Ethernet to CAN Converter Communication Protocol

Document version: 4
Date: April 5, 2021
Written by: Oleksandr Bogush, Axiomatic Technologies Corporation.

Contents

Introduction

The following document describes a proprietary communication protocol used by Axiomatic Ethernet to CAN and WiFi to CAN converters.

The protocol is used to transmit CAN messages over a TCP/IP network. In addition to CAN messages, the protocol also defines control and status messages necessary to communicate with the converter.

The document contains terminology and acronyms from CAN and TCP/IP protocols. Their meaning is explained in the appropriate CAN and TCP/IP documentation.

The document version contains a number and an optional letter. The number reflects changes in the actual protocol (all changes must be backward compatible), and the letter is used to change the protocol description.

Protocol Basics

The protocol is binary based. It runs on top of the standard UDP or TCP internet protocols. It uses a protocol independent message structure described below. This structure is also implemented in other Axiomatic proprietary communication protocols [1].

Protocol Message Structure

All protocol messages use the following protocol message structure, see Figure 1:

Figure 1. Protocol Message Structure

The protocol message contains:

- 11-byte *Message Header*;
- Variable size *Message Data* field, from 0 up to 245 bytes.

The overall size of the protocol message is limited to 256 bytes to protect the messages from fragmentation during transmission over the internet and to simplify handling in embedded systems with limited RAM resources.

The protocol messages are transmitted in the ascending octet order. All numerical values in all message fields, unless explicitly stated, are presented least significant byte (LSB) first.

Message Header

The protocol *Message Header* contains:

- 4-byte *Axiomatic Tag,* AXIO in capital letters;
- 2-byte *Protocol ID*;
- 2-byte *Message ID*;
- 1-byte *Message Version* (0 by default);
- 2-byte *Message Data Length.*

The protocol *Message Header* format is presented below:

Table 1. Protocol Message Header Format

Octet Offset Octet	0	1	2	3
0	A	X	I	O
	0x41	0x58	0x49	0x4F
	Axiomatic Tag			
4	Protocol ID		Message ID	
8	Message Version (0 by default)	Message Data Length		Message Data (optional, if Message Data Length > 0)

The *Axiomatic Tag* is used for the message header identification.

The *Protocol ID* defines a proprietary protocol carried by this message. This field allows different protocols to use the same protocol independent message structure. The *Protocol ID* equal, for example, 0x36BA, is presented as: (0xBA, 0x36) – LSB first and the most significant byte (MSB) second:

Table 2. Protocol ID Presentation

Octet / Offset Octet	0	1	2	3
4	0xBA	0x36	Message ID	
	Protocol ID=14010=0x36BA			

The *Message ID* defines a type of the message within the specified protocol and the *Message Data Length* – its length. The *Message Data Length* should be zero for messages without the *Message Data* field.

The *Message Version* field is used to distinguish between different message data formats in messages with the same *Message ID*. Different versions of the same message should have backward compatible data formats.

This protocol message header format allows parsing of the protocol messages without any knowledge of the message data content. Each *Message Data* field is then parsed by individual protocol-specific parsers based on: *Protocol ID, Message ID and Message Version* fields.

Message Data

The *Message Data* field format depends on the protocol and the message type defined in the *Message Header*.

Protocol ID

The proprietary communication protocol described in this document uses the *Protocol ID* = 14010 – the project number of the converter first used this protocol.

Message IDs

The following *Message IDs* are defined in the current version of the protocol.

Table 3. Message IDs.

Message ID	Message Versions	Message Name
0	0	Undefined Message
1	0	CAN and Notification Stream
2	0	Status Request
3	0,1	Status Response
4	0,1	Heartbeat

The *Undefined Message* has no parser associated with it. Messages with IDs not shown in Table 3 are not processed by the current version of the protocol. They are treated the same way, as *Undefined Messages*.

Connection Lifetime

The connection lifetime for a pair of nodes communicating using the proprietary communication protocol depends on the upper-level IP protocol and the message traffic between the nodes.

TCP Protocol

If the TCP protocol is used, the connection is maintained by the standard means of this protocol.

