Date	Version	Change	Author
13-Jan-12	1	Document created	Marc Andre
07-Jun-12	8	Adapted to latest characteristics structure	Marc Andre
22-Oct-12	20	Many minor updates. BLE_QNH parameter	man
02-Apr-13	27	Update messages lengths	Eko
21-May-13	28	Added axis orientation picture	Man
08-Oct-13	29	Added correction for acceleration unit	man
04-Nov-13	30	Added BLE protocol details	Eko
05-Nov-13	31	Added details on Status2 flags. Added details for file transfer.	man
20-Nov-13	32	Updated System characteristic with pressure and QNH. Added details for notify and device info service.	Man
16-Dec-13	33	Added more detail on notification update rate settings.	Man
29-Apr-14	34	Fixed typo for BLE_MsgRate_Sys	Man

1 Introduction

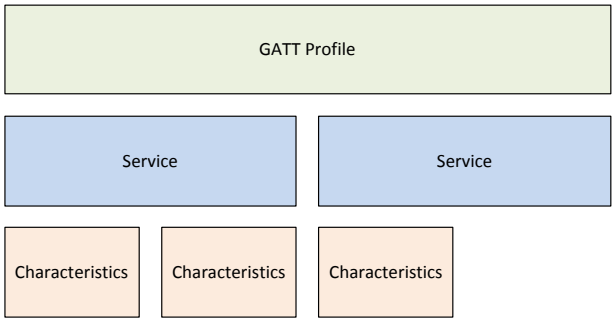
This document describes the communication interface between the Flytec SensBox device and a Bluetooth device.

For a simple developer using a SDK, you might directly jump to the sensor description (Section 3). In Section 0 the implementation for iPhone is described.

2 Bluetooth Low Energy "Protocol"

Bluetooth low energy is based on the GATT profile. This profile allows reading/writing of attributes. Attributes are called characteristics and can contain up to MTU-3 bytes for simple attributes or longer for long (multi-packet) attributes. MTU>=23. Absolute maximum is 512 bytes.[1] A characteristic is updated at the time. Characteristics are grouped to services. A device can contain multiple services and each service can contain multiple characteristics.

For achievable data rates and packet size consult TI forum[2]. All characteristics for SensBox will be limited to 19 bytes.



2.1 Pairing / Bonding – Encryption

No pairing or bonding is needed to connect to the sensor box and for standard operation. This might change with added advanced features.

2.2 BLE Settings in Flytec SensBox

Field	Duration
Advertising Interval	50 ms
Slave Latency	0 ms
Interval Minimum	20 ms
Interval Maximum	50 ms
Conn Supervision Timeout	5 s

[1] See Bluetooth Spec v.4, Volume 3, page 479
[2] http://e2e.ti.com/support/low_power_rf/f/538/t/108807.aspx and http://e2e.ti.com/support/low_power_rf/f/538/t/88642.aspx

3 Services

3.1 Sensor Service

Sensor service is the main service in SensorBox. It allows reading the primary values like pressure, variometer, GPS, etc.

UUID: <aba27100 143b4b81 a444edcd 0000f020>

3.1.1 Notification

All characteristics support notification (starting firmware version FS.02.XX). The update rate can be set using BLE_MsgRate_XXX settings through the communication service. The update rate affects the number of updates on the BLE interface. If notify is enabled the rate also sets the notify rate. If notify is disabled the rate sets the number of updates that the characteristic gets from the sensors.

The value written to BLE_MsgRate_XXX must be calculated as follows:

$$value_{rate} = \frac{100}{rate_{Hz}}$$

As Example, for 1Hz the value is 100 and for 5Hz the value is 20.

The total number of messages per second must not exceed 25. Please not that internally the different sensors have different update rates. The default message rates are the maximum recommended rates for each characteristic.

3.1.2 Characteristics

3.1.2.1 Navigation

UUID: <aba27100 143b4b81 a444edcd 0000f022>
Size: 19 bytes
Abstract: Primary navigational data
Rate: 5Hz, Adjustable with BLE_MsgRate_Nav

Field	Unit	Encoding	Bytes
Date/Time (UTC)	Unixtime	Uint32	4
Latitude	$° * 10^7$	Int32	4
Longitude	$° * 10^7$	Int32	4
GPS Height MSL	m	Int16	2
Pressure Altitude	m	Int16	2
Vario 1Hz	cm/s	Int16	2
Status			1

Remarks:
Date/Time is from the internal real-time clock after startup and is updated with GPS time as soon as the first satellite fix is obtained. Thus a jump in time has to be expected upon the first GPS fix.

