

Fido

Data Processing Reference

This material, including documentation and any related computer programs, is protected by copyright controlled by Nokia. All rights are reserved. Downloading and copying of the material for use with the Fido tracebox is permitted. Redistribution of any or all of this material requires a prior written consent of Nokia. This material also contains confidential information, which may not be disclosed to others without the prior written consent of Nokia.

CONTENTS

1. INTRODUCTION	4
2. TRACE DATA PROCESSING	5
2.1. XTI version 3 trace data protocol: basic concepts	5
2.1.1. Introduction	5
2.1.2. STP packets	5
2.1.3. Trace data transport with STP packets	6
2.1.4. Fido: what it does	7
2.2. XTI version 3 trace data protocol: complementary knowledge	7
2.2.1. Synchronizing	7
2.2.2. STP endianness	8
2.2.3. Timestamps	9
2.3. Fido output data messages	9
2.3.1. Introduction	9
2.3.2. OST messages	9
2.3.3. PhoNet messages	10
2.4. Exception handling	11
2.4.1. Introduction	11
2.4.2. PhoNet exception messages	11
2.4.3. Behaviour in case of illegal STP packets	11
2.4.4. Behaviour in case of too long messages	12
2.4.5. Behaviour when the collecting was not closed	12
2.4.6. Behaviour in case of traced device overflow	13
2.4.7. Behaviour in case of Fido buffer overflow	13
2.5. Markers	13
2.5.1. Introduction	13
2.5.2. OST marker messages	14
2.5.3. PhoNet marker messages	15
3. USER DATA UPLOAD TO TRACED DEVICES	16
3.1. Concepts and organization	16
3.2. Device messages targeted to Fido	16
3.2.1. Formats	16
3.2.2. The <i>Bit rate</i> message	17
3.2.3. Acknowledgements	17
4. REFERENCES	19

CHANGE HISTORY

Version : 1.0
Status : Draft
Date : 02-Jun-2009
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.
Comments : Document was created

Version : 1.1
Status : Draft
Date : 12-Jun-2009
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.
Comments : Elaboration of terms and C manifest constants (#define...). Remark: numeric values of manifest constants were not changed.

Version : 1.2
Status : Draft
Date : 16-Jun-2009
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.
Comments : Elaboration of terms.

Version : 1.3
Date : 19-Jun-2009
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.
Comments : Bugs fixed.

Version : 1.4
Date : 22-Jun-2009
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.
Comments : Fix in references.

Version : 1.5
Date : 12-Jan-2010
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.
Comments : Bugs fixed. Behaviour in case of traced device overflow revised.

Version : 1.6
Date : 19-Feb-2010
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.
Comments : Legal notices completed.

Version : 1.7
Date : 12-Mar-2010
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.
Comments : Legal notices revised.

1. Introduction

The main goal of this document is to explain how Fido handles the trace data. The reader should find answers to the following questions:

- What are the STP packets (i.e. Fido input messages)?
- How trace data transport with STP packets is organized?
- How Fido processes the trace data?
- How Fido output data are formatted?
- How Fido handles exceptional situations like lost of synchronization or illegal input data?
- What are the marker messages and how are they formatted?
- How to upload the user data from Fido to the traced device?

Thus, this document is helpful for developers, who are

- debugging software built into sophisticated devices (for example, mobile phones)
- working out PC applications capable of being used as Fido controllers and trace data receivers/analysers.

Remark that when you are reading this document you have on some occasions to study /3/ and Fido Online Help.

2. Trace data processing

2.1. XTI version 3 trace data protocol: basic concepts

2.1.1. Introduction

Fido supports five trace data protocols: STI, C-STI, XTI version 2, ST-XTI and XTI version 3. In XTI version 3 the trace data transport is provided by STP (System Trace Protocol) packets. More exactly, XTI version 3 bases on STP version 1 (see /1/)¹. The current document deals only with XTI version 3.

The detailed STP specification can be found in /1/. A short overview is presented below.

2.1.2. STP packets²

STP packets are low-level message units used for transporting the high-level trace data messages from traced device to an external device.

