

# **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

<b>1. INTRODUCTION</b>	<b>7</b>
<b>1.1. Supported trace data protocols</b>	<b>7</b>
1.1.1. Legacy protocols	7
1.1.2. XTI version 3 protocol	7
<b>1.2. Scope of this document</b>	<b>7</b>
<b>1.3. How to read this document</b>	<b>8</b>
<b>2. TRACE DATA PROCESSING</b>	<b>9</b>
<b>2.1. XTI v3 / STP v1 basic concepts</b>	<b>9</b>
2.1.1. STP v1 packets	9
2.1.2. XTI v3 trace data transport with STP v1 packets	9
2.1.3. How Fido handles the stream of STP v1 packets	11
2.1.4. Synchronizing	11
<b>2.2. XTI v3 / STP v2 basic concepts</b>	<b>12</b>
2.2.1. STP v2 packets versus STP v1 packets	12
2.2.2. STP v2 packets: full list	13
2.2.3. XTI v3 trace data transport with STP v2 packets	14
2.2.4. How Fido handles the stream of STP v2 packets	16
2.2.5. Synchronizing and versioning	17
<b>2.3. XTI version 3 trace data protocol: complementary knowledge</b>	<b>17</b>
2.3.1. STP endianness	17
2.3.2. Calculation of timestamps in STP v1	18
2.3.3. Calculation of timestamps in STP v2	19
<b>2.4. Fido output data messages</b>	<b>19</b>
2.4.1. OST messages	19
2.4.2. PhoNet messages	20
<b>2.5. Exception handling</b>	<b>21</b>
2.5.1. PhoNet exception messages	21
2.5.2. Behaviour in case of illegal STP packets	22
2.5.3. Behaviour in case of too long messages	22
2.5.4. Behaviour when the STP v1 collecting was not closed	22
2.5.5. Behaviour in case of STP v1 traced device overflow	23
2.5.6. Behaviour in case of STP v2 master errors	23
2.5.7. Behaviour in case of STP v2 global errors	24
2.5.8. Behaviour in case of Fido buffer overflow	24
<b>2.6. Markers</b>	<b>24</b>
2.6.1. OST marker messages	24

2.6.2. PhoNet marker messages	25
<b>3. SENDING DATA TO TRACED DEVICES</b>	<b>27</b>
<b>3.1. UART</b>	<b>27</b>
3.1.1. Concepts and organization	27
3.1.2. Formats of messages targeted to Fido	27
3.1.3. The <i>Bit rate</i> message	28
3.1.4. Acknowledgements	29
3.1.5. Example	29
<b>3.2. SWD</b>	<b>30</b>
3.2.1. Overview	30
3.2.2. Data transfer in SWD	30
3.2.3. Fido SWD messages	30
3.2.4. Protocol violation and its results	30
3.2.5. Implementation: sending data to traced device	31
3.2.6. Implementation: reading data from traced device	31
3.2.7. SWD messages wrapped into OST messages	32
3.2.8. SWD messages wrapped into PhoNet messages	32
<b>4. REFERENCES</b>	<b>33</b>

## 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 STP v1 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.

## Version 3.1

Version : 2.0  
Date : 21-Jun-2010  
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.  
Comments : Extended. XTI v3 / STP v2 is the new protocol.

Version : 2.1  
Date : 31-Aug-2011  
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.  
Comments : New chapter: SWD.

Version : 3.0  
Date : 14-Sep-2011  
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.  
Comments : Document revised.

Version : 3.1  
Date : 18-Jan-2012  
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.  
Comments : New SWD messages wrapper format

## ACRONYMS

Acronym	Description
C-STI	Custom Serial Trace Interface
DAB	Debug Access Port
DTS	Debug and Test System
JTAG	Joint Test Action Group
MIPI	Mobile Industry Processor Interface
ST-XTI	eXtended Trace Interface implemented in STMicroelectronics STn8810 devices
STI	Serial Trace Interface
STM	System Trace Module
STP	System Trace Protocol
SWD	Serial Wire Debug
UART	Universal Asynchronous Receiver/Transmitter
XTI	eXtended Trace Interface

# 1. Introduction

## 1.1. Supported trace data protocols

### 1.1.1. Legacy protocols

Fido supports the following legacy protocols:

1. STI
2. C-STI
3. XTI version 2
4. ST-XTI

The support of legacy protocols is inherited from a Fido predecessor device called Musti. Fido and Musti process the trace data of those four protocols in an identical way. Detailed processing rules descriptions are presented in Musti documentation<sup>1</sup>. The current document does not deal with legacy protocols.

### 1.1.2. XTI version 3 protocol

In XTI version 3 the trace data is transported by STP (System Trace Protocol) packets. The STP has two versions: the older version 1 (see /1/) and the latest version 2 (see /4/). The detailed XTI version 3 data processing rules are presented in this document.

When you connect your Fido box to traced device via MIPI 60-pin cable, you will meet two specific terms: dedicated mode and muxed mode. Those terms specify the sets of connector pins selected for data transport. In both modes the debug data protocol is the standard STP version 1. The details are presented in /5/.

## 1.2. Scope of this document

The main goal of this document is to explain how Fido handles the XTI v3 trace data. The reader should find answers to the following questions:

- What are the STP version 1 and 2 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 loss 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 in processor-controlled devices (for example, mobile phones)

---

<sup>1</sup> The legacy protocols and Musti documentation are not public. To get more information please contact Liewenthal Electronics <http://www.liewenthal.ee/projects/fido/>.

- Elaborating PC applications which task is to control Fido and to retrieve and analyse the trace data outputted by Fido.

## 1.3. How to read this document

Readers interested only in STP v1 may omit chapter 2.2. Readers unfamiliar with STP v1 and interested only in STP v2 may omit chapters 2.1 and 2.2.1. Readers experienced in STP v1 and interested in STP v2 will discover that chapters 2.1.2 and 2.1.3 generally fall together with chapters 2.2.3 and 2.2.4 respectively.

Remark: when you are reading this document you will on some occasions have to study:

- Fido Programmer's Reference (/3/)
- Fido Online Help (/6/).

## 2. Trace data processing

### 2.1. XTI v3 / STP v1 basic concepts

#### 2.1.1. STP v1 packets<sup>2</sup>

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 v1 packets consist of series of 4-bit nibbles. The first nibble is the opcode that determines the packet type. The opcode is followed by 0...9 byte payload. Fido recognizes the following STP v1 packets:

Mnemonic	Opcode (in hex)	Payload (data field)	Length in nibbles	Remarks
NULL	0		1	Filler packet, ignored
MASTER	1	M[7:0]	3	Specifies the STP master for the following packets
OVRF	2	O[7:0]	3	Overflow in the traced device internal STP buffers (the following STP packets stream is not complete)
C8	3	C[7:0]	3	Specifies the channel for the following data packets
D8	4	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	5	D[15:0]	5	
D32	6	D[31:0]	9	
D64	7	D[63:0]	17	
D8TS	8	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, see 2.1.2). In this document summarily denoted as DnnTS. Remark that Fido calculates the timestamps assigned to output data itself. Timestamps from STP packets are ignored (read 2.3.2).
D16TS	9	D[15:0]+T[7:0]	7	
D32TS	A	D[31:0]+T[7:0]	11	
D64TS	B	D[63:0]+T[7:0]	19	

#### 2.1.2. XTI v3 trace data transport with STP v1 packets

The STP v1 documentation does not specify how to apply the STP packets for transportation of long messages. The data transport rules presented below are Fido-specific.