GPS Height MSL is the GPS height above the geoid. The height above the ellipsoid can be read from the GPS Secondary characteristic.

Pressure Altitude is by default the non-QNH corrected barometric altitude. By setting the "BLE_UseQNH_Nav" setting to '1', this parameter can be changed to return the QNH corrected altitude. The "BLE_UseQNH_Nav" setting is not persistent and is reset for each BLE session. The Movement characteristic (3.1.2.2) also contains pressure altitude, but with more precision. The movement characteristic has its own setting for QNH/QNE.

Vario is low-pass filtered for 1Hz update rate.

Status: see *3.1.3.1* Status

3.1.2.2 Movement

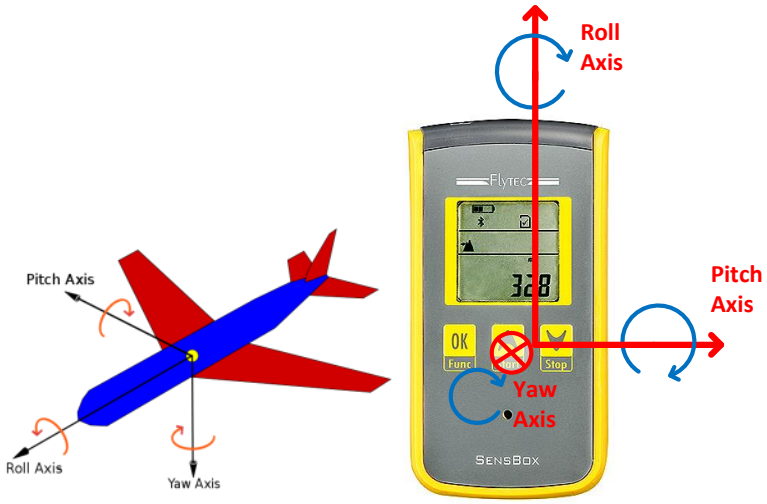
UUID: <aba27100 143b4b81 a444edcd 0000f023>
Size: 19 bytes
Abstract: Primary movement data
Rate: 10Hz, Adjustable with BLE_MsgRate_Mov

Field	Unit	Encoding	Bytes
Pressure Altitude	cm	Int32	4
Vario 8Hz	cm/s	Int16	2
Ground Speed	dm/s	Int16	2
GPS Heading	° * 10	Int16	2
Pitch	° * 10	Int16	2
Yaw	° * 10	Int16	2
Roll	° * 10	Int16	2
Acceleration	"g" * 10	UInt16	2
Status			1

Remarks:
Pressure Altitude is by default the non-QNH corrected barometric altitude. By setting the "BLE_UseQNH_Mov" setting to '1', this parameter can be changed to return the QNH corrected altitude. The "BLE_UseQNH_Mov" setting is not persistent and is reset for each BLE session. The Navigation characteristic (3.1.2.2) also contains pressure altitude, but with less precision. The navigation characteristic has its own setting for QNH/QNE.

Vario is filtered for 8Hz output.

Pitch / Roll / Yaw are the Euler angles. Pitch is in the range of +-90°, Roll is in the range of +-180° and yaw is in the range of 0..360°. The axes are defined as:



Acceleration is the total length of the acceleration vector in g multiplied by 10. Thus the normal value would be in the range of 10.

Status: see *3.1.3.1* Status

3.1.2.3 GPS Secondary

UUID: <aba27100 143b4b81 a444edcd 0000f024>
Size: 8 bytes
Abstract: Secondary data from GPS.
Rate: 1Hz, Adjustable with BLE_MsgRate_Gps

Field	Unit	Encoding	Bytes
Horizontal Accuracy	dm	Uint16	2
Vertical Accuracy	dm	Uint16	2
GPS Height Ellipsoid	m	Int16	2
Number of satellites		Uint8	1
Status			1

Remarks:
GPS Height Ellipsoid is the GPS height above ellipsoid. The height above geoid (msl) can be found in the Navigation characteristic (3.1.2.1).

Number of satellites shows the number of used satellites. The fix status can be found in the status byte (3.1.3.1).

