Bertec Device Interface Library

Developer Documentation

Version 1.82
March 2014

Copyright © 2009-2014 BERTEC Corporation. All rights reserved. Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without express written permission of BERTEC Corporation or its licensees.

"Measurement Excellence", "Dominate Your Field", BERTEC Corporation, and their logos are trademarks of BERTEC Corporation. Other trademarks are the property of their respective owners.

Printed in the United States of America.

Bertec's authorized representative in the European Community regarding CE:

Bertec Limited
31 Merchiston Park
Edinburgh EH10 4 PW
Scotland, United Kingdom

CE

SOFTWARE LICENSE AGREEMENT

This License Agreement is between you ("Customer") and Bertec Corporation, the author of the Bertec Device DLL software and governs your use of the of the dynamic link libraries, example source code, and documentation (all of which are referred to herein as the "Software").

PLEASE READ THIS SOFTWARE LICENSE AGREEMENT CAREFULLY BEFORE DOWNLOADING OR USING THE SOFTWARE. NO REFUNDS ARE POSSIBLE. BY DOWNLOADING OR INSTALLING THE SOFTWARE, YOU ARE CONSENTING TO BE BOUND BY THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, DO NOT DOWNLOAD OR INSTALL THE SOFTWARE.

• Bertec Corporation grants Customer a non-exclusive right to install and use the Software for the express purposes of connecting with Bertec Devices for data gathering purposes. Other uses are prohibited.

• Customer may make archival copies of the Software provided Customer affixes to such copy all copyright, confidentiality, and proprietary notices that appear on the original.

• The Customer may not resell the Software or otherwise represent themselves as the owner of said software.

The binary redistributables are royalty free to the original Licensee and can be distributed with applications, provided that proper attribution is made in the documentation and end user agreement. Binary redistributables include:

1. BertecDeviceDLL.dll
2. ftd2xx.dll

Note that the FTD2XX.DLL is a USB driver provided by Future Technology Devices that enables communication with the Bertec Device.

The binary redistributables cannot be used by third parties to build applications or components.

Customer created binary redistributables from the Software source code cannot be used by anyone, including the original license holder, to create a product that competes with Bertec Corporation products. Neither the original nor altered source code may be distributed.

EXCEPT AS EXPRESSLY AUTHORIZED ABOVE, CUSTOMER SHALL NOT: COPY, IN WHOLE OR IN PART, SOFTWARE OR DOCUMENTATION; MODIFY THE SOFTWARE; REVERSE COMPILE OR REVERSE ASSEMBLE ALL OR ANY PORTION OF THE SOFTWARE; OR RENT, LEASE, DISTRIBUTE, SELL, MAKE AVAILABLE FOR DOWNLOAD, OR CREATE DERIVATIVE WORKS OF THE SOFTWARE OR SOURCE CODE.

Customer agrees that aspects of the licensed materials, including the specific design and structure of individual programs, constitute trade secrets and/or copyrighted material of Bertec Corporation. Customer agrees not to disclose, provide, or otherwise make available such trade secrets or copyrighted material in any form to any third party without the prior written consent of Bertec Corporation. Customer agrees to implement reasonable security measures to protect such trade secrets and copyrighted material. Title to Software and documentation shall remain solely with Bertec Corporation.

No Warranty

THE SOFTWARE IS BEING DELIVERED TO YOU "AS IS" AND BERTEC CORPORATION MAKES NO WARRANTY AS TO ITS USE, RELIABILITY OR PERFORMANCE. BERTEC CORPORATION DOES NOT AND CANNOT WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THE SOFTWARE. BERTEC CORPORATION MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AS TO NONINFRINGEMENT OF THIRD PARTY RIGHTS, TITLE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. YOU ASSUME ALL RISK ASSOCIATED WITH THE QUALITY, PERFORMANCE, INSTALLATION AND USE OF THE SOFTWARE INCLUDING, BUT NOT LIMITED TO, THE RISKS OF PROGRAM ERRORS, DAMAGE TO EQUIPMENT, LOSS OF DATA OR SOFTWARE PROGRAMS, OR UNAVAILABILITY OR INTERRUPTION OF OPERATIONS. YOU ARE SOLELY RESPONSIBLE FOR DETERMINING THE APPROPRIATENESS OF USE OF THE SOFTWARE AND ASSUME ALL RISKS ASSOCIATED WITH ITS USE.