The trace messages can be simultaneously transferred by 256 masters and on 256 channels per master. A typical sequence of STP v1 packets transporting one single trace message is:

<sup>2</sup> The older versions of STP specification call them STP messages.

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

A trace message starts with the MASTER packet followed by the C8 packet. Those two packets present the source of trace message. Data is transported by Dnn packets. Finally, a timestamped DnnTS packet transports the last bits and also indicates the trace message end. Exception: if the next trace message is sent by the same master and on the same channel, the MASTER and C8 packets can be omitted, for example:

		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 can 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 bouquet of data, the traced device sends the MASTER packet. This master is now considered as the current master and the following packets are considered to belong to it. The current master stays valid until the next MASTER packet. There is no default master. Data sent before the first valid MASTER packet is discarded.
2. To specify the channel used by the current master, the traced device sends the C8 packet. This channel is now considered as the current channel. The following packets are considered to belong to the current channel. The current channel stays valid until the next C8 or MASTER packet is received. It is important to note that the MASTER packet invalidates the current channel. The default current channel is 0x00. Consequently, if the traced device has specified only the current master, the current channel is 0x00<sup>3</sup>. Read also the important remark from 3.1.2.
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 the current channel.
  - 3.2. There is no specific packet marking the start of trace message. Any Dnn/DnnTS packet can carry the first bytes of a trace message.
  - 3.3. The last bytes of trace messages must be carried by a DnnTS packet<sup>4</sup>.
4. STP v1 packets transporting parts of trace messages may be interleaved by other trace messages. 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.

Note: in typical hardware implementations the traced device built-in System Trace Module (STM) inserts the MASTER and C8 packets into STP v1 packets stream automatically. The programmer's responsibility is to specify the data.

<sup>3</sup> 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.

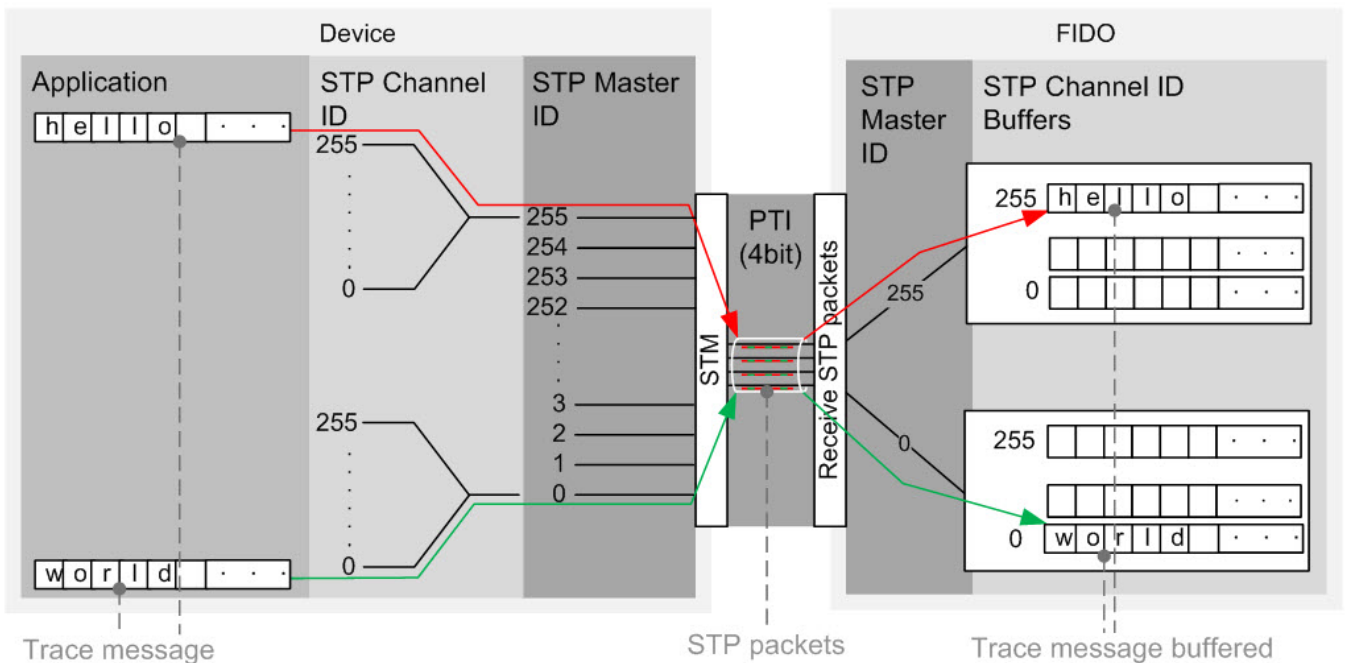
<sup>4</sup> Consequently, if the DnnTS packet is also the first one then the whole trace message was transported by one STP packet.

### 2.1.3. How Fido handles the stream of STP v1 packets

Fido extracts the trace message components packed into Dnn and DnnTS packets and according to their master and channel distributes them between different buffers. The buffers are handled in the following way:

	Buffer	
Received STP v1 packet	There is no open buffer related to the master and channel of the received STP v1 packet.	There is already an open buffer related to the master and channel of the received STP v1 packet.
Dnn	Fido creates a new buffer and stores the data into it.	Fido appends the data to the buffer.
DnnTS	Fido creates a new buffer, stores the data into it and closes the buffer.	Fido appends the data to the buffer and closes the buffer.

When the assembling of a particular trace message has ended (i.e. the buffer is closed), Fido supplies the received data with a header (see 2.4), calculates the timestamp (see 2.3.2) and stores the complete output message in a user-specified file or sends on to the user application software. The following figure illustrates the data processing rules presented above:



About the reaction to OVRF packets read 2.5.5.

### 2.1.4. Synchronizing

If at the moment of connection the traced device is already sending data (the so-called hot plug-in), Fido has to wait for a pause in input data stream. Data received before the pause is discarded. The length of pause is checked. If it exceeds the resynchronization limit (see below), Fido supposes that there are no more pending (unfinished) trace messages and that the first STP packet received after the pause denotes

the beginning of a new trace message. In other words, in case of hot plug-in the normal operating starts when a pause long enough is detected.

