

RS232 protocoll description for CHIPDRIVE & KartenZwerg

BASICS

CHIPDRIVE extern uses a half duplex transmission, i.e. data is transmitted in one direction only at a time. All data transmission sequences are initiated by the PC.

The command sequence set by the PC may have a character spacing of 0 to 50 ms. The answer to a completed data block is returned within 0 to 200 ms (CWT < 50ms; BWT < 200ms).

The following protocol description contains the data sent by the PC marked by '(')' and data sent by CHIPDRIVE marked by '[']'.

(QS) denominates the checksum which is to be calculated for each case. [nc] denominates a unspecified spacing byte reserved for future use.

The COM-port has to be configured as according to this table:

type	bps	DPS	RTS	DTR	ps
CHIPDRIVE extern	9600	8E2	+12V	+12V	15
CHIPDRIVE micro	9600	8E2	+12V	+12V	15
CHIPDRIVE extern II	9600	8E2	-12V	+12V	15
CHIPDRIVE twin	9600	8E2	-12V	+12V	15
CHIPDRIVE intern	9600	8E2	+12V	+12V	15
KartenZwerg	9600	8N1	+12V	+12V	16

The parameter 'ps' denominates the maximum number of bytes that can be transmitted with a single command. For write commands with less than 'ps' bytes a end character is required. (See explanation on write commands.)

COMMUNICATION SEQUENCES

The following order procedures has to be respected for accessing card data. The commands for reading, writing and PIN-related functions depend on the exact card type and will be explained later.

1. activate card
2. read access
3. read error counter. Cancel operation in case that the chip access counter is exhausted
4. PIN-entry
5. read access / write access
6. change PIN
7. deactivate card.

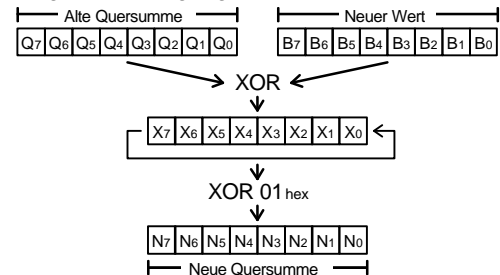
Steps 3,4 and 6 are only relevant for write access to cards featuring PIN security (4442/4428)

Step 2 should always be performed (at least one byte) prior to writing to the card.

The commands for retrieving the card status and displaying LED signals can always be processed by the terminal - even while a card is activated.

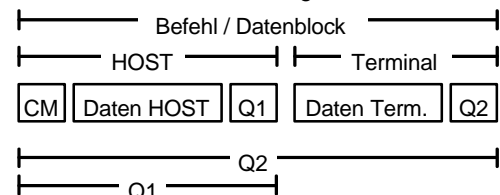
CHECKSUM CALCULATION

The checksum calculation starts with value 00 hex. Every byte send or received is calculated into the checksum according to following algorithm:



Please note, that the value X is rotated to the left 1 bit; bit 7 pushed out to the left is inverted and placed in the position of bit 0.

Each command is concluded with a checksum byte sent as last character by the pc issuing a command and the terminal answering to it. Note that the terminal does not start a new checksum calculation but continues with the values of the command string.



The following sequences contain the properly calculated checksum (unless variable data is contained!). The start value for the checksum calculation is 0x00. For CHIPDRIVE twin the start value for the second reader unit is 0X01.

GENERAL COMMANDS

• Card status check

(03,07)[ss,nc]

Encoding of the value [ss] :

0x: no card / no change

8x: no card / change

4x: card present / no change

Cx: card present / change

The low Nibble contains no data.

