

BLUETECHNIX  
Embedding Ideas

---

# BltTofApi SDK

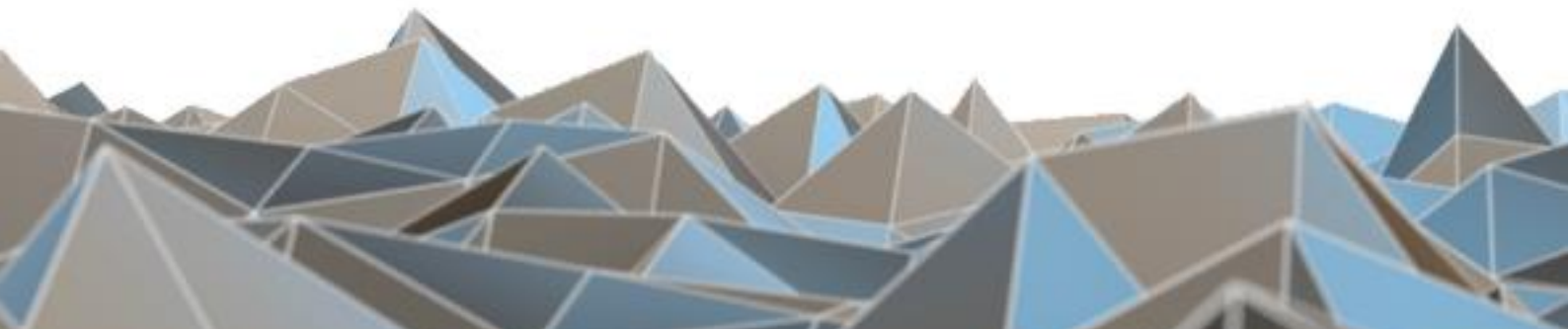
---

Software User Manual

---

Version 8

---



## Contact

Bluetechnix R&D GmbH  
Gutheil-Schoder-Gasse 17, 1230 Wien  
AUSTRIA  
[office@bluetechnix.com](mailto:office@bluetechnix.com)  
<http://www.bluetechnix.com>

Date: 2015-05-21

## Table of Contents

---

1	Introduction .....	5
1.1	Purpose of the document .....	5
1.2	References .....	5
2	Overview.....	6
3	Specification.....	7
4	Interface.....	8
5	References .....	16
6	Document Revision History.....	17
A	List of Figures and Tables .....	18

© Bluetechnix R&D GmbH 2015  
All Rights Reserved.

The information herein is given to describe certain components and shall not be considered as a guarantee of characteristics.

Terms of delivery and rights of technical change reserved.

We hereby disclaim any warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Bluetechnix makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. Bluetechnix specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Bluetechnix takes no liability for any damages and errors causing of the usage of this board. The user of this board is responsible by himself for the functionality of his application. He is allowed to use the board only if he has the qualification. More information is found in the General Terms and Conditions (AGB).

#### **Information**

For further information on technology, delivery terms and conditions and prices please contact Bluetechnix <http://www.bluetechnix.com>.

#### **Warning**

Due to technical requirements components may contain dangerous substances.

# 1 Introduction

## 1.1 Purpose of the document

This document explains the usage of the Bluetechnix ToF API. It does not cover device specific information or any implementation related information. It only describes the interface.

## 1.2 References

Document Name	Version	Path to Document	Hereinafter referred to as
<b>bta.h</b>	1.6.0	Inc/	
<b>bta_frame.h</b>	1.6.0	Inc/	
<b>bta_status.h</b>	1.6.0	Inc/	
<b>bta_discovery.h</b>	1.6.0	Inc/	
<b>bta_flash_update.h</b>	1.6.0	Inc/	
<b>bta_event.h</b>	1.6.0	Inc/	

Table 1.1: Reference Overview

## 2 Overview

In order to create a common interface for our products we define the interfaces between a ToF device and an application. The main part of this model is the BltTofApi which is written in C for platform independency. The already existing BltTofApiExt (used by BltTofSuite) is able to access the BltTofApi interface and will therefore be compatible with any device with existing lib implementing the BltTofApi.