The STP packets consist of series of nibbles (4-bit half-bytes). The first nibble (ID, opcode) determines the packet type. The opcode is followed by 0...9 byte payload. Fido recognizes the following STP packets:

Mnemonic	Opcode (ID)	Payload (Data field)	Length in nibbles	Remarks
NULL	0000		1	Filler packet, used for synchronization only
MASTER	0001	M[7:0]	3	Specifies the master for the following packets
OVRF	0010	O[7:0]	3	Overflow in traced device internal STP buffers (the following STP packets stream is not complete)
C8	0011	C[7:0]	3	Specifies the channel for the following data packets
D8	0100	D[7:0]	3	Data packets used for transporting 1, 2 4 or 8 bytes of data. In this document summarily denoted as Dnn.
D16	0101	D[15:0]	5	
D32	0110	D[31:0]	9	
D64	0111	D[63:0]	17	
D8TS	1000	D[7:0]+T[7:0]	5	Timestamped (specially marked) data packets. Besides transporting data, these packets are used to indicate the end of a data block (i.e. high-level trace message). In this document summarily denoted as DnnTS.
D16TS	1001	D[15:0]+T[7:0]	7	
D32TS	1010	D[31:0]+T[7:0]	11	
D64TS	1011	D[63:0]+T[7:0]	19	

¹ STP has also version 2 but it is under development yet.

² The older versions of STP specification call them STP messages.

2.1.3. Trace data transport with STP packets

In STP trace messages may be simultaneously transferred on 256 channels belonging to 256 masters. A typical sequence of STP packets transporting one single trace message is:

MASTER	C8	Dnn	Dnn	...	DnnTS
--------	----	-----	-----	-----	-------

Each trace message must start with the originator (master and channel). Data is transported by Dnn packets. Finally, timestamped DnnTS packet indicates the trace message end.

If the next trace message is sent by same master and on same channel, the MASTER and C8 packets may be omitted:

		Trace message #1			Trace message #2 from the same master and on the same channel		
MASTER	C8	Dnn	...	DnnTS	Dnn	...	DnnTS

Trace messages can be interleaved. The example presented below shows how two trace messages from two different masters may be transported simultaneously:

Trace message #1 from master #2 on channel #3 starts				Trace message #2 from master #3 on channel #4				Trace message #1 continues				
MASTER #2	C8 #3	Dnn	...	MASTER #3	C8 #4	Dnn	...	DnnTS	MASTER #2	C8 #3	Dnn	...

Synopsis:

1. To specify the master sending the following bunch of data, the traced device sends the MASTER packet. This master is considered as the current and the following packets are considered to belong to it. The current master stays valid until the next MASTER packet. There is no any default master. In Fido data sent before the first MASTER packet is discarded.
2. To specify the channel used by the current master, the traced device sends the C8 packet. This channel is considered as the current and the following packets are considered to belong to it. The current channel stays valid until the next C8 or MASTER packet (or in other words, each MASTER packet also invalidates the current channel). The default channel is 0x00. It means that if the traced device has specified only the current master, the current channel is supposed to be 0x00³. Read also the important remark from 3.2.1.
3. The contents of trace messages is transported by a sequence of Dnn and DnnTS packets:
 - 3.1. It is supposed that data from Dnn/DnnTS packets belong to the current master and current channel.
 - 3.2. There is no any specific packet marking the start of transport. Any Dnn/DnnTS packet can carry the first bytes of trace message.
 - 3.3. The last bytes of the trace messages must be carried by a DnnTS packet⁴.
4. STP packets transporting components of trace messages may be interleaved. It means that the traced device may temporarily interrupt the transporting of a message by sending another MASTER and/or C8 STP packet(s), thus switching over to transporting another message.

³ There are so-called HW-type masters that do not support channels and therefore do not send C8 packets. For Fido it means that they have only one channel: 0x00.

⁴ Consequently, if the DnnTS packet is also the first one then the whole trace message was transported by one STP packet.

Note that in typical hardware implementations the traced device built-in System Trace Module (STM) inserts the MASTER and C8 packets into STP packets stream automatically. The programmer's responsibility is to specify the data and the channels to be used.

