RS232-Communication
&
XML-Description

M. Zimmermann

Version 2.0

1 Revision History

Date	Version	Change Details	Firmware Min
11.10.2018	1.0	First Release, M.Zimmermann	
21.12.2018	1.1	<RI>, Command representation adapted	
28.03.2019	1.2	Drive-CMDs ergänzt	
17.04.2019	1.3	K17a spezific Settings added	
04.07.2019	1.4	Sequence RD/Write commands explanation added	
06.09.2019	1.5	Mode Correction implemented	
24.09.2019	1.6	Drive Settings / Filter Setting Explanation integrated	
10.03.2020	2.0	Added at Homing description, 'Save Marked Position as Home' Added CMD <Df>, for User specic Communication Command Timing extended, new Timeout at RX of extended data	V2.5-000 K11a K11b K17a K17e

Index

2 Introduction

Target of this document

This document shall describe the main functions of **adlos**'s KannMotion RS232 Communication protocol.
The '***COM-Watch***' shall support **adlos** business partners and customers, in product maintenance, product firmware updates, field return qualifications … of smart/intelligent battery packs and power products of **adlos**'s engineering department.
The software is as it is, and in principle for free for **adlos** customers, the software is not made for a broad range of standard users, it's made in principle for technical engineers which are used in working w. windows based software and have some understanding of technical things.
This document has not the aim, to describe all functions, working steps and behinds of diagnostic interface or connected product, it shall only describe the main functions and give some ideas about the functionality.

Licence agreement

'***COM-Watch***', afterwards named software, is distributed as Freeware for **adlos** partners and customers. You may use this software on any number of computers for as long as you like, for personal and also for commercial use. The software is **NOT** Public Domain software. We allow the free distribution of the software, but we retain ownership and copyright of the software and its source code in its entirety. You may use and/or distribute the software only subject to the following conditions:

- You may not modify the program or documentation files in any way
- You must include all the files that were in the original distribution
- You may not decompile or otherwise reduce the Software to a human perceivable form
- You may not sell the software or charge a distribution fee
- You understand and agree with this license and with the disclaimer of warranty and the limitation of liability printed below

For distribution in print-media or internet you will need the permission of **adlos.** Private distribution does not seek a permission.

Disclaimer of warranty

The software and related documentation are provided "as is", without warranty of any kind. **adlos** disclaims all warranties, express or implied, including, but not limited to, the implied warranties of design, merchantability, fitness for a particular purpose. **adlos** does not warrant that the functions contained in the software will meet your requirements, or that the operation of the software will be uninterrupted, error-free or complete, or that defects in the software or documentation will be corrected.

Limitation of liability

Under no circumstances, including negligence, shall **adlos** be liable for any lost revenue or profits or any incidental, indirect, special, or consequential damages that result from the use or inability to use the or related products or documentation, even if the author has been advised of the possibility of such damages.

Copyright

3 KannMotion Rel-2 Command Set (Basics)

For detailed command and protocol settings, please refer to your device specific manual !

RS232 Standard Settings

38400 Baud, 8 Data, 1 Stop, Parity=none (Standard), *might be different refer to device manual*

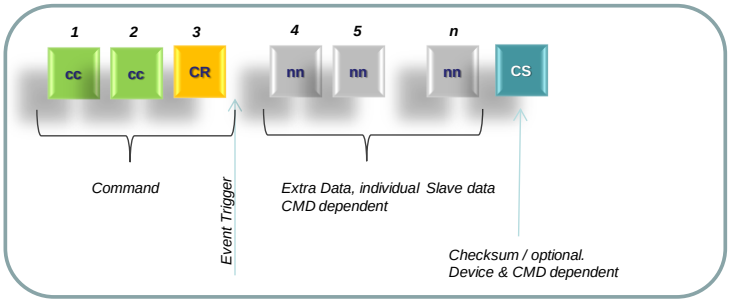
Protocol Basics

In principle every command consists on a 2-Byte-Instruction followed by a Trigger Byte (CR).
The Command Bytes has to be inside a certain number range, for protocol consistency (clear distance between Trigger Byte and Command)

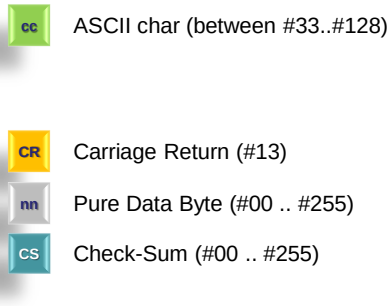
After trigger byte is detected, interpretation of 2-command bytes (word) will be done, and if command is known, the command is sent back to the transmitter and the receiver prepares it's buffer for additional data if needed …

Hint: if command is known, the device does answer the command immediately to inform the master, that command was received (and or is waiting for additional data)

