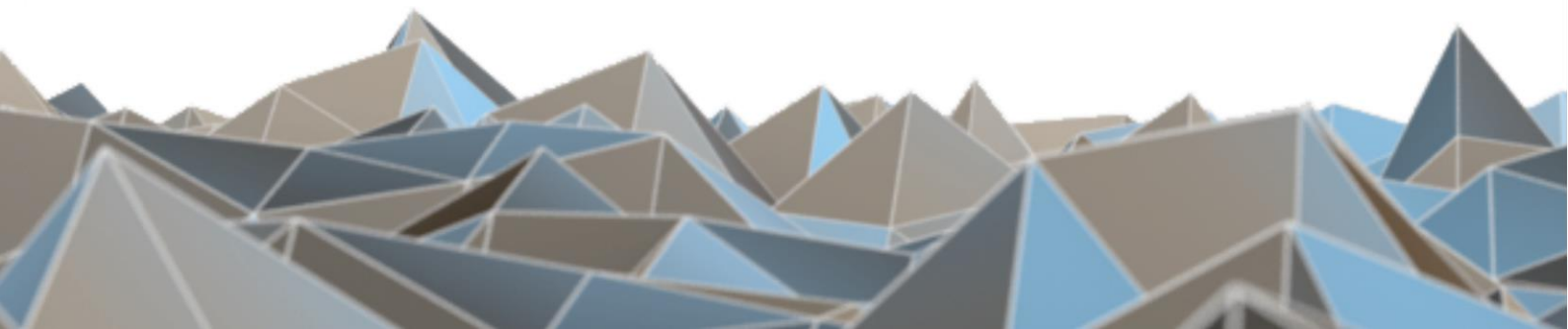


BLUETECHNIX
Embedding Ideas

BltTofApi v2.5 SDK

User Manual

Version 2





Contact

BECOM BLUETECHNIX GmbH
Gutheil-Schoder-Gasse 17, 1230 Wien
AUSTRIA
office@bluetechnix.com
<http://www.bluetechnix.com>

Date: 2018-03-07

Table of Contents

1	Introduction	5
1.1	Purpose of the document	5
2	Overview.....	6
3	BTA_Config parameters.....	8
3.1	General.....	8
3.2	BTAgetFrame() vs. frameArrived()	8
3.3	LibParams	9
3.4	USB connection (P100 based cameras).....	9
3.4.1	BTAgetFrame() vs. frameArrived() callback	9
3.4.2	Other BTA_Config parameters	10
3.4.3	Hotplugging	10
3.4.4	LibParams.....	10
3.5	Ethernet connection.....	11
3.5.1	BTAgetFrame() vs. frameArrived() callback	11
3.5.2	Other BTA_Config parameters	11
3.5.3	Hotplugging	12
3.5.4	LibParams.....	12
3.6	Bltstream connection.....	12
3.6.1	BTAgetFrame() vs. frameArrived() callback	12
3.6.2	Other BTA_Config parameters	13
3.6.3	Bltstream handling.....	13
3.6.4	LibParams.....	13
4	Metadata	14
5	Lens Parameters	15
6	Build instructions.....	16
6.1	Windows.....	16
6.2	Linux.....	16
6.3	ARM (Linux).....	16
6.4	Tegra TX2 (Linux)	16
7	Recommended documents.....	17
8	Document Revision History.....	18
A	List of Figures and Tables.....	19

© BECOM BLUETECHNIX GmbH 2018

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 library v2.5.

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 BltTofSuite is able to access the BltTofApi interface as is any other user application. The BltTofApi interface guarantees ease of use and compatibility with all Bluetechnix products and projects.

