OPTIN SENSOR PROTOCOL

COMMUNICATION PROTOCOL SPECIFICATION

Authors:
Optin Team
hwdev@optin.hu

Revised by:
Dénes Attila ALMÁSI

May 8, 2014
Document version: EN-1.1.519

Table of content

1 Introduction

The **Optin Sensor Protocol** is a payload-agnostic client-server protocol with small overhead.

- It's simplicity makes it a perfect choice for devices with scarcer resources.
- It provides numerous optional solutions to guarantee that the critical packets reach their destiny and that the number of lost packets are reduced.
- The handling of the non-critical packets are not burdened by the restrictions that come with the application of the security tasks.

This specification can be divided into two chapters:

- General message format, which is valid for every message.
- The specific parts of the messages types.

Attention: the device-specific content of the *DATA* messages can be found in the appendix.

2 Legal Notice

The Optin Kft. (henceforth Author) provides access to use or implement the OSP, to copy or publish the protocol-specification under the conditions that EVERY copy of the protocol-specification or any part that shows up in other documents, specifications, articles or every system that implements the OSP contains the followings:

- The name of the Author:
  (Optin Kft.)
- a link or URL to the OSP specification on the Author's website:
  www.en.optin.hu/products-services/optin-sensor-protocol/

3 The foundations of the communication

The communication with the protocol happens always through packets. The packets are atomic from the protocol's point of view (The smallest intelligent units with meaning). The packets can be embedded into other protocols and another protocol can be embedded in the packets into a maximum of one point (See: *DATA* and *COMMAND* packets).
This specification doesn't define whether these packets should be sent in a whole or divided into optional chunks.

3.1 Packet structure

Every packet consists of the following parts: a fixed header and the body of the packet. The fixed header identifies the type of the packet, sets its length and includes flags that define the state of the package. The body of the packet is optional, can contain a specialized header, which is present once in the packet and data payload which can contain one or more data.

Fixed header:	MsgType + Flags
	Length
Packet body:	Specialized header
	Data payload

3.2 Definitions

3.2.1 In-flight-uniqueness

An X semantic y value is ”in-flight-unique” if there is no packet in a given moment and in a given direction on the communication channel, nor in the communication caches, memories that's X value is also y.
Examples are the *MessageID* and the *CommandID*. The in-flight-unique values help to identify the packets unequivocally in a window of time defined by the receiver.

3.2.2 Identifiers

The identifiers are positive numbers. The value 0 is always reserved and indicates an error or an invalid entry.

3.2.3 String/binary types

String can only occur at the end of the packets, its length is limitless and there is no closing byte. The end of the string can be calculated with the packet's *Length* field.
In terms of the protocol the strings and binaries are equal: byte-sequences with arbitrary length.

3.2.4 Client and Server

The Client is the one who initiates the communication and connects to the Server. The current 1.1. version of the protocol restricts that DATA packet can only be composed by the Client (like sensor or set of sensors) and sent to the Server, which works as a data collector.

3.2.5 Byte order

In the OSP, like in most network protocols, big-endian coding is used for values with more bytes. This means that the most significant byte (MSB) is in front and the LSB is at the end.
The Length field is an exception, which has a special coding, see below.

3.3 Guarantees

The specification guarantees the followings:

3.3.1 MessageID guarantees

If the MessageID is present in the packet:

- It's always 1 byte long.
- It's always the first byte after the Fixed Header.
- The Client can only set AckReq bit to packets which contain the MessageID.
- The server can request back the packet that belongs to the last in-flight K MessageID with the RESEND packet. The value of the K is defined implicitly by the parties (For examples the server might know it from the ModuleID or the DeviceType).

4 Fixed header

Every packet starts with a fixed header, which is a minimum of 2 bytes and a maximum of 5 bytes long depending on the changing Length field.

The format of the fix header is the following:

	8-5.bit	**4.bit**	**3.bit**	**2.bit**	**1.bit**
1st byte	MsgType	Cached	Saved	AckReq	Reserved
2nd byte	Length ([1-4] byte, chaining)				

Table 1: The structure of the fixed header

1st byte:
Contains the type of the packet (MsgType) and the status flags (Cached, Saved, Ack/Req).

2nd byte:
The field that defines the packet's size (at least 1 byte).

4.1 Message Type (MsgType)

Position: 1st byte, 8-5th bit

The packet's type, it is a 4 bit unsigned, enum value. Its meaning is shown in Chart 2.