Indemnification

You agree to indemnify and hold Bertec Corporation, parents, subsidiaries, affiliates, officers and employees, harmless from any claim or demand, including reasonable attorneys' fees, made by any third party due to or arising out of your use of the Software, or the infringement by you, of any intellectual property or other right of any person or entity.

Limitation of Liability

IN NO EVENT WILL BERTEC CORPORATION BE LIABLE TO YOU FOR ANY INDIRECT, INCIDENTAL, SPECIAL, PUNITIVE, CONSEQUENTIAL, OR OTHER DAMAGES WHATSOEVER, OR ANY LOSS OF REVENUE, DATA, USE, OR PROFITS, EVEN IF BERTEC CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, AND REGARDLESS OF WHETHER THE CLAIM IS BASED UPON ANY CONTRACT, TORT OR OTHER LEGAL OR EQUITABLE THEORY.

This License is effective until terminated. Customer may terminate this License at any time by destroying all copies of Software including any documentation. This License will terminate immediately without notice from Bertec Corporation if Customer fails to comply with any provision of this License. Upon termination, Customer must destroy all copies of Software.

Software, including technical data, is subject to U.S. export control laws, including the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations in other countries. Customer agrees to comply strictly with all such regulations and acknowledges that it has the responsibility to obtain licenses to export, re-export, or import Software.

This License shall be governed by and construed in accordance with the laws of the State of Ohio, United States of America, as if performed wholly within the state and without giving effect to the principles of conflict of law. If any portion hereof is found to be void or unenforceable, the remaining provisions of this License shall remain in full force and effect. This License constitutes the entire License between the parties with respect to the use of the Software.

Should you have any questions concerning this Agreement, please write to:

Bertec Corporation, 6171 Huntley Road, Suite J, Columbus, Ohio 43229

TABLE OF CONTENTS

Introduction ______ ***6***

Definitions, Acronyms, and Abbreviations ______ ***7***

Using the DLL ______ ***8***

Using polled data to pull the data yourself ______ **9**

Using callbacks to get the data pushed to you ______ **9**

Error checking and handling ______ **10**

Data processing ______ **10**

Data format ______ **10**

Bertec Device DLL Functions ______ ***11***

bertec_Init ______ **11**

bertec_Close ______ **11**

bertec_GetState ______ **11**

bertec_Start ______ **11**

bertec_Stop ______ **12**

bertec_DataCallback typedef ______ **12**

bertec_RegisterDataCallback ______ **12**

bertec_UnregisterDataCallback ______ **12**

bertec_StatusCallback typedef ______ **13**

bertec_RegisterStatusCallback ______ **13**

bertec_UnregisterStatusCallback ______ **13**

bertec_DeviceSortCallback typedef ______ **14**

bertec_RegisterDeviceSortCallback ______ **14**

bertec_UnregisterDeviceSortCallback ______ **14**

bertec_RestartData ______ **15**

bertec_AllocatePollBuffer ______ **15**

bertec_GetAllocatePollBuffer ______ **15**

bertec_DataPoll ______ **15**

bertec_ClearPollBuffer ______ **17**

bertec_ClearStatusError ______ **17**

bertec_ClearOverflowError ______ **17**

bertec_SetAveraging 17
bertec_SetLowpassFiltering 18
bertec_ZeroNow 18
bertec_EnableAutozero 18
bertec_AutozeroState 18
bertec_TransducerSerialNumber 19
bertec_SetAcquireRate 19
bertec_GetZeroLevelNoiseValue 19
bertec_SetUsbThreadPriority 20
bertec_GetMayBeMissingSyncCable 20
bertec_ResetSyncCounters 20
Troubleshooting *21*
Document Revision History *22*

INTRODUCTION

The Bertec Device DLL for Windows provides the end-user developer or data acquisition expert a common and consistent method to gather data from Bertec equipment. Instead of directly communicating with the USB devices and implementing different protocols and calibrations for each, the Bertec Device DLL manages all interaction with the USB devices, and provides the calibrated captured data to your program or data analysis project. The DLL also provides zeroing of the plate data (either on-demand for tare loading, or automatic for low or no loading), sample averaging, and low-pass filtering.