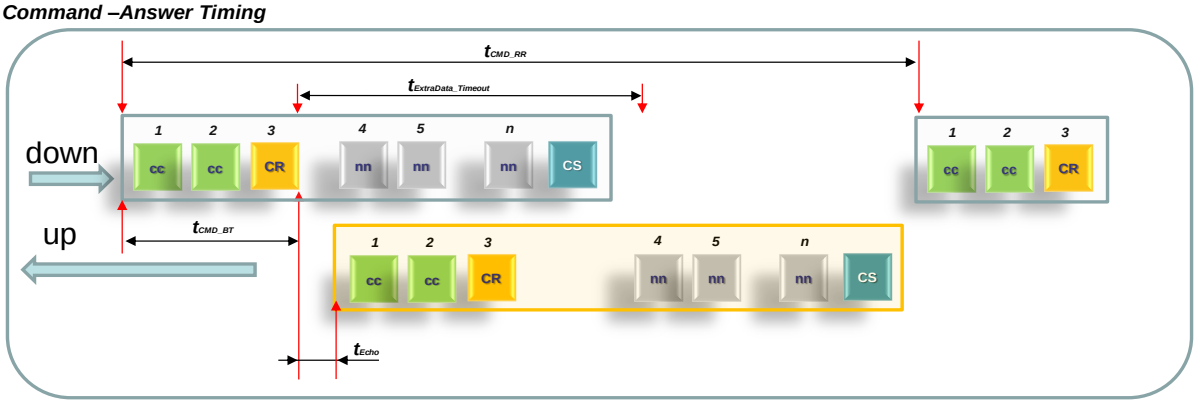
Command Frame



Legend



Command –Answer Timing



Short	*Description*	*min*	*typ*	*max*	*Unit*
t_{Echo}	Echo Delay time	0.05	1	2	ms
t_{CMD_RR}	Command Repetition Rate	0 [1)]			ms
t_{CMD_BT}	Commando Time Consumption	781 [2)]			us
$t_{ExtraData\_Timeout}$	Commando extra data timeout	625us + 312.5us x Bytecount			us

1) Only if Receive buffer of device is not full, in principle the device can handle several commands without delay, but this cannot be guaranteed due to different command timing needs (e.g. EEprom write CMD, needs time..), to get no conflicts its recommended to wait at least to the command echo..
2) Standard Baud rate of 38000 Baud
3) ExtradataTimout is: Time= (expected Bytecount+2) * BitTime * 1.2

Command Set example

CMD	*Description*	*double CMD*	*Additional data*	*Answer*	*time*
ID$_{<CR>}$	**Ident Device (Firmware)**	no	no	Echo + 8-Data Bytes +CK	norm
Commands					
CR$_{<CR>}$	**System Software-Reset Restart**	yes	no	Echo	norm
CS$_{<CR>}$	**Save Parameters**	yes	no	Echo	ext.
Read Data Instructions					
RI$_{<CR>}$	**Read Device Info**	no	no	Echo + 20-Data Bytes +CK	norm
RE$_{<CR>}$	**Read EEPROM data**	no	yes	Echo + n-Data Bytes +CK	norm
Drive specific sample Instructions					
Dr$_{<CR>}$	**DriveRUN (Rotate) <rpm>**	no	rpm + CK	Echo + enDrvErrCode +CK	norm
Dp$_{<CR>}$	**DrivePosition (GotoPos) <Mode> <Pos>**	no	mode,pos,CK	Echo + enDrvErrCode +CK	norm

Keywords

CK	:byte	*0..255*	Checksum of full Command or Answer Frame (Sum, followed by complement of 2)
ACK	:byte	*0x06*	
NAK	:byte	*0x15*	

time

norm	Standard device answer time, e.g. typ.	1.1ms

Usually Read commands are defined without an ending Command checksum, because they are not critical for application. The answer of a read command is usually maintained with a checksum, zhis to avoid data inciconsitence on the masters side !

As a consequence, normally write CMD's are defined with checksum, this to avoid that wrong parameters might be stored or executed on slave side!

Checksum Calculation

Checksum of full Command or Answer Frame (Sum, followed by complement of 2).

➔ Means: as an receiver, the sum of each byte of the full frame shall allways be **<0>,** otherwise Message is not correct

Standard Command details

<ID> Identify Command

This is a very important command to get information which firmware is running on the connected device, it also returns firmware version info.

Send: **ID<CR>**
Answer: **ID<CR> nnnnnn <V> <M> <CK>**

```
Nnnnnn  : Firmware Number                  6-Byte ASCII char     e.g. '300416'
<V>     : Software Version                 data byte [0..255]    e.g. 0x01
<M>     : Software Minor-SubVersion  data byte [0..255]    e.g. 0x01
<CK>    : Checksum of full answer          data byte [0..255]    e.g. 0x36
```

<RI> Read Device Information

This is a very important command to get information about Device Hardware and if mounted, system ID number.

