

BLUETECHNIX  
Embedding Ideas

---

# BltTofApi SDK

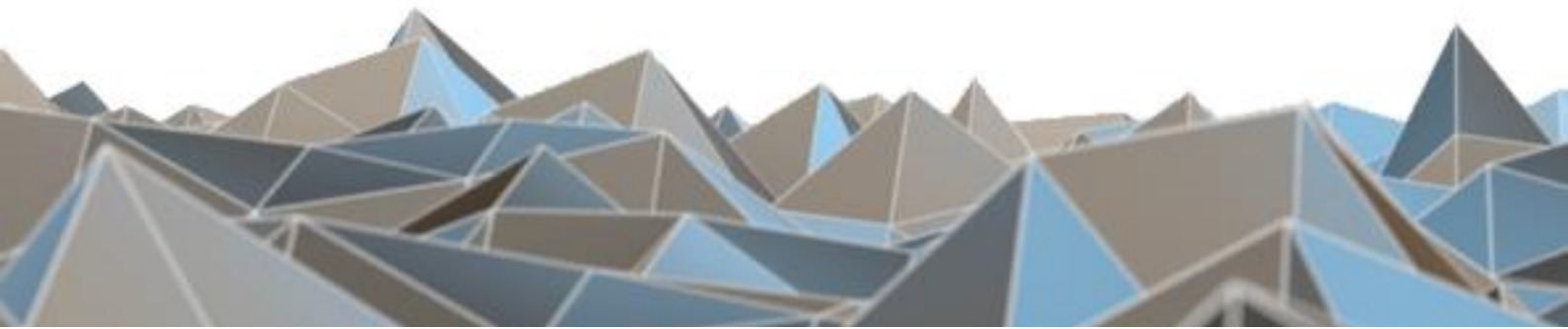
---

Software User Manual

---

Version 6

---



Contact

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

Date: 2014-09-12

## Table of Contents

1	Introduction.....	5
1.1	Purpose of the document .....	5
1.2	References.....	5
2	Software Architecture.....	6
2.1	Overview.....	6
2.2	Interfaces.....	6
2.2.1	Initialization .....	6
2.2.2	Connection Parameters .....	7
2.2.3	Frame Mode Parameter .....	7
2.2.4	Lens Calibration File Parameter.....	7
2.2.5	Callback Functions Parameters .....	7
2.2.6	Connecting .....	8
2.2.7	Connection Status .....	8
2.2.8	Querying Device Information .....	8
2.2.9	Frame Retrieval .....	8
2.2.10	Frame Cleanup.....	10
2.2.11	Register Operations .....	10
2.2.12	Device Reset.....	11
2.2.13	Disconnecting .....	11
2.2.14	Flash Update.....	11
2.2.15	Firmware Update.....	12
3	References .....	13
4	Document Revision History .....	14
A	List of Figures and Tables .....	15

© Bluetechnix R&D GmbH 2014

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>	1.0.0	Inc/	
<b>bta_frame.h</b>	1.0.0	Inc/	
<b>bta_status.h</b>	1.0.0	Inc/	
<b>bta_discovery.h</b>	1.0.0	Inc/	
<b>bta_flash_update.h</b>	1.0.0	Inc/	
<b>bta_event.h</b>	1.0.0	Inc/	

Table 1.1: Reference Overview

## 2 Software Architecture

### 2.1 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 is able to access the BltTofApi interface and will therefore be compatible with any device with existing lib implementing the BltTofApi.

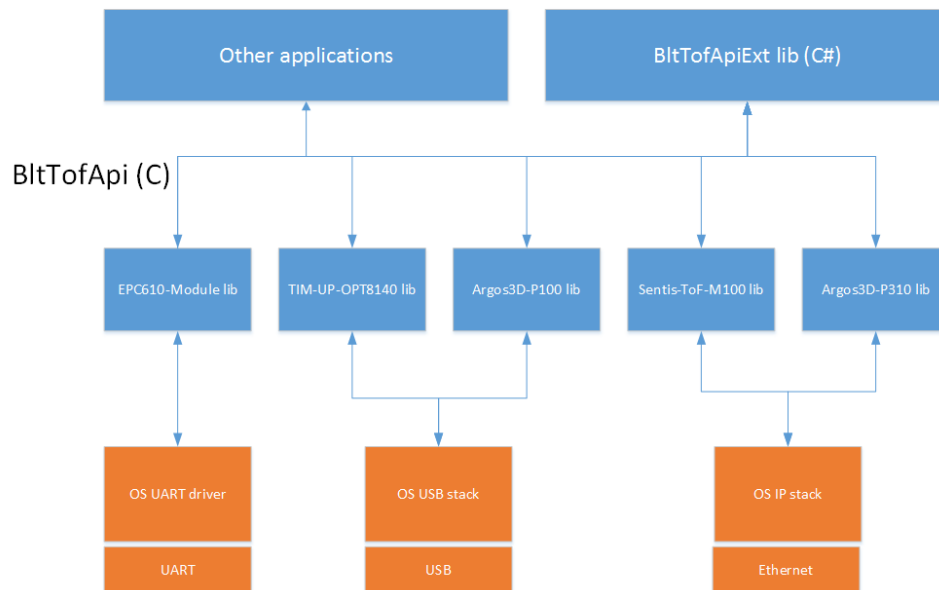


Figure 2-1: Interfacing concept

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.