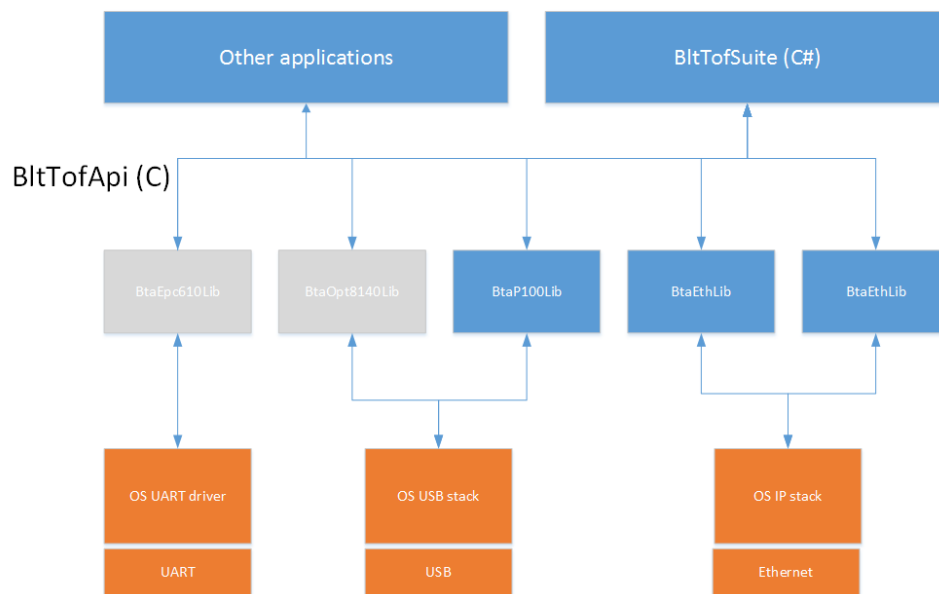


Figure 2-1: Interfacing concept

Every ToF system built by or for Bluetechnix shall be accessible by this common interface. A lib implementing this interface shall be written in C only and compile on any platform. The Interface is kept as simple as possible and covers all functionalities of all ToF sensors.

Before implementing and building the following examples, please check the support wiki Section [BltTofApi Build instructions](#). When a status returned by any interface is not comprehensible, consult the list of error codes in the support wiki Section [BltTofApi Error codes](#).

### 3 Specification

The pixel order and coordinate system as received from any BtaXXXLib is as described in the following image.