Send: **RI<CR>**
Answer: **RI<CR> <HWi> <HWn> <SNR> <SYi> <SYn> <CST> <RES> <COR> <CK>**

```
<HWi>   : Hardware Index / Release   data byte   [0..255]   e.g. 0x01
<HWn>   : Hardeware Art-Number       3-data byte [0..16777215]    e.g. 100500
<SNR>   : Serial Number              4-data byte [0..0xFFFF]      e.g. 2018110001
<SYi>   : System Main-Version        data byte   [0..255]   e.g. 0x01
<SYn>   : System Art-Number          3-data byte [0..16777215]    e.g. 190055
<CST>   : Quality Check stamp        data byte   [0..255]   e.g. 0x03
<RES>   : Reserved                   3-data byte [0..16777215]    e.g. 0
<COR>   : CPU Information            4-data byte [0..0xFFFF]      e.g. 0
<CK>    : Checksum of full answer    data byte [0..255]           e.g. 0x36
```

Note: MultiByte returns are transfered in liddle endian orientation (Low Significant byte first)
System: motor/gear/controller combination // Hardware: PCB Information

<RE> Read EEPROM data

This command allows to read out EEPROM data.

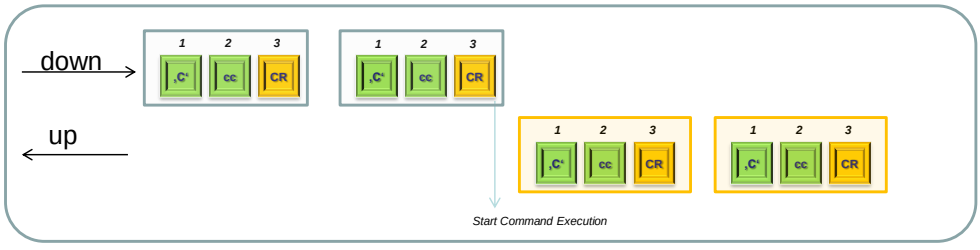
Send: **RE<CR> <ADR> <CNT>**
Answer: **RE<CR> <DATA> <CK>**

```
<ADR>   : First-Adress in EEPROM to read     data byte [0..255]    e.g. 0x01
<CNT>   : Number of Data bytes to read       data byte [0..255]    e.g. 0x10
<DATA>  : EEPROM content                     cnt-data bytes e.g. 0,1,2..
<CK>    : Checksum of full answer            data byte [0..255]    e.g. 0x36
```

<C_> Critical Commands

All commands in this block are somehow critical for the connected device. For this instance, these commands have to be send ***<TWICE>, without another command in between***, to be excepted.

Command –Answer Timing (Double Command Cx<CR>)



Drive specific Commands

The Drive Commands are in the same way as the typical and device specific integrated commands. They got here a special chapter due to the fact, that they have some common properties or return values.

Drive Commands overview

CMD	*Description*	*double CMD*	*Additional data*	*Answer*	*time*
RD<CR>	**Read Drive State**	no		Echo + Data +CK	norm
Dr<CR>	**Drive RUN (Rotate) <rpm>**	no	rpm + CK	Echo + enDrvErrCode +CK	norm
Dp<CR>	**Drive to Position (GotoPos) <Mode> <Pos>**	no	mode,pos,CK	Echo + enDrvErrCode +CK	norm
Dc<CR>	**Drive Configuration Write <Pid> <value>**	no	<Pid> <value>	Echo + enDrvErrCode +CK	norm
DC<CR>	**Drive Configuration Read <Pid>**	no	<Pid>	Echo + Data +CK	norm
Dh<CR>	**Drive Do Homing**	no	<mode> <Dir> <timeout>	Echo + enDrvErrCode +CK	norm
Ds<CR>	*Drive Sequence Write*	no	<Did> <data32>	Echo + enDrvErrCode +CK	norm
DS<CR>	*Drive Sequence Read*	no	<Did>	Echo + enDrvErrCode + data32 +CK	norm
Df<CR>	**Drive forward Data User Sequence**	no	<data32>	Echo + enDrvErrCode +CK	norm

