

Fido

Target Emulator 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. OUTLINE	4
1.1. What is the Target Emulator	4
1.2. How Target Emulators communicate with Fido	4
1.2.1. TCP/IP connection between Fido and Target Emulator	4
1.2.2. Channels	4
1.2.3. Data formats	4
1.2.4. Communication protocol	5
1.3. How Fido handles Target Emulators	5
1.4. About this document	6
2. PROGRAMMER'S COOKBOOK	7
2.1. About example application	7
2.2. Connecting	7
2.3. Creating and sending data	8
2.4. Receiving data	8
3. REFERENCES	9

CHANGE HISTORY

Version : 1.0
Status : Draft
Date : 28-Jan-2010
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.
Comments : Document was created.

Version : 1.1
Status : Draft
Date : 19-Feb-2010
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.
Comments : Legal notices completed.

Version : 1.2
Status : Draft
Date : 12-Mar-2010
Owner : Viktor Leppikson / Liewenthal Electronics Ltd.
Comments : Legal notices revised.

1. Outline

1.1. What is the Target Emulator

The Target Emulator (in the earlier documentation referred also as Virtual Target or Virtual Traced Device) is a software application or electronic device that is able to send trace data to Fido over TCP connection. Usually, Target Emulators are built to imitate physical traced devices (targets) connected to Fido over specific cables designed for trace data transport.

Fido supports five trace data protocols (STI, C-STI, XTI version 2, ST-XTI and XTI version 3). However, Fido can cooperate only with Target Emulators following the XTI version 3 protocol. In other words, the data sent by Target Emulators must be packed into System Trace Protocol (STP) packets or, using another terminology, the data is sent as a sequence of Parallel Trace Interface (PTI) nibbles (4-bit half-bytes). Detailed specifications concerning XTI version 3 protocol, STP and PTI are in /1/ and /4/.

1.2. How Target Emulators communicate with Fido

1.2.1. TCP/IP connection between Fido and Target Emulator

To start with, the Target Emulator must establish TCP/IP connection with Fido. Any Fido software (*Fido.exe*) instance running on a PC listens for incoming TCP/IP connection attempts on a specific IP port. The factory default for this port is 7654, but the user can apply a command line parameter to replace it by another one. About Fido Ethernet connections read /2/.

1.2.2. Channels

A Target Emulator may be built in two ways:

1. A simple Target Emulator that is only able to send data to Fido. According to the terminology used in this document we tell that such an Emulator supports only the trace data channel.
2. A full scale Target Emulator that can also receive data (for example, control commands) from Fido. Full scale Target Emulators support two channels: the trace data channel and the return channel¹.

1.2.3. Data formats

There are five different data formats: three for the trace data channel and two for the return channel. Target Emulator developers are free to decide which and how many formats to implement.

The possible data formats for the trace data channel are:

1. Binary: the emulator must put each PTI nibble into a single byte onto bits [3 : 0]. Bits [7 : 4] are ignored.
2. Packed binary: the emulator must put two PTI nibbles into one byte. Nibble on bits [7 : 4] is generated earlier than nibble on bits [3 : 0].
3. Text: nibbles are sent as hexadecimal digits coded by ASCII. Example: if we have nibbles 1010 and 1111, the Emulator sends characters 'A' and 'F' (0x41 and 0x46).

¹ We are speaking about channels because we need some analogy with cables having four wires for nibbles (corresponding to trace data channel) and one wire for sending data to the traced device (corresponding to return channel).

The possible data formats for the return channel are²:

1. Binary: data from Fido to the Emulator is sent as a sequence of bytes. Fido encodes source data bytes 0xFC, 0xFD, 0xFE and 0xFF, prefixing them by 0xFC and XOR-ing by 0x80. For example, if the source data contains byte 0xFD, the Emulator gets two bytes: 0xFC and 0x7D. Byte 0xFF in Fido output is the BREAK signal indicator. 0xFD and 0xFE are reserved for future developments.
2. Text: a source data byte is sent from Fido to Emulator as a pair of ASCII-coded hexadecimal digits. Pairs are separated by sequences of spaces. The length of sequence may have any value except 0. For example, if the source data consists of bytes 10101111 and 01010001 and the separating sequences are consisting of only one space, Fido sends to Emulator the “AF 51 “ six-byte string (0x41, 0x46, 0x20, 0x35, 0x31, 0x20). Character ‘!’ (0x21) marks the BREAK signal³.

1.2.4. Communication protocol

Each software application or electronic device must right after establishing the TCP/IP connection introduce itself. This is performed by so-called extended HTTP command (more about Fido HTTP commands read in /2/).

The extended HTTP command for Target Emulators must be formatted as follows:

1. PTI xtiv3:<trace data channel format> TRACEBOX/<version><CR><LF><CR><LF>
in case of simple emulator (see 1.2.2).

2. PTI xtiv3:<trace data channel format>/<return channel format>
TRACEBOX/<version><CR><LF><CR><LF>

in case of full-scale emulator.

The format names are:

- binary
- binary.packed
- text

Example strings:

```
"PTI xtiv3:binary TRACEBOX /1.1.1.1\r\n\r\n"
"PTI xtiv3:binary/text TRACEBOX /1.1.1.1\r\n\r\n"
"PTI xtiv3:binary.packed/binary TRACEBOX /1.1.1.1\r\n\r\n"
```

The data format(s) stay valid until the end of current communication session. It is not possible to change it (them) meanwhile.

With sending this extended command all the preparations have been completed. There is no any response from Fido.

1.3. How Fido handles Target Emulators

A traced device is connected to Fido box with the special cable. The Fido box, in turn, is connected to computer via USB. To cooperate with a Target Emulator, Fido creates a virtual box. This box is considered to be connected to the Target Emulator and handled as any real box. About details see /5/.

² Remark that binary and text protocols specified for return channel are very different from the binary and text protocols specified for the trace data channel.

³ It is better to write the Emulator software so that any string consisting of characters other than hexadecimal digits ‘0’...’1’, ‘A’...’F’ (or ‘a’...’f’) and ‘!’ is interpreted as a separator.

1.4. About this document

The aim of this document is to explain how to write Target Emulator software. The discussion bases on an example program (see /3/).

2. Programmer's Cookbook

2.1. About example application

The example application (see /3/) is written in C. The code consists of one .h and five .c files. As the differences between Windows and Linux environments are taken into consideration, you should be able to compile and link this code under Windows as well as under Linux without any modifications.

The example application may send trace data to Fido or receive data from Fido, but not the both during one run. In other words, it may run in the send mode or in the receive mode. The running mode is specified by command line parameters (see the `help()` function from the `Main.c`). In both modes the trace data channel uses the binary protocol (see 1.2). The return channel uses the text communication protocol.

In the send mode the example application sends to Fido the so-called ASCII trace messages. Those messages start with identifier (its value is 0x20) followed by regular C string (sequence of ASCII-coded characters terminated by 0x00). All the messages contain the same text ("Hi, Trace Message from Target Emulator!"). You may use the command line parameters to specify the number of trace messages. If you omit it, the messages are sent in an infinite loop.

In the receive mode the example application accepts any data. The easiest way to test the receive mode is to create a short binary file, and using the Fido Control Panel (see /5/), send it to the example application. You may also use some external application capable to cooperate with Fido. The received data in hexadecimal format is shown on the screen.

It is important to remember that Fido can perform tracing only when the user has specified how to handle the output trace data. There are three options (see also /2/ and /5/):

1. Using the Fido Control Panel, you may select and open a file for storing the trace data.
2. An external application calls Fido and opens a particular TCP/IP connection called as output trace data channel. In that case Fido will pump its output to this application.
3. Using a browser, you may download the Fido output into a file.

Specifying the output trace data handling mode we activate the tracing. Until the tracing is not activated, Fido can neither receive data from traced devices nor send data to them. Naturally, this concept is valid for the Target Emulators as well.

2.2. Connecting

The steps the Target Emulator has to perform are:

1. Create the socket.
2. Establish the TCP/IP connection with *Fido.exe*.
3. Send the extended HTTP command (see 1.2.4).

Creating a socket and establishing TCP/IP connections are routine operations. In the example application those steps are implemented in function `open_connection()` from file `Connect.c`.

The connection stays on until:

- The Target Emulator itself closes the socket (see function `close_connection()` from file `Connect.c`). It can create another socket and establish new connection later.
- *Fido.exe* exits. While *Fido.exe* runs normally, it never closes the connection with Target Emulator.

Fido.exe can concurrently serve several Target Emulators⁴. Along with emulators *Fido.exe* can also serve physical traced devices.

2.3. Creating and sending data

To send trace data, the Target Emulator has to do the following:

1. Convert the data into a sequence of STP packets
2. Send those packets to Fido.

Trace data transport with STP packets is discussed in /1/ and /4/. Let us emphasize that until the tracing is not activated (see 2.1), Fido refuses to receive data⁵ from Target Emulator. To avoid loss of data, it is advisable to start the Target Emulator only when Fido is ready to operate.

An STP packet may transport 8, 16, 32 or 64 data bits. The STP specification does not set down how to divide the trace data between packets. If we need to minimize the number of packets, the best way is to use as much 64-bit packets as possible. In that case the shorter packets are used only at the end of packet sequence.

In the example code the functions used for sending are in files `SendMain.c` and `SendBasic.c`. Function `send_main()` from `SendMain.c` controls the sending. Functions from `SendBasic.c` are designed to be re-usable in other Target Emulators. To employ them you only need to store your connection parameters into the `struct fido_vt_t` (see `TargetEmulator.h`).

2.4. Receiving data

Uploading data from Fido to Target Emulator follows the protocol used for uploading to physical traced devices /1/. Shortly, the steps of communication are as follows:

1. As the target may sleep, Fido sends the *BREAK* signal. In case of physical traced devices, Fido uses its UART interface. In case of Target Emulators, the character(s) replacing the *BREAK* signal are sent via TCP connection (see also 1.2).
2. The traced device answers with the *Set bit rate* command. The command as well as the other ones (their specifications are in /1/) are sent as any other trace message, i.e. it is converted into a sequence of STP packets. For Target Emulators the value declared as bit rate is fictional, because the UART is not used. To avoid possible complications⁶, use some standard bit rate like 115200.
3. Fido sends the number of bytes it has to upload. This number is a four-byte little-endian integer.
4. The target answers with the *Data length acknowledge* command.
5. Fido sends the data bytes.
6. To complete the uploading, the target answers with the *Data acknowledge* command.

In the example code the functions used for receiving are in file `ReceiveMain.c`. STP packets are sent with functions from `SendBasic.c`.

⁴ Currently, the maximal number of ports allowed for Fido is 8.

⁵ Fido has several options to control the data flow. In addition to opening and closing the output data channel you may also open, close, connect and disconnect Fido ports (see /5/).

⁶ If the bit rate is unreal (not supported with real Fido UART interface), the uploading breaks off.

3. References

1. Fido Data Processing Reference. The PDF-version is downloadable from <http://www.liewenthal.ee/projects/fido/documents>
2. Fido Programmer's Reference. The PDF-version is downloadable from <http://www.liewenthal.ee/projects/fido/documents>
3. Fido Virtual Target Example Program. .c and .h source files are downloadable from <http://www.liewenthal.ee/projects/fido/download/>
4. MIPI Alliance Standard for Test & Debug – Parallel Trace Interface. Draft Version 0.9. 5. June 2006.
5. Fido Help. Accessible from the Fido Control Panel *Help* menu.