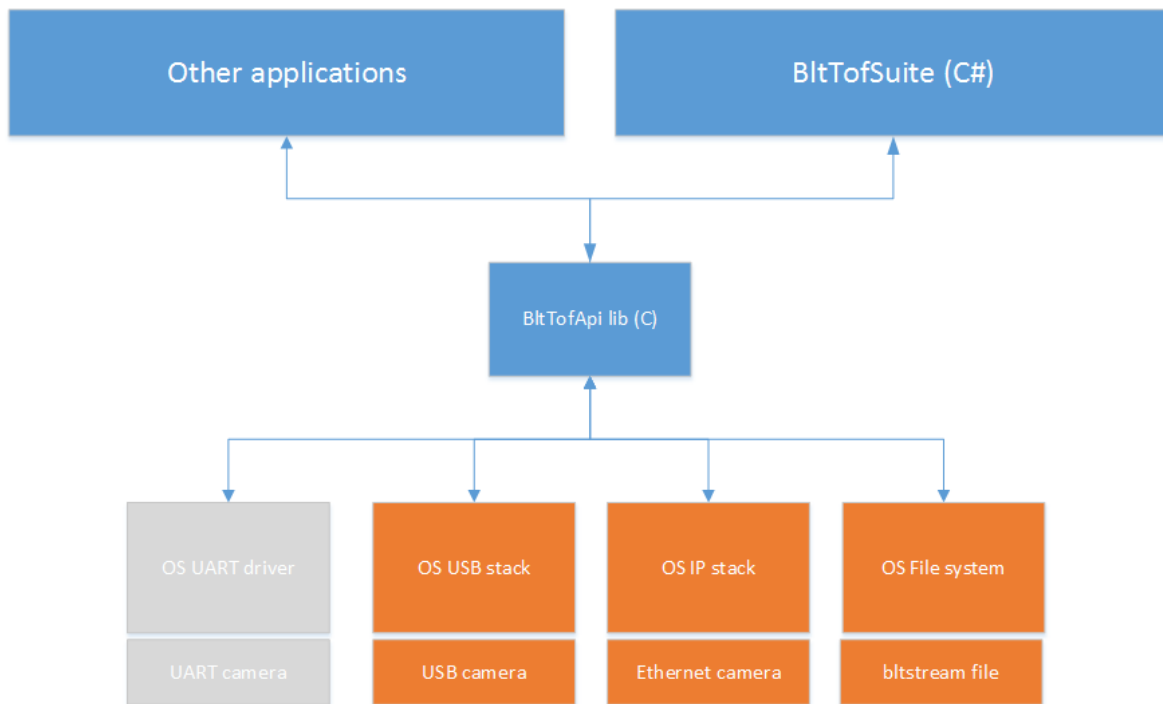


Figure 2-1: Interfacing concept

Every ToF system built by or for Bluetechnix shall be accessible by this common interface. The Interface is kept as simple as possible and covers all functionalities of all ToF sensors.

Any coordinates retrieved from the API follows the pixel order and coordinate system defined in the following image.