### 2.2 Interfaces

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.

#### 2.2.1 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.

Code:

```

BTA_Config config;
BTAinitConfig(&config);

```

## 2.2.2 Connection Parameters

A library might define different parameter sets for different connection modes (For example if only TCP configuration interface parameters are set, no data interface connection will be established to the sensor).

Example for Sentis-ToF-M100 lib and connection mode UDP/TCP:

```
uint8_t udpDataIpAddr[] = { 224, 0, 0, 1 };
config.udpDataIpAddr = udpDataIpAddr;
config.udpDataIpAddrLen = 4;
config.udpDataPort = 10002;
uint8_t tcpDeviceIpAddr[] = { 192, 168, 0, 10 };
config.tcpDeviceIpAddr = tcpDeviceIpAddr;
config.tcpDeviceIpAddrLen = 4;
config.tcpDataPort = 10001;
```

Example for opening a specific P100 device (serial number 000039):

```
config.serialNumber = 0x27;
```

## 2.2.3 Frame Mode Parameter

The frame mode can be configured in order to get the desired data channels from the sensor / library:

```
config.frameMode = BTA_FrameModeXYZAmp;
```

## 2.2.4 Lens Calibration File Parameter

If this parameter is left empty the default lens configuration will be used. Otherwise it has to contain the name (path) of a valid lens calibration file.

```
config.calibFileName = "xyz_calibration_00.bin";
```

## 2.2.5 Callback Functions Parameters

Callback functions are optional. In order to automatically get frames and events from the library, callbacks can be configured.

Example for an implementation of callback functions:

```
void infoEvent(BTA_EventId eventId, int8_t *msg) {
    printf("infoEvent %d: %s\n", eventId, msg);
}

void frameArrived(BTA_Frame *frame) {
    printf("Got frame %d", frame.ulFrameCounter);
}
```

These callback function must also be set before calling BTAopen:

```
config.infoEvent = &infoEvent;  
config.frameArrived = &frameArrived;
```

If any of these parameters are left null, those callbacks will not be executed.

## 2.2.6 Connecting

Now that the configuration structure is filled in, the connection is ready to be opened:

```
BTA_Handle btaHandle;  
status = BTAOpen(&config, &btaHandle);  
if (status == BTA_StatusOk) {  
    // ok  
}
```

The first infoEvents should already fire while the library is connecting to the sensor. That, however depends on the library implementation.

## 2.2.7 Connection Status

Additionally it is possible to get the service state and connection state:

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

## 2.2.8 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;  
status = BTAgetDeviceInfo(btaHandle, &deviceInfo);  
if (status == BTA_StatusOk) {  
    // ok  
}  
printf("Device type: 0x%x\n", deviceInfo->deviceType);  
BTAFreeDeviceInfo(deviceInfo);
```

## 2.2.9 Frame Retrieval

Once the connection is established, the frameArrived callback (if not null) is called whenever a frame is received from the sensor. The implementation of the callback function frameArrived should copy the data relevant to the application and return immediately. No heavy operations should be performed, or the library is not able to continue to capture frames from the device.

A frame can also actively be requested from the library:

```
BTA_Frame *frame;  
status = BTAgetFrame(btaHandle, &frame);
```



```
if (status == BTA_StatusOk) {  
    // ok  
}
```

The frame structure contains all 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;  
uint16_t yRes;  
status = BTAgetAmplitudes(frame, (void **)&amplitudes, &dataFormat, &unit,  
&xRes, &yRes);  
if (status == BTA_StatusOk) {  
    if (dataFormat == BTA_DataFormatUInt16) {  
        if (unit == BTA_UnitUnitLess) {  
            // as expected -> process amplitude data: amplitudes[i]  
        }  
    }  
}
```

The first pixel value in the buffer corresponds to the upper left pixel (sensor point of view).

For extracting the Cartesian coordinates, do the following:

```
uint16_t *xCoordinates, *yCoordinates, *zCoordinates;  
status = BTAgetDistances(frame, (void **)&xCoordinates, (void  
**) &yCoordinates, (void **)&zCoordinates, &dataFormat, &unit, &xRes, &yRes);  
if (status == BTA_StatusOk) {  
    if (dataFormat == BTA_DataFormatSInt16) {  
        if (unit == BTA_UnitMillimeter) {  
            // as expected -> process point in space: ( xCoordinates[i],  
yCoordinates[i], zCoordinates[i] )  
        }  
    }  
}
```

The Channels X, Y and Z are given in the predefined Cartesian coordinate system. See Figure 2-2.