2.1.4. Fido: what it does

Fido extracts the trace message components packed into Dnn and DnnTS packets and, depending on their originator (i.e. the current master and the current channel) distributes them between different buffers. When the assembling of a particular trace message has ended, Fido supplies it with a header and sends on to user application software.

For all the originators the assembling rules are the same. There are no any exceptions.

Synopsis:

- When a Dnn STP packet has been received and there is no any uncompleted output message related to its originator, Fido considers that the assembling on that master and on that channel has begun and creates a new buffer. The received data byte(s) are stored at the beginning of this buffer. No any data checking is provided.
- When a DnnTS STP packet has been received and there is no any uncompleted output message related to its originator, Fido creates a new buffer, stores the data byte(s) and ends the assembling. Thus, in that case the user will get a Fido message wrapping data from a single STP packet.
- When a Dnn STP packet has been received and there is an uncompleted output message related to its originator, Fido appends the received data byte(s) to the appropriate buffer. No any checking is provided.
- When a DnnTS STP packet has been received and there is an uncompleted output message related to its originator, Fido appends the received data byte(s) to the appropriate buffer and ends the assembling.
- When the assembling has been ended, Fido calculates the timestamp (see 2.2.3), supplies the collected data with OST (see 2.3.2) or PhoNet (see 2.3.3) header and sends on to the user application.

About reaction to the OVRF packets read 2.4.6.

2.2. XTI version 3 trace data protocol: complementary knowledge

2.2.1. Synchronizing

The STP data transport does not use any special packets denoting that sending of a new message has started (see 2.1.4). Therefore the synchronizing is provided by timeouts. Pause in STP data stream exceeding a certain limit⁵ suggests that sending of one or more simultaneous trace messages has ended and the next STP data packets would carry new trace messages. Consequently, the traced device must at first finish all the messages it currently is dealing with (i.e. send the DnnTS packet(s)) and only after that take a break. If it fails to do so, Fido detects an exception (see 2.4).

⁵ This limit is one of the Fido setup parameters. For details see Fido Online Help and /3/.

2.2.2. STP endianness

The STP specification (see /1/) does not specify in uniquely understandable way in what order the data bytes in Dnn/DnnTS packets must be transferred. It only declares that the most significant nibble must be transferred first.

Lets take as an example a memory region containing C string “Hi!”

String:	H (0x48)	i (0x69)	! (0x21)	terminating zero (0x00)
Memory address:	N	N+1	N+2	N+3

Suppose that a processor reads this field as a four-byte integer. Then a little-endian processor interprets it as 0x00216948 and a big-endian processor as 0x48692100. Due to the ambiguity in STP standard, hardware designers are free to select, in a D32 packet what is the byte the System Trace Module (STM) outputs first: N + 3 (0x00) or N (0x48).

To overcome this hardware variety, several software fixes can be applied:

1. Swap the data bytes in Dnn/DnnTS packets in traced device software before writing them to STM. Cons: adds significant load on the traced device processor.
2. Swap the data bytes in Dnn/DnnTS packets in Debug and Test System (DTS) like Fido. Cons: additional knowledge about endiannesses must be inserted into DTS.
3. Include additional information into high-level trace messages or specific knowledge into DTS so that the trace messages can be interpreted in consistent manner. Cons: too traced device and DTS dependent, not general enough.

These three methods do not conflict with each other. They can be used in mix. In Fido, however, the second method is implemented. As different masters may have different architecture, Fido needs to know the endianness information about each of the 256 masters⁶.

Note that the master’s endianness and the order of bytes in STP packets are not in one-to-one relationship – depending on the STM implementation, for example, a little-endian master may still set a byte from lower memory address as the first byte in STP packet. Therefore, in this documentation the term STP endianness⁷ denotes the order of bytes in STP data packets and it must be clearly distinguished from the master’s endianness itself.

Synopsis:

- If the master is specified as big-endian, the data bytes are inserted into the buffer in the same order as they arrived.
- If the master is specified as little-endian, the 2-, 4- or 8-byte data word must be swapped. Example: if the D64 packet data bytes arrived in order [B1, B2, B3, B4, B5, B6, B7, B8], byte B8 is inserted into the buffer at first and byte B1 at last. In this way the exact copy of master’s memory word is recovered.