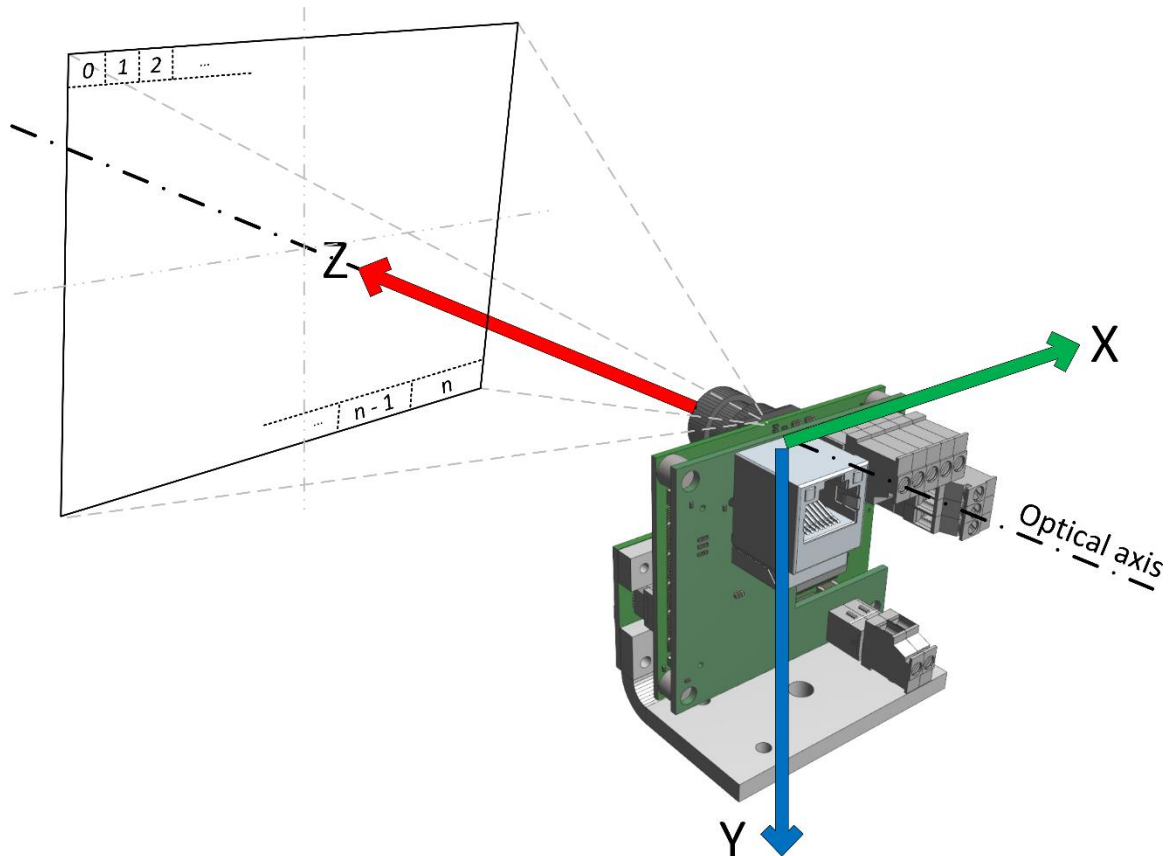


Figure 3-1: ToF coordinate system

## 4 Interface

The functions in bta.h define the interfaces of the SDK. The headers bta\_frame.h, bta\_status.h, bta\_event.h, bta\_discovery.h and bta\_flash\_update.h contain more declarations and are included by bta.h.

The interface is best described going through a well documented example. For further help, also make use of the doxygen-style documentation of the header files.

---

```
static void errorHandling(BTA_Status status) {
    if (status != BTA_StatusOk) {
        char statusString[100];
        BTAstatusToString(status, statusString, strlen(statusString));
        printf("error: %s\n", statusString);
        printf("Hit <Return> to end the example\n");
        fgetc(stdin);
        exit(0);
    }
}
```

---

```
static void wait(int ms) {
    #if defined(PLAT_LINUX) || defined(linux)
        usleep(ms * 1000);
    #elif defined(PLAT_WINDOWS) || defined(WIN32) || defined(WIN64)
        Sleep(ms);
    #endif
}
```

---

```
// Example for an implementation of the infoEvent callback handler
static void BTA_CALLCONV infoEvent(BTA_EventId eventId, int8_t *msg) {
    char eventIdString[100];
    BTAeventIdToString(eventId, eventIdString, sizeof(eventIdString));
    printf("    infoEvent: (%s) %s\n", eventIdString, msg);
}
```

---

```
// Example for an implementation of the frameArrived callback handler
static void BTA_CALLCONV frameArrived(BTA_Frame *frame) {
    BTA_Status status;
    BTA_Frame *frameClone;
    status = BTAcloneFrame(frame, &frameClone);
    errorHandling(status);
    // The frameClone pointer can now be stored for later processing, or
    // a new thread can handle that, // but this function should return now.
    // Wherever the processing is done, before losing the last reference to it,
    // the clone must be free'd
    BTAfreeFrame(&frameClone);
    // (Do not free or alter the frame that was passed as a parameter)
}
```