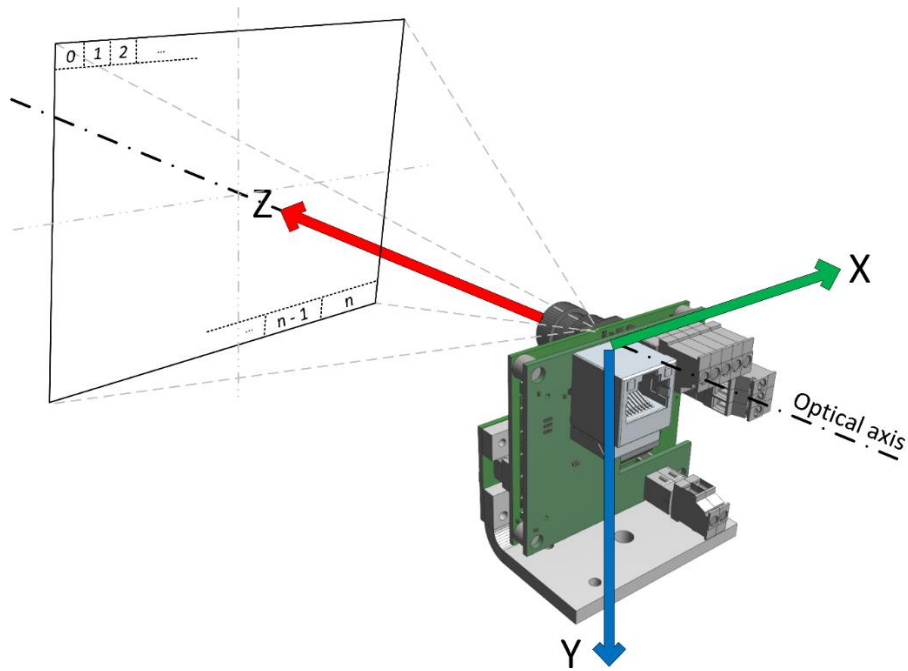


Figure 2-2: ToF coordinate system (here with a Sents3D-M100)

3 BTA_Config parameters

3.1 General

This section only explains parameters that behave the same with all camera interfaces. For device/interface specific information, please refer to the corresponding section.

- **DeviceType:** In order for the lib to know what interface to use to connect to a camera. If none is specified (i.e. 0), all interfaces are probed sequentially.

Possible values are:

- 1: Ethernet
- 2: USB
- 3: UART
- 15: Bltstream (file)
- **FrameMode:** Setting this parameter is the same as calling `BTAsetFrameMode`, but with more immediate effect.
- **InfoEventEx:** Register a callback for the identification of the source of an error and for logging purposes. You are very welcome to include these logs when contacting Bluetechnix support.
- **InfoEventFilename:** Set a filename (relative or absolute) for automatic logfile generation.
- **Verbosity:** Change the amount of infoEvents.

3.2 BTAgetFrame() vs. frameArrived()

Every application needs considering which is best in which situation. The methods can also be mixed. In the corresponding interface section there are tables describing the behavior and the functionality of the BltTofApi library. The behavior of the BltTofApi library depends on these connection parameters:

- **frameQueueLength:** The BltTofApi library can take care of queueing frames. This parameter sets the length of the queue in [frames]. The library automatically starts a thread to fill the queue. Frames can be dequeued by calling `BTAgetFrame()`.
- **frameQueueMode:** In the case that the mentioned queue is full and a new frame is ready to be enqueued, this parameter decides what to do:
 - **BTA_QueueModeDoNotQueue:** Invalid unless `frameQueueLength == 0`.
 - **BTA_QueueModeDropOldest:** The oldest frame in the queue is thrown away and the new frame is enqueued.
 - **BTA_QueueModeDropCurrent:** The frame that is to be enqueued is not enqueued but thrown away instead.
 - **BTA_QueueModeAvoidDrop:** Invalid parameter.

- **frameArrived:** By providing this parameter to the BTAopen function the callback functionality is activated. The library automatically starts a thread, parses frames from the corresponding interface and delivers them via calling frameArrived().
- **frameArrivedEx:** This callback has the same functionality as frameArrived, with an extra parameter to distinguish multiple cameras.

3.3 LibParams

EnableTestPattern: Enable the test pattern. Frames retrieved from the camera are overwritten with test data.

InfoEventVerbosity: Change the logging verbosity while connected.

UndistortRgb: The undistortion of RGB frames is on by default (when intrinsic data available). With this parameter it can be turned off.

EnableJpgDecoding: The camera can send RGB data as JPG which is decoded by default. With this parameter one is able to get the JPG data undecoded.

FramesParsedCount: A counter of frames parsed by the API. Read to clear.

FramesParsedPerSec: A counter of frames parsed per second (averaged).

3.4 USB connection (P100 based cameras)

When operating USB cameras please be aware of the behavior of the SDK described below.