⁶ The array of endiannesses is one of the Fido setup parameters. For details see Fido Online Help and /3/.

⁷ The parallel term is “transport endianness”

2.2.3. Timestamps

Fido calculates the timestamp values itself and does not turn any attention to timestamps attached to the DnnTS packets. In other words, the DnnTS signals that the sending of a message has been ended but the timestamp value from it is not used.

The timestamp bases on the values retrieved from the clock of PC on which *Fido.exe* is running. However, external applications may set another base. About details see /3/.

2.3. Fido output data messages

2.3.1. Introduction

For historical reasons, Fido supports two output message protocols: OST (Open System Trace, see /2/) and Nokia legacy protocol (called also as PhoNet protocol). The default protocol is OST. Data is transported from Fido to external applications via TCP/IP connection. About details read /3/.

A Fido output message may consist of the following components:

1. Header.
2. Timestamp.
3. Data field.

2.3.2. OST messages

When the output data protocol is OST, Fido output messages are formatted according to the following general rules:

Byte	Name	Contents and remarks
0	Version	Always OST_VERSION (0x10)
1	Entity ID	Always 0x00
2	Protocol ID	Always OST_PROTOCOL_XTIV3 (0x84)
3	Length	Number of bytes following this byte. However, if the length exceeds 255, byte 3 must be 0.
[4 : 7]	Extended length	This field is present only when byte 3 is 0. Contains the length of message starting from byte 8. Little endian format is used.
4 or 8	Tracebox port ⁸	Depends on the Fido port and its operating mode: <ul style="list-style-type: none"> • If a port is emulating Musti⁹ channel CH1, the tracebox port value is 0x0A. • If a port is emulating Musti channel CH2, the tracebox port value is 0x0B. • Otherwise, the tracebox port value is calculated as Fido port ID + 0x0B (for example, for messages from device connected to Fido port 0x07 this byte gets value 0x12).
5 or 9	Master ID	STP master (see 2.1)

⁸ The parallel term is “sender object”

⁹ Musti box is one of the predecessors of Fido tracing system.

6 or 10	Channel ID	STP channel (see 2.1)
[7 : 14] or [11 : 18]	Timestamp	This field consists of two sections: 1. The flag (bits [63 : 60]) is the remnant inherited from Fido predecessors. Its value is always 1101. 2. The timestamp value itself (i.e. bits [59 : 0]) in <u>big-endian</u> mode. The unit is nanosecond
[15 : n] or [19 : n]	Data field	Created from traced device data bytes.

2.3.3. PhoNet messages

When the output data protocol is PhoNet, Fido output messages are formatted according to the following general rules:

Byte	Name	Contents and remarks
0	Media	Always TB_MEDIA_TCPIP (0x1D)
1	Receiver device	Always TB_DEV_PC (0x10) ¹⁰
2	Sender device	Always TB_DEV_TRACEBOX (0x4C)
3	Not used	Always TB_TRACEBOX (0x7C)
4, 5	Message length	Integer (n-5) in <u>big-endian</u> format. Specifies the number of bytes following the message length field (i.e. starting from byte 6).
6	Receiver object	May be set by application providing data download. Default value is 0x00.
7	Sender object	As tracebox port (OST byte 4 or 8), specified in 2.3.2.
8	Transaction ID	Counter incremented right before the sending to PC, possible values are 0, 1,...255, 0, 1... The origin (i.e. through which port the data was acquired) is not taken into consideration.
9	Message ID	Always TB_TRACE_MSG (0x94).
10	Master ID	Master that has sent this data (see 2.1)
11	Channel ID	Channel on which the data was sent (see 2.1)
[12 : 19]	Timestamp	As specified in 2.3.2
[20 : n]	Body	Created from traced device data bytes.

¹⁰ The receiver and sender devices are actually 6-bit values. Furthermore, the receiver and sender objects (i.e. bytes 6 and 7) are actually 10-bit values and consist of two parts: two bits are stored as the least significant bits of bytes 1 and 2 and the remaining 8 bits in bytes 6 and 7. In messages created by Fido, however, the maximum value for sender objects is 0x13 and the receiver object assigned by PC cannot increase 0xFF. Therefore, we can handle all the header components as 8-bit values; the only exception is the message length.

