Technical Information
for 'programmable CAN-terminals without CANopen'
Version 1.0

Protocol for downloading applications via CAN-bus into programmable terminals by MKT Systemtechnik

to be used in CAN networks **without** CANopen !

Dokument-Nr: 85116
Original: C:\cbproj\UptWin1\DOKU\art85116_CanTerminalDownloadProtocol.odt
Autor: Wolfgang Büscher (Software-Entwicklung)

Table of Contents

1. Revisions of this document

Revision number	Date	Author	Notes, remarks
V1.0	2007-04-23	W. Büscher	Created document nr 85116 with OpenOffice
	2014-06-18	W. Büscher	removed 'Preliminary' on the front cover page

2. Introduction

This document describes the protocol used to download an „application“ from MKT's programming tool into the programmable terminal via CAN-Bus. It only applies to terminals *WITHOUT* CANopen protocol (because CANopen uses a totally different protocol to transfer strings via SDO).

Purpose

This document was written by a programmer for other programmers, to simplify the task of implementing your own download host. Basic programming skills, and knowledge of CAN networking are assumed.
The document is not targeted for the general user of the programmable terminals.

Disclaimer

The software mentioned in this document, and all accompanying documents, are distributed „as is“ without any warranties, either expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the software, the accompanying written materials, and any accompanying hardware.

Names of products, which are registered trademarks, are not explicitly marked as such in this document. Therefore, a missing❋,✳, or ❋ character doesn't mean that a name is not a registered trademark or similar.

CANdb is registered trademark of Vector Informatik GmbH
NTCAN API is copyright (c) by ESD electronic system design GmbH
PCAN Dongle and the PCAN API is copyright (c) by PEAK-Service GmbH
Microsoft, Windows, Win95, WinNT, WinXP are registered trademarks of Microsoft Corporation

3. Terms and abbreviations

Application
: Here: The „program“ for the user-programmable terminal which was typically created with MKT's programming tool. This „application“ is usually saved in a *.cvt file.

Host
: Here: The „tool“ which sends the data into the programmable terminal. This is usually MKT's programming tool, but it may be something else too (for example, the application which you want to write yourself)

Target
: Usually a programmable terminal „without CANopen“ by MKT Systemtechnik .

4. Principle

The host sends the application into the terminal on a line-by-line basis. Every line consists of a simple string of „printable“[1] ASCII characters. How text strings are sent from host to target (and vice versa) is explained in a later chapter. But the principle („using text strings“) remains the same, regardless of the protocol actually used.

To set up the communication, some special commands must be sent to establish the connection, and to erase the FLASH memory in the target. The next chapter explains a typical conversation between the host and the target. We'll discuss the details of how to send strings via CAN in a later chapter.

Note: The very same protocol (on the „text string“ layer) is used via RS-232 too. The only difference is, we use the 3964R protocol to send the strings via RS-232, and the protocol explained in another chapter to send the same strings via CAN. In future, we may introduce other protocols to send the same strings (and commands) via IRDA, Bluetooth, ZigBee, Ethernet, TCP/IP, or whatever ;-)

[1] „printable“ character in this context means, only character codes 32 (decimal) and above will appear in the text line. The text line will not contain any ASCII „control“ character like carriage return, new line, NULL, or similar.

4.1 Opening the connection between host and target

To establish the connection, a few special commands must be sent from the host to the target. All these commands begin with an exclamation mark to tell them from „normal“ data lines.

The following lines were copied from the „Error and Message“ window of the programming tool, with the option „Show messages from higher CAN protocols“ set. Green lines are sent from the programming tool (host) to the terminal (target), and blue lines are the „answers“ from the target back to the host.

The command „!comp_date!“ queries the compilation date of the target firmware, and also switches the terminal into transfer mode. Note that the first two strings are sent „slowly“ (without handshake frames), because the target must not send anything as long as it is not in the „connected“ state. The other commands (like „!hardware_name!“ to poll the target's hardware name, etc) are optional and can be ommited to simplify things.

```
!comp_date!
!comp_date!=Apr 12 2007
!hardware_name!
!hardware_name!=A7_TFT_L
!software_name!
!software_name!=CVTv1
!software_art_nr!
!software_art_nr!=11092
!software_step!
!software_step!=10
!receive_program!
!ready!
```

The command „!receive_program!“ is very important, because the target needs to erase the FLASH memory device before it can receive the application („program“). Depending on the hardware, it may take a few seconds until the FLASH memory is erased. The answer („!ready!“) will be sent when the erase processed is finished.

```
!receive_program!
!ready!
```