The DLL provides data results in either a threaded callback or a non-threaded polled-data mode, depending on the needs and abilities of the data acquisition application.

The DLL exports its functionality as a set of C-level functions and structures for a Windows 32-bit environment, which can be used by most development environments or data acquisition programs that can accept C-level function calls. As long as your development system or data acquisition software can understand C structures and C function calls, then you should have no problems with using the DLL and the header file.

Sample code is provided in the BertecExample.cpp file.

If you are doing development in the .NET environment, please refer to the BertecDeviceNET.pdf file. This can also be used as a reference for COM-based development.

If you are doing development using JAVA, please refer to the BertecDeviceJAVA.pdf file.

If you have any developmental questions on using this library or SDK, please contact Bertec Corporation for support.

DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

Balance plate: a Bertec device that measures pressure and movement that is optimized for balance diagnostics.

Force plate: a Bertec device that measures pressure and movement.

Center of Pressure (CoP): The point on the surface of the platform through which the ground reaction force acts. It corresponds to the projection of the subject's center of gravity on the platform surface when the subject is motionless.

USING THE DLL

Getting data from an attached device generally consists of just a few function calls:

1. Call `bertec_Init`
2. Call either `bertec_AllocatePollBuffer` or `bertec_RegisterDataCallback`
3. Call `bertec_Start`
4. Poll using `bertec_DataPoll`, or use the threaded callback
5. Call `bertec_Stop`
6. End by calling `bertec_Close`

Step 1: `bertec_Init`

Calling `bertec_Init` will set up the internal data in the DLL and locate attached Bertec USB devices. It is the first thing you will need to do to use the Bertec Device DLL. The `bertec_Init` function returns a handle that you must store and use later to pass to the other `bertec_XXX` functions.

Step 2: `bertec_AllocatePollBuffer` or `bertec_RegisterDataCallback`

Depending on how your application works, you will either want to poll for the data yourself (pull) and process it, or else use the faster callback functionality (push). If polling, you will need to first tell the system how much internal buffer memory it should allocate. If using callbacks, you will need to register the callback with the system.

Step 3: `bertec_Start`

To actually gather data, you must call `bertec_Start` with the handle that `bertec_Init` returned to you. Doing so will start the data gathering process, and will start calling your callbacks if you have registered them.

Step 4: poll using `bertec_DataPoll`, or use the threaded callback

If you're using the data polling method, you will need to repeatably call `bertec_DataPoll` to gather the data; otherwise, the callbacks will be used.

Step 5: `bertec_Stop`

Once you have completed your data gathering call `bertec_Stop` to end all of the USB data reads and end buffering of the data. The system still remains active and data gathering can be resumed by calling `bertec_Start` again.

Step 6: `bertec_Close`

Once you are completely done with the system, you will need to call `bertec_Close` to stop all USB devices and release the device driver memory. Failing to do so could result in the Bertec devices continually running, and possible memory leaks.

USING POLLED DATA TO PULL THE DATA YOURSELF

In order to use polled data, you will need to inform the DLL how much internal buffer space it will need to reserve for itself. You do this using the `bertec_AllocatePollBuffer`, passing it the time in seconds you wish it to buffer for. If you don't do this, then calling `bertec_DataPoll` will fail. Here is a very simplistic example with no error handling:

```
bertec_AllocatePollBuffer(handle,2.750);
bertec_Start(handle);
int channelsOut;
double yourDataBuffer[8000];
while (bertec_DataPoll(handle,8000,yourDataBuffer,&channelsOut) > 0)
{
  processYourData(...);
}
bertec_Stop();
```

The `bertec_DataPoll` function will return either the number data items placed into the buffer, up to the size of the buffer, or else an error code.

The error codes are defined in the BERTECIF.H file, along with what to do about each one. See the section on Error Codes for more information.

USING CALLBACKS TO GET THE DATA PUSHED TO YOU

Callbacks are a very fast way to get data from the attached devices, and are the preferred method. The callbacks are performed in a separate thread from your application's main thread – this must be taken into consideration when designing your application.