2.4. Exception handling

2.4.1. Introduction

The following abnormal situations may occur:

- The traced device has sent something that Fido is not able to identify (see also 2.4.3).
- Fido is not able to complete the collecting because the device has not sent the DnnTS STP message (see 2.4.5).
- The number of bytes in the device data message exceeds the limits specified for the converted messages (see 2.4.4).
- The device signals about internal overflow (see 2.4.6).
- The device data amount is too large and Fido buffers have overflowed (see 2.4.7).

When the normal processing cannot be continued, Fido buffers may contain data that are not converted to the output messages yet. The main problem is how to undertake them. Fido behaviour in these situations depends on the current output data protocol. If the protocol is OST, data that Fido cannot manage are simply discarded. If the protocol is PhoNet, the abovementioned data are converted into specific output messages (see 2.4.2).

2.4.2. PhoNet exception messages

The exception messages have the following common format:

Byte	Name	Contents and remarks
[0 : 8]		As presented in 2.3.3.
9	Message ID	Always TB_TRACE_EXCEPTION_MSG (0x95)
[10 : 19]		As presented in 2.3.3
20	Exception code	<p>The following values are allowed:</p> <ul style="list-style-type: none"> • TB_EXCEPTION_LONG_MESSAGE_FIRST (0x01), see 2.4.4. • TB_EXCEPTION_LONG_MESSAGE (0x02), see 2.4.4. • TB_EXCEPTION_LONG_MESSAGE_LAST (0x03), see 2.4.4. • TB_EXCEPTION_UNFINISHED_MESSAGE (0x04), see 2.4.3, 2.4.5. • TB_EXCEPTION_OVRF_PACKET (0x05), see 2.4.6. • TB_EXCEPTION_LONG_UNFINISHED_MESSAGE (0x06), see 2.4.4, 2.4.5. • TB_EXCEPTION_LONG_OVRF_PACKET (0x07), see 2.4.4, 2.4.6. • TB_EXCEPTION_OVRF (0x08), see 2.4.6
[21 : n]	Data field	

2.4.3. Behaviour in case of illegal STP packets

If the device has sent a packet that cannot be identified as a legal STP packet, Fido temporarily stops the processing. To resume it, Fido has to wait for a pause in the STP input message stream. If the pause has

been long enough (its minimum length is a configurable setup parameter, see Fido Online Help or /3/), Fido becomes ready to receive the next input burst (read also 2.2.1).

As the processing was temporarily broken off, collecting of some output data messages (see 2.1.4) may stay uncompleted. If the output data protocol is OST, Fido clears all the open buffers, i.e. discards a part of device data. If the output data protocol is PhoNet, the contents of buffers with uncompleted messages are converted into exception messages (see 2.4.2). The exception code in this case is `TB_EXCEPTION_UNFINISHED_MESSAGE` (0x04). Timestamp attached to the exception message marks the moment when the error occurred. If some of the uncompleted messages are too long, they are handled as presented in 2.4.4

In addition, Fido generates the `TB_ERROR_RESERVED_STP_PACKET` (0x27) error marker (see 2.5).

In extreme cases, however, Fido box software may get totally confused and hangs. In that case the behaviour does not depend on the output data protocol. To overcome the problem, the user has for a while to close the relevant port. It clears the box internal buffers. To indicate the crash, Fido creates the `TB_ERROR_DATA_CORRUPTED` (0x06) error marker (see 2.5)

2.4.4. Behaviour in case of too long messages

The total number of bytes in a Fido output message cannot exceed 0xFFFF. If the device has sent a message that cannot be wrapped into a single Fido message and the protocol is OST, Fido creates an output message with maximal length and discards the surplus data. If the protocol is PhoNet, Fido creates a sequence of exception messages (see 2.4.2). The exception codes in this sequence are:

- `TB_EXCEPTION_LONG_MESSAGE_FIRST` (0x01) for the first message.
- `TB_EXCEPTION_LONG_MESSAGE_LAST` (0x03) for the last message.
- `TB_EXCEPTION_LONG_UNFINISHED_MESSAGE` (0x06) for the last message if the collecting was not completed due to illegal data (see 2.4.3) or because the DnnTS STP packet closing the collecting was not sent (see 2.4.5).
- `TB_EXCEPTION_LONG_OVRF_PACKET` (0x07) for the last message if the collecting was not completed because the device informed about internal overflow (see 2.4.6).
- `TB_EXCEPTION_LONG_MESSAGE` (0x02) for the middle messages.