At this point, the target is ready to receive the application. All following lines can be sent more or less „directly“ from the *.CVT file (which contains the application) :

```
; File:  C:\cbproj\UptWin1\programs\arm7_special\Arm7JoystickTest.CVT
; Saved: 2007-04-23 10:13:26
; Type:  program file for user programmable terminal
; Producer: programming tool compiled Apr 20 2007
Encryption=0
(...)
```

4.2 Sending the application

The application is sent line-by-line from the host to the target, using the same format as used to store the application in a *.CVT (or sometimes *.UPT-) file. The line separators (carriage return and new line) are stripped from the text file, because the CAN protocol will automatically add 0x0D (CR) and 0x0A (NL) to mark the end of a text line.

4.3 Closing the connection between host and target

After downloading the application into the target, an additional command must be sent to let the target „finish“ the transfer. This „finishing“ will flush some remaining data from RAM to FLASH, etc. The target will acknowledge this command when finished :

```
!finish_transfer!
!transfer_finished!
```

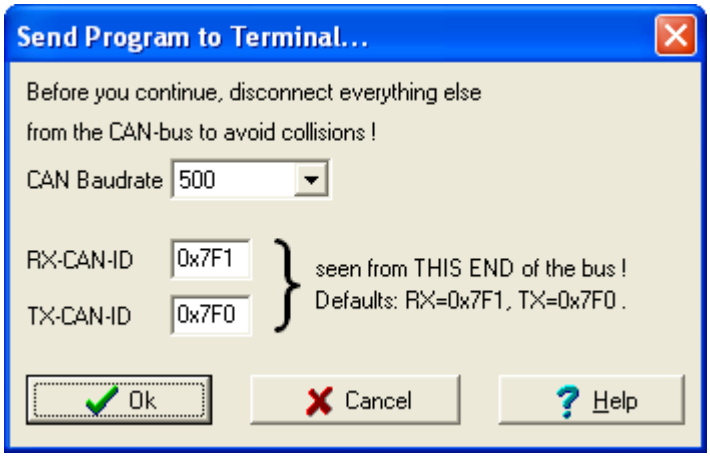
Additionally, you may want to „restart“ the target after this, so the terminal reboots with the new application. In contrast to all other commands mentioned before, the „!run!“ command will not be acknowledged by the terminal :

```
!run!
```

After this, you should see the new application in the terminal's display.

5. Protocol to send strings via CAN („without CANopen“)

The protocol used to send text strings from host to target is quite simplistic (at least, in this case). Only two CAN identifiers are used. You can see these CAN-Identifiers in MKT's programming tool when starting the transfer, in a message box like this one:

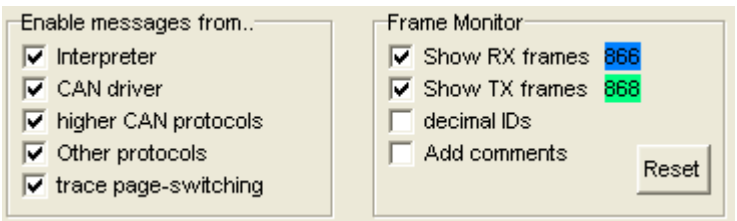


The „TX-CAN-ID“ is used for all CAN messages sent from the host to the target. In MKT's programming tool, such CAN messages can be displayed in GREEN colour (as in the examples below, where we'll look at the protocol at the lower level).
The „RX-CAN-ID“ is used for all CAN messages sent back from the target to the host. These messages are displayed in BLUE colour (at least, in MKT's programming tool).
Black text (below) are comments and remarks added by the author of this document.

The programming tool can display single CAN messages in the following format:

<timestamp>: <CAN-ID> <data bytes> ,
where timestamp is an integer number in milliseconds, CAN-ID is a 3-digit hex number (with 0x as hex prefix), and the up to 8 CAN data bytes are displayed as two digit hex numbers.

To see the net data (in the upper protocol layer), and individual CAN messages (in the lower protocol layer), configure MKT's programming tool as follows, before starting the transfer. You will find these settings in the tabsheet named „Errors“ (which can actually display a lot more than just „Errors“):



We start with sending the command „!comp_date!“, which -as explained in the previous chapter- establishes the connection between host and target. Because at this step, the connection is not established yet, the terminal doesn't send handshake frames in the first step. So the host uses a large gap between two unacknowledged CAN messages (to make sure the target doesn't miss a frame). After reception of the „!comp_date!“ command, the terminal begins sending response frames, which will make the transfer run much faster (explaination further below).

```
Created program-transfer-THREAD . <--- this is a message from MKT's tool
02515379: 0x7F0 21 63 6F 6D 70 5F 64 61  ; <- from HOST to TARGET
02517329: 0x7F0 74 65 21 0D 0A           ; <- from HOST to TARGET
02517381: 0x7F1 21 63 6F 6D 70 5F 64 61  ; <- from TARGET to HOST (1st part of response)
02517381: 0x7F0 11                       ; <- from HOST to TARGET (acknowledge sequence)

!comp_date!    ; <- net data of the previously transferred string
```

At this step, the command „!comp_date!“ has just been sent. The terminal will now send the response string. Because the connection is now established, all CAN messages will be achknowledged immediately with extra handshake telegrams. So the sender only has to wait for the acknowledge telegram, before sending the next data telegram. An acknowledge telegrams contains exactly one data byte with the „sequence number“ (beginning with 0x11 for the first message of one text line). So by looking at the first data byte in a CAN message, we can tell if it's „data“ or „acknowledge“. Let's continue...

```
02517383: 0x7F1 74 65 21 3D 41 70 72 20
02517410: 0x7F0 12                      ; second acknowledge telegram (!)
02517410: 0x7F1 31 32 20 32 30 30 37 0D
02517411: 0x7F0 13                      ; third acknowledge telegram
02517412: 0x7F1 0A                      ; end of sent line, 0x0A = NEW LINE character
02517413: 0x7F0 14                      ; fourth acknowledge telegram

!comp_date!=Apr 12 2007   ; <- net data of the previously transferred string
```

At this step, the terminal (target) has sent its first complete line back to the host. Please note the appended carriage return (0x0D) and new line (0x0A) characters, which mark the end of a line. cknowledge telegrams contains exactly one data byte with the „sequence number“ (beginning with 0x11 for the first message of a text line). For the next line of text, the acknowlege sequence begins at 0x11 again, which means „the recipient acknowledges the first CAN message of the text line). This way, missing CAN frames can easily be detected.

```
02517425: 0x7F0 21 68 61 72 64 77 61 72
02517549: 0x7F1 11
02517550: 0x7F0 65 5F 6E 61 6D 65 21 0D
02517688: 0x7F1 12
02517688: 0x7F0 0A
02517830: 0x7F1 13

!hardware_name!   ; <- net data of the previously transferred string
```

At this step, the string „!hardware_name!“ has been completely send from host to terminal. Note that this is much faster now, because the response messages allow us to use eliminate unnecessary delays. In the next step, the terminal will send the answer (with the hardware name) back to the host :

```
02517839: 0x7F1 21 68 61 72 64 77 61 72
02517847: 0x7F0 11
02517848: 0x7F1 65 5F 6E 61 6D 65 21 3D
02517848: 0x7F0 12
02517850: 0x7F1 41 37 5F 54 46 54 5F 4C
02517851: 0x7F0 13
02517852: 0x7F1 0D 0A
02517853: 0x7F0 14

!hardware_name!=A7_TFT_L   ; <- net data of the previously transferred string
```

Etc, etc .. Note that TWO CAN MESSAGES may travel in the same direction, without a large gap, and without an acknowlege message „in between“. In older firmware revisions, we used an extra delay between the last response message (here: 02517830: 0x7F1 13) and the begin of the next string transmission (here: 02517839: 0x7F1 21 68 ...), but it later turned out to be unnecessary because modern CAN controllers do have at least a two-stage receive buffer.

6. Protocol to send strings via CAN using CANopen

If the target (terminal, or whatever) has a CANopen protocol stack implemented, the lower protocol level is totally different: We use a special object in the CANopen object dictionary, and use an SDO channel to send the application line-by-line into that object. Details are beyond the scope of this document; please refer to the CANopen protocol specification. Look for data type „Visible String“, because the text lines are transferred as „visible strings“ from the host into the target (and vice versa).

7. Protocol to send strings via RS-232 using 3964R

Most terminals with an RS-232 interface support downloading the application via this port, too. But be aware: The RS-232 interface may be occupied by a GPS receiver, or other devices; so you must switch the terminal into the „Download“ mode manually (through the terminal's system menu). This will cause the terminal firmware to stop any activities which may have „occupied“ the serial port (like communicating with the GPS, etc).

The download protocol uses 3964R frames to send the strings mentioned in chapter 4 .
The 3964R protocol specification can be found in the internet, eg.

http://de.wikipedia.org/wiki/3964R