USB2XServer

A Telnet Server that makes the USB2X easier to use in own applications

Manual

Version: 1.00
September 2nd, 2008

Trinamic Motion Control GmbH & Co KG
Sternstraße 67
D - 20 357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60

http://www.trinamic.com

Table of Contents

1 Life support policy

TRINAMIC Motion Control GmbH & Co. KG does not authorize or warrant any of its products for use in life support systems, without the specific written consent of TRINAMIC Motion Control GmbH & Co. KG.

Life support systems are equipment intended to support or sustain life, and whose failure to perform, when properly used in accordance with instructions provided, can be reasonably expected to result in personal injury or death.

Information given in this data sheet is believed to be accurate and reliable. However no responsibility is assumed for the consequences of its use or for any infringement of patents or other rights of third parties, which may result form its use.

Specifications are subject to change without notice.

2 Overview

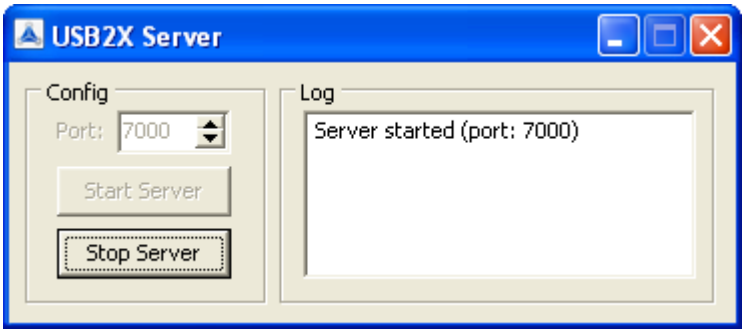
USB2XServer is a Windows application that implements a Telnet server interface to the Trinamic USB2X devices. It provides easy to use ASCII commands for all features of the USB2X device. Thus it makes it easy to access all interface and features of the USB2X device, regardless of the programming language that is to be used (as with every modern programming language like C++, Delphi, C# or Visual Basic and also script languages like Tcl/Tk it is often easier to open a Telnet connection than to use a DLL). It is also possible to use it either from a native Windows programme or from a .NET programme. It is even possible to use it from a different PC over a network.

3 Install USB2Xserver

There is no special installation needed. Just copy it anywhere on the hard disk of your PC. It may also be put into the installation routine of other software that makes use of it.

4 Starting USB2Xserver

The USB2X server can be started just by double clicking on it. Its main window then appears and the server opens the pre-configured port (the default port is port 7000) and is ready to accept connections.



The USB2X server can of course also be started form other applications just by using the CreateProcess() function of Windows. In this case, the following command line options can also be used:

- `-m`: start up with the main window minimized (don't show the main window)
- `-p<number>`: start up using a different port number (where <number> is the port number)

If the number of the port used by USB2XServer should be needed to be changed manually, first click the "Stop Server" button. Then the port number can be changed. After you have done that, click the "Start Server" button again to re-enable the server with the new port number.

5 Testing USB2Xserver

For a first test, the Telnet client that is included in Windows can be used. To do this, first start the USB2Xserver manually. Then, open a command line and type "`telnet 127.0.0.1 7000`". This means that a Telnet connection to port 7000 on the local machine is to be opened. You will see the version information of the USB2Xserver and then you are ready to explore the USB2Xsever commands. This should look like this:

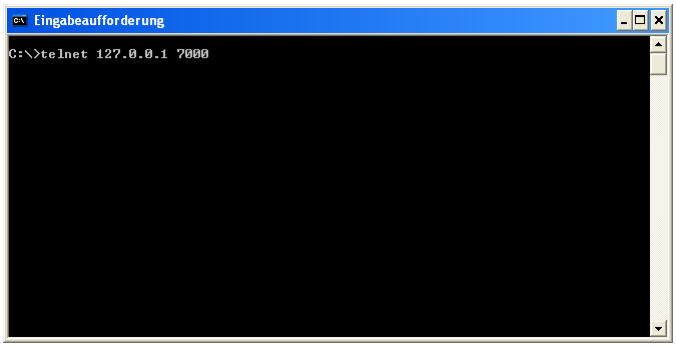


Figure 5-1: Starting the Telnet client

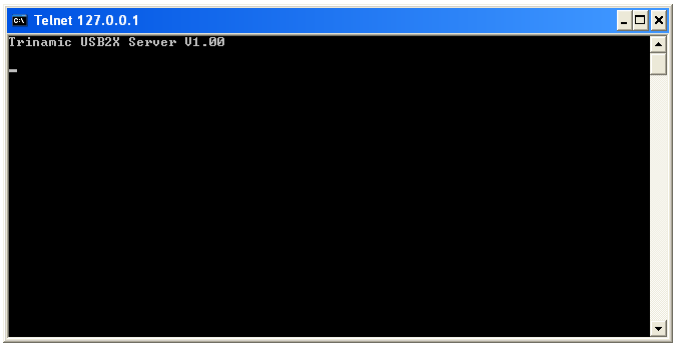


Figure 5-2: Telnet connection successfully opened

If one USB2X device is connected to the PC you can type „open", and the USB2server will open a connection to the connected USB2X device. For a further test, type "info" for information about the firmware revision of the USB2X device. The command "?" gives an overview of all available commands. Please note that every command must be terminated by the carriage return or enter key.

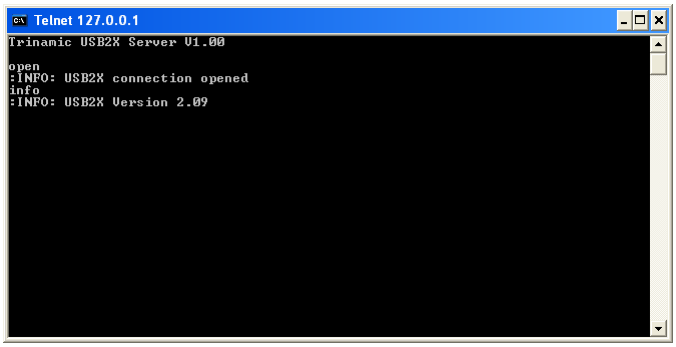


Figure 5-3: First commands

6 Commands

The following chapters describe all the commands provided by the USB2Xsever in detail. It is important that all commands are case-sensitive.

6.1 USB2X connection commands

These commands are used for choosing the USB2X device that is to be used and for connecting to it or disconnecting from it. When only one USB2X device is connected to the PC it is sufficient just to use the "open" command without any parameters.

6.1.1 The "?" command

This command prints a help screen with an overview of all available commands.

6.1.2 The "list" command

The "list" command prints out the identification string of all connected USB2X devices. It is normally not needed when only one USB2X device is connected to the PC.

6.1.3 The "open" command

This command is used to open the connection between the PC and the USB2X device. It expects the identification string of a USB2X command as parameter. When the parameter is omitted a connection to the first USB2X device found will be opened.
So in most cases, when only one USB2X is connected to the PC it is sufficient to use the "open" command without any parameter. An "open" command must be issued before any other operation with the USB2X device can be carried out.

6.1.4 The "close" command

This command closes the connection between the PC and the USB2X device. No further operation with the USB2X device can be carried out after this command has been issued.

6.1.5 The "info" command

This command returns the firmware revision number of the connected USB2X device.

6.1.6 The "logout" command

The "logout" command terminates the Telnet connection, but does not stop the USB2Xserver.

6.1.7 The "Quit" command

The "Quit" command closes all Telnet connections to the USB2Xserver and then stops and terminates the USB2Xserver. Use this command with care as other application logged in to this instance of USB2Xserver will then also lose their connections to the USB2Xserver.
Please watch the spelling of this command: it starts with a capital letter.

6.2 CAN commands

6.2.1 Command "ci" - Initialize the CAN interface

Syntax: `ci <bitrate>` (<bitrate> = 10, 20, 50, 100, 125, 250, 500, 800 or 1000)
`ci off` (switch off the CAN interface)
Example: `ci 500` (set the CAN bit rate to 500kBit/s)

This command initializes the CAN interface. Mainly it sets up the CAN bit rate that is to be used. The bit rate is specified directly in kBit/s.

6.2.2 Command "cw" – Send a CAN frame with 11 bit identifier

Syntax:

`cw [r] <id> <data>` (<id>=0..2047, <data>=up to eight data bytes)

Examples:

`cw 0x720 0x01 0x02 0x03` (send a CAN frame with ID 0x720 and three data bytes

`cw r $100 0 0 0 0` (send a CAN RTR frame with ID 0x100 and DLC field set to 4)

This command sends a CAN frame with the specified 11 bit ID and the specified data bytes. The data can be up to eight data bytes, separated by spaces. Hexadecimal numbers must be preceeded with a 0x or a $ sign. If the parameter [r] is specified, an RTR frame will be sent. The DLC of the RTR frame depends on the number of (dummy) data bytes given in the command.

6.2.3 Command "cW" – Send a CAN frame with 29 bit identifier

Syntax:

`cW [r] <id> <data>` (<id>=0..2047, <data>=up to eight data bytes)

Examples:

`cW 0xff720 0x01 0x02 0x03` (send a CAN frame with ID 0xff720 and three data bytes

`cW r $100 0 0 0 0` (send a CAN RTR frame with ID 0x100 and DLC field set to 4)

This command sends a CAN frame with the specified 29 bit ID and the specified data bytes. The data can be up to eight data bytes, separated by spaces. Hexadecimal numbers must be preceeded with a 0x or a $ sign. If the parameter [r] is specified, an RTR frame will be sent. The DLC of the RTR frame depends on the number of (dummy) data bytes given in the command.

6.2.4 Command "cr" – Read a CAN frame

Syntax:

`cr`

This command tries to read a CAN frame from the input buffer of the USB2X device. If a CAN frame could be read it will be printed using the following format:

`:CAN: <id> : <frametype><datatype> : <data>`

Where <id> is the CAN-ID of the frame, printed as a hexadecimal number preceeded by "0x", <frametype> can be "s" for standard format (11 bit ID) or "x" for extended format (29 bit ID), <datatype> can be "D" for normal data frames or "R" for RTR frames, and <data> can be up to eight data bytes, printed as hexadecimal numbers , preceeded by "0x" and separated by spaces.

Examples:

`:CAN: 0x720 : sD 0x80 0x00 0x22`

If there is no CAN frame available, the following message will be printed:

`:CAN: no message`

6.3 IIC commands

6.3.1 Command "ii" – Initialize the IIC interface

Syntax:

`ii <bitrate>` (<bitrate> = 11, 23, 35, 46, 58, 93, 140, 187, 234 or 375)

Example:

`ii 375` (initialize the IIC interface with 375kBit/s)

The "ii" command initializes the IIC interface and sets the IIC bit rate. The bit rate is given directly in kBit/s (one of the values given above).

6.3.2 Command "iw" – Write to an IIC device

Syntax:

`iw <address> <data>`

Example:

`iw 200 1 2 3 4 5 6 7 8`

The "iw" command writes data to an IIC device connected to the IIC bus. The <address> parameter must be an even number between 0 and 254 (where 0 writes to all devices connected to the bus), and <data> can be up to 254 bytes of data. All data bytes must be separated by spaces. Hexadecimal numbers must be preceeded either by a $ sign or by 0x.

6.3.3 Command "ir" – Read from an IIC device

Syntax:

`ir <address> <length>`

Example:

`ir 201`

This command reads data from a device connected to the IIC bus. The <address> parameter must be an odd number between 1 and 255, and <length> is the number of bytes to read from the device. This can be any value between 1 and 128.
If data could be read successfully it will be printed using the following format:
`:IIC: <data>`
where <data> are the data bytes in hexadecimal form, preceeded by 0x and separated by spaces.
Example:
`:IIC: 0x21 0x42 0x78`
If no data could be read, an error message is given:
`:IIC: no data`

6.4 LIN commands

6.4.1 Command "li" – Initialize the LIN interface

Syntax:

`li <bitrate>` (<bitrate> = 2400, 9600 or 19200)

Example:

`li 19200`

This command initializes the LIN interface and sets the LIN bit rate to the given value.

6.4.2 Command "lw" – Write to the LIN bus

Syntax:

`lw <address> <data>` (<address>=0..255, <data>=two, four or eight data bytes)

Example:

`lw 0x1f 1 2 3 4` (send four data bytes to LIN device 0x1f)

This command writes data to a device connected to the LIN bus. The <address> parameter must be a value between 0 and 255 (the LIN ID), and <data> must be either two, four or eight data bytes (separated by spaces).

6.4.3 Command "lr" – Read from a LIN device

Syntax:

`lr <address> <length>` (<address>=0..255, <length>=2, 4 or 8)

Example:

`lr 0x55 2` (request 2 bytes from device 0x55)

The "lr" command tries to read data from a LIN device. If data could be read it will be printed using the following format:

`:LIN: <data>`

where <data> will be two, four or eight data bytes in hexadecimal format, preceeded by 0x and separated by spaces.

6.5 SPI commands

6.5.1 Command "si" – Initialize the SPI

Syntax:

`si <p1> <p2> <p3>` (<p1>=0..255, <p2>=0..255, <p3>=0..255)

This command initializes the SPI. Please see the following two tables for a description of the possible values for the three parameter bytes.

Bit	7	6	5	4	3	2	1	0
<p1>	0	Base Clock[2:0]			Clock Polarity	Clock Phase	Clock Divisor [1:0]	
<p2>	0	Slave Select Delay [2:0]			Inter-Byte Delay [3:0]			
<p3>	SPI Enable	0	0	0	0	Push Slave Data	Enable Slave Request	3V3

Parameter Name	Parameter Coding
Base Clock [2:0]	0: CPU clock frequency = 1.5 MHz 1: CPU clock frequency = 3.0 MHz 2: CPU clock frequency = 4.5 MHz (only allowed for 5V operation) 3: CPU clock frequency = 6.0 MHz (only allowed for 5V operation) 4: CPU clock frequency = 7.5 MHz (only allowed for 5V operation) **SPI clock frequency = CPU clock frequency / SPI clock divider**
Clock Polarity	0: SPI clock line idle level = low 1: SPI clock line idle level = high
Clock Phase	0: 1st clock edge latches bit 7, 2nd clock edge shifts out bit 6 onto the bus 1: 1st clock edge shifts out bit 7 onto the bus, 2nd clock edge latches bit 6
Clock Divisor [1:0]	0: SPI clock divider = 2 1: SPI clock divider = 8 2: SPI clock divider = 32 3: SPI clock divider = 128 **SPI clock frequency = CPU clock frequency / SPI clock divider**
Slave Select Delay [2:0]	Minimum delay after slave select going low until 1st byte / after last byte until slave select going high: **min_ss_delay [CPU clock cycles] = 6 + 3 * 2^Slave Select Delay** 0: min_ss_delay = 9 cycles 1: min_ss_delay = 12 cycles 2: min_ss_delay = 18 cycles 3: min_ss_delay = 30 cycles 4: min_ss_delay = 54 cycles 5: min_ss_delay = 102 cycles 6: min_ss_delay = 198 cycles 7: min_ss_delay = 390 cycles **Note**: Due to SPI hardware constraints the actual delay act_ss_delay varies: **min_ss_delay ≤ act_ss_delay ≤ min_ss_delay + SPI bit time** with **SPI bit time [CPU clock cycles] = SPI clock divider**
Inter-Byte Delay [3:0]	Delay between any two bytes of an SPI transfer: **byte_delay [CPU clock cycles] = 8 * 2^Inter-Byte Delay** for 1 ≤ Inter-Byte Delay ≤ 8 (0 means no delay) 0: byte_delay = 0 cycles (back-to-back byte transfer without delay) 1: byte_delay = 16 cycles (only for SPI clock divider 2 or 8) 2: byte_delay = 32 cycles (only for SPI clock divider 2, 8 or 32) 3: byte_delay = 64 cycles (only for SPI clock divider 2, 8 or 32) 4: byte_delay = 128 cycles 5: byte_delay = 256 cycles 6: byte_delay = 512 cycles 7: byte_delay = 1024 cycles 8: byte_delay = 2048 cycles **Note**: The delay after the first byte of a Slave Request transfer is up to 15 CPU clock cycles plus one SPI bit time longer than the delay between the following bytes. This is because the first byte of a Slave Request transfer contains the number of the following bytes and must be processed before the following bytes can be transferred.
SPI Enable	0: SPI hardware disabled 1: SPI hardware enabled
Push Slave Data	Only evaluated if Enable Slave Request = 1 0: Data received in reaction to SPI slave request must be read using SPI-Read command 1: Data received in reaction to SPI slave request are forwarded to the USB host automatically
Enable Slave Request	0: SPI slave request line is ignored, SPI slave can not request a transfer 1: SPI slave can request USB2X to start a transfer by a falling edge on the SPI slave select line. The first byte received from the slave is interpreted as remaining byte count of this transfer.
3V3	0: logic high level of SPI bus lines is 5V 1: logic high level of SPI bus lines is 3.3V (only allowed for CPU clock frequency = 1.5 MHz or 3.0 MHz)

Table 1 SPI-Init Parameters

Please also note the following things when using SPI:

- IIC cannot be used simultaneously with SPI. To re-enable IIC after SPI has been used, disable the SPI interface by executing an "si" command with the "SPI Enable" bit (Bit 7 of data byte 3) set to zero. This will disable SPI (all SPI pins will be tri-stated then) and re-initialise IIC. After that, the IIC bit rate must also be set up again using the "ii" command.

The bit rates of all the other interfaces are calculated using a base clock value of 7.5MHz. So, when setting the base clock frequency to a value different than 7.5MHz, all bit rates of the other interfaces (IIC, LIN, RS485, CAN) will be wrong. So, set the clock frequency back to 7.5MHz when disabling SPI to get all other

bit rates right or use a base clock frequency of 7.5MHz when planning to use SPI simultaneously with LIN, RS485 or CAN.

6.5.2 Command "sw" – Write to SPI

Syntax:

```
sw <data>
```

Example:

```
sw 0 1 2 3
```
(write 4 data bytes to RS485)

The "sw" command writes data to the SPI. The data bytes must be separated by spaces. Hexadecimal numbers must be preceeded with a $ sign or 0x. Up to 255 data bytes can be written. The data read back from the SPI will be printed using the following format:

```
:SPI: <data>
```

with the data bytes given in hexadecimal format, preceeded by 0x and separated by spaces.

6.5.3 Command "sr" – Read from SPI

Syntax:

```
sr
```

The "sr" command reads the last SPI data out of the buffer of the USB2X device, if there is data to read. The data will be printed in a similar format as with the "sw" command. If there is no data to read, the error message `:SPI: no data` will be printed.

6.6 RS485 commands

6.6.1 Command "ri" – Initialize the RS485 interface

Syntax:

```
ri <bitrate>
```
(<bitrate>=2400, 9600 or 19200)

Example:

```
ri 9600
```
(sets the RS485 bitrate to 9600bps)

This command sets the bit rate of the RS485 interface. The bit rate can be 2400, 9600 or 19200.

6.6.2 Command "rw" – Write to RS485 interface

Syntax:

```
rw <data>
```

Example:

```
Rw 1 17 0x42
```
(send three data bytes through RS485)

This command sends data through the RS485 interface. Up to 255 data bytes can be given. The data bytes must be separated by spaces. Hexadecimal numbers must be preceeded by a $ sign or 0x.

6.6.3 Command "rr" – Read from the RS485 interface

Syntax:

```
rr
```

The "rr" command reads data from the RS485 receive buffer of the USB2X device. Data that has been read will be printed using the following format:
:RS485: <data>
where <data> can be up to 127 data bytes, separated by spaces. Hexadecimal numbers must be preceeded by a $ sign or 0x. If the receive buffer is empty the error message :RS485: no data will be printed.

6.7 TMCL commands

The USB2XServer can also handle the communication with Trinamic Motion Control Modules supporting TMCL. It translates TMCL commands given in text form (like in the TMCL-IDE, please see [TMCL] for details) into their binary representation and sends them out via CAN, IIC or RS485. The result of a TMCL command is then also printed in text form.

6.7.1 Command "tc" – TMCL communication via CAN

Syntax:

```
tc <id> <command>
```

Example:

```
tc 1 MVP ABS, 0, 1000
```

This command sends a TMCL command to a module via the CAN interface. Before using this command for the first time the CAN interface must be initialized using the "ci" command (please see chapter 6.2.1 for details).
The "tc" command expects the CAN ID of the module as the first parameter and a TMCL command in ASCII form as the second parameter (in the above example, the command "MVP ABS, 0, 1000" will be sent to the module with ID 1).
The return value will be displayed in the following format:
:TMCL: <ReplyId> <ModuleId> <Status> <Value> where <ReplyId> is the CAN ID of the reply message, <ModuleId> is the ID of the module, <Status> is the TMCL status and <Value> is the TMCL return value.
In the above example, this could be
:TMCL: 2 1 100 1000

6.7.2 Command "ti" – TMCL communication via IIC

Syntax:

```
ti <id> <command>
```

Example:

```
ti 2 MVP ABS, 0, 1000
```

This command sends a TMCL command to a module via the IIC interface. Before using this command for the first time the IIC interface must be initialized using the "ii" command (please see chapter 6.3.1 for details).
The "ti" command expects the IIC write ID of the module (this must be an even number between 0 and 254) as the first parameter and a TMCL command in ASCII form as the second parameter (in the above example, the command "MVP ABS, 0, 1000" will be sent to the module with ID 2).
The return value will be displayed in the following format:
:TMCL: x <ModuleId> <Status> <Value>, where <ModuleId> is the ID of the module, <Status> is the TMCL status and <Value> is the TMCL return value. As there is no reply ID with TMCL IIC communication, an x is displayed in place of the reply ID.

In the above example, this could be
:TMCL: x 1 100 1000

6.7.3 Command "tr" – TMCL communication via RS485

Syntax:

```
tr <id> <command>
```

Example:

```
tr 1 MVP ABS, 0, 1000
```

This command sends a TMCL command to a module via the RS485 interface. Before using this command for the first time the RS485 interface must be initialized using the "ri" command (please see chapter 6.6.1 for details).
The "ri" command expects the RS485 ID of the module as the first parameter and a TMCL command in ASCII form as the second parameter (in the above example, the command "MVP ABS, 0, 1000" will be sent to the module with ID 1).
The return value will be displayed in the following format:
:TMCL: <ReplyId> <ModuleId> <Status> <Value> where <ReplyId> is the CAN ID of the reply message, <ModuleId> is the ID of the module, <Status> is the TMCL status and <Value> is the TMCL return value.
In the above example, this could be
:TMCL: 2 1 100 1000

7 Revision History

7.1 Document Revision

Version	Comment	Author	Description
1.00	2-Sep-08	OK	Initial version

Table 6.1: Document Revision

7.2 Software Revision

Version	Comment	Description
1.01	First release	

Table 6.2: Software Revision

8 References

[USB2X] USB2X manual
[TMCL] TMCL reference and programming manual