Status: see *3.1.3.1* Status

3.1.2.4 System

UUID: <aba27100 143b4b81 a444edcd 0000f025>
Size: 16 bytes
Abstract: Delivers system and status information for the Flytec SensBox.
Rate: 1Hz, Adjustable with BLE_MsgRate_Sys

Field	Unit	Encoding	Bytes
Date/Time (UTC)	Unixtime	Uint32	4
Battery Level	%	Uint8	1
Logging level	%	Uint8	1
Temperature	°C * 10	Int16	2
Status			1
Status2			1
QNH	Pa * 10	UInt16	2
Pressure	mPa	Int32	4

Remarks:
Date/Time is from the internal real-time clock after startup and is updated with GPS time as soon as the first satellite fix is obtained. Thus a jump in time has to be expected upon the first GPS fix.

Battery level is given in percent, but only knows discrete steps. The most significant bit is used to indicate charging. Thus > 100% means charging.

Logging level gives the used logging space in % of total available space.

Status and *Status2:* see *3.1.3.1* Status and

Pressure: Current air pressure in mPa, filtered for 1Hz.

3.1.3 Status Flags

3.1.3.1 Status

This status byte is sent with all characteristics and shows the main status of the system.

Bit	Field	Remarks
0..2	GPS Status	0: No Fix 1: 2D Fix 2: 3D Fix 3: 2D Fix + DGPS 4: 3D Fix + DGPS 5..7: Reserved
3	Time Status GPS	0 = Not yet updated 1 = Updated from GPS
4	Charging	1 = Charging
5	Bat Low	Battery low warning
6	Logging	Logging is started
7	Reserved	

3.1.3.2 Status2

This status byte is sent with the status characteristics and shows the additional status of the system.

Bit	Field	Remarks
0	SD card inserted	1 = SD Card inserted
1	Comm. Protocol Ready	1 = Device ready to receive Comm. Protocol messages
2	FILE_READY	1 = Device ready to transfer files. (Must be set to file transfer mode prior using File transfer event, see 3.3.2.2)
3..7	Reserved	

3.2 DeviceInfo Service

The device info service is specified by Bluetooth SIG. [3]

The SensBox fills the following information:

- FirmwareVersion and SoftwareVersion are the same value (e.g. “FS.02.04”)
- Serialnumber is a 5 character string (e.g. “01000”)
- Model is “SensBox”
- Manufacturer is “Flytec”
- HardwareVersion is “1” (at this time).

3.3 Communication Service

This service allows sending commands to the SensBox and reading back values. It is used for configuration or actions. It is also used to transmit large quantities of data towards the Master (i.e iPhone) through notification mechanism. The achieved transmission speed is at maximum 1700 bytes/s.

UUID: <aba27100 143b4b81 a444edcd 0000f010>

3.3.1 Characteristics

3.3.1.1 Control

UUID: <aba27100 143b4b81 a444edcd 0000f011>
Size: 1 byte
Abstract: Controls the communication from the iPhone side.

SensBox communication control is used to reset the communication state machine in case it is stuck. When read it returns the communication state machine. ***NOTE:*** *Not yet implemented.*

Field	Unit	Encoding	Bytes
Control/Status	None	uint8	1

Control *(Write only register): Writing ‘0’ will reset the state machine to idle.*
Status *(Read only register)*: 0=idle, 1=Write pending, 2=Read pending, 3=Response wait

[3] https://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.device_information.xml

3.3.1.2 Data Transfer

UUID: <aba27100 143b4b81 a444edcd 0000f012>
Size: min 1, up to 18
Abstract: Data transfer for the SensBox communication.

Field	Unit	Encoding	Bytes
Header		uint8	1
Payload		uint8[]	0-17

The idea of this transport layer is to transfer a data buffer up to 252 bytes between a BLE client (the Master) and the main processor (the Host) through the Bluetooth CPU (the Slave). The BLE client is intended to be a BLE capable smartphone as the iPhone.

This transfer occurs in chunks of max 18 bytes due to the Slave's characteristics limitations.
The Master (BLE client) performs a request by writing an amount (up to 252) of data, tagged with an identifier. The Master can require a response from the Host. This response can be an acknowledge to a write command or the response to a read command. These details are left to a higher level communication protocol implementation. The response from the Host can be an amount of data up to 252 bytes and must contain the same header tag as the request in order to be able to complete the transfer.

Due to the characteristic length limitations, the Master must send the data in chunks of 17 bytes + 1 byte header. As long as the header's tag is the same, the chunk will be appended to the internal data buffer, up to the size of the buffer. The Master signals the last chunk by setting a specific bit (Last) into the header. Once the message is completed, the data is transferred from the Slave to the Host. If the Master requested a response, the Slave will reject any further request from the Master until the Host provided a response and the Master completely read it or the Master writes a reset command into the Control register (not yet implemented).

