This example has been tested with Delphi versions 1, 2 and 3.

### 4.3 Excel

The easiest way to transfer data to Excel is to use the PicoLog software application.

If, however, you need to do something that is not possible using PicoLog, you can write an Excel macro which calls `usbtc08.dll` to read in a set of data values. The Excel Macro language is similar to Visual Basic.

The example `usbTC0832.XLS` reads in 20 values of the cold junction temperature and channel 1 temperature, one per second, and assigns them to cells A1..B20.

### 4.4 LabVIEW

While it is possible to access all of the driver routines described earlier, it is easier to use the special LabVIEW access routine.

To use this routine, copy `usbtc08.vi` from the SDK to your LabVIEW `user.lib` directory. You will then need `usb_tc08` sub-vi and an example sub-vi which demonstrate how to use them. You can use one of these sub-vis for each of the channels that you wish to measure. The sub-vi returns a temperature for thermocouple types.

### 4.5 Visual Basic

The SDK contains the following files, created in Visual Basic 6:

```
usbtc0832.vbp
usbtc0832.bas
usbtc0832.Frm
usbtc0832.vbw
```

## 5 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	<b>P*</b>
2	USBTC08_ERROR_NO_CHANNELS_SET	<b>P</b>
3	USBTC08_ERROR_INVALID_PARAMETER	<b>P</b>
4	USBTC08_ERROR_VARIANT_NOT_SUPPORTED	<b>P*</b>
5	USBTC08_ERROR_INCORRECT_MODE	<b>P</b>
6	USBTC08_ERROR_ENUMERATION_INCOMPLETE	<b>S</b>
7	USBTC08_ERROR_NOT_RESPONDING	<b>S</b>
8	USBTC08_ERROR_FW_FAIL	<b>S</b>
9	USBTC08_ERROR_CONFIG_FAIL	<b>S</b>
10	USBTC08_ERROR_NOT_FOUND	<b>S</b>
11	USBTC08_ERROR_THREAD_FAIL	<b>S</b>
12	USBTC08_ERROR_PIPE_INFO_FAIL	<b>S</b>
13	USBTC08_ERROR_NOT_CALIBRATED	<b>S</b>
14	USBTC08_ERROR_PICOPP_TOO_OLD	<b>S</b>

Key	
<b>S</b>	<p>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:</p> <ol style="list-style-type: none"> <li>1. Uninstall the software and restart the PC</li> <li>2. Reinstall the software and restart the PC</li> <li>3. If this does not work, download the latest version of the software from <a href="http://www.picotech.com/download.html">http://www.picotech.com/download.html</a> and install this, then restart the PC</li> <li>4. If this does not work, contact <a href="mailto:support@picotech.com">support@picotech.com</a></li> </ol>
<b>P</b>	<p>Errors in this category are handled internally by software applications, and only developers need to be aware of their meanings.</p>
<b>P*</b>	<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.</p>

## 6 Glossary

**CJC.** A method of compensating for ambient temperature variations in thermocouple circuits.

**Cold junction compensation.** See **CJC**.

**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 no two QNaNs have the same value.

In C/C++, the **int \_isnan(double)** function in the **<float.h>** header 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  $-270\text{ }^{\circ}\text{C}$  to  $+1370\text{ }^{\circ}\text{C}$  temperature range. Sensitivity is approx  $41\text{ }\mu\text{V}/^{\circ}\text{C}$ .

**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  $1760\text{ }^{\circ}\text{C}$ . Low sensitivity ( $10\text{ }\mu\text{V}/^{\circ}\text{C}$ ) 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  $1760\text{ }^{\circ}\text{C}$ . Low sensitivity ( $10\text{ }\mu\text{V}/^{\circ}\text{C}$ ) 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  $-270\text{ }^{\circ}\text{C}$  to  $+400\text{ }^{\circ}\text{C}$  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 Mbps (12 megabits per second), and is much faster than a COM port.

**USBTC08\_INFO.** This structure is used to receive information from the [usb\\_tc08\\_get\\_unit\\_info\(\)](#) function and is defined in the `usbtc08.h` header file. Note: if the programming language you are using does not support structures, use the [usb\\_tc08\\_get\\_formatted\\_info\(\)](#) function.

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



# Index

## C

Contact details 3

## D

DLLs 1

Driver routines

- usb\_tc08\_close\_unit 18
- usb\_tc08\_get\_formatted\_info 23
- usb\_tc08\_get\_last\_error 24
- usb\_tc08\_get\_minimum\_interval\_ms 21
- usb\_tc08\_get\_single 28
- usb\_tc08\_get\_temp 29
- usb\_tc08\_get\_unit\_info 22
- usb\_tc08\_legacy\_get\_cold\_junction 36
- usb\_tc08\_legacy\_get\_cycle 39
- usb\_tc08\_legacy\_get\_driver\_version 37
- usb\_tc08\_legacy\_get\_temp 35
- usb\_tc08\_legacy\_get\_version 38
- usb\_tc08\_legacy\_run 33
- usb\_tc08\_legacy\_set\_channel 34
- usb\_tc08\_open\_unit 13
- usb\_tc08\_open\_unit\_async 15
- usb\_tc08\_open\_unit\_progress 17
- usb\_tc08\_run 27
- usb\_tc08\_set\_channel 26
- usb\_tc08\_set\_mains 20
- usb\_tc08\_stop 19
- usb\_tc08\_temp\_deskew 31

## E

Error codes 24, 42

## F

Functions, list of 12

Further information 1

## I

Installation 1

## L

Legal information 2

## M

Modes of operation

- Get single mode 4, 7

- Legacy mode 4, 10

- Streaming mode 4, 5

## P

Programming 1

- C 40

- C++ 40

- Delphi 40

- Excel 41

- LabVIEW 41

- Visual Basic 41

## Q

QNaN 29, 31

## T

Troubleshooting 42







## 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  
[www.picotech.com](http://www.picotech.com)

usbtc08pg.en r3 2013-03-11

Copyright © 2005-2013 Pico Technology Limited. All rights reserved.