UDP Protocol

If the UDP protocol is used, the connection is considered to be lost after 10 seconds of inactivity on one of the nodes. Inactivity here means no protocol messages for a certain period of time.

To avoid disconnection on inactivity, it is recommended that the node communicating with the converter constantly send a *Heartbeat* or a *Status Request* message.

Heartbeat Message

The *Heartbeat* message is preferable, since it does not require a response message from the converter and it has a standard sending interval defined in the *Heartbeat* section of this document. The *Heartbeat* message with an undefined (all zeros) *Health Data* field can be potentially used for monitoring quality of the connection on the converter side in the future extensions of the protocol, if all other fields including the *Message Number* and the *Time Interval* are set.

A blank *Heartbeat* message with all data fields set to zeros is also acceptable, but only for maintaining a connection with the converter. A converter itself cannot use such a message; it must define all the message data fields.

Status Request

If the *Status Request* message is used to maintain a connection between nodes, it must not be sent more often than the *Heartbeat* message.

CAN and Notification Stream

The *CAN and Notification Stream* is the main message type used by the Ethernet to CAN converter. It encodes CAN messages. In addition to CAN messages, it can be used to send short notification messages. The *CAN and Notification Stream* has *Message ID* = 1.

The notification messages are not defined in the current version of the protocol. This feature is left for the future use.

Message Data Structure

Each *CAN and Notification Stream* message consists of *CAN* and *Notification Frames* following each other in an arbitrary order. For example:

Figure 2. CAN and Notification Stream Example

CAN Frames (CF) are used to transmit various types of CAN messages. *Notification Frames* (NF) are intended to communicate status of the CAN interface. They also can be used to send run-time error information, etc.

CAN Frames

CAN Frames have the following format:

$$\mathbf{CF} = \{CB,[TSB_1,\ldots,TSB_4],IDB_1,IDB_2,[IDB_3,IDB_4],[DB_1,\ldots,DB_8]\}, \quad (1)$$

Where:

CB – *Control Byte*;

$TSB_1,\ldots,TSB_4$ – Optional one to four bytes of the *Time Stamp*, LSB first;

$IDB_1,IDB_2,[IDB_3,IDB_4]$ – Two or four byte *CAN ID* with *Remote Frame* bit, LSB first;

$DB_1,\ldots,DB_8$ – Optional up to eight *CAN Data* bytes.

Due to a variety of CAN message types and different length of the timestamp, the length of the *CAN Frame* is variable. It is determined by the information in the first *Control Byte* (CB) of the frame.

Control Byte

Control Byte (CB) of the *CAN Frame* contains the following bits:

Figure 3. CAN Frame. Control Byte

Bit 7 = C_Bit: *Control Bit*
0: *CAN Frame*.
1: *Notification Frame*.
This bit defines a type of the frame. It should be always 0 for *CAN Frames*.

Bit 6:5 = TS_Bit[1:0]: *Time Stamp Length Bits*
Refer to Table 4 for the TS_Bit settings.

Table 4. CAN Frame. Time Stamp Length Bits

TS1_Bit	TS0_Bit	Time Stamp Length in bytes
0	0	0 – No Time Stamp
0	1	1
1	0	2

TS1_Bit	TS0_Bit	Time Stamp Length in bytes
1	1	4

Bit 4 = EID_Bit: *Extended CAN ID Bit*
0: CAN ID is standard
1: CAN ID is extended

Bit 3:0 = L_Bit[3:0] : *CAN Data Length Bits*
Refer to Table 5 for L_Bit settings.

Table 5. CAN Frame. CAN Data Length Bits

L3_Bit	L2_Bit	L1_Bit	L0_Bit	Number of Data bytes
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	Undefined
1	-	-	-	
1	1	1	1	

Time Stamp

An optional *Time Stamp* (TS) carries a time interval between the current and the previous CAN messages received on the CAN bus. It is filled by the Ethernet to CAN converter for incoming CAN messages. This field is not used for encoding outcoming CAN messages sent to the Ethernet to CAN converter by external nodes.