Data read by the Master is also done in chunks of 17 bytes length + 1 byte header. As long as the 'Last' bit is not set in the header, the Master must read the characteristic. Only when the Master read the last chunk, it will be able to perform a further request.

Header: The header is used to specify the message. A message can be a read request, a write request and also part of a bigger data transfer.

Bit	Name	Encoding
0	Last	
1	Response	
7:2	Tag	uint8

The tag must change between subsequent messages in order to avoid response's misinterpretation.

A higher protocol level must be implemented in order to use this transport layer.

3.3.1.3 File Transfer

UUID: <aba27100 143b4b81 a444edcd 0000f013>

Size: min 1, up to 20
Abstract: File transfer for the SensBox communication through notifications.

Field	Unit	Encoding	Bytes
Header		uint8	1
Payload		uint8[]	0-19

After a file 'Read' command (see Read (FileOP = 0x05)), SensBox will send the requested data through notifications. The characteristic's notification configuration must be enabled.

Data to be transferred is packed in packets of 19 bytes and preceded by a header used to rearrange packets upon receiving as they may arrive in different order than sent. This mechanism is similar to an UDP transfer over IP.

The Header is incrementing number (0 to 255) and starts at 0 for each file 'Read' command. If a packet is missing after 255 new packets then it is completely lost and the transfer must be aborted and restarted. This behavior is unlikely to happen in real world.
Nevertheless, a workaround to this limitation is to request file read of sizes of 256*19 (4'864) bytes which ensure that the Header will never overlap.

3.3.2 Communication protocol

A communication protocol is needed on top of the transport layer.

The 252 bytes long payload that can be transferred between Master and Host are further divided into:

Field	Encoding	Bytes
ID	uint8	1
Payload	uint8[]	0-251

3.3.2.1 Host acknowledge (ID 0x00)

Field	Encoding	Description
ID	uint8	0x00
Error Code	uint8	Error code. 0: SUCCESS, 0xff: unknown ID, otherwise context dependent.

This message can be sent by the Host, if the Master requested a response. The payload of this message is the error code. As a general rule, an error code of zero indicates a success and a '0xff' indicates that an unknown message ID has been received. Otherwise, errors codes are specific to the request and must be decoded by the Master.
In case of a write with acknowledge (or what the Host think that it received), this message is the only possible response. In case of a read, the response is either this message indicating a failure or the request's response.
Please note that this message cannot be sent to the Host.

3.3.2.2 Events (ID 0x01)

Field	Encoding	Description
ID	uint8	0x01
Event Code	uint16	Refer to the 'Events list and description' topic for further information about the events.
Payload (optional)	byte[]	Refer to the 'Events list and description' topic for further information about the events.

This message is sent by the Master to the Host in order to control or to notify the latter. In general (up to now), these events doesn't carry a payload. The first two bytes of the general message payload carry the event code. In case of a response request, the general case is that the Host issues a success "Host acknowledge" message. If the event is not known, the Host generates a failure "Host acknowledge" message. If an event derogates to this rule, specific description can be found in the event specification.

3.3.2.2.1 Events list and description

Event	Code	Payload	Remarks
PEV	0x0001	0	Sends Pilot Event in IGC file
Start Log	0x0002	0	Starts IGC logging
Stop Log	0x0003	0	Stops IGC logging
Shutdown	0x0004	0	Shutdown SensBox device
DumpFlights	0x0005	0	Dumps all flights from Flash to IGC. This command also shut down the device.
Disconnect	0x0006	0	Disconnect BLE
File Transfer	0x0007	0	Shutdown SensBox to File Transfer Mode. Check Status2 to wait until mode is reached. This can take a few seconds if ICG file has to be written first.

3.3.2.3 Settings (ID 0x02)

Field	Encoding	Description
ID	uint8	0x02
Settings [up to 251]	char	NULL terminated string used to read or write a SensBox setting. Don't forget that NULL char is part of the field.

The query message (from Master to SensBox) may contain a 'Setting' string with the following formats:

- 'setting_name=setting_value' which means that the Master performs a setting write request.
- 'setting_name' which means that the Master performs a setting read request.

If the 'response' bit is set the SensBox will issue one of the following responses:

- 'Host acknowledge' message containing:
  - 0x00 for a successful setting write request.

- 0x01 if an error occurred during a write request.
- 0x02 if an error occurred during a read request.
- 0x03 if the SensBox didn't recognized the Master's query.