3.4.1 BTAgetFrame() vs. frameArrived() callback

Q... frameQueueing enabled (BTA_Config parameters frameQueueLength and frameQueueMode)

C... frameArrived callback registered (BTA_Config parameters frameArrived or frameArrivedEx)

Q	C	Behaviour	BTAgetFrame	frameArrived
		Internal capturing is disabled. No thread is running in background.	Calling BTAgetFrame triggers the readout of a frame via USB. The call might take even longer when no frame is available.	Disabled.
	X	Internal capturing by the SDK is enabled. A thread repeatedly reads frames from USB and delivers them via callback. With very low frame rates or slow frame triggering the internal thread blocks the USB interface. Do not use this configuration in that case.	Disabled.	Frames are delivered as soon as the capture thread gets them. This is the fastest way for a frame from camera to the user.
X		Internal capturing by the SDK is enabled. A thread repeatedly	Calling BTAgetFrame delivers a frame from the queue. The	Disabled.

		reads frames from USB and queues them. With very low frame rates or slow frame triggering the internal thread blocks the USB interface. Do not use this configuration in that case.	call is fast, but the frame might be old.	
X	X	Internal capturing by the SDK is enabled. A thread repeatedly reads frames from USB, delivers them via callback and queues them. With very low frame rates or slow frame triggering the internal thread blocks the USB interface. Do not use this configuration in that case.	Calling BTAgetFrame delivers a frame from the queue. The call is fast, but the frame might be old.	Frames are delivered as soon as the capture thread gets them. This is the fastest way for a frame from camera to the user.

3.4.2 Other BTA_Config parameters

- All parameters starting with tcp, udp, uart or bltstream are ignored.
- PON and serial number must be set to null or match the value in the camera's register exactly.
- calibFileName is supported. Please use files provided by Bluetechnix.
- zFactorsFileName is supported. Please use files provided by Bluetechnix.
- AverageWindowLength: If > 1 the camera is configured for sequencing and a filter is instantiated in order to combine sequences of the same frame into one frame, averaging valid values.

3.4.3 Hotplugging

The USB cable must be connected and the camera must be up and running before BTAopen is called. If the USB connection is somehow lost, the BTA_Handle must be closed with BTAClose() and reopened.

3.4.4 LibParams

PauseCaptureThread: By disabling the capture thread (see Section 3.4.1 for how to enable it) it is possible to control the load on the USB port.

DisableDataScaling: The lib computes metric distance units from phase data by default. This parameter turns it off.

3.5 Ethernet connection

When operating Ethernet cameras please be aware of the behavior of the SDK described in Section 3.5.1).

Ethernet connections are divided into the Control Interface and the Data Interface. “Control” is for register operations, firmware updates and flash readouts. “Data” is for sensor data stream. A control interface can be established via UDP or TCP (depending on the camera), while the data interface only supports UDP. The API can establish a control only or a data only connection if so desired.

3.5.1 BTAgetFrame() vs. frameArrived() callback

Q... frameQueueing enabled (BTA_Config parameters frameQueueLength and frameQueueMode)

C... frameArrived callback registered (BTA_Config parameters frameArrived or frameArrivedEx)

Q	C	Behaviour	BTAgetFrame	frameArrived
		If a UDP data connection is configured, BTAopen will return an error because it is not possible to retrieve a frame	Disabled.	Disabled.
	X	The internal thread handling UDP packets delivers the frames via callback.	Disabled	Frames are delivered as soon as the UDP packets are parsed. This is the fastest way for a frame from camera to the user.
X		The internal thread handling UDP packets queues the frames.	Calling BTAgetFrame delivers a frame from the queue. The call is fast, but the frame might be old.	Disabled
X	X	The internal thread handling UDP packets delivers the frames via callback and queues them.	Calling BTAgetFrame delivers a frame from the queue. The call is fast, but the frame might be old.	Frames are delivered as soon as the UDP packets are parsed. This is the fastest way for a frame from camera to the user.

3.5.2 Other BTA_Config parameters

- All parameters starting with uart or bltstream are ignored.
- The three connections udpData, udpControl and tcpControl can be configured completely individually. At least one of the three connections must be provided. If tcpControl and udpControl are configured both, then both connections are tried to establish.
- PON: The device itself is only aware of its serial number which is matched to a string of one or several PONs separated by a slash. The PON parameter must be contained in that string in order to be a match.
- Serial number: must match the value in the camera’s register exactly.

- calibFileName is not supported. Please use BTAflashUpdate.
- zFactorsFileName is not supported.
- AverageWindowLength is not supported.