---



```
#ifdef FOR_EXPERTS_ONLY
// Example for an implementation of the progressReport callback handler
static void BTA_CALLCONV progressReport(BTA_Status status, uint8_t percentage) {
    if (percentage == 0 && status == BTA_StatusOk) {
        printf("Flash update started");
    }
    else if (percentage == 100 && status == BTA_StatusOk) {
        printf("Flash update finished with success");
    }
    else if (status == BTA_StatusOk){
        printf("Flash update progress: %d", percentage);
    }
    else {
        char statusString[100];
        BTastatusToString(status, statusString, strlen(statusString));
        printf("Flash update failed: %s", statusString);
    }
}
#endif
```

```
// Discovery
//-----
// The feature of listing available devices is currently only supported by BtaP100Lib.

// Initialization
//-----
// First, the library must be configured via the configuration c-structure.
// The configuration structure must be initialized with standard values using the function
// BTAinitConfig. The specific implementation of the library defines which parameters are
// required and which can be left out. The required parameters must then be set to a valid
// value before calling BTAopen.

BTA_Config config;
printf("BTAinitConfig()\n");
status = BTAinitConfig(&config);
errorHandling(status);

// Connection Parameters
//-----
// Depending on the library and the device, different connection parameters must be set.
// Redundant connections (like TCP and UDP control connections) are used sequentially and
// exclusively
// (UDP is tried first and only if it fails, TCP is connected)
// Unnecessary information is ignored (BtaP100Lib ignores ethernet parameters)

// UDP data connection (normally multicast address)
uint8_t udpDataIpAddr[] = { 224, 0, 0, 1 };
config.udpDataIpAddr = udpDataIpAddr;
config.udpDataIpAddrLen = 4;
config.udpDataPort = 10002;
// TCP control connection (device's IP address)
uint8_t tcpDeviceIpAddr[] = { 192, 168, 0, 10 };
config.tcpDeviceIpAddr = tcpDeviceIpAddr;
config.tcpDeviceIpAddrLen = 4;
config.tcpControlPort = 10001;
```

```
// UDP control outbound connection (device's IP address)
uint8_t udpControlOutIpAddress[] = { 192, 168, 0, 10 };
config.udpControlOutIpAddress = udpControlOutIpAddress;
config.udpControlOutIpAddressLen = 4;
config.udpControlOutPort = 10003;
// UDP control inbound connection (normally the host's IP address)
uint8_t udpControlInIpAddress[] = { 192, 168, 0, 1 };
config.udpControlInIpAddress = udpControlInIpAddress;
config.udpControlInIpAddressLen = 4;
config.udpControlInPort = 10004;

// Optional settings for advanced functionalities

// If you want to receive status updates from the library,
// register a callback function for informative events.
config.infoEvent = &infoEvent;
// ...and set the verbosity for infoEvents.
config.verbosity = 9;

// If you want to receive the frames immediately when they arrive,
// register a callback function for incoming frames.
config.frameArrived = &frameArrived;

// Choose whether and how queueing of frames is done.
// Queueing does not affect the frameArrived callback at all
// If you don't specify frame queueing, you can still use the frameArrived callback,
// but you will get an error on BTAgetFrame()
// We choose 'DropOldest', so we always get the most recent frame
config.frameQueueMode = BTA_QueueModeDropOldest;
// Set the length of the frame queue.
config.frameQueueLength = 1;

// The frame mode can be configured in order to get the desired data channels in a frame from
// the sensor / library:
config.frameMode = BTA_FrameModeDistAmp;

// This parameter is only used by BtaP100Lib.
// If this parameter is left empty, the default lens configuration will be used.
// It has to contain the name (and path) of a valid lens calibration file encoded in ASCII.
config.calibFileName = (uint8_t *)"calibFile.bin";

// Connecting
//-----
// Now that the configuration structure is filled in, the connection is ready to be opened
// The first infoEvents should fire while the library is connecting to the sensor. That, however
// depends on the library implementation and the configured verbosity.

BTA_Handle btaHandle;
printf("BTAopen()\n");
status = BTAopen(&config, &btaHandle);
errorHandling(status);

// Connection Status
//-----
// It is possible to get the service state and connection state (not all libraries are able to
// detect their connection status):

printf("Service running: %d\n", BTAisRunning(btaHandle));
```



```
printf("Connection up: %d\n", BTaisConnected(btaHandle));

// Querying Device Information
//-----
// The following example shows, how general information on the device can be retrieved.
// The resulting struct may be only partly filled depending on the device's capabilities.

BTA_DeviceInfo *deviceInfo;
printf("BTAGetDeviceInfo()\n");
status = BTAGetDeviceInfo(btaHandle, &deviceInfo);
errorHandling(status);
printf("Device type: 0x%x\n", deviceInfo->deviceType);
printf("BTAFreeDeviceInfo()\n");
BTAFreeDeviceInfo(deviceInfo);

// Frame-Rate
//-----
// Read and change the frame rate via these functions

float frameRate;
printf("BTAGetFrameRate()\n");
status = BTAGetFrameRate(btaHandle, &frameRate);
errorHandling(status);
printf("Framerate is %f\n", frameRate);
frameRate = 5;
printf("BTASetFrameRate(5)\n");
status = BTASetFrameRate(btaHandle, frameRate);
errorHandling(status);

// Integration time
//-----
// Read and change the main integration time via this functions:

uint32_t integrationTime;
printf("BTAGetIntegrationTime()\n");
status = BTAGetIntegrationTime(btaHandle, &integrationTime);
errorHandling(status);
printf("Integration time is %d\n", integrationTime);