Or

- A 'Settings' message containing the Master's setting read response in the 'setting_name=setting_value' string format.

A list of all settings can be found in a separate document.

3.3.2.4 Files (ID 0x03)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	This field indicates the file operation type
Payload [up to 250]	byte[]	FileOP dependent

This message is intended to offer file like interface access to the SensorBox. The SensBox must be in a certain state before it will accept commands through this interface.
All commands require that the 'response' bit is set. In case where the FileOP received from the Master is unknown or if the message is invalid, the SensBox will respond with a "Host acknowledge" message containing the error code 0x01. Moreover, if the SensBox is unable to process the message due to its state it will issue a "Host acknowledge" message containing the error code 0x02.

3.3.2.4.1 FindFirst (FileOP = 0x01)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	0x01
FolderName [up to 250]	char	NULL terminated string containing the folder name to be browsed.

This message is sent in order to query the first file from a given folder. The root of the file system is "/". Please note that '.' And '..' folders are filtered.

The response message (from SensBox to the Master) may be one of the following:

- A "Host acknowledge" message containing an error code.
  - 0x03 if the 'FolderName' directory cannot be open.
  - 0x07 if the file name is empty.
- A FileEntry (FileOP=0x08) message containing a file entry.

3.3.2.4.2 FindNext (FileOP = 0x02)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	0x02

This message is sent in order to query the subsequent files from a folder opened by the 'FindFirst' command.
Please note that '.' And '..' folders are filtered.

The response message (from SensBox to the Master) may be one of the following:

- A "Host acknowledge" message containing an error code.
  - 0x05 if the 'FindFirst' command hasn't been issued.
  - 0x07 if the file name is empty.
- A FileEntry (FileOP=0x08) message containing a file entry.

3.3.2.4.3 Open (FileOP = 0x03)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	0x03
Mode	uint16	This field specifies in which mode the file is open.
FileName [up to 248]	char	NULL terminated string containing the file name to be opened.

This message is sent in order to open a file.

The 'Mode' may be one or more of the following flags:

- 0x0001: open the file 'FileName' in plain read mode. The file must exist. The data is transferred without compression.
- 0x0002: open the file 'FileName' in ICG optimized Huffmann compression mode. The file must exist. The data is transferred with Huffmann tree compression optimized for ICG. See 4.1 for details.

The response message (from SensBox to the Master) may be one of the following:

- A "Host acknowledge" message containing an error code.
  - 0x04 if a file is already open.
  - 0x03 if the file cannot be open.
- A FileHandle (FileOP=0x09) message containing a file handle. This handle must be used for subsequent accesses concerning that file.

3.3.2.4.4 Close (FileOP = 0x04)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	0x04
Handle	uint32	This field specifies the handle of a previously opened file.

This message is sent in order to close a file that has been opened.

The 'Handle' is a value that has been returned by the 'Open' command.

The response message (from SensBox to the Master) may be one of the following:

- A "Host acknowledge" message containing an error code.

  - 0x05 if no file is open.
  - 0x06 cannot close the file as a data transfer is in progress.
- A FileCRC (FileOP=0x0A) message containing the transferred data CRC.

3.3.2.4.5 Read (FileOP = 0x05)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	0x05
Handle	uint32	This field specifies the handle of a previously opened file.
Size	uint32	This field specifies the size in byte to transmit from the file.

This message is sent in order to read a specified amount 'Size' of bytes from an opened file with handle 'Handle'.

The 'Handle' is a value that has been returned by the 'Open' command.
The 'Size' is the size in byte to be transferred. This value may be '2^{31}-1' if the Master intends to transfer the full file. If the file is open in compressed mode, this command will transfer 'Size' compressed bytes which may represent a variable amount of bytes in the file.

The response message (from SensBox to the Master) may be one of the following:

- A "Host acknowledge" message containing an error code.
  - 0x05 if no file is open.
  - 0x06 if another transfer is in progress.
- A FileReminder (FileOP=0x0B) message containing the remaining bytes from the current position to the end of the file.

The SensBox will start the data transfer through the characteristic 'File Transfer',
'UUID = <aba27100 143b4b81 a444edcd 0000f013>' using the notification mechanism.

3.3.2.4.6 Write (FileOP = 0x06)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	0x06
TBD	byte[]	This command is not yet defined.

This message is sent in order to write data to a file that has been opened in Write mode.

The response message (from SensBox to the Master) will be a "Host acknowledge" message containing the error code 0x01 as this command has not been implemented yet.