Similarly, when a data error has occurred, Fido starts to discard input data and resumes the normal operating only after pause exceeding the resynchronization limit.

The reason of behaviour presented above is that the STP v1 data transport does not use any special packets denoting that sending of a new message has started (see 2.1.2 and 2.1.3). Therefore the synchronizing is provided by timeouts.

Programmers developing the device software should know that Fido has two setup parameters<sup>5</sup> specifying the limits for pauses in data stream:

1. The maximal pause allowed between trace data messages during composing of output messages. When a pause in the trace data stream has exceeded this value, Fido checks the presence of uncompleted output messages. If they are present, Fido announces trace data error. The factory default of this pause is 100ms.
2. Minimal pause in trace data stream needed for resynchronization after data errors and in case of hot plug-in. The factory default of this pause is 1ms.

Generally, the device software developers are free to set the pauses in trace data stream. However, they have to take into consideration that:

1. Fido setup parameters may have maximal and minimal allowed values.
2. The minimal pause needed for resynchronization should not exceed the maximal pause allowed between components.

Obviously, the users of Fido should know what are the input data stream properties and set the proper values for setup parameters beforehand.

## 2.2. XTI v3 / STP v2 basic concepts

### 2.2.1. STP v2 packets versus STP v1 packets

1. Each STP packet starts with an opcode (packet ID). In STP v1 the opcodes (from 0x0 until 0xB) occupy one nibble. In STP v2 the opcode may occupy one (from 0x0 until 0xE), two (from 0xF, 0x1 until 0xF, 0xE) and three nibbles (from 0xF, 0x0, 0x0 until 0xF, 0x0, 0xB). Moreover, the opcode of ASYNC packet occupies 22 nibbles.
2. In STP v1 the timestamp (TS) occupies 2 nibbles. STP v2 has several timestamp formats. One of them coincides with the STI v1 timestamp; the other three have variable length up to 16 nibbles. The traced device must specify the format sending the VERSION packet.
3. Timestamp values from the STP v1 packets are not used. Timestamp values from STP v2 packets are used for calculation the final timestamp included into the output message (read 2.3.3).
4. In STP v1 there can be 256 masters and the current master is specified by MASTER packet having a two-nibble payload. In STP v2 the possible number of masters is 65536 and the current master may be specified by either the M8 or the M16 packets.
5. In STP v1 each master can have 256 channels and the current channel is specified by C8 packet having a two-nibble payload. In STP v2 each master may have 65536 channels. The current channel can be specified by either the C8 or the C16 packets.

---

<sup>5</sup> For details see /6/ and /3/.

6. STP v1 data packets can transport 8, 16, 32 or 64 bits of data. In STP v2 there are also data packets transporting 4 bits of data. Furthermore, in addition to ordinary and timestamped data packets defined in STP v1 there are also marked data packets and marked timestamped data packets.
7. In STP v1 the last bits of a trace message must be carried by the timestamped DnnTS data packet (read 2.1.2). In STP v2 a timestamped data packet does not mark the end of trace message. To denote the end of trace message in STP v2, the traced device has to dispatch the last data bits with marked data message or send a specific FLAG or FLAG\_TS packet.

### 2.2.2. STP v2 packets: full list

Fido recognizes the following STP v2 packets:

Mnemonic	Opcode (nibbles in hex)	Payload (data field)	Remarks
NULL	0		Filler packet, , may be used for synchronization, see 2.2.5
NULL_TS	F, 0, 1	TS	
M16	F, 1	Master ID	Specifies the 16-bit current master (i.e. the master for the following packets).
M8	1	Master ID, lower bits	Specifies the 8 lower bits of the current master. The 8 higher bits are not affected.
C16	F, 3	Channel ID	Specifies the 16-bit current channel (i.e. channel for the following data packets).
C8	3	Channel ID, lower bits	Specifies the 8 lower bits of the current channel. The 8 higher bits are not affected.
D4	C	4 data bits	Packets used for transporting data. In this document summarily denoted as Dnn.
D8	4	8 data bits	
D16	5	16 data bits	
D32	6	32 data bits	
D64	7	64 data bits	
D4M	F, D	4 data bits	Packets used for transporting the last bits of a trace message. In this document summarily denoted as DnnM.
D8M	F, 8	8 data bits	
D16M	F, 9	16 data bits	
D32M	F, A	32 data bits	
D64M	F, B	64 data bits	
D4TS	F, C	4 data bits + TS	Packets used for transporting data. In this document summarily denoted as DnnTS.
D8TS	F, 4	8 data bits + TS	
D16TS	F, 5	16 data bits + TS	
D32TS	F, 6	32 data bits + TS	
D64TS	F, 7	64 data bits + TS	
D4MTS	D	4 data bits + TS	Packets used for transporting the last bits of a trace message. In this document summarily denoted as DnnMTS.
D8MTS	8	8 data bits + TS	
D16MTS	9	16 data bits + TS	
D32MTS	A	32 data bits + TS	
D64MTS	B	64 data bits + TS	

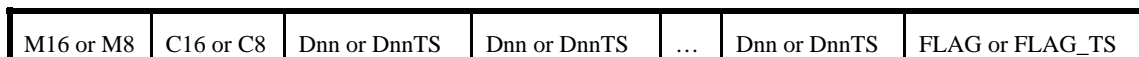
FLAG	F, E		Used to inform Fido about the end of trace message.
FLAG_TS	E	TS	
MERR	2	Error info, 8 bits	Master error
GERR	F, 2	Error info, 8 bits	Global error
ASYNC	F, ... , F, 0 (21 Fs in row)		Used for synchronization, see 2.2.5.
VERSION	F, 0, 0	Version number, 4 bits	
USER	F, 0, 2	Up to 16 nibbles of user data including the nibble presenting the length	Those messages Fido discards
USER_TS	F, 0, 3	As in USER + TS	
TIME	F, 0, 4	Time synchronization, payload depends on version	
TIME_TS	F, 0, 5	As in TIME + TS	
TRIG	F, 0, 6	Position of a trigger event, 8 bits	
TRIG_TS	F, 0, 7	As in TRIG + TS	
FREQ	F, 0, 8	Timestamp clock frequency, 32 bits	
FREQ_TS	F, 0, 9	As in FREQ + TS	
XSYNC	F, 0, A	Cross synchronization event identifier, 8 bits	
XSYNC_TS	F, 0, B	As in XSYNC + TS	

### 2.2.3. XTI v3 trace data transport with STP v2 packets

In STP v2 trace messages can be transferred simultaneously by 65536 masters and on 65536 channels per master. A typical sequence of STP v2 packets transporting one single trace message is:



or



A trace message starts with the MASTER packet followed by the C8 packet. Those two packets present the source of trace message. Data is transported by Dnn and/or DnnTS packets. Finally, a marked DnnM or DnnMTS packet transports the last bits and indicates the trace message end. As an alternative, all the data bits are transported by Dnn and/or DnnTS packets and the message end is denoted by FLAG or FLAG\_TS packet.