To use callbacks, simply register your callback function with the optional user data value with `bertec_RegisterDataCallback`. You can register multiple unique callback functions, and they will each get called in turn. Unlike using the polled data functionality, you do not need to first allocate memory or a buffer. Here is a very simplistic example without error checking:

```
void myDataCallback(int samples, int channels, double * data, void * user)
{
  processYourData(...);
}
bertec_RegisterDataCallback(myDataCallback, NULL);
bertec_Start();
..your main program runs...
bertec_Stop();
```

ERROR CHECKING AND HANDLING

When using the data polling, you will need to check the return value from `bertec_DataPoll`. If the value is greater than zero, then there is valid data in the buffer that you passed to `bertec_DataPoll` of this total amount. If the return value is zero, there is no data on the wire to process. A value less than zero indicates an error condition. The error codes are defined in the BERTECIF.H file, along with what to do about each one. See the section on Error Codes for more information.

When using the callback, errors are given using the `samples` parameter. If this value is less than zero, it indicates an error condition. Again, the error codes are defined in the BERTECIF.H file, along with what to do about each one. See the section on Error Codes for more information.

Status errors are also sent using the `bertec_StatusCallback` function. These callbacks are sent whenever the status of the DLL changes, either to an error condition or when cleared.

Generally speaking, the Bertec Device DLL will automatically handle most error conditions that you would otherwise have to design for. If the user unplugs a device, and then reconnects it, the DLL will handle this and restart the device. However, the system *will not* attempt to locate freshly attached devices that are plugged in while the DLL is in use. If the Bertec device is turned off, or disconnected from the USB converter box, the DLL will notice the lack of incoming data and attempt to restart the device once it is reconnected.

DATA PROCESSING

Since data can flow into the computer in a very rapid rate, it is critical that your program handle it as promptly as possible – buffering it in a large pre-allocated memory block is preferred. Should data not be read fast enough, it will start to be lost, and you will get notifications either through the return code from `bertec_DataPoll` or the `samples` error value in the callback.

DATA FORMAT

The buffered data that is presented via callbacks or data polling is an array of double values, already calibrated for each device, grouped in sets of samples. What each value means is specific to the channel of that device. Each sample contains all channels of all attached transducers, starting from `bertec_State::transducers[0]` and going up to the value in `bertec_State::transducerCount.`

Each sample is therefore arranged like this:

sample 0: tr0ch0,tr0ch1,tr0ch2,tr0ch3, tr1ch0,tr1ch1

sample 1: tr0ch0,tr0ch1,tr0ch2,tr0ch3, tr1ch0,tr1ch1

etc.

Both the callback and data poll return to your code the total number of channels for all transducers. In the above samples, this value would be 6.

BERTEC DEVICE DLL FUNCTIONS

All of the functions are "C" exported function calls. For each function call, the name is called out, and then the function call definition is provided, along with documentation. For .NET and Java interfaces, please see the appropriate documentation.

BERTEC_INIT

```
bertec_Handle bertec_Init(void)
```

The `bertec_Init` function initializes the library and does initial communication with the USB devices. It returns a handle pointer to the `bertec_State` data that you can use to iterate through the attached devices, and their corresponding channels. You must first call the `bertec_Init` function before using any other method. You should only call this once. It does *not* start the data gathering process – you must call `bertec_Start` to do so. Calling this multiple times will close the existing handle and re-init the system.

BERTEC_CLOSE

```
void bertec_Close(bertec_Handle theHandle)
```

Call the `bertec_Close` to shut down all devices, unregister all callbacks, and stop acquisition. You must pass it the same handle that `bertec_Init` provided. Failing to call this when you have completed data acquisition will leave the devices running and possibly introduce memory leaks. Calling this multiple times will have no effect.

BERTEC_GETSTATE

```
bertec_State * bertec_GetState(bertec_Handle theHandle)
```

This function will return the current state of the system. Do not delete this pointer, or otherwise modify the data. You may call this function whenever your program needs to. If you do not pass in the correct handle, the function will return a NULL value.

BERTEC_START

```
int bertec_Start(bertec_Handle theHandle)
```

This function starts the data gathering process, invoking the callbacks if they are registered, and buffering incoming data as needed. The function will return a zero value if the process is started correctly, otherwise it will return a `BERTEC_ERROR_INVALIDHANDLE` return code.

BERTEC_STOP

`int bertec_Stop(bertec_Handle theHandle)`

This function stops the data gathering process. Callbacks will no longer be called, and calling the data polling function will return an error. The function will return a zero value for success, otherwise it will return a `BERTEC_ERROR_INVALIDHANDLE` return code.