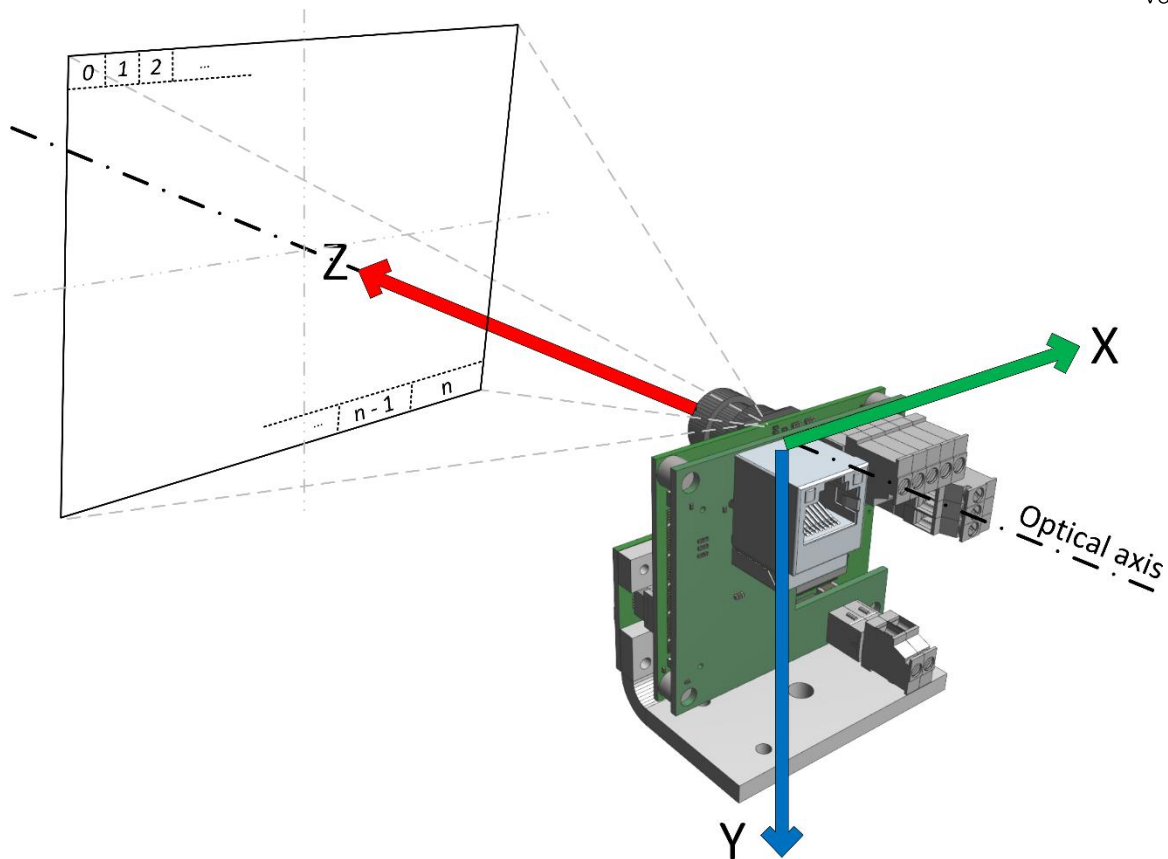


Figure 2-2: ToF coordinate system

### 2.2.10 Frame Cleanup

A successful call to `getFrame` always demands for a call to:

```
BTAfreeFrame(&frame);
```

### 2.2.11 Register Operations

Register operations are done via `readRegister` and `writeRegister`. Example for one register at address 5:

```
uint32_t regValue;
status = BTAreadRegister(btaHandle, 5, &regValue, 0);
if (status == BTA_StatusOk) {
    // ok
}

status = BTAwriteRegisters(btaHandle, 5, &regValue, 0);
if (status == BTA_StatusOk) {
    // ok
}
```

The application must guarantee that register accesses are done exclusively. A read/write is only called when the last read/write returned.

### 2.2.12 Device Reset

If the device and the library choose to implement this functionality, a device reset can be performed:

```
status = BTAsendReset(btaHandle);  
if (status == BTA_StatusOk) {  
    // ok  
}
```

### 2.2.13 Disconnecting

In order to disconnect the sensor and stop the service, simply call:

```
void BTAClose(&btaHandle);
```

### 2.2.14 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`. 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 the implementation of the callback function is:

```
void 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 {  
        printf("Flash update failed: %d", status);  
    }  
}
```

An example for updating the pixel list of an EPC610-Module is:

```
BTA_FlashUpdateConfig config;  
config.target = BTA_FlashTargetPixelList;
```

```
config.data = /*read byte-stream from data source like file*/;  
BTA_Status status = BTAflashUpdate(btaHandle, &config, &progressReport);  
if (status == BTA_StatusOk) {  
    // ok  
}
```

### 2.2.15 Firmware Update

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().

```
BTA_Status status = BTAfirmwareUpdate(btaHandle, "firmware.bin");
```

### **3 References**

## 4 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

Table 4.1: Revision history

## **A List of Figures and Tables**

### **Figures**

Figure 2-1: Interfacing concept.....	6
Figure 2-2: ToF coordinate system .....	10

### **Tables**

Table 1.1: Reference Overview .....	5
Table 4.1: Revision history .....	14