Exception: if the next trace message is sent by same master and on same channel, the M16/M8 and C16/C8 packets can be omitted, for example

		Trace message #1				Trace message #2 from the same master and on the same channel				
M16	C16	Dnn	.....	DnnM	Dnn	.....				FLAG

Trace messages can be interleaved. The example presented below shows how two trace messages from two different masters can 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			
M16 #2	C8 #3	Dnn	...	M8 #3	C8 #4	Dnn	...	FLAG	M8 #2	C8 #3	Dnn	...

Synopsis:

1. To specify the master sending the following bouquet of data, the traced device sends the M8 or M16 packet. This master is now considered as the current and the following packets are considered to belong to it. The current master stays valid until the next M8 or M16 packet. If the first master packet is M8 specifying the 8 lower bits, the higher 8 bits are set to zero. In STP v2 0 master 0 is considered as the default master.

Examples:

- If the master packet is [1111 0001] [0010 0011 0100 0101] (i.e. M16, see table in 2.2.2), the current master is always set to 0010 0011 0100 0101.
- If the new master packet is [0001] [1001 1010] (i.e. M8) and the current master is as presented above, the new current master will be 0010 0011 1001 1010
- If the first master packet after synchronizing and versioning (see 2.2.5) is [0001] [1001 1010] (i.e. M8), the current master is set to 0000 0000 1001 1010.

2. To specify the channel used by the current master, the traced device sends the C8 or C16 packet. This channel is now considered as the current and the following packets are considered to belong to it. The current channel stays valid until the next C8, C16, M8 or M16 packet. If the first channel packet is C8 specifying the 8 lower bits, the higher 8 bits are set to zero. The M8 or M16 packets set the current channel to 0<sup>6</sup>. Consequently, if a master uses only channel 0 or does not support channels as such, sending of C8 and C16 packets is unnecessary. Read also the important remark from 3.1.2

Examples:

- If the channel packet is [1111 0011] [0010 0011 0100 0101] (i.e. C16, see table in 2.2.2), the current channel is always set to 0010 0011 0100 0101.
- If the new channel packet is [0011] [1001 1010] (i.e. C8) and the current channel is as presented above, the new current channel will be 0010 0011 1001 1010.
- If the first channel packet after master packet or after synchronizing and versioning is [0011] [1001 1010] (i.e. C8), the current channel is set to 0000 0000 1001 1010.

3. The contents of trace messages is transported by a sequence of Dnn/DnnTS/DnnM/DnnMTS packets:
  - 3.3. It is supposed that any Dnn/DnnTS/DnnM/DnnMTS and FLAG/FLAG\_TS packet belongs to the current master and current channel.
  - 3.4. There is no any specific packet marking the start of trace message. Any Dnn/DnnTS/DnnM/DnnMTS packet can carry the first bits of trace message.

<sup>6</sup> Obviously, the traced device may specify the master with M8 packet and the channel with C16 packet or the master with M16 packet and the channel with C8 packet.

3.5. The last bits of the trace messages can be carried by a DnnM/DnnMTS packet<sup>7</sup> as well as by a Dnn/DnnTS packet. In the last case the FLAG/FLAG\_TS packet denotes the end of trace message.

4. STP packets transporting parts of trace messages can be interleaved. It means that the traced device may temporarily interrupt the transporting of a message by sending another M8/M16 and/or C8/C16 STP packet(s), thus switching over to transporting another message.

Note: in typical hardware implementations the traced device built-in System Trace Module (STM) inserts the M16/M8 and C16/C8 packets into STP v2 packets stream automatically. The programmer's responsibility is to specify the data and the channels.

#### 2.2.4. How Fido handles the stream of STP v2 packets

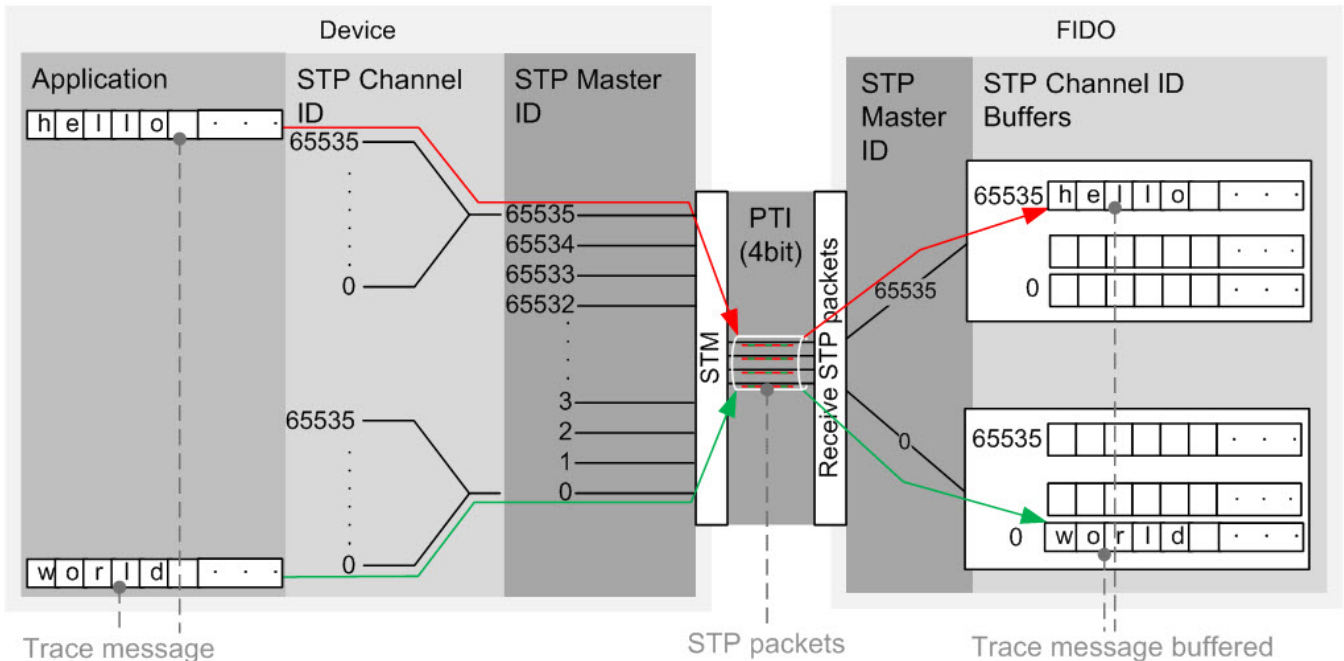
Fido extracts the trace message components packed into Dnn/DnnTS/DnnM/DnnMTS packets and, depending on their originator (i.e. the current master and the current channel) distributes them between different buffers. The buffers are handled in the following way:

Received STP v2 packet	Buffer	
		There is no open buffer related to the master and channel of the received STP v2 packet.
Dnn/DnnTS	Fido creates a new buffer and stores the data into it.	Fido appends the data to the buffer.
DnnM/DnnMTS	Fido creates a new buffer, stores the data into it and closes the buffer.	Fido appends the data to the buffer and closes the buffer.
FLAG/FLAG_TS	Fido creates a new empty buffer and closes it. <sup>[JL1]</sup>	Fido closes the buffer.