printf("BTASetIntegrationTime(%d)\n", integrationTime);
status = BTASetIntegrationTime(btaHandle, integrationTime);
errorHandling(status);

// Modulation frequency
//-----
// Read and change the main modulation frequency via this functions

uint32_t modulationFrequency;
printf("BTAGetModulationFrequency()\n");
status = BTAGetModulationFrequency(btaHandle, &modulationFrequency);
errorHandling(status);
printf("Modulation frequency is %d\n", modulationFrequency);
printf("BTASetModulationFrequency(%d)\n", modulationFrequency);
status = BTASetModulationFrequency(btaHandle, modulationFrequency);
errorHandling(status);

// Global offset
//-----
// Global stands for 'all pixels', meaning, that this offset is applied to all pixels.
```

```
// It is, for all current devices, valid for the currently set modulation frequency.
// It can only and should be set for all predefined modulation frequencies (see device's SUM).
// When changing the modulation frequency, the global offset reads differently.

float offset;
printf("BTAGetGlobalOffset()\n");
status = BTAGetGlobalOffset(btaHandle, &offset);
errorHandling(status);
printf("Global offset is %f\n", offset);
printf("BTASetGlobalOffset(%f)\n", offset);
status = BTASetGlobalOffset(btaHandle, offset);
errorHandling(status);

// Frame mode
//-----
// The frame mode defines what channels the sensor delivers to the library and/or
// what data the library puts into a frame

printf("BTASetFrameMode(BTA_FrameModeXYZAmp)\n");
status = BTASetFrameMode(btaHandle, BTA_FrameModeXYZAmp);
errorHandling(status);

// Register read/write
//-----
// Register operations are done via readRegister and writeRegister.
// Most devices support multi-read and multi-write. The last parameter can be used to take
// advantage of that feature.
// For a normal read/write null can be passed.
// Example for one register at address 5:
uint32_t regValue;
printf("BTAReadRegister()\n");
status = BTAReadRegister(btaHandle, 5, &regValue, 0);
errorHandling(status);
printf("BTAWriteRegister()\n");
status = BTAWriteRegister(btaHandle, 5, &regValue, 0);
errorHandling(status);
//The application must guarantee that register accesses are done exclusively. A read/write is
// only called when the last read/write returned.

// Wait a little, so the frame in the lib's queue is fresh regarding the just written parameters
wait(1000);

// Frame Retrieval
//-----
// Once the connection is established, the frameArrived callback (if not null) is called
// whenever a frame is received from the sensor.
// But a frame can also actively be requested from the library

BTA_Frame *frame;
printf("BTAGetFrame()\n");
status = BTAGetFrame(btaHandle, &frame, 300);
errorHandling(status);

// The frame structure contains all the necessary information which can be accessed directly
// or using the helper functions.
// For getting the buffer with amplitudes, for example:
uint16_t *amplitudes;
BTA_DataFormat dataFormat;
BTA_Unit unit;
```

```

uint16_t xRes, yRes;
printf("BTAggetAmplitudes()\n");
status = BTAggetAmplitudes(frame, (void **)&amplitudes, &dataFormat, &unit, &xRes, &yRes);
errorHandling(status);
if (dataFormat == BTA_DataFormatUInt16) {
    // This dataformat tells us that it's ok to cast (void *) to (uint16_t *)
    if (unit == BTA_UnitUnitLess) {
        printf("Got amplitude data\n");
        // -> access amplitude data simply as amplitudes[i]
        uint32_t ampAvg = 0;
        for (int y = 0; y < yRes; y++) {
            for (int x = 0; x < xRes; x++) {
                ampAvg += amplitudes[x + y*xRes];
            }
        }
        if (xRes != 0 && yRes != 0) {
            printf("The average amplitude is %d\n", ampAvg / xRes / yRes);
        }
    }
}
// The first pixel value in the buffer corresponds to the
// upper left pixel (sensor point of view).

// In order to extract the Cartesian coordinates, do the following:
void *xCoordinates, *yCoordinates, *zCoordinates;
printf("BTAggetXYZcoordinates()\n");
status = BTAggetXYZcoordinates(frame, &xCoordinates, &yCoordinates, &zCoordinates,
                                &dataFormat, &unit, &xRes, &yRes);
errorHandling(status);
if (dataFormat == BTA_DataFormatSInt16) {
    // This dataformat tells us that it's ok to cast (void *) to (int16_t *)
    if (unit == BTA_UnitMillimeter) {
        printf("Got 3D data\n");
        // -> cast the void* to int16_t* and access data points simply as:
        // ((int16_t *)xCoordinates)[i]
        // ((int16_t *)yCoordinates)[i]
        // ((int16_t *)zCoordinates)[i]
        uint32_t radiusMin = 0xffffffff;
        int16_t xCoordinate, yCoordinate, zCoordinate;
        for (int y = 0; y < yRes; y++) {
            for (int x = 0; x < xRes; x++) {
                if (((int16_t *)zCoordinates)[x + y*xRes] > 0) {
                    uint32_t radius = (((int16_t *)xCoordinates)[x + y*xRes] *
                                        ((int16_t *)xCoordinates)[x + y*xRes];
                    radius += (((int16_t *)yCoordinates)[x + y*xRes] *
                               ((int16_t *)yCoordinates)[x + y*xRes];
                    radius += (((int16_t *)zCoordinates)[x + y*xRes] *
                               ((int16_t *)zCoordinates)[x + y*xRes];
                    if (radius < radiusMin) {
                        radiusMin = radius;
                        xCoordinate = ((int16_t *)xCoordinates)[x + y*xRes];
                        yCoordinate = ((int16_t *)yCoordinates)[x + y*xRes];
                        zCoordinate = ((int16_t *)zCoordinates)[x + y*xRes];
                    }
                }
            }
        }
        printf("The nearest point is (%d, %d, %d) [mm]\n", xCoordinate, yCoordinate, zCoordinate);
    }
}