BERTEC_DATACALLBACK TYPEDEF

BERTEC_REGISTERDATACALLBACK

BERTEC_UNREGISTERDATACALLBACK

```
void bertec_DataCallback(bertec_Handle theHandle, int samples, int channels,
double * data, void * userData)
```

```
int bertec_RegisterDataCallback(bertec_Handle theHandle, bertec_DataCallback fn,
void * userData)
```

```
int bertec_UnregisterDataCallback(bertec_Handle theHandle, bertec_DataCallback fn,
void * userData)
```

To use the data callback functionality in the system, you will need to register your callback function with `bertec_RegisterDataCallback`. The system can support multiple data callbacks. To stop using the callback without stopping data acquisition, call `bertec_UnregisterDataCallback`. Calling `bertec_Close` will automatically unregister all callbacks.

In order to properly unregister the callback, you must call `bertec_UnregisterDataCallback` with the same values as you registered it with – both the callback itself, and the value of `userData`. Failing to do so will not unregister the callback and will return a negative result code.

Both functions return zero for success.

Your data callback function should be a C-style function, using the "standard call" specification. This is the default for most development environments.

The callbacks that are registered will be called each time there is data left in the internal buffer to be processed – each callback will be called with the same data values. The `userData` parameter is set when your register the callback, and is not used by the Bertec Device DLL, but is to be used by your callback for whatever it needs – for example, in C++ it could be used to pass a pointer to a class object. The pointer to the data that contains the samples will contain a total number of values equal to samples times channels. Do not delete, free, or otherwise modify this data buffer – it is maintained internally by the Device DLL.

Since data gathering is time-critical, it is important that you process the data as fast as possible and return.

You should register your callbacks before calling `bertec_Start` in order to insure that no data is lost.

See the section on Returned Data for information about the format and type of the data block.

BERTEC_STATUSCALLBACK TYPEDEF

BERTEC_REGISTERSTATUSCALLBACK

BERTEC_UNREGISTERSTATUSCALLBACK

`void bertec_StatusCallback(bertec_Handle theHandle, int status, void * userData)`

`int bertec_RegisterStatusCallback(bertec_Handle theHandle, bertec_StatusCallback fn, void * userData)`

`int bertec_UnregisterStatusCallback(bertec_Handle theHandle, bertec_StatusCallback fn, void * userData)`

To use the status callback functionality in the system, you will need to register your callback function with `bertec_RegisterStatusCallback`. The system can support multiple callbacks. To stop using the callback, call `bertec_UnregisterStatusCallback`. Calling `bertec_Close` will automatically unregister all callbacks.

In order to properly unregister the callback, you must call `bertec_UnregisterStatusCallback` with the same values as you registered it with – both the callback itself, and the value of `userData`. Failing to do so will not unregister the callback and will return a negative result code.

Both functions return zero for success.

Your status callback function should be a C-style function, using the “standard call” specification. This is the default for most development environments.

The callbacks that are registered will be called each time there is a *change* in the status of code of the DLL – they will *not* be called unless the `status` value changes from N to X. The `status` value passed to the callback is the same as in the `bertec_State::status` value, and is provided only as a convenience factor.

The `userData` parameter is set when your register the callback, and is not used by the Bertec Device DLL, but is to be used by your callback for whatever it needs – for example, in C++ it could be used to pass a pointer to a class object.

Since data gathering is time-critical, and this callback is made in the context of the gathering process, it is important that you process the status change as fast a possible and return.

BERTEC_DEVICESORTCALLBACK TYPEDEF

BERTEC_REGISTERDEVICESORTCALLBACK

BERTEC_UNREGISTERDEVICESORTCALLBACK

`void bertec_DeviceSortCallback (bertec_TransducerInfo* pInfos, int transducerCount, int* orderArray, void * userData)`

`int bertec_RegisterDeviceSortCallback(bertec_Handle bHand, bertec_DeviceSortCallback fn, void * userData)`

`int bertec_UnregisterDeviceSortCallback(bertec_Handle bHand, bertec_DeviceSortCallback fn, void * userData)`

To use the device sort order callback functionality in the system, you will need to register your callback function with `bertec_RegisterDeviceSortCallback`. The system can support multiple callbacks. To stop using the callback, call `bertec_UnregisterDeviceSortCallback`. Calling `bertec_Close` will automatically unregister all callbacks.