When the assembling of a particular trace message has ended (i.e. the buffer is closed), Fido supplies the received data with a header (see 2.4), calculates the timestamp (see 2.3.3) and stores the complete output message in a user-specified file or sends on to the user application software. If the buffer is empty, the output message will contain header and timestamp but no data..

The following figure illustrates the data processing rules presented above:

<sup>7</sup> Consequently, if the DnnM/DnnMTS packet is also the first one then the whole trace message was transported by one STP packet.



About the reaction to MERR and GERR packets read 2.5.6 and 2.5.7.

### 2.2.5. Synchronizing and versioning

Synchronization may be provided:

- By ASYNC packet followed by VERSION packet
- By sequence of 22 NULL/NULL\_TS packets followed by VERSION packet.

The VERSION packet payload contents (one nibble) present the version number. For Fido version 0 is illegal (see 2.5.2). Versions 1, 2, 3 and 4 are distinguished by timestamp formats: 1 is the STP v1 timestamp, 2 is the relative timetamp, 3 is the absolute timestamp and 4 is the absolute timestamp encoded using gray code.

The ASYNC packet not followed by the VERSION packet is illegal (see 2.5.2). But the VERSION packet not preceded by the ASYNC packet is permissible.

The VERSION packet sets the current master and current channel to 0.

## 2.3. XTI version 3 trace data protocol: complementary knowledge

### 2.3.1. STP endianness

The STP specifications (see /1/ and /4/) does not specify in uniquely understandable way in what order the data bytes in Dnn/DnnTS/DnnM/DnnMTS packets must be transferred. It only declares that the most significant nibble must be transferred first.

To exemplify the problem, take 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 which byte in a D32 packet 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 method number 2 is implemented. As different masters may have different architecture, Fido needs to know the endianness information of each of the 256/65536 masters<sup>8</sup>.

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 of STP packet. Therefore, in this documentation the term STP endianness<sup>9</sup> 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.

### 2.3.2. Calculation of timestamps in STP v1

Fido calculates the timestamp values on its own. Timestamps attached to the DnnTS packets are ignored.

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

<sup>8</sup> The array of endiannesses is one of the Fido setup parameters. For details see /6/ and /3/.

<sup>9</sup> The parallel term is “transport endianness”

### 2.3.3. Calculation of timestamps in STP v2

To set the base for timestamp calculations, Fido needs two source values:

1. The value retrieved from the clock of PC on which *Fido.exe* is running. Remark that the external application controlling Fido may set another base. About details see /3/.
2. The first timestamp extracted from a DnnTS or DnnMTS packet. The timestamp format must be 2, 3 or 4 (see 2.2.5). If this source value is not acquired, only the value from PC clock is applied.

## 2.4. Fido output data messages

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.

### 2.4.1. 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 <sup>10</sup>	Depends on the Fido port and its operating mode: <ul style="list-style-type: none"> <li>• If a port is emulating Musti<sup>11</sup> channel CH1, the tracebox port value is 0x0A.</li> <li>• If a port is emulating Musti channel CH2, the tracebox port value is 0x0B.</li> <li>• 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).</li> </ul>
5 or 9	Master ID	STP master (see 2.1 and 2.2).

<sup>10</sup> The parallel term is “sender object”

<sup>11</sup> Musti box is one of the predecessors of Fido tracing system.

6 or 10	Channel ID	STP channel (see 2.1 and 2.2).
[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 for XTI v3 / STP v1 messages and for XTI v3 / STP v2 messages when Fido-set timestamp is used. When traced device originating timestamps are used with STP v2 protocol, then the flag is set as follows: a. 1100 – timestamp value received from traced device. b. 1110 – timestamp value approximated by Fido (for example when traced device set timestamp seems incorrect). c. 1101 – timestamp value set by Fido. 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.

### 2.4.2. 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) <sup>12</sup>
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 2.4.1.
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.

<sup>12</sup> 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.

9	Message ID	Always TB_TRACE_MSG (0x94).
10	Master ID	Master that has sent this data (see 2.1 and 2.2).
11	Channel ID	Channel on which the data was sent (see 2.1 and 2.2).
[12 : 19]	Timestamp	As specified in 2.4.1
[20 : n]	Body	Created from traced device data bytes.

## 2.5. Exception handling

The following abnormal situations may occur:

- The traced device has sent something that Fido is not able to identify (see also 2.5.2).
- Fido is not able to complete the collecting because the device has not sent the DnnTS STP message (only for STP v1, see 2.5.4).
- The number of bytes in the device data message exceeds the limits specified for the converted messages (see 2.5.3).
- The traced device signals about internal errors (see 2.5.5, 2.5.6 and 2.5.7).
- The device data amount is too large and Fido buffers have overflowed (see 2.5.8).

When the normal data processing cannot be continued Fido buffers may contain data that are not converted to output messages yet. The main problem is how to undertake them. The behaviour in these situations depends on the current output data protocol. If the protocol is OST, the data that Fido cannot manage are simply discarded.. If the protocol is PhoNet, the abovementioned data are converted into specific output messages (see 2.5.1).

### 2.5.1. PhoNet exception messages

The exception messages have the following common format:

Byte	Name	Contents and remarks
[0 : 8]		As presented in 2.4.2.
9	Message ID	Always TB_TRACE_EXCEPTION_MSG (0x95)
[10 : 19]		As presented in 2.4.2
20	Exception code	<p>The following values are allowed:</p> <ul style="list-style-type: none"> <li>• TB_EXCEPTION_LONG_MESSAGE_FIRST (0x01), see 2.5.3.</li> <li>• TB_EXCEPTION_LONG_MESSAGE (0x02), see 2.5.3.</li> <li>• TB_EXCEPTION_LONG_MESSAGE_LAST (0x03), see 2.5.3.</li> <li>• TB_EXCEPTION_UNFINISHED_MESSAGE (0x04), see 2.5.4.</li> <li>• TB_EXCEPTION_OVRF_PACKET (0x05), see 2.5.5, 2.5.6 and 2.5.7.</li> <li>• TB_EXCEPTION_LONG_UNFINISHED_MESSAGE (0x06), see 2.5.3.</li> <li>• TB_EXCEPTION_LONG_OVRF_PACKET (0x07), see 2.5.3</li> <li>• TB_EXCEPTION_OVRF (0x08), see 2.5.5, 2.5.6 and 2.5.7.</li> </ul>
[21 : n]	Data field	

### 2.5.2. Behaviour in case of illegal STP packets

If the traced device has sent a packet that cannot be identified as a legal STP v1 or STP v2 packet, Fido temporarily stops the processing. To resume the normal work:

- In case of STP v1, Fido has to wait for a pause in the STP v1 input message stream. If the pause has been long enough (its minimum length is a configurable setup parameter, see /6/ or /3/), Fido becomes ready to receive the next input burst (read also 2.1.4).
- In case of STP v2, Fido has to wait for the new synchronization (see 2.2.5).