3.3.2.4.7 Abort (FileOP = 0x07)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	0x07

This message is sent in order to abort all file operations. This command will not close the file. Also note that the CRC send on the close command may not correspond to the data received by the Master.

The response message (from SensBox to the Master) will be A "Host acknowledge" message containing one of the following error code:

- 0x05 if no file is open.
- 0x00 on successful abort.

3.3.2.4.8 FileEntry (FileOP = 0x08)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	0x08
Size	uint32	This field specifies the file size in bytes.
Date	uint32	This field specifies the date and time in unix format.
Type	uint8	This field specify the type of the entry: • 0x01: file • 0x02: folder
Name [Up to 12]	char	This field specifies the name of the file.

This message is sent as a response from the SensBox to the Master to a 'FindFirst' or 'FindNext' command.

3.3.2.4.9 FileHandle (FileOP = 0x09)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	0x09
Handle	uint32	This field specifies the file handle.

This message is sent as a response from the SensBox to the Master to an 'Open' command.

3.3.2.4.10 FileCRC (FileOP = 0x0A)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	0x0A
Type	uint8	This field specifies the type of the CRC. • 0x00: CRC16
CRC [up to 64]	byte[]	This field contains the CRC value of the read/write bytes from the file. Its type depends of the 'Type' field. • 0x00: uint16

This message is sent as a response from the SensBox to the Master to a 'Close' command.

3.3.2.4.11 FileReminder (FileOP = 0x0B)

Field	Encoding	Description
ID	uint8	0x03
FileOP	uint8	0x0B
Reminder	uint32	This field specifies the difference between the end of the file and the current position of an opened file.

This message is sent as a response from the SensBox to the Master to a 'Read' command.

3.3.2.4.12 Error codes

Code	Description
0x01	Bad command message or format
0x02	SensBox is not in file transfer mode. Not ready to reply to messages.
0x03	File or folder not found
0x04	A file is already open. Close file first.
0x05	Not file is open.
0x06	Transfer is in progress. Must abort before close or restart.
0x07	No more file in folder

4 File Transfer

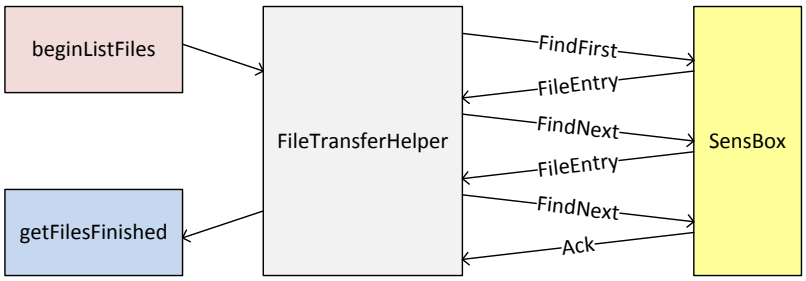
File transfer is implemented for transferring small files such as ICG tracks from the SD card. The SensBox must be set into a specific File Transfer mode using event message, see 3.3.2.2. In this mode the SensBox doesn't supply any other characteristics except for updating the Status message. Also ICG file recording is disabled and USB is disconnected. The file transfer mode can be exited with shutdown only either through the event message or through user interface.

The transfer of data is done using Communication Service, see 3.3.2.4 for details. Please refer to the iOS demo implementation as reference.

4.1 Listing files

File listing is initiated with FindFirst message. Upon receipt of the reply (FileEntry) a FindNext message has to be sent until the SenBox replies with simple ack. The nack signals the end of the file listing. In normal case 0x07 "error" is reported when the listening is finished, but any error should be considered as finish of listing.

Demo implementation:



4.2 Retrieving file

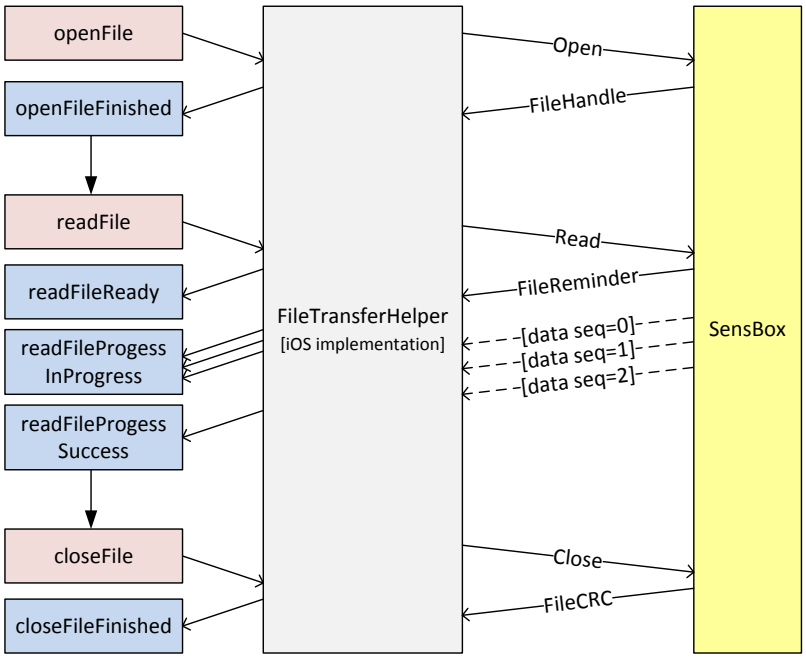
For retrieving a file, the file must be opened, read and then closed.

Open command will reply with a FileHandle if successful or with a NACK including error code if failed. See 3.3.2.4.12 for error code description.

After successful open a Read command must be sent. The read command will reply with a FileReminder message. The size returned by this message is the uncompressed length of the data to be received. Immediately after this command the SensBox starts sending data using notify on the FileTransfer characteristic (see 3.3.1.3). The file transfer contains a header byte with a sequence Id. The sequence starts with 0 on each ready and increases until 255 where it wraps to 0. The sequence must be used to reorder packets as they might arrive mixed. A reorder buffer of 10 items is usually sufficient. If the open was requested using compression the data must be decompressed. All uncompressed bytes must be counted. File transfer is finished when the number of bytes received matches the FileReminder message.

After successful read the file must be closed using Close message. The close message replies with a FileCRC message which contains a CRC16 of all (uncompressed) data sent (usually the CRC of the full file). This CRC16 can be compared with a calculated CRC16 of the received data.

Demo implementation:



4.3 ICG Optimized Huffmann Tree compression

Because the memory size is very limited on the SensBox, it cannot implement classic compression algorithms such as Zip files.

The SensBox offers a specific ICG file optimized compression which can reduce the transfer size of usual ICG files to 40-50%. Very short ICG files or non-ICG files can become larger, up to 200% of the original size in the worst case.

The algorithm uses a static Huffmann tree which was determined from different ICG files. The Huffmann tree compresses the most common 15 characters while it escapes all other characters. Data is sent MSB first. The receiving algorithm must count the number of bytes received and terminate when the file is completed.

Character	Hex	Huffmann Code
0	30	11
1	31	010
9	39	000
3	33	0010
4	34	1001
6	36	0111
7	37	0011
-	2D	01100
2	32	10110
5	35	10111
8	38	10101
<LF>	A	101000
<CR>	D	100011
B	42	011010
E	45	100010
N	4E	100000
S	53	011011
W	57	100001