```

```

}
// The Channels X, Y and Z are given in the predefine Cartesian coordinate system.
// See Figure "ToF coordinate system".

// Frame Cleanup
//-----
// A successful call to getFrame or cloneFrame always demands for a call to BTAfreeFrame

printf("BTAfreeFrame()\n");
status = BTAfreeFrame(&frame);
errorHandling(status);

#ifdef FOR_EXPERTS_ONLY
// Flash update
//-----
// A transfer of data to the device, typically to be saved in the device's flash memory can be
// performed by calling the library's function BTAflashUpdate. It is mainly used to perform a
// firmware update. The struct BTA_FlashUpdateConfig is passed containing all the information
// needed. The library defines which fields in the structure have to be filled depending on
// the type of the update and/or data. Thus, the user must know how to configure the update
// and what data to pass as parameter.

// An example for updating the pixel list is:
BTA_FlashUpdateConfig flashUpdateConfig;
flashUpdateConfig.target = BTA_FlashTargetPixelList;
uint32_t dummydata = 0x12345678;
flashUpdateConfig.data = (uint8_t *)&dummydata;
flashUpdateConfig.dataLen = 4;
printf("BTAflashUpdate()\n");
status = BTAflashUpdate(btaHandle, &flashUpdateConfig, (FN_BTA_ProgressReport)&progressReport);
errorHandling(status);

// This function simplifies the process of a firmware update allowing the user to pass a
// connection handle and the name of a binary firmware file. Internally it uses BTAflashUpdate()
printf("BTAfirmwareUpdate()\n");
status = BTAfirmwareUpdate(btaHandle, (uint8_t *)"fw.bin",
(FN_BTA_ProgressReport)&progressReport);
errorHandling(status);
#endif

#ifdef THIS_SECTION_COMPROMISES_REGISTER_VALUES_PERMANENTLY
// Storing and restoring register settings
//-----
// The register values can be stored in flash memory in order to be preserved beyond a
// reboot/power cycle
printf("BTAwriteCurrentConfigToNvm()\n");
status = BTAwriteCurrentConfigToNvm(btaHandle);
errorHandling(status);

// The restoring of the default configuration typically requires a reboot in order to take
// immediate effect
printf("BTArestoreDefaultConfig()\n");
status = BTArestoreDefaultConfig(btaHandle);
errorHandling(status);
#endif

// Grabbing the stream