In order to properly unregister the callback, you must call `bertec_UnregisterDeviceSortCallback` with the same values as you registered it with – both the callback itself, and the value of `userData`. Failing to do so will not unregister the callback and will return a negative result code.

Both functions return zero for success.

Your data callback function should be a C-style function, using the “standard call” specification. This is the default for most development environments.

The callbacks that are registered will be called each time the library reloads the list of devices attached to the computer. The `userData` parameter is set when your register the callback, and is not used by the Bertec Device DLL, but is to be used by your callback for whatever it needs – for example, in C++ it could be used to pass a pointer to a class object.

The pInfos pointer is the list of devices discovered, and the orderArray should be manipulated by your callback to re-order which device is #1, which is #2, etc. By default, the USB hardware orders the devices based on internal identifiers, which may or may not be order you wish to have, and the orderArray is filled in with [0,1,2,3…] . By examining the pInfos array (typically the serial# of the device), and changing the index values in orderArray, you can tell the library to move a device in front of others; the ordering of devices is reflected in the outgoing data stream. This allows your project to always have a consistent ordering of devices without the overhead of dealing with the USB system ordering.

Do not delete, free, or otherwise modify the pInfos array – it is maintained internally by the Device DLL. You should not delete or free the orderArray, but it is expected that you change it to reflect your new device order.

BERTEC_RESTARTDATA

`void bertec_RestartData(bertec_Handle theHandle)`

When using the callback, if an error is indicated, the callbacks will be suspended and no longer called. In order to resume the callback processing, you must call this with the handle in order re-enable them.

BERTEC_ALLOCATEPOLLBUFFER

`void bertec_AllocatePollBuffer(bertec_Handle theHandle, double timeInSeconds)`

To use data polling instead of callbacks, you must first call `bertec_AllocatePollBuffer` to set up the internal buffer and tell the system how much data you wish to buffer. The system will take the passed value and create an internal buffer larger enough to at least hold that amount for all channels. If you do not do this, then the data poll will return an error. You should call this function before calling `bertec_Start.` The allocated memory is released when `bertec_Close` is called.

Note that calling this again will re-allocate the internal buffer, resulting in a loss of any data remaining in the internal buffer.

BERTEC_GETALLOCATEPOLLBUFFER

`double bertec_GetAllocatePollBuffer(bertec_Handle theHandle)`

Returns the current value that was set in `bertec_AllocatePollBuffer.` This can be used to double-check that the value passed to `bertec_AllocatePollBuffer` was accepted. Returns zero if the handle is incorrect.

BERTEC_DATAPOLL

`int bertec_DataPoll(bertec_Handle theHandle, int bufSize, double * dataBuf, int * channels)`

Instead of using the callbacks, you can use the `bertec_DataPoll` function to periodically pull the data from the internal buffer set up with the `bertec_AllocatePollBuffer` call. You must first have a buffer set up large enough to hold at least once sample set of data, which is the sum of all the values of `bertec_State::transducers[]::channelCount`. This sum value is returned to you in the channels variable that you provide to the function; if you provide NULL for this value, then the channel count is not provided.

You must call this function with a large enough buffer, often enough, to keep up with the data being gathered and buffered by the system. Failing to do so will result in an error condition and loss of data.

The function returns a positive number for how many total data entry values, in terms of samples times channels. Thus if the function returns 20, and you have 5 channels, then there are 100 data samples in the buffer. See the section on Returned Data for information about the format and type of the data block.

If the function returns a negative number, it will be one of the BERTEC_DATA_POLL errors:

Error	Value	Means
`BERTEC_DATA_POLL_NOUSERBUFFER`	-1	The buffer size passed was too small to hold one sample. Increase the size of the buffer.
`BERTEC_DATA_POLL_NOPOLLBUFFER`	-2	There are no internal buffers allocated – you need to call `bertec_AllocatePollBuffer` first.
`BERTEC_DATA_POLL_CHECKSTATUS`	-3	A status change has occurred – either a device has been unplugged or some other error occurred. You will need to check the device's status and call `bertec_ClearStatusError` to clear this error.
`BERTEC_DATA_POLL_OVERFLOW`	-4	The polling wasn't performed for long enough, and data has been lost. You will need to call `bertec_ClearOverflowError` to clear this condition
`BERTEC_DATA_POLL_NODEVICES`	-5	There are apparently no devices attached. Attach a device.
`BERTEC_DATA_POLL_NOTSTARTED`	-6	You have not called `bertec_Start` yet.
`BERTEC_DATA_SYNCHRONIZING`	-7	Synchronizing, data not available yet (this value is in `bertec_State::status` during callbacks, as the return code when polling)
`BERTEC_DATA_SYNCHRONIZE_LOST`	-8	If multiple plates are connected with the Bertec Sync option, the plates have lost sync with each other - check the sync cable.
`BERTEC_DATA_SEQUENCE_MISSED`	-9	One or more plates have missing data sequence - data may be invalid
`BERTEC_DATA_SEQUENCE_REGAINED`	-10	The plates have regained their data sequence
`BERTEC_NO_DATA_RECEIVED`	-11	No data is being received from the devices, check the cables

`BERTEC_DEVICE_HAS_FAULTED`	-12	The device has failed in some manner. Power off the device, check all connections, power back on
`BERTEC_DATA_POLL_DEVICESREADY`	-50	There are plates to be read
`BERTEC_AUTOZEROSTATE_WORKING`	-51	Currently finding the zero values
`BERTEC_AUTOZEROSTATE_ZEROFOUND`	-52	The zero leveling value was found

BERTEC_CLEARPOLLBUFFER

`int bertec_ClearPollBuffer(bertec_Handle bHand)`

Clears all of the data that is currently in the polling buffer. Only useful if *not* using callbacks.

BERTEC_CLEARSTATUSERROR

`bertec_State * bertec_ClearStatusError(bertec_Handle bHand)`

Call this when you have received an error code in either the data poll or callback functions in order to continue to get data.

BERTEC_CLEAROVERFLOWERROR

`int bertec_ClearOverflowError(bertec_Handle bHand)`

If you get an overflow condition in your error code, you will need call this function to clear it. It will return how many samples have been lost. Note that you might have lost additional data since your code was notified of this error and when you call this function.

BERTEC_SETAVERAGING

`int bertec_SetAveraging(bertec_Handle bHand,int samplesToAverage)`

Averages the number of samples from the devices, reducing the apparent data rate and result data by the value of `samplesToAverage` (ex: a value of 5 will cause 5 times less data to come out). The `samplesToAverage` value should be >= 2 in order for averaging to be enabled. Setting `samplesToAverage` to 1 or less will turn off averaging (the default).

BERTEC_SETLOWPASSFILTERING

`int bertec_SetLowpassFiltering(bertec_Handle bHand,int samplesToFilter)`

Performs a running average of the previous `samplesToFilter`, making the input data stream to appear smoother. The `samplesToFilter` value should be >= 2 in order to turn the filter on. Setting `samplesToFilter` to 1 or less will turn filtering off (the default). This does not affect the total number of samples gathered.

BERTEC_ZERONOW

`int bertec_ZeroNow(bertec_Handle bHand)`

By default, the data from the devices is not zeroed out. Calling the `bertec_ZeroNow` function with *any* load on the device will sample the data for a fixed number of seconds, and then use the loaded values as the zero baseline (this is sometimes called "tare" for simpler load plates). Calling this after calling `bertec_Start` will cause your data stream to rapidly change values as the new zero point is taken. This can be used in conjunction with `bertec_EnableAutozero`.

BERTEC_ENABLEAUTOZERO

`int bertec_EnableAutozero(bertec_Handle bHand,int enableFlag)`

The DLL has the ability to automatically re-zero the plate devices when it detects a low- or no-load condition (less than 40 newtons for at least 3.5 seconds). Calling this function with a non-zero value for `enableFlag` will cause the DLL to monitor the data stream and continually reset the zero baseline values. Call this function with a zero value for `enableFlag` to turn this off. This functionality will not interrupt your data stream, but you will get a sudden shift in values as the DLL applies the zero baseline initially.

BERTEC_AUTOZEROSTATE

`int bertec_AutozeroState(bertec_Handle bHand)`

This function returns the current state of the autozero system. Your program will need to poll this on occasion to find the current state – there is no callback for when it changes.