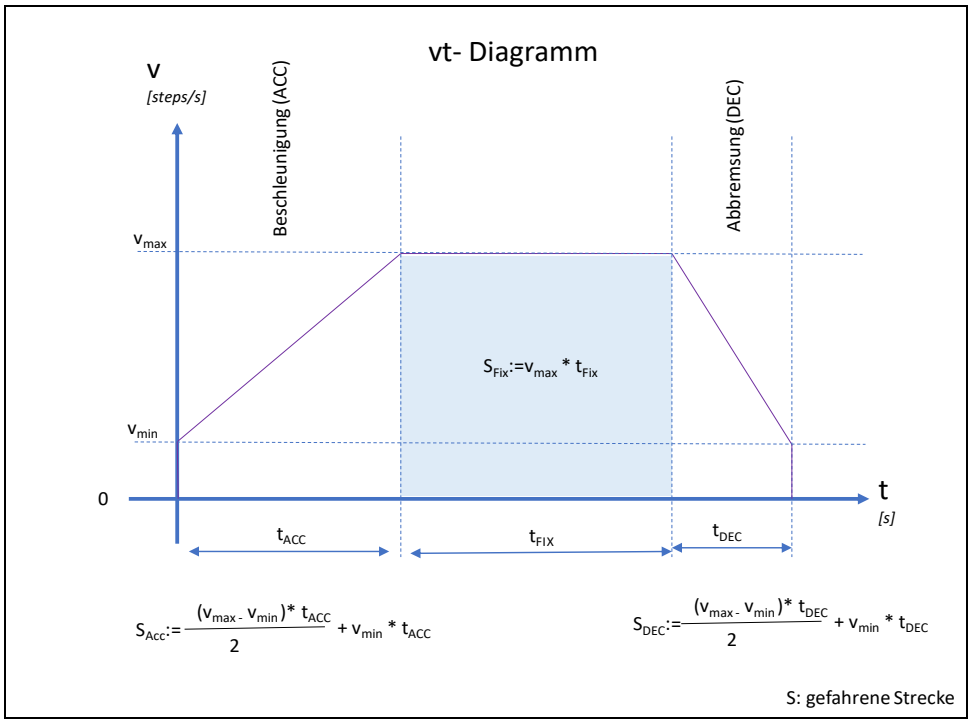
Keywords

CK	:byte	*0..255*	Checksum of full Command or Answer Frame (Sum, followed by complement of 2)
ACK	:byte	*0x06*	
NAK	:byte	*0x15*	

Return Type Enumeration <enDrvErrCode>

Value	Meaning	Description
0	eMS_OK	Ok, no Error
-1	eMS_ERR_OUTofRange	Parameter out of valid Range
-2	eMS_ERR_ParamisWrProtected	Parameter is Read only, can not been written
-3	eMS_ERR_CMDnotAccepted	Command could not been accepted, maybe drive was in wrong mode
-4	eMS_ERR_CMDnotKnown	Command not known
-5	eMS_ERR_ParamNotKnown	Parameter is not known
-6	eMS_ERR_ActionFailed	Action Failed
-7	eMS_ERR_NoMoveUntilRestart	Moving actually not allowed until Restart/ReInit (Due to params change)

Vt-Diagram



<RD> Read Drive State

This command requests the current state of the motor.

```
Send:      RD<CR>
Answer:    RD<CR> <State-Date> <CK>
```

State data: see XML (State, ErrorBits, Actual-Position, Target Position, Actual Speed..)

<Dr> Drive Run (rotate)

This command sets the motor into motion w. dedicated speed. Speed = 0 means Stop.

```
Send:      Dr<CR> <01rpm> <CK>
Answer:    Dr<CR> <enDrvErrCode> <CK>

    <01rpm>  : Speed, in signed integer 16-Bit [1=0.1rpm]   [-:CW] /[+:CCW]
               0= SoftStop    -1= FastStop
    <CK>     : Checksum
    Return   : enDrvErrCode        see Return Type Enumeration <enDrvErrCode>
```

Examples

```
    100rpm   -> 1000 = 0x03E8             -> Dr#0D#E8#03#CK
    -100rpm  -> -1000= 0xFC18             -> Dr#0D#18#FC#CK
```

<Dp> Drive to Position (GotoPos)

This command sets the motor into motion w. dedicated mode and destination-position.

```
Send:      Dp<CR> <mode> <pos> <CK>
Answer:    Dp<CR> <enDrvErrCode> <CK>

    <mode>   : Control mode (8-Bit)
               0 = Microsteps
               1 = in Micrometer OR 0.1° (depending on Drive Configuration)
               2 = relative Move in Microsteps
    <pos>    : Target Position, signed integer 32-Bit [-:CW] /[+:CCW]
    <CK>     : Checksum

Return:    enDrvErrCode        see Return Type Enumeration <enDrvErrCode>
```

Note
Mode=1: Control mode depending on general Control setting, for Captive-Actuators it will be 1-Mikrometer/Unit, for rotative drives it will be 0.1°/Unit

Examples

```
    Mode0/128mStp   ->     0x00 0x0000000080     -> Dp#0D#00#80#00#00#00#CK
    Mode1/409.6°    ->     0x