A	41	1010011
V	56	10100101
<Escape>		10100100

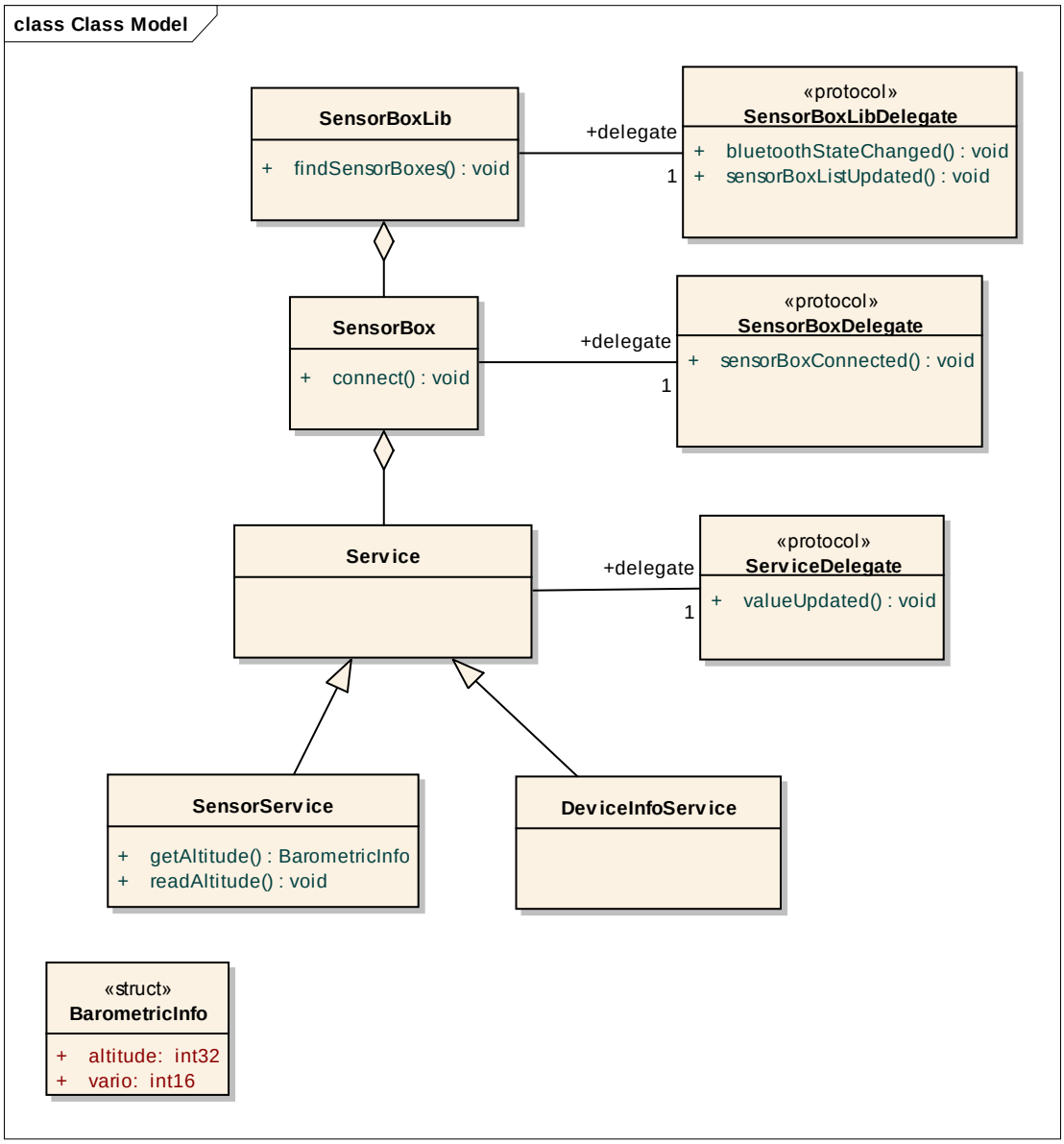
A demo decompression implementation in C can be found in the iOS implementation.

5 iPhone implementation

An open-source library will be provided for iPhone developers to access the SensBox.

The library is based on asynchronous calls. All calls are non-blocking. As soon as the operation (connect, read value, etc) has completed, a callback function is called. This principle is based on the asynchronous methods from CoreBluetooth.

5.1 Class Model



Not all functions and structs are shown. The diagram is only used as concept drawing for accessing the basic service. Communication service is not shown.

5.2 Example

The example code is very preliminary and not cleaned up! It reads a “temp” value, which is just for example purpose and doesn’t exist like that in the real SensorBox.

```
// Write status of Bluetooth
- (void)bluetoothStateChanged
{
    SensorBoxManager* mgr = [SensorBoxManager managerInstance];
    self.lableBLEStatus.text = [mgr getBluetoothStateText];
}

- (void)sensorBoxListUpdated
{
    if (self.sensorBoxLib.sensorBoxes.count > 0 )
    {
        // Connect to the first box found
        SensorBoxManager* mgr = [SensorBoxManager managerInstance];
        self.SensBox = [mgr.sensorBoxes objectAtIndex:0];
        [self.SensBox setDelegate:self];
        [self.SensBox connect];
    }
}

- (void)sensorBoxConnected
{
    self.labelConnected.text = @"connected";

    SensorService* s1 = self.sensorBox.sensorService;
    [s1 setDelegate:self];
    [s1 readTemp];
}


-(void) valueUpdated:(Service*)service characteristicUUID:(NSString*) uuid
{
    self.labelValue.text = [[NSString alloc] initWithFormat:@"%d", [s getTemp]];
    // Immediately read again
    [s readTemp];
}

- (IBAction)buttonStartPush:(id)sender
{
    // Scan for boxes. As soon as a box is found an event is
    // called where the connect is done
    SensorBoxManager* mgr = [SensorBoxManager managerInstance];
    [mgr findSensorBoxes:5];
}
```