MsgType's value	Short name	Description
0x00	[RESERVED]	*Not in use!*
0x01	CONECT/DISCONNECT	A packet initiating a connection or disconnection with the Server
0x02	COMMAND	Command type packet
0x03	ACKNOWLEDGE	Acknowledgement packet for an earlier packet to be acknowledged
0x04	PINGREQ	Ping request
0x05	PINGRESP	Ping response
0x06	FIRMWARE	Firmware packet
0x07	RESEND	Requests resend
0x08	DATA	General data packet
[0x09 - 0x0F]	[RESERVED]	*For further development*

Table 2: The meaning of the MsgType values

4.2 Cached bit (C)

Position: 1st byte, 4th bit

If this bit has a value of **1**, than it's a packet that was attempted to be sent by the device at least once. This is possible in the following situations:

- If the server requested a packet to be resent with the *RESEND* command.
- If the receipt to the packet to be acknowledged has not yet arrived and it was resent automatically.
- The Client detected that the packet was not sent and tries to send it again.

4.3 Saved bit (S)

Position: 1st byte, 3rd bit

If this bit is set to **1** than it's a packet which was reread from a persistent storage (internal Flash or SD card). This results in the transmission of an earlier data. Depending on the application, configuring the bit is justified for example in the following situations:

- There was no connection with the server for a long time.
- While roaming, the data forwarding was disabled or reduced and the device was reconnected to a domestic network.
- Mobile data forwarding was disabled and authorized again.

When the packets produced during one of the above mentioned or similar network dropout are sent, the packet's Saved bit must be configured.

4.4 AckReq bit (A)

Position: 1st byte, 2nd bit

If this bit is set to **1**, it is a packet that must be acknowledged by the server. The signification can happen by sending an ACKNOWLEDGE packet.

4.5 Length field

Position: 2nd byte

The length of the complete message, including the fixed header section. This field's length alternates between 1-4 bytes. The first byte's highest-order bit (8. bit) is not a part of the represented number. If this is turned on the following byte also becomes a part of the length and so on. This way every byte contributes with 7 bit to the length.

For example: the length field of a 64 byte message simply consists of 1 byte with a 0x40 value. However, if this message's length is 321 byte $(2 * 128 + 65)$, the field becomes two byte long. The value of the first byte is $65 + 128 = 193$ according to the LSB first coding. The highest-order bit indicates that there is at least a one byte follow-up. The second byte's value is 2.

Since the protocol limits the length field in 4 bytes, the applications can send a maximum number of 268 435 455 bytes (256 MB) data in a packet.

In Figure 1. we can see the decoding algorithm of the length field. When the algorithm finishes the *value* variable will contain the length of the message in bytes.

```
// Pseudo C code
int multiplier = 1;
int value = 0;
do {
  digit = next_byte_from_stream;
  value += (digit & 127) * multiplier;
  multiplier *= 128;
} while ((digit & 128) != 0)
```

Figure 1: The decoding algorithm of the *Length* field.

5 Packet types

In the following description we mark in a short way the state of the **C** = Cached, **S** = Saved and **A** = AckReq bits as well as the **R** = Reserved bit.
The X symbol stands for an arbitrary value, the 0 for a fix zero, the 1 for a fix one value.

E.g.:

Fixed header:

bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1st byte	MessageType				C	S	A	R
	0	0	0	1	0	0	0	X
2nd byte	Length							

This means the following:
Based on the MessageType this is a CONNECT packet in which the *Cached*, the *Saved* and the *AckReq* flag must always have a fix 0 value.

The violation of the rules of the flag bits must always result in the immediate termination of the communication. The incorrect operation of any one of the three bits can start a process that leads to a series of communicational errors.

An example for this would be the irresponsible use of the AckReq bit: it would be pointless to request ACKNOWLEDGE to a packet that has no MessageID.

5.1 CONNECT/DISCONNECT

The packet that indicates the intention to connect, the authentication and the confirmation or the refusal of the connection.
The fixed header is identical, but the packet body depends on whether the Client or the Server sends it. The CONNECT packet may not have any of the *Cached*, *Saved* or *AckReq* bit set.
Connection is always initiated by the Client towards the Server.

Fixed header:

bit	7	6	5	4	3	2	1	0
1st byte	MessageType				C	S	A	R
	0	0	0	1	0	0	0	X
2nd byte	Length							

5.1.1 Client ⇒ Server

The Client indicates the request to connect to the server, the first packet after the creation of the TCP Channel has to be a CONNECT packet. To every further CONNECT command the Server has to terminate the communication: this is also the indication to request disconnection from the Client.

Packet body:

	Description
1st byte	DeviceType (MSB)
2nd byte	DeviceType (LSB)
3rd byte	ModuleID (MSB)
4th byte	ModuleID (...)
5th byte	ModuleID (...)
6th byte	ModuleID (LSB)
7th byte	ProtocolVersion
8th byte	Password (string)
...	
nth byte	Password (string)

DeviceType: the Client's device identifier.

Value	Meaning
0x0000	Invalid
0x0001	IRIS.base FW 2.0

ModuleID: The Client's configured ID. A 4 byte value.

ProtocolVersion: The OSP version requested to be used by the Client.

5.1.2 Server ⇒ Client

The Server's response to the first CONNECT message in the TCP Channel. The Server doesn't answer to the DISCONNECT stance (to a CONNECT message in a live connection).

Packet body:

	Description
1st byte	ResponseCode
2nd byte	Timestamp (MSB)
3rd byte	Timestamp (...)
4th byte	Timestamp (...)
5th byte	Timestamp (LSB)

ResponseCode: The answer to the request for connection.

Value	**Meaning**
0x00	Denied
0x01	Succesful
0x02	Incorrect ModuleID
0x03	Incorrect DeviceType
0x04	Incorrect ProtocolVersion
0x05	Incorrect authentication
...	*Reserved values*

Timestamp: The moment of connection at the Server's side in Unix Time format. The Client can use it to set its clocks.

5.2 PINGREQ

Both parties can send this to each other. Its function is solely to test the integrity of the connection.
Any party can send one PINGREQ in a given moment, and cannot send any more until exactly one PINGRESP packet arrives.

The violation of the guideline can result in any decision depending on the implementation of the other party. (For example to disconnect, to ignore the unnecessary PINGREQ packets or to answer to all).
A PINGREQ packet can only consist of the following fixed header (2 bytes).

Fixed header:

bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1st byte	MessageType				C	S	A	R
	0	1	0	0	0	0	0	X
2nd byte	Length							

5.3 PINGRESP

Both parties can send this to each other. A PINGRESP packet must always be preceded by exactly one PINGREQ packet going in the other direction.
An unexpected PINGRESP packet can result in any decision depending on the implementation of the receiving party.
A PINGRESP packet can only consist of the following fixed header (2 bytes).

Fixed header:

bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1st byte	MessageType				C	S	A	R
	0	1	0	1	0	0	0	X
2nd byte	Length							

5.4 COMMAND

To execute remote commands. The packet contains the commands in a textual or binary form. (It’s unimportant for the protocol).
The COMMAND packet always contains an in-flight-unique command identifier.

Fixed header:

bit	7	6	5	4	3	2	1	0
1st byte	MessageType				C	S	A	R
	0	0	1	0	0	0	0	X
2nd byte	Length							

5.4.1 Server ⇒ Client

Note: It's the Server's responsibility to send commands that the Client can understand. These commands can be found in the documentation of that specific device that implements the OSP.

Packet body:

	Description
1st byte	CommandID
2nd byte	Script (string)
...	
nth byte	Script (string)

CommandID: The command's in-flight-unique identifier. It is important to define it, because this way the latter responses to the consecutive commands can be identified unambiguously.

Script: The command and it's parameters in a textual or binary form.

5.4.2 Client ⇒ Server

The Client's response to one of the commands sent by the Server.

Packet body:

	Description
1st byte	CommandID
2nd byte	ExitCode
3rd byte	Response (string)
...	
nth byte	Response (string)

CommandID: The command's in-flight-unique identifier. The response to the command with the same CommandID. *If the Client sends back an invalid ID the Server ignores it.*

ExitCode: The command's return code that depends on the device.

Response: The Client's answer to the command. (Can also be an empty string)

5.5 FIRMWARE

The basic unit of the communication procedure. The Client identifies which firmware to download with it's name and requests it in any arbitrarily sized chunks from the Server.

Fixed header:

bit	7	6	5	4	3	2	1	0
1st byte	MessageType				C	S	A	R
	0	1	1	0	0	X	0	X
2nd byte	Length							

5.5.1 Client ⇒ Server

The Client indicates to the Server which chunk of the firmware it requests.

Packet body:

	Description
1st byte	ChunkID (MSB)
2nd byte	ChunkID (LSB)
3rd byte	FirmwareName (string)
...	
22nd byte	FirmwareName (string)

ChunkID: The identifier of the firmware chunk. It's a 2 byte, unsigned value. A maximum of 65535 packets is possible..
The protocol doesn't define the size of these chunks.
FirmwareName: A 20 character long text field that identifies the firmware.

5.5.2 Server ⇒ Client

Sends a firmware chunk during the firmware update. The packet contains the globally explicit name (version) of the firmware and the chunk.
If the Client requested a version or a chunk, that doesn't exist, than the Server sends a FIRMWARE packet with an empty name and a 0 chunk ID. (With empty ChunkData)

Packet body:

	Description
1st byte	ChunkID (MSB)
2nd byte	ChunkID (LSB)
3rd byte	FirmwareName (string)
...	
22nd byte	FirmwareName (string)
23rd byte	ChunkData (string)
...	
nth byte	ChunkData (string)

ChunkID: The identifier of the firmware chunk. It's a 2 byte unsigned value, which means that a maximum of 65535 chunks are possible during a single firmware update.

FirmwareName: A 20 character long text field that identifies the firmware.

ChunkData: The binary of the firmware chunk. The protocol doesn't define the sizes of the chunks, only the 256 MB ceiling in the Length field of the fixed head gives an absolute limit.

5.6 DATA

General data packets sent always from the Client to the Server. The following statements are true to these packets:

- The DATA packets can be stored (Cache flag) on the client's side and are resendable.
- The DATA packets can be saved (Save flag) in the client's side.
- The DATA packets (and only those) can request ACKNOWLEDGE.

Fixed header:

bit	7	6	5	4	3	2	1	0
1st byte	MessageType				C	S	A	R
	1	0	0	0	X	X	X	X
2nd byte	Length							

5.6.1 Client ⇒ Server

Sends data packets. The packet has one in-flight-unique ID generated by the Client. The packet has a DataType field, which refers to the semantics of the package's contents and the Client and the Server uses it by mutual agreement.

Packet body:

	Description
1st byte	MessageID
2nd byte	DataType (MSB)
3rd byte	DataType (LSB)
4th byte	Payload (string)
...	
nth byte	Payload (string)

MessageID: The in-flight-unique identifier generated by the Client to the DATA packet. The designation of the MessageID-s is continuous, which means that the Server can detect any data drop-out and can send a RESEND request to the Client with the missing MessageID.
DataType: This parameter defines the structure and content of the Payload. It's a 2 byte, unsigned value.
Reserved DataType values:

Value	Description
0	An error message on the RESEND packet, see below.
1-9	Reserved for development, testing.
>10	Device specific data packet, see in the appendix of the device's OSP protocol

The 0 value DataType: These packets can be sent from the Client to the Server if the Server sent a RESEND packet with an invalid ID. In this case the server acknowledges that the ID is invalid and moves on.
The server has to ignore every subsequent 0 valued DataType.

Payload: A data content that is specified by the DataType, its length is optional and it's structure is irrelevant to the protocol. The detailed description of the defined DataTypes can be found in the appendix.

5.6.2 Server ⇒ Client

The server can never send it to the client.
It depends on the implementation how the Client treats the violation of this rule.

5.7 RESEND

Requests a DATA packet with an in-flight-unique MessageID to be resent.
In the latest version (1.1.) of the protocol DATA packets can be sent in the Client ⇒ Server direction, this way the RESEND packet should only be interpreted in the Server ⇒Client direction.

Fixed header:

bit	7	6	5	4	3	2	1	0
1st byte	MessageType				C	S	A	R
	0	1	1	1	0	0	0	X
2nd byte	Length							

5.7.1 Server ⇒ Client

Packet body:

	Description
1st byte	MessageID

MessageID: The identification of the DATA packet requested to be resent by the Server.

5.7.2 Client ⇒ Server

This specification doesn't define circumstances in which the Client would send such a request to the Server.
It depends on the implementation how the server reacts to the arrival of such packets.

5.8 ACKNOWLEDGE

A packet type that is used to acknowledge a given DATA packet with an in-flight-unique MessageID. Ther current (1.1.) version of the protocol allows a DATA packet to be sent only in the Client ⇒ Server direction, which means that an ACKNOWLEDGE packet should only be interpreted in the Server ⇒ Client direction.

Fixed header:

bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1st byte	MessageType				C	S	A	R
	0	0	1	1	0	0	0	X
2nd byte	Length							

5.8.1 Server ⇒ Client

The Server acknowledges the arrival of a DATA packet. The ACKNOWLEDGE includes the acknowledged message's in-flight-unique identifier.

Packet body:

	Description
1st byte	MessageID

MessageID: The acknowledged packet's in-flight-unique identifier. In case of an invalid ID the Client ignores the packet.

5.8.2 Client ⇒ Server

Not permitted in the latest 1.1. version of the protocol.
It depends on the implementation how the Server reacts to the arrival of such packets.