Timestamps attached to those exception messages are calculated using the timestamp of the last normal message and/or the values periodically read from the system clock.

This situation is not handled as an error: the processing continues; error message is not created; LEDs do not indicate error.

2.4.5. Behaviour when the collecting was not closed

The traced device is supposed to dispatch the components of an output message (see 2.2.1) in one burst. To detect the end of burst, Fido checks the length of pauses in the input data stream. If this pause exceeds a configurable limit (see Fido Online Help or /3/), the burst is considered to have ended.

If the burst has been ended, but there are still buffers with uncompleted messages and the output protocol is OST, Fido creates regular output messages containing the data received so far. If the protocol is PhoNet, Fido converts the contents of buffers with uncompleted messages into exception output messages (see 2.4.2). The exception code is set to `TB_EXCEPTION_UNFINISHED_MESSAGE` (0x04). If some of the uncompleted messages are too long, they are handled as presented in 2.4.4.

Timestamps attached to the output messages in that case mark the end of timeout: i.e. the moment when Fido has become ready to resume normal data processing.

In addition, Fido generates the TB_ERROR_PAUSE_IN_STP_STREAM_LONG (0x23) error marker (see 2.5).

2.4.6. Behaviour in case of traced device overflow

When the OVRF¹¹ packet was detected, Fido closes the buffer corresponding to the current master and channel. If the protocol is OST, data already stored in the buffer are converted into regular output message. If the protocol is PhoNet, Fido creates two exception messages:

1. First, it puts the contents of buffer into the TB_EXCEPTION_OVRF_PACKET (0x05) exception message (see 2.4.2). If the uncompleted message is too long, it is handled as presented in 2.4.4. Buffers corresponding to the other masters as well as to the channels on the current master are not affected.
2. Second, it creates the TB_EXCEPTION_OVRF (0x08) exception message. The STP/OVRF message data bits (see 2.1.2) are copied into the exception message data field (i.e. onto byte 21, see 2.4.2).

Timestamps attached to the output messages are calculated using the timestamp of the last normal message and/or the values periodically read from the system clock.

Error marker is not created and data processing is not stopped.

2.4.7. Behaviour in case of Fido buffer overflow

Behaviour in this case is similar to the behaviour described in 2.4.3. Fido temporarily stops the processing. To resume it, Fido has to wait for a pause in the STP input packet stream. If the pause has been long enough (its minimum length is specified by the same configuration parameter as the pause for resuming after illegal STP data) and there is already some free space in the box memory, Fido continues to process data.

If the output data protocol is OST, Fido clears all the open buffers, i.e. discards a part of device data. If the output data protocol is PhoNet, the contents of buffers with uncompleted messages are converted into exception messages (see 2.4.3).

In addition, Fido generates the TB_ERROR_OVERFLOW (0x26) error marker (see 2.5).

2.5. Markers

2.5.1. Introduction

Markers are specific output messages inserted between the regular output messages containing device tracing data. Marker messages are created when:

- The user has ordered Fido to generate a marker (see Fido Online Help and /3/).
- A data processing error has occurred (see 2.4).
- The external application has set new time basis (see 2.2.3 and /3/). In that case Fido generates two markers located one after another: the first of them has timestamp calculated according to the old time base and the second has timestamp calculated according to the new time base.

¹¹ OVRF packets are sent only by so-called HW-type masters.

2.5.2. OST marker messages

Compare with format specified in 2.3.2.