The *Time Stamp* is measured in milliseconds and can be: 0, 1, 2 or 4 byte long depending on the TS_Bit value in the *Control Byte*.

For 1-byte *Time Stamp*:
Bit 0:7 in TSB_1 = TS_Bit[7:0] : *1-byte Time Stamp*

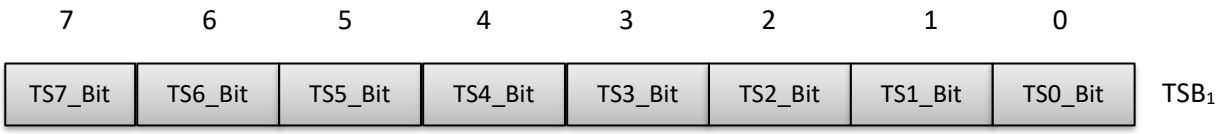

Figure 4. CAN Frame. One-byte Time Stamp

For 2-byte *Time Stamp*:
Bit 0:7 in TSB_1, Bit 0:7 in TSB_2 = TS_Bit[15:0] : *2-byte Time Stamp*

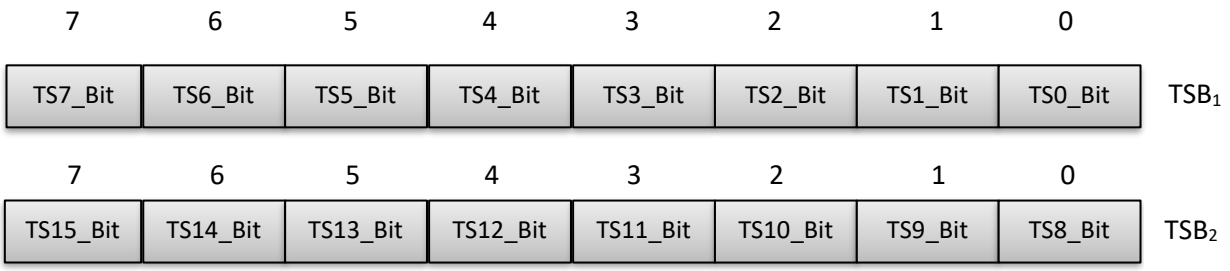

Figure 5. CAN Frame. Two-byte Time Stamp

For 4-byte *Time Stamp*:

Bit 0:7 in TSB_1, Bit 0:7 in TSB_2, Bit 0:7 in TSB_3, Bit 0:7 in TSB_4 = TS_Bit[31:0] : *4-byte Time Stamp*