As the processing was temporarily broken off, collecting of some output data messages (see 2.1.3 and 2.2.3) 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 2.5.1). 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 2.5.3

In addition, Fido generates the `TB_ERROR_RESERVED_STP_PACKET` (0x27) error marker (see 2.6).

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 2.6).

### 2.5.3. 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 2.5.1). 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 2.5.2) or because the DnnTS STP v1 packet closing the collecting was not sent (see 2.5.4).
- `TB_EXCEPTION_LONG_OVRF_PACKET` (0x07) for the last message if the collecting was not completed because the device informed about internal error (see 2.5.5, 2.5.6 and 2.5.7).
- `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.

### 2.5.4. Behaviour when the STP v1 collecting was not closed

The traced device is supposed to dispatch the components of an output message (see 2.1.4) 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 2.5.1). 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 2.5.3.

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 2.6).

### 2.5.5. Behaviour in case of STP v1 traced device overflow

When the `OVRF`<sup>13</sup> packet was detected and there is an uncompleted buffer corresponding to the current master and channel, Fido closes it. If the protocol is OST, data already stored in the closed buffer are converted into regular output message. If the protocol is PhoNet, Fido behaviours in the following way:

1. The contents of closed buffer are put into the `TB_EXCEPTION_OVRF_PACKET` (0x05) exception message (see 2.5.1). If the uncompleted message is too long, it is handled as presented in 2.5.3. Buffers corresponding to the other masters as well as to the channels on the current master are not affected.
2. The `TB_EXCEPTION_OVRF` (0x08) exception message is created. The `OVRF` message data bits (see 2.1.1) are copied into the exception message data field (i.e. onto byte 21, see 2.5.1).

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

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

### 2.5.6. Behaviour in case of STP v2 master errors

The STP v3 `MERR` packet corresponds to STP v2 `OVRF` packet. When the `MERR` packet was detected and there is an uncompleted buffer corresponding to the current master and channel, Fido closes it. If the protocol is OST, data already stored in the closed buffer are converted into regular output message. If the protocol is PhoNet, Fido behaviours in the following way:

1. The contents of closed buffer are put into the `TB_EXCEPTION_OVRF_PACKET` (0x05) exception message (see 2.5.1). If the uncompleted message is too long, it is handled as presented in 2.5.3. Buffers corresponding to the other masters as well as to the channels on the current master are not affected.
2. The `TB_EXCEPTION_OVRF` (0x08) exception message is created. The `MERR` message data bits (see 2.2.2) are copied into the exception message data field (i.e. onto byte 21, see 2.5.1).

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

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

---

<sup>13</sup> `OVRF` packets are sent only by so-called HW-type masters.

### 2.5.7. Behaviour in case of STP v2 global errors

When the GERR packet was detected, Fido closes all the uncompleted buffers. If the protocol is OST, data already stored in the closed buffer(s) are converted into regular output message(s). If the protocol is PhoNet, Fido behaviours in the following way:

1. The contents of closed buffer(s) are put into the TB\_EXCEPTION\_OVRF\_PACKET (0x05) exception message(s) (see 2.5.1). If some uncompleted message(s) are too long, they are handled as presented in 2.5.3.
2. One TB\_EXCEPTION\_OVRF (0x08) exception message is created. The GERR message data bits (see 2.2.2) are copied into the exception message data field (i.e. onto byte 21, see 2.5.1).

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

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

### 2.5.8. Behaviour in case of Fido buffer overflow

Behaviour in this case is identical with the behaviour described in 2.5.2 except that the generated error marker is TB\_ERROR\_OVERFLOW (0x26).

## 2.6. Markers

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 2.5).
- The external application has set new time basis (see 2.3.2 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.

### 2.6.1. OST marker messages

Compare with format specified in 2.4.1.

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"> <li>• TB_MARKER_USER_ASSIGNED (0x02) for markers set by the user.</li> <li>• TB_MARKER_TIMEBASE (0x03) for markers informing that the time basis has been adjusted</li> </ul>

		<ul style="list-style-type: none"> <li>• Error code in case of error markers: <ul style="list-style-type: none"> <li>○ TB_ERROR_DATA_CORRUPTED (0x06) – see 2.5.2.</li> <li>○ TB_ERROR_RESERVED_STP_PACKET (0x27) – see 2.5.2.</li> <li>○ TB_ERROR_PAUSE_IN_STP_STREAM_LONG (0x23) – see 2.5.4.</li> <li>○ TB_ERROR_OVERFLOW (0x26) – see 2.5.8.</li> </ul> </li> </ul>
6 or 10	Sender object	<p>Allowed values are:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• In case of error markers depends on port on which the error has occurred. See 2.4.1.</li> </ul>
[7 : 14] or [11 : 18]	Timestamp	As specified in 2.4.1. Markers will always have Fido-set timestamp.
[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	<p>If the marker type is TB_ERROR_RESERVED_STP_PACKET (0x27), this field contains the description of not recognized STP packet:</p> <ul style="list-style-type: none"> <li>• In case of STP v1 the field includes one byte presenting the illegal opcode (see also 2.1.1) located on the lower bits.</li> <li>• In case of STP v2 the field presents all the correct nibbles and the first incorrect nibble. For example, if the traced device tries to set version 0 (see 2.2.5), the field contains nibbles F, 0, 0 (VERSION packet correct opcode) and 0 (VERSION packet illegal payload). Each nibble is stored in separate byte, the lower bits are used.</li> </ul> <p>Otherwise this field is omitted.</p>
[m : n]	Optional comments	<ul style="list-style-type: none"> <li>• 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).</li> <li>• 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.</li> <li>• Error markers do not have comments.</li> </ul>

## 2.6.2. PhoNet marker messages

Compare with format specified in 2.4.2.

Byte	Name	Contents and remarks
[0 : 6]		As in 2.4.2.
7	Sender object	As in 2.6.1.