Byte	Name	Contents and remarks
0	Version	Always OST_VERSION (0x10)
1	Entity ID	Always OST_ENTITY_DEFAULT (0xF0).
2	Protocol ID	Always OST_PROTOCOL_TRACEBOX (0x81)
3	Length	Number of bytes following this byte. However, if the length exceeds 255, byte 3 must be 0.
[4 : 7]	Extended length	This field is present only when byte 3 is 0. Contains the length of message starting from byte 8. Little endian format is used.
4 or 8	Message ID	Always TB_MARKER (0xFF).
5 or 9	Marker type	Allowed values are: <ul style="list-style-type: none"> • TB_MARKER_USER_ASSIGNED (0x02) for markers set by the user. • TB_MARKER_TIMEBASE (0x03) for markers informing that the time basis has been adjusted • Error code in case of error markers: <ul style="list-style-type: none"> ○ TB_ERROR_DATA_CORRUPTED (0x06) – see 2.4.3. ○ TB_ERROR_RESERVED_STP_PACKET (0x27) – see 2.4.3. ○ TB_ERROR_PAUSE_IN_STP_STREAM_LONG (0x23) – see 2.4.5. ○ TB_ERROR_OVERFLOW (0x26) – see 2.4.7.
6 or 10	Sender object	Allowed values are: <ul style="list-style-type: none"> • TB_OBJ_TRACEBOX_MAIN (0x01) in case of markers set by user as well as in case of markers informing about changes in time basis. • In case of error markers depends on port on which the error has occurred. See 2.3.2.
[7 : 14] or [11 : 18]	Timestamp	As specified in 2.3.2
[15 : 18] or [19 : 22]	Marker index	Four-byte <u>big-endian</u> integer. Index of the first marker created after <i>Fido.exe</i> startup is 1.
19 or 23	Length of the optional binary parameters field	If the marker type is not TB_ERROR_RESERVED_STP_PACKET (0x27), zero
[20 : m] or [24 : m]	Optional binary parameters	If the marker type is TB_ERROR_RESERVED_STP_PACKET (0x27), this field contains the not recognized STP packet. Otherwise omitted.
[m : n]	Optional comments	<ul style="list-style-type: none"> • In case of TB_MARKER_USER_ASSIGNED (0x02), this field (if not omitted) may contain any data the user wants to append to the marker. If the marker is set from the control panel, the comment is a regular C string (ASCII characters, terminating zero byte). • In case of TB_MARKER_TIMEBASE (0x03) this field contains regular C string: “<i>Old timestamp!</i>” for the first marker, “<i>New timestamp!</i>” for the second marker.

		• Error markers do not have comments.
--	--	---------------------------------------

2.5.3. PhoNet marker messages

Compare with format specified in 2.3.3.

Byte	Name	Contents and remarks
[0 : 6]		As in 2.3.3.
7	Sender object	As in 2.5.2.
8	Transaction ID	As in 2.3.3.
9	Message ID	Always TB_MARKER (0xFF)
10	Marker type	As in 2.5.2.
11	Spare byte	Always zero
[12 : 19]	Timestamp	As in 2.3.3.
[20: 23]	Marker index	As in 2.5.2
24	Length of the optional binary parameters field	
[25 : m]	Optional binary parameters	
[m : n]	Optional comments	

3. User data upload to traced devices

3.1. Concepts and organization

Actually, uploading data to the traced devices (the return channel problem) is out of STP scope. Fido, however, is designed with taking into consideration the solutions implemented in its predecessors as well as in the traced devices. Therefore, the return channel is formed from UART as channel directed from Fido to traced device and from device feedback (or acknowledge) messages sent to Fido with STP packets.

The user may send his/her data to traced device by commands applied from the control panel or from the external control application. The formatting rules for data and commands are presented in /3/.

When Fido has received the user data, it organizes their upload on to the device. The transfer protocol is as follows:

1. To start, the Fido prepares the device (for example, wakes the device up) sending the break signal. The device must reply with the *Bit rate* message (see 3.2.2). Fido stops the break signal after 200ms or when the reply has arrived. The response timeout is 2000ms: if the reaction to break signal has not reached Fido inside this interval, the upload is considered to be failed. Fido tries to establish the bit rate proposed by the device as exactly as possible. Error exceeding 2% means that the requested bit rate is not achievable, i.e. the upload has failed. The lowest accepted bit rate is 100b/s.
2. When Fido has accepted the bit rate and configured itself, it dispatches the number of bytes waiting for the upload (4-byte little-endian integer). The device has to acknowledge the data length (see 3.2.3). The acknowledgement timeout is 100ms: if the reaction has not reached Fido inside this interval, the upload is considered to be failed.
3. When the device has acknowledged the length, Fido starts the actual data upload.
4. At last, the device has to acknowledge (see 3.2.3) that it has got all the transmitted data. The acknowledgement timeout is 100ms: if the reaction has not reached Fido inside this interval, the upload is considered to be failed.