3.5.3 Hotplugging

The UDP data connection is established regardless of the presence of a camera. The UDP/TCP control connection can only be established if the camera responds to the first alive message. If the control connection is subsequently lost, the library tries to reconnect until BTAClose is called or the connection was reestablished.

3.5.4 LibParams

KeepAliveMsgInterval: Set and get the interval at which alive messages are sent in order to keep the connection alive / check if the connection is alive.

CrcControlEnabled: Enable and disable the usage of a crc checksum control interface communication. When disabled, the crc checksum is only used for file transmissions, not for register operations and the like.

BytesReceivedStream: A counter of bytes received over the data interface by the API. Read to clear.

3.6 Bltstream connection

When reading a stream from file please be aware of the behavior of the SDK described below.

3.6.1 BTAgetFrame() vs. frameArrived() callback

Q... frameQueueing enabled (BTA_Config parameters frameQueueLength and frameQueueMode)

C... frameArrived callback registered (BTA_Config parameters frameArrived or frameArrivedEx)

Q	C	Behaviour	BTAgetFrame	frameArrived
		BTAopen will return an error because it is not possible to retrieve a frame	Disabled.	Disabled.
	X	The internal thread reading the frames from the bltstream file delivers the frames via callback.	Disabled	Frames are delivered as recorded (with the same timing as they were grabbed).
X		The internal thread reading the frames from bltstream queues the frames.	Calling BTAgetFrame delivers a frame from the queue.	Disabled
X	X	The internal thread reading the frames from the bltstream file delivers the frames via callback and queues them.	Calling BTAgetFrame delivers a frame from the queue.	Frames are delivered as recorded (with the same timing as they were grabbed).

3.6.2 Other BTA_Config parameters

All parameters except `bltstreamFilename`, `frameQueueMode`, `frameQueueLength`, `frameArrived` and `frameArrivedEx` are ignored.

3.6.3 Bltstream handling

The `bltstream` file has a plain text header at the beginning. In `BTAopen` the library reads some information from the `bltstream` and starts a thread. That thread starts reading frames and (after some buffering) provides them via the API. Jumping to specific index inside the stream rather than accessing the frames sequentially can take longer.

File format v1 limitation: Total frame count is not supported, i.e. it is not known how many frames there are in the file.

3.6.4 LibParams

StreamTotalFrameCount: Get the total amount of frames stored in the current `bltstream` file (only supported since file format v2).

StreamAutoPlaybackSpeed: Get and set the playback speed. Playback is timed by the frame timestamps (as recorded) times this factor.

StreamPos: Get and set the index of the current frame. The stream, i.e. frames in the file are sequentially numbered starting with 0.

StreamPosIncrement: Set the index of the current frame relatively to the current index.

4 Metadata

The library supports channels to be accompanied by metadata. Metadata is only available if it was provided by the connected camera. Metadata can only be a stream of n bytes identified by an id (BTA_MetadataId). A frame's channels' metadatas can be looped through by using the following code or using the function BTAgetMetadata().

```
uint32_t chInd;
BTA_Frame *frame = ...
if (frame->channels) {
    for (chInd = 0; chInd < frame->channelsLen; chInd++) {
        BTA_Channel *channel = frame->channels[chInd];
        uint32_t mdInd;
        for (mdInd = 0; mdInd < channel->metadataLen; mdInd++) {
            BTA_Metadata *metadata = channel->metadata[mdInd];
            uint32_t metadataId = metadata->id;
            void *metadata = metadata->data;
            uint32_t metadataLen = metadata->dataLen;
            // work with metadata bytestream
        }
    }
}
```

The BTAgetMetadata function does the same job:

```
BTA_Status BTAgetMetadata(BTA_Channel *channel, uint32_t metadataId, void **metadata,
uint32_t *metadataLen)
```

It takes a channel and a metadataId. If the metadata by the given id is present in the channel, the void* bytestream and its length metadataLen in [bytes] are returned. Otherwise BTA_StatusInvalidParameter is returned.