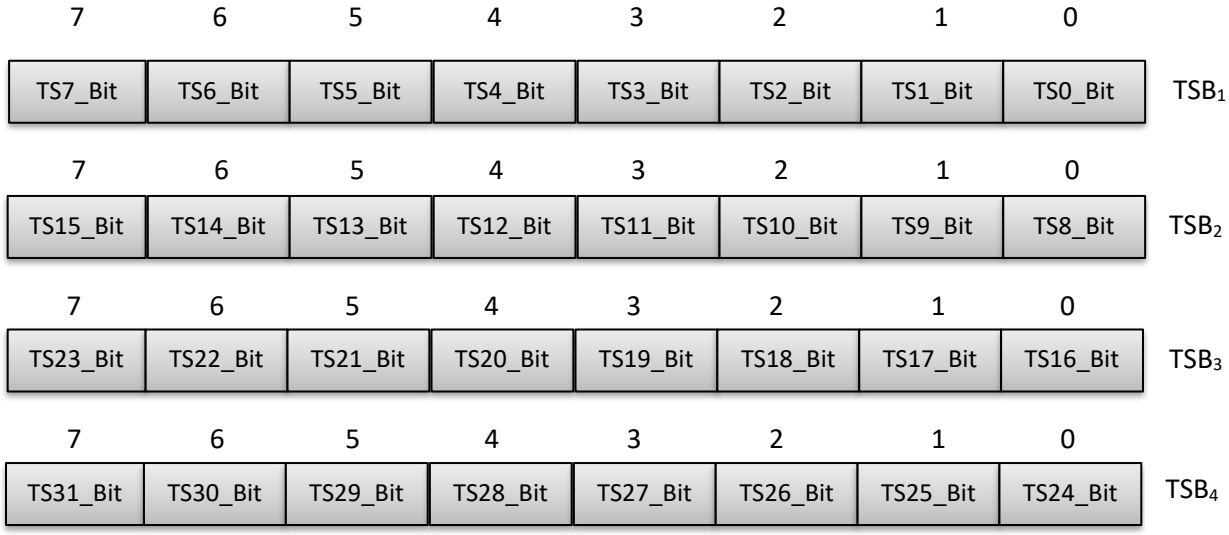

Figure 6. CAN Frame. Four-byte Time Stamp

CAN Identifier

CAN Identifier (*CAN ID*) structure is different for *Standard* and *Extended CAN ID*.

Standard CAN ID

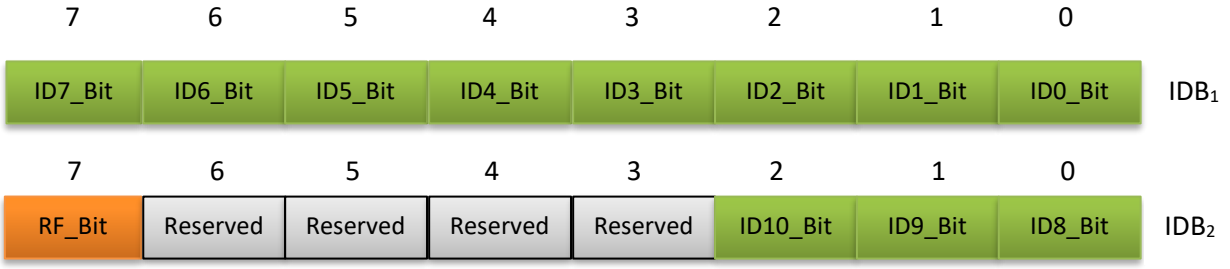

Figure 7. CAN Frame. Standard CAN ID.

Bit 0:7 in IDB_1, Bit 0:2 in IDB_2 = ID_Bit[10:0] : *CAN Standard Identifier*

Bit 3:6 in IDB_2 = Reserved.

Bit 7 in IDB_2 = RF_Bit : *Remote Frame Bit*
0: *CAN Frame* is a regular data frame
1: *CAN Frame* is a remote request for a data frame