UART channel parameters other than the bit rate are fixed: 8 data bits, no parity, 1 stop bit, neither software nor hardware handshake.

3.2. Device messages targeted to Fido

3.2.1. Formats

The device may send a message to Fido at the very beginning when the actual STP endianness (see 2.2.2) is unknown. Therefore, Fido must be able to interpret the message correctly even when the endianness stored in the setup parameters has wrong value. Therefore, all the device messages targeted to Fido must strictly follow some specific rules:

- The message must be transported on channel 0x01.
- All the Dnn STP packets used for transport must have the same number of data bytes. For example, when the first four bytes are transported with D32, the following bytes must be also transported with D32. Exception: the closing DnnTS packet and a few Dnn packets right before it can have less (but

not more) data bytes. Furthermore, number of bytes in those closing STP packets must be in decreasing order. Thus, if D32 is the base transport packet, the last three bytes must be transported with D16 and D8TS (but not with D8 and D16TS).

- The message must start with four-byte or eight-byte message header (*ID* denotes the message identifier, see 3.2.2 and 3.2.3):
 - *[0x00, 0x00, ID, 0x00]*, in that case transport with D64 packets is not allowed.
 - *[0x00, 0x00, 0xFF, 0x00, 0x01, 0x01, ID, 0x00]*
- The traced device may freely select between those two headers.

If those rules are violated, Fido misinterprets the message. Remark that all the data sent by the device are also converted into regular output data messages (see 2.1.4).

Important: obviously, the traced device should never transport ordinary trace messages starting with 0x00 on channel 0x01. Fido may interpret those messages misleadingly as messages targeted to itself.

3.2.2. The *Bit rate* message

This message has two alternatives:

1. If the *ID* (see 3.2.1) is 0x04, the bit rate must be presented as a four-byte big-endian integer. For example, if the bit rate is 115200 b/s and four-byte header is selected, the device must send to Fido the following sequence of bytes:

*0x00, 0x00, 0x04, 0x00,
0x00, 0x01, 0xC2, 0x00*

Transporting may be organized as follows:

D16 0000, D16 0400, D16 0001, D16TS C200

2. If the *ID* is 0x03¹², the bit rate must be presented as ASCII text. For example, if the bit rate is 2285714.286 bits per second and the eight-byte header is selected, the device must send to Fido the following sequence of bytes:

*0x00, 0x00, 0xFF, 0x00, 0x01, 0x01, 0x03, 0x00,
0x32, 0x32, 0x38, 0x35, 0x37, 0x31, 0x34, 0x2E, 0x32, 0x38, 0x36*

Transporting may be organized as follows:

D32 0000FF00, D32 01010300, D32 32323835, D32 3731342E, D16 3238, D8TS36

Fido uses the ANSI C *scanf* “%lg” format string to convert the text to float-point C variable.

3.2.3. Acknowledgements

Actually, the acknowledgements are four-byte or eight-byte headers (see 3.2.1). There is no any data payload following the header.

1. Acknowledgement with *ID* 0x01 confirms that the device has got the number of data bytes. For example, a four-byte header may include the following bytes: *0x00, 0x00, 0x01, 0x00*

¹² This alternative should be regarded as obsolete

2. Acknowledgement with ID 0x02 confirms that all the data had reached the device. Example: *0x00, 0x00, 0x02, 0x00*

4. References

1. MIPI Alliance Standard for System Trace Protocol Specification. Version. 1.00. 15 Dec 2006.
2. MIPI Alliance Specification for Open System Trace (OST) Base Protocol. Version 0.80.00, Revision 1.02. 1 February 2009. Draft.
3. Fido Programmers Reference. The PDF-version is downloadable from <http://www.liewenthal.ee/projects/fido/documents>