5 Lens Parameters

The method `BTAgetLensParameters` allows for reading the stored calibration data from the camera. Calibration data includes Intrinsic and Extrinsic (relative to the camera's defined reference point) parameters. The usage of the example is explained in the following example:

```
uint32_t intDataLen = 10;
BTA_IntrinsicData *intData = (BTA_IntrinsicData *)malloc(intDataLen * sizeof(BTA_IntrinsicData));
uint32_t extDataLen = 10;
BTA_ExtrinsicData *extData = (BTA_ExtrinsicData *)malloc(extDataLen * sizeof(BTA_ExtrinsicData));
status = BTAgetLensParameters(btaHandle, intData, &intDataLen, extData, &extDataLen);
errorHandling(status);
if (status == BTA_StatusOk) {
    for (int i = 0; i < intDataLen; i++) {
        printf(" Intrinsic data for lens %d:\n", i);
        printf("   xRes: %d\n", intData[i].xRes);
        printf("   yRes: %d\n", intData[i].yRes);
        printf("   fx: %f\n", intData[i].fx);
        printf("   fy: %f\n", intData[i].fy);
        printf("   cx: %f\n", intData[i].cx);
        printf("   cy: %f\n", intData[i].cy);
        printf("   k1: %f\n", intData[i].k1);
        printf("   k2: %f\n", intData[i].k2);
        printf("   k3: %f\n", intData[i].k3);
        printf("   k4: %f\n", intData[i].k4);
        printf("   k5: %f\n", intData[i].k5);
        printf("   k6: %f\n", intData[i].k6);
        printf("   p1: %f\n", intData[i].p1);
        printf("   p2: %f\n", intData[i].p2);
        printf("   lensId: %d\n", intData[i].lensId);
        printf("\n");
    }
    for (int i = 0; i < extDataLen; i++) {
        printf(" Extrinsic data for lens %d:\n", i);
        printf("   rotMat: %f %f %f\n", extData[i].rot[0], extData[i].rot[1], extData[i].rot[2]);
        printf("           %f %f %f\n", extData[i].rot[3], extData[i].rot[4], extData[i].rot[5]);
        printf("           %f %f %f\n", extData[i].rot[6], extData[i].rot[7], extData[i].rot[8]);
        printf("   trlVec: %f\n", extData[i].trl[0]);
        printf("           %f\n", extData[i].trl[1]);
        printf("           %f\n", extData[i].trl[2]);
        printf("\n");
    }
    printf("\n");
}
else {
    char statusString[100];
    BTASTatusToString(status, statusString, (uint16_t)sizeof(statusString));
    printf("\nBTAgetLensParameters() returned %s\n\n", statusString);
}
```

For further details, please look at the reference manual on the Bluetechnix support page.

6 Build instructions

6.1 Windows

Please take a look at the example project for a reference on how to compile an application using a BTA library.

Dependencies:

- Define the macro `#define PLAT_WINDOWS`
- Microsoft Visual C++ Redistributable for Visual Studio 2015 (can be downloaded from the Microsoft homepage)

When using a **USB (P100)** camera, consider:

- The libusb0 driver must be installed (Provided by Bluetechnix)

6.2 Linux

Please refer to the Makefile of the provided example for building your own applications.

Dependencies:

- libusb-0.1.0
- libpthread-2.15

When using a **USB (P100)** camera, consider:

- Copy the .rules file provided by Bluetechnix to `/lib/udev/rules.d/`

6.3 ARM (Linux)

See Section 6.2

6.4 Tegra TX2 (Linux)

See Section 6.2

7 Recommended documents

The newest version of this document and the reference manual can be downloaded from:

https://support.bluetechnix.at/wiki/Bluetechnix_ToF_API_v2



8 Document Revision History

Version	Date	Author	Description
1	2018 02 20	AFA	Initial Draft
2	2018 03 07	AFA	Added Chapter LensParameters

Table 8.1: Revision history

A List of Figures and Tables

Figures

Figure 2-1: Interfacing concept	6
Figure 2-2: ToF coordinate system (here with a Sentis3D-M100)	7

Tables

Table 8.1: Revision history	18
-----------------------------------	----