Extended CAN ID

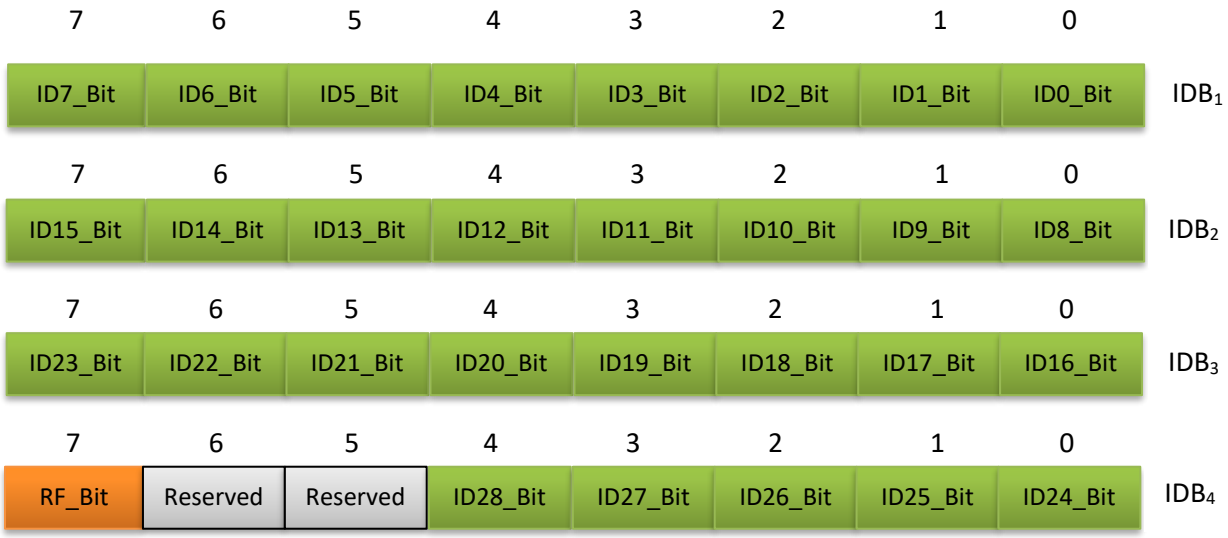

Figure 8. CAN Frame. Extended CAN ID

Bit 0:7 in IDB_1, IDB_2, IDB_3 and Bit 0:4 in IDB_4 = ID_Bit[28:0] : *CAN Extended Identifier*

Bit 5:6 in IDB_4 = Reserved

Bit 7 in IDB_4 = RF_Bit : *Remote Frame Bit*
0: *CAN Frame* is a regular data frame
1: *CAN Frame* is a remote request for a data frame

CAN Data Bytes

Optional *CAN Data* bytes are placed after *CAN ID* in the same order they appear in the CAN message. The number of *CAN Data* bytes is specified by the L_Bit field in the *Control Byte* (CB).

Notification Frames

The format of the *Notification Frames* is presented below:

NF = {NIDB,NDB_1,…,NDB_4}, **(2)**

Where:

NIDB – *Notification Identifier* byte;
NDB_1,…,NDB_4 – *Notification Data* bytes.

In opposite to the variable size *CAN Frames*, *Notification Frames* have a 5-byte fixed size format. *Notification Identifier* byte (NIDB) carries the *Notification Identifier*, which defines information sent by

the *Notification Frame* and the meaning of the four *Notification Data* bytes (NDB) associated with the frame.

Notification Identifier

The *Notification Identifier* byte (NIDB) has the following format:

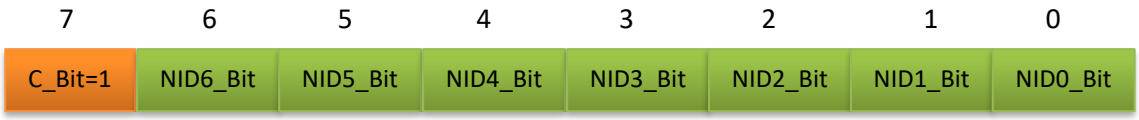

Figure 9. Notification Frame. Notification ID Byte

Bit 7 = C_Bit: *Control Bit*
0: *CAN Frame*.
1: *Notification Frame*.
This bit defines a type of the frame. It should be always 1 for *Notification Frames*.

Bit 0:6 = NID_Bit[6:0] : *Notification Identifier*
Seven bits of the *Notification Identifier* can determine up to 128 different notification messages.

Notification Data

The four *Notification Data* bytes carry information defined by the *Notification Identifier.*

There are no notification messages supported by the current version of the protocol.

Status Request

The *Stratus Request* message with *Message ID* = 2 is sent from a node to the Ethernet to CAN converter to request its status. The converter must respond with the *Status Response* message.

The *Status Request* message does not contain any data.

Status Response

The *Status Response* message with *Message ID* = 3 is sent by the Ethernet to CAN converter in response to the *Stratus Request* message. Two message versions {0,1} are defined for this message.

Status Response Data Fields

The *Status Response* message sends the following data:

SRM = {HDB_1,…, HDB_4, $CANRxDEB_1$,…,$CANRxDEB_4$, $CANTxDEB_1$,…,$CANRxDEB_4$, $CANBusOffB_1$,…,$CANBusOffB_4$, CT},

(3)

Where:

HDB_1,…, HDB_4 – 4-byte *Health Data*, LSB first,
$CANRxDEB_1$,…, $CANRxDEB_4$ – 4-byte *CAN Receive Error Counter*, LSB first,

$CANTxDEB_1$,…, $CANTxDEB_4$ – 4-byte *CAN Transmit Error Counter*, LSB first,
$CANBusOffB_1$,…, $CANBusOffB_4$ – 4-byte *CAN Bus Off Counter*, LSB first,
CT – 1-byte *Converter Type* (Defined only in *Message Version = 1*).