8	Transaction ID	As in 2.4.2.
9	Message ID	Always TB_MARKER (0xFF)
10	Marker type	As in 2.6.1.
11	Spare byte	Always zero
[12 : 19]	Timestamp	As in 2.4.2.
[20: 23]	Marker index	As in 2.6.1
24	Length of the optional binary parameters field	
[25 : m]	Optional binary parameters	
[m : n]	Optional comments	

## 3. Sending data to traced devices

Uploading data to the traced devices (the reverse channel problem) is out of the STP scope. The current Fido version supports two types of reverse channel:

- UART
- SWD (see 3.2).

### 3.1. UART

#### 3.1.1. Concepts and organization

The user may send data to the 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 the upload to the traced device. The transfer protocol is as follows:

1. To start, Fido prepares the traced device (for example, wakes the device up) sending the break signal. The device must reply with the *Bit rate* message (see 3.1.3). 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 within this interval, the upload is considered as 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 3.1.4). The acknowledgement timeout is 100ms: if the reaction has not reached Fido within this interval, the upload is considered as failed.
3. When the device has acknowledged the length, Fido starts the actual reverse channel data upload.
4. At last, the device has to acknowledge (see 3.1.4) that it has got all the transmitted data. The acknowledgement timeout is 100ms: if the reaction has not reached Fido within this interval, the upload is considered as failed.

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

Remark also that the protocol presented here matches with the user data upload protocols implemented in Fido predecessors.

#### 3.1.2. Formats of messages targeted to Fido

The device may send a message to Fido at the very beginning when the actual STP endianness (see 2.3.1) is unknown. Therefore, Fido must be able to interpret the message correctly even when the endianness stored in the setup parameters has wrong value. For that reason, all the device messages targeted to Fido must strictly follow the next rules:

- The message must be transported on channel 0x01.
- All the Dnn STP packets used for transport must have the same number of data nibbles. 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 nibbles. Furthermore, number of nibbles in those closing STP packets must be in de-

creasing order. Thus, if D32 is the base transport packet and STP v1 is the protocol., the last three bytes must be transported with D16 and D8TS (but not with D8 and D16TS). For example, if D32 is the base transport packet:

- In case of STP v1 the last three bytes must be transported with D16 and D8TS (but not with D8 and D16TS).
- In case of STP v2 the last packets must be D16/D16TS and D8M/D8MTS or D16/D16TS, D8/D8TS and FLAG/FLAG\_TS.
- The message must start with four-byte or eight-byte message header (*ID* denotes the message identifier, see 3.1.3 and 3.1.4):
  - *[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 2.4).

**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 it.

### 3.1.3. The *Bit rate* message

This message has two alternatives:

1. If the *ID* (see 3.1.2) 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* (STP v1)
- *D16 0000, D16 0400, D16 0001, D16M C200* (STP v2)

2. If the *ID* is 0x03<sup>14</sup>, 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, D8TS 36* (STP v1)
- *D32 0000FF00, D32 01010300, D32 32323835, D32 3731342E, D16 3238, D8M 36* (STP v2)

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

---

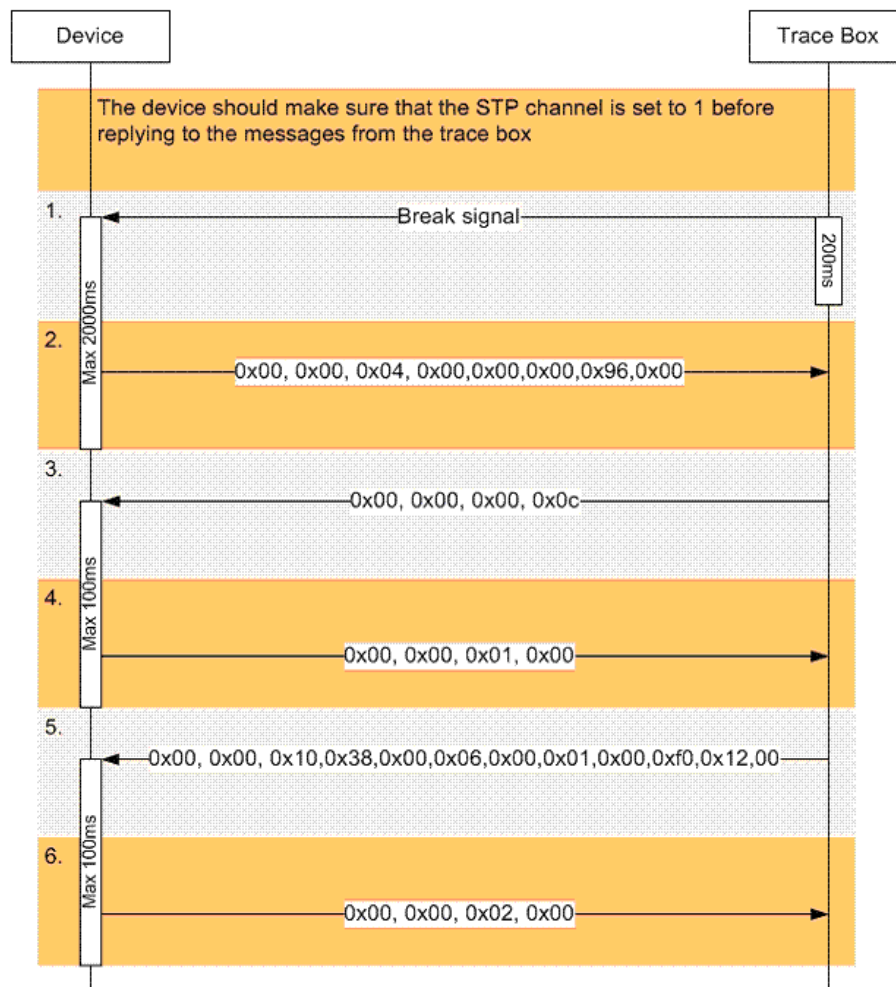
<sup>14</sup> This alternative should be regarded as obsolete

### 3.1.4. Acknowledgements

Actually, the acknowledgements are four-byte or eight-byte headers (see 3.1.2). 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*
2. Acknowledgement with ID 0x02 confirms that all the data had reached the device. Example: *0x00, 0x00, 0x02, 0x00*

### 3.1.5. Example



1. The trace box sends BREAK signal (very long-lasting LOW – approximately 200 ms) for waking up the UART in the device (in case the UART is sleeping).
2. The device must then send the Set Bit Rate control command over STP - within 2000ms.
3. The trace box sends 4 bytes indicating the length of the message that it is going to send.
4. The device must then send the Acknowledge Data Length control command - within 100ms.
5. The trace box sends the actual message.
6. The device must then send the Acknowledge Data Received control command - within 100ms.

## 3.2. SWD

### 3.2.1. Overview

The serial wire debug (SWD) is an alternative to JTAG transport layer for accessing the debug access port in ARM-Cortex based devices. A short overview about SWD is presented in /9/. The detailed specifications can be found in /7/ and /8/.

The current version of Fido software supports only single-chip traced devices (i.e. devices having only one debug port). The debug port has one or more access points. The access points are indexed with integers from 0 to 255. Each access point may have one or more related to it processors.