```



```
//-----  
// The Blt libraries are able to write frames to disk. Very conveniently, all the frame data  
// is stored in a *.bltstream file. These files can later be replayed by the BtaStreamLib as  
// if the same sensor was connected.  
  
BTA_GrabbingConfig grabbingConfig;  
printf("BTAAinitGrabbingConfig()\n");  
status = BTAinitGrabbingConfig(&grabbingConfig);  
errorHandling(status);  
grabbingConfig.filename = (uint8_t *)"test.bltstream"; // (ASCII coded)  
printf("BTAAstartGrabbing()\n");  
status = BTAstartGrabbing(btaHandle, &grabbingConfig);  
errorHandling(status);  
  
// Grabbing is taking place. Let it grab for 2 sec  
wait(2000);  
  
printf("BTAAstopGrabbing()\n");  
status = BTAstartGrabbing(btaHandle, 0);  
errorHandling(status);  
  
// Device Reset  
//-----  
// If the device and the library choose to implement this functionality, a device reset can be  
// performed  
  
status = BTASendReset(btaHandle);  
errorHandling(status);  
  
// Disconnecting  
//-----  
// When work is done and no other threads need to access the library's functions,  
// disconnect the sensor and stop the service by simply calling BTAClose  
  
printf("BTAClose()\n");  
status = BTAClose(&btaHandle);  
errorHandling(status);
```

## 5 References



## 6 Document Revision History

Version	Date	Author	Description
1	2014 06 18	AFA	Initial Draft
2	2014 07 22	AFA	Changed coordinate system
3	2014 07 30	AFA	Minor corrections and changes Added BTAGetDeviceInfo
4	2014 08 05	AFA	Correction for BTAGetFrame Made BTAReadRegister and BTAWriteRegister device-independent
4.1	2014 08 14	SSY	Added lens calibration file parameter Added BTAfirmwareUpdate()
5	2014 08 27	AFA	Check for timeliness of examples: very minor changes
6	2014 09 11	ROB	Serial number in BTAOpen adjusted to v1.0.0
7	2014 09 16	AFA	Updated Concept figure, added Links to the Bluetechnix support wiki
8	2015 05 19	AFA	Updates for Version 1.6 release

Table 6.1: Revision history

## A List of Figures and Tables

### Figures

Figure 2-1: Interfacing concept .....	6
Figure 3-1: ToF coordinate system .....	7

### Tables

Table 1.1: Reference Overview .....	5
Table 6.1: Revision history .....	17