The following values are returned from `bertec_AutozeroState` :

State	Value	Means
`AUTOZEROSTATE_NOTENABLED`	0	Autozeroing is currently not enabled.
`AUTOZEROSTATE_WORKING`	1	Autozero is currently looking for a sample to zero against.

AUTOZEROSTATE_ZEROFOUND	2	The zero level has been found. Note that once this value is returned, `bertec_AutozeroState` will always return it until you disable Autozeroing.

BERTEC_TRANSDUCERSERIALNUMBER

`int bertec_TransducerSerialNumber(bertec_Handle bHand,int transducerIndex,char *buffer,int bufferSize)`

This is a convenience function that will return a transducer's serial number. This is the same as accessing the `bertec_state` structure. Call this function with a valid handle value, the index of the transducer you want to get the serial number for (zero-based), the character buffer, and the size of the buffer. Returns 0 if the serial number was successfully gotten, or else BERTEC_ERROR_INVALIDHANDLE.

BERTEC_SETACQUIRERATE

`int bertec_SetAcquireRate(bertec_Handle bHand,int rateValue)`

The DLL defaults to a data push rate of approximately 5Hz. On faster equipment that might be used in a near-realtime environment, you can change this up to a rate of 20Hz (`speedValue` equals 1).

Note that this directly effects how often the data *callback* is given data, and how much data it is given. ***It does not effect how the device is read, only how much is pushed to your application in the callback. This has no effect on the DataPoll function.*** For example, at the default `speedValue` of 10, the data callback is called with about 240 samples every 210ms. Changing this value to 2 causes the callback to be called with 52 samples every 47ms, and a value of 1 is equal to about 30 samples every 31ms. At the other extreme, a `speedValue` of 50 equates to 1025 samples at around 1016ms.

The number of samples and effective values of `rateValue` are dependent upon both the Bertec Device and the Windows platform you are using.

There are very few conditions where you will need to change the rate.

Plates that do advanced checksum encoding will ignore the rate change, since they control the rate themselves.

BERTEC_GETZEROLEVELNOISEVALUE

`double bertec_GetZeroLevelNoiseValue(bertec_Handle bHand,int transducerIndex,int channelIndex)`

Returns the zero level noise value for a device and channel. Either `bertec_ZeroNow` or `bertec_EnableAutozero` must have been called prior to this function being used. The value returned is a computed value that can be used for advanced filtering. Valid values are always zero or positive; negative values indicate either no zeroing or some other error.

BERTEC_SETUSBTHREADPRIORITY

`void bertec_SetUsbThreadPriority(bertec_Handle bHand,int priority)`

This function allows your code to change the priority of the internal USB reading thread. Typically, this is not something you will need to do unless you feel that the USB interface needs more or less of the thread scheduling that Windows performs. The priority value can range from -15 (lowest possible) to 15 (highest possible – this will more than likely prevent your GUI from running). The default system scheduling of the USB reading thread should be suitable for most applications.

BERTEC_GETMAYBEMISSINGSYNCCABLE

`int bertec_GetMayBeMissingSyncCable(bertec_Handle bHand)`

If there are multiple devices, this function will attempt to detect if the sync cable is missing, disconnected, or loose. Will return a value of 1 if the cable appears to be missing, or returns 0 if the sync cable appears to be good. Only valid when used with multiple devices connected through 650x amps with a sync cable connection.

BERTEC_RESETSYNCCOUNTERS

`int bertec_ResetSyncCounters(bertec_Handle bHand)`

If there are multiple devices, this function will reset the internal counters that account for sync offset and drifts. This is an advanced function that is typically not used and does nothing if there is only a single device connected. Returns 0 for success.

TROUBLESHOOTING

If your application will not launch, make sure that both the BertecDeviceDLL.dll and ftd2xx.dll are in the same folder as your application.

For any other issues, please contact Bertec Technical Support.

DOCUMENT REVISION HISTORY

Date	Revision	Description	Author
01/15/2009	1.00	Initial Revision	Todd Wilson
03/28/2012	1.80	Updated with current version	Todd Wilson
06/30/2012	1.81	Removed Sync Drift function; added Sync Cable and Sync Counter Reset functions and additional status codes.	Todd Wilson
3/24/2014	1.82	Added sort ordering, usb thread priority functions, corrected typographical errors	Todd Wilson