If the user has decided to apply the SWD communication mechanism, Fido performs the following preparatory tasks:

1. Searches for all the existing processors.
2. To each detected processor assigns a unique index (4-byte integer).

To communicate with the traced device (i.e. to upload the user's commands and other data to traced device as well as read responses to commands and other messages from traced device), the user has to select one of the detected processors that will perform those tasks. In other words, this processor will act as the traced device reverse channel. By default the reverse channel processor is the processor that Fido has detected first (see /3/ and /6/).

If no one processor was found, the traced device, most likely does not support SWD and the communication via methods presented below is not possible.

### 3.2.2. Data transfer in SWD

According to /7/, data in SWD is transferred in four-byte words. The order of data bytes in a word is better to demonstrate with an example: the bytes from sequence  $\{0x40, 0x41, 0x42, 0x43\}$  are located as follows:

31.....24	23.....16	15.....8	7.....0
01000011	01000010	01000001	01000000
0x43	0x42	0x41	0x40

In SWD operations bit 0 is the first transferred bit.

### 3.2.3. Fido SWD messages

The SWD communication mechanism implemented in Fido is based on messages organized in the following way:

1. The first byte of a message must be 0xC0 (START MARK).
2. The last byte of a message must be 0xC1 (END MARK)
3. Character 0x7D (PREFIX MARK) means that the following to it character is the original character XOR-ed by 0x20. This opens the way to transfer messages containing in their body characters 0x7D, 0xC0 or 0xC1.

### 3.2.4. Protocol violation and its results

The Fido SWD reverse channel protocol rules are violated when

1. In the current message the PREFIX MARK is directly followed by another PREFIX MARK, START MARK or END MARK
2. The current message has not been ended (i.e. the END MARK is not detected) but a new START MARK appears.
3. The new message has not been started (i.e. the START MARK is not detected) but an END MARK appears.

When a protocol violation is noticed, the current message (i.e. the bytes from START MARK until the violation) is deleted. The normal data transfer is renewed when a message with correct beginning is detected. Bytes between the violation point and the acceptable START MARK are ignored. Bytes arriving before the first START MARK are also ignored.

Remark that a START MARK appearing without the preceding to it END MARK may denote the beginning of a correct message. For example, if the byte sequence is

*0xC0, 0x40, 0x40, 0xC0, 0x41, 0x41, 0xC1*

the first three bytes are deleted but the last four bytes form a regular message.

Sometimes the protocol violation may be applied deliberately. So, as the beginning and the end of a message must be aligned to the four-byte boundary (see 3.2.2), the paddings from several START MARKS (only the last of them denotes the actual start) and END MARKS (only the first of them denotes the actual end) are applicable.

Similarly, if the sending of a message must be broken off, it can be done by sending four-byte words like {0xC0, 0xC1, 0xC0, 0xC1} or {0x7D, 0x7D, 0x7D, 0x7D}. Remark that in word {0xC0, 0xC1, 0xC0, 0xC1} the first 0xC0 violates the protocol and forces the receiver to delete the unfinished message. The following to it 0xC1 also violates the protocol but has no any affect. The two last characters actually form an empty message, which the receiver should discard.

### **3.2.5. Implementation: sending data to traced device**

The user may send 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/.

Fido converts the user's data into a regular SWD message according to rules presented in 3.2.3 and 3.2.4. and transfers it in four-byte words to the traced device (see 3.2.2). There is no any wake-up or some other introductory procedure. The traced device has approximately 10 seconds for reading a single word. If it fails to do so, Fido breaks the sending off and reports about failure.

Fido uses the {0xC0, 0xC1, 0xC0, 0xC1} sequence to signal that a new connection has started. More about this sequence see in 3.2.4.

### **3.2.6. Implementation: reading data from traced device**

Traced device must send messages formatted according 3.2.3. Fido does not use timeouts while receiving SWD messages – traced device does not have to care about pauses between outputting four-byte words.

If the traced device has sent an {0xC0, 0xC1} empty SWD message, Fido discards it.

Fido restores the original traced device message by removing START and END MARK and converting characters prefixed with PREFIX MARK to their original values. After that it wraps received message into an OST (see 3.2.7) or PhoNet (see 3.2.8) message and sends on to the external application or stores

into a user-specified file<sup>15</sup>. Resulting message contains information about the processor that sent the SWD message and, contrast to typical trace message, is not timestamped by Fido.

The total length of resulting OST or Phonet message cannot exceed 0xFFFF bytes. Too long traced device sent messages are simply truncated before wrapping into OST or Phonet.

### 3.2.7. SWD messages wrapped into OST messages

SWD messages wrapped into OST have the following format:

Byte	Name	Contents and remarks
[0 : 1]		As presented in 2.4.1.
2	Protocol ID	Always OST_PROTOCOL_WRAPPER (0x99)
3 or [3 : 7]		As presented in 2.4.1.
4 or 8	Message type	Always TB_WRAPPER_TYPE_SWD (0x01)
5 or 9	Tracebox port <sup>16</sup>	See 2.4.1.
6 or 10	Processor	Index of the processor that sent the SWD message (0x01...0xFF).
[7 : n] or [11 : n]	Data field	Message sent over SWD.

### 3.2.8. SWD messages wrapped into PhoNet messages

SWD messages wrapped into PhoNet have the following format:

Byte	Name	Contents and remarks
[0 : 8]		As presented in 2.4.2.
9	Message ID	Always TB_WRAPPER_MSG (0x99)
10	Message type	Always TB_WRAPPER_TYPE_SWD (0x01)
11	Processor	Index of the processor that sent the SWD message (0x01...0xFF).
[12 : n]	Data field	Message sent over SWD.

<sup>15</sup> Fido does not analyse the received SWD messages. Therefore the traced device may use the SWD connection for transporting any kind of data (including the trace messages).

<sup>16</sup> The parallel term is “sender object”

## 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 Programmer's Reference. The PDF-version is downloadable from <http://www.liewenthal.ee/projects/fido/documents>
4. MIPI Alliance Specification for System Trace Protocol (STP). Draft version 2.00.00 revision 088. 24 Feb 2010.
5. MIPI 60 Pin QTH Trace Cable. Datasheet. The PDF-version is downloadable from <http://www.liewenthal.ee/projects/fido/documents>
6. Fido Online Help. Accessible from the Fido software control panel *Help* menu.
7. ARM Debug Interface v5. Architecture Specification. The PDF-version is downloadable for registered ARM costumers from <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ih0031a/index.html>.
8. ARM Debug Interface v5. Architecture Specification Supplement. The PDF-version is downloadable for registered ARM costumers from <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prdc008772b/index.html>.
9. Serial wire debug. <http://www.arm.com/products/system-ip/debug-trace/coresight-soc-components/serial-wire-debug.php>