Health Data

The 4-byte *Health Data* field contains the health status information of the Ethernet to CAN converter. It is described in [2].

Converter Type

The following *Converter Types* are defined in *Message Version = 1*:

Table 6. Converter Types

Converter Type	Name
0[1]	Ethernet to CAN converter with CAN Voltage Output
1	WiFi to CAN converter
2	WiFi to CAN converter with CAN datalogging capability

[1]Default value if the *Converter Type* is not defined (e.g., in the *Message Version* = 0)

Heartbeat

The *Heartbeat* message with *Message ID* = 4 is sent by the Ethernet to CAN converter every 1s to maintain a link with the connected node and to inform the node about the converter status. Two message versions {0,1} are defined for this message.

Heartbeat Data Fields

The *Heartbeat* message sends the following data:

HM = {MNB_1,…, MNB_4, TIB_1,…, TIB_4, HDB_1,…, HDB_4, CT}, **(4)**

Where:

MNB_1,…, MNB_4 – 4-byte *Message Number*, LSB first,
TIB_1,…, TIB_4 – 4-byte *Time Interval* in milliseconds elapsed from the last *Heartbeat* message, LSB first,
HDB_1,…, HDB_4 – 4-byte *Health Data*, LSB first,
CT – 1-byte *Converter Type* (Defined only in *Message Version = 1*).

If the *Heartbeat* message is used only to maintain a connection between the node and the converter, the node can use a blank *Heartbeat* message with all data fields set to zero.

Message Number

The *Message Number* is a value of a free-running global counter, which is incremented every time the *Heartbeat* message is generated. One counter is used for all connected nodes.

The nodes can examine the *Message Number* consistency to check the state of the data link between the node and the converter in case a connectionless UDP communication protocol is used to carry protocol messages.

The *Message Number* can be set to zero in a blank *Heartbeat* message.

Time Interval

The *Heartbeat Message* sends the *Time Interval* in milliseconds elapsed from the last *Heartbeat* message. This value is close to 1000 for 1s heartbeat interval and can be used by nodes to estimate delays in communication between the nodes and the Ethernet to CAN converter.

The *Time Interval* can be set to zero in a blank *Heartbeat* message.

Health Data

The *Health Data* field is the same as in the *Status Response* message [2]. In can be set to zero in a blank *Heartbeat* message.

Converter Type

The *Converter Type* field is the same as in the *Status Response* message. In can be set to zero in a blank *Heartbeat* message.

References

[1] O. Bogush, "Ethernet to CAN Converter Discovery Protocol. Document version: 1A," Axiomatic Technologies Corporation, April 5, 2021.

[2] O. Bogush, "Ethernet to CAN Converter Health Status. Document version: 3," Axiomatic Technologies Corporation, April 5, 2021.

Document Version History

Document Version	Date	Author	Changes
4	April 5, 2021	Olek Bogush	Added WiFi to CAN converters in Introduction section. Removed Health Data field format. It is now a part of the "Ethernet to CAN Converter Health Status. Document Version 3". Added *Converter Type* and different versions of the *Status Response* and *Heartbeat* messages. Updated *References* section.
3	October 26, 2016	Olek Bogush	Added the document version description in the *Introduction* section. Changed the *Protocol Basics* section. Defined the protocol independent message structure. Generalized the protocol message header format to support different *Protocol IDs*. Added *Connection Lifetime* subsection.

Document Version	Date	Author	Changes
			Added a blank *Heartbeat* message. Added *References* section.
2	June 27, 2016	Olek Bogush	Added *Flash Memory Driver Initialization Operational Status* field. Updated reference to the *Ethernet to CAN Converter Health Status* document.
1A	February 5, 2016	Olek Bogush	Made the document generic. Removed references to the project 14010, except for the protocol ID. Replaced the *Ethernet to CAN Gateway* term with the term: *Ethernet to CAN Converter*. Corrected Table 7. Heath Status aggregation rules.
1	October 28, 2015	Olek Bogush	Initial version.