• **LED on** (6A,0F,B4)[01]

• **LED off** (6B,0F,B0)[01]

• **LED color** (6F,cc,6A,0F,QS)[01]

Only terminals featuring a bi-color LED can be process this command. The encoding of cc is:

00: off, 01: red, 02: green, 03: yellow

• **activate card** (60,0F,9C)[01]

• **deactivate card** (61,0F,98)[01]

CHIPTYPE 'I2C'

- **I2C read starting with address A**

Init sequence for I2C up to 16 kbit:

```
(7C,64,41,a1,a2,64,40,a3,0F,QS)[01]
```

a1:=Hi.A shl 1 or A0, a2:=Lo.A,
a3:=a1 or 01

Init sequence for I2C 32/64 kbit:

```
(7C,64,42,A0,a1,a2,64,40,A1,0F,QS)[01]
```

a1:=Hi.A, a2:=Lo.A

read n bytes:

```
(10 or (n-1),QS)[d1 ... dn,QS]
```

with n: 1..15 Bytes

- **I2C write starting with address A**

Init sequence for I2C up to 16kbit:

```
(7C,64,41,A0,00,64,40,A1,0F,36)[01]
```

```
(7E,10,DA)[xx,QS]
```

```
(7E,66,6E,a2,a1,pm,0F,QS)[01]
```

a1:=Hi.A shl 1 or A0 hex, a2:=Lo.A

pm specifies the page mode with (pm-1) being the
number of bytes per page

Init sequence for I2C up to 64kbit:

```
(7C,64,42,A0,00,00,64,40,A1,0F,0B)[01]
```

```
(7E,10,DA)[QS,QS]
```

```
(7F,66,6E,a2,a1,A0,0F,QS)[01]
```

a1:=Hi.A, a2:=Lo.A

The pagemode for cards with 32kbit and more is
preset to 16 byte.

write n data bytes (the lowest byte of the address
counter is incremented):

for n: 1..(ps-1) bytes:

```
(40 or (n-1),d1 ... dn,0F,QS)[01]
```

for n = ps bytes:

```
(4E,d1 ... dn,QS)[01]
```

For setting the high byte of the address counter
(also required for continuous writing) the
following sequence is required:

for I2C up to 16 kbit:

```
(7E,66,6E,a2,a1,pm,0F,QS)[01]
```

for I2C 32/64 kbit:

```
(7F,66,6E,a2,a1,A0,0F,QS)[01]
```

CHIPTYPE '2-wire'

- **read error counter**

```
(70,64,42,31,00,00,65,0F,80)[01]
```

```
(13,27)[d1,nc,nc,nc,QS]
```

The remaining number of trials for PIN entry is
represented by the number of bits set to 1 in 'd1'
(max. 3 trials = 07h).

- **PIN-entry**

```
(72,6E,00,39,03,0F,B5)[01]
```

```
(40,e1,0F,QS)[01]
```

```
(72,6E,01,33,03,0F,F5)[01]
```

```
(42,p1,p2,p3,0F,QS)[01]
```

```
(72,6E,00,39,03,0F,B5)[01]
```

```
(40,FF,0F,E4)[01]
```

e1: this value represents the new error counter
value. For each PIN entry one bit has to be set
to zero in comparison to the counter value read
from the card. The first trial bit 0 (0x06), the
second trial bit 1 (0x04) and for the last trial bit 2
(0x00).

p1 through p3 correspond to the first through
third byte of the PIN.

If the PIN entry was correct the error counter will
read as 0x07.

- **change PIN**

```
(72,6E,01,39,03,0F,A5)[01]
```

```
(42,p1,p2,p3,0F,QS)[01]
```

p1 thru p3 correspond to the three bytes of the
PIN. Changing the PIN is possible only after
successful PIN entry.

- **read 2-wire starting with address a**

Init sequence:

```
(70,64,42,30,a,00,65,0F,QS)[01]
```

with a: 0..255

read n bytes:

```
(10 or (n-1),QS)[d1 ... dn,QS]
```

with n: 1..15 Bytes

dx: data of the card in sequential order

- **write 2-wire starting with address a**

Init sequence:

```
(72,6E,a,38,03,0F,QS)[01]
```

with a: 0..255

write n data bytes:

for n: 1..(ps-1) bytes:

```
(40 or (n-1),d1 ... dn,0F,QS)[01]
```

for n = ps bytes:

```
(4E,d1 ... dn,QS)[01]
```

CHIPTYPE '3-wire', special functions

- **read error counter**

```
(70,A0,42,CE,FD,FD,80,50,0F,17)[01]
(10,21)[d1,QS]
```

The number of remaining trials corresponds to the number of Bits set to '1' in d1.

- **PIN entry**

```
(73,67,6E,FD,F2,02,0F,8C)[01]
(40,e1,0F,QS)[01]
(73,67,6E,FE,CD,02,0F,45)[01]
(40,p1,0F,QS)[01]
(73,67,6E,FF,CD,02,0F,55)[01]
(40,p2,0F,QS)[01]
(73,67,6E,FD,F3,02,0F,84)[01]
(40,FF,0F,E4)[01]
```

e1: this value represents the new error counter value. For each PIN entry one bit has to be set to zero in comparison to the counter value read from the card. The first trial bit 0 (0xFE), the second trial bit 1 (0xFC) and so on.

p1: first byte of the PIN

p2: second byte of the PIN

If the PIN entry was correct the error counter reads as 0xFF.

- **change PIN**

```
(73,67,6E,FE,F3,02,0F,B4)[01]
(40,p1,0F,QS)[01]
(73,67,6E,FF,F3,02,0F,A4)[01]
(40,p2,0F,QS)[01]
```

p1: first byte of the new PIN

p2: second byte of the new PIN

Prior to changing the PIN a correct PIN entry of the old PIN is required.

CHIPTYPE '3-wire' read/write

- **read 3-wire starting with address A**

Init sequence:

```
(70,A0,42,a1,a2,00,80,50,0F,QS)[01]
a1:=Hi.A shl 6 or 0E hex, a2:=Lo.A
with A: 0..1023
```

read n bytes:

```
(10 or (n-1),QS)[d1 ... dn,QS]
with n: 1..15 bytes
```

dxx: data of the card in sequential order.

The address is automatically incremented, so data can be read to the end of the card without issuing another init sequence.

- **write 3-wire starting with address A**

Init sequence:

```
(73,67,6E,a2,a1,02,0F,5F)[01]
a1:=Hi.A shl 6 or 33 hex, a2:=Lo.A
with A: 0..1023
```

write n bytes of data (the low byte of the address counter is incremented):

for n: 1..(ps-1) bytes:

```
(40 or (n-1),d1 ... dn,0F,QS)[01]
```

for n = ps bytes:

```
(4E,d1 ... dn,QS)[01]
```

To change the high byte of the address counter a new init sequence needs to be issued. Writing across a 8-bit low address byte rollover is not allowed.

Transparent mode (used for T=0, T=1,
T=14 implementation)

- **activate card**

(60,0F,9C)[01]

- **deactivate card**

(61,0F,98)[01]

- **read ATR of the card**

(80,6F,00,05,qs)[d1,d2,...]

(A0,6F,00,05,qs)[d1,d2,...]

Both sequences need to be sent 3 times alternately. Data bytes received need to be decoded and checked according to ISO specification. A card with 'active high reset' will answer to the first sequence, cards with 'active low reset' will answer to the second sequence.

- **inverse convention**

Cards using the 'inverse convention' according to ISO can be recognized by checking the ATR.

The following rules need to be observed for such cards:

1. the bit order of all bytes sent needs to be inverted: (D0->D7, D7->D0 and all inverted).
2. the serial interface of the host needs to be reconfigured from 'even parity' to 'odd parity' umgestellt werden.
3. the terminal needs to be set to 'odd parity' by issuing following commands:

- odd Parity: (6F,80,6A,0F,qs)[01]

- even Parity: (6F,40,6A,0F,qs)[01]

Please note that the ATR sequence should always be performed with both settings (odd + even parity) in order to make sure that cards using the inverse convention are recognized.

- **communicating with the card**

(6F,nn,05,qs,t1,t2,...)[r1,r2,...]

nn: denominates the number of data bytes to be transmitted to the card.

tn: data bytes to be transmitted transparently to the card.

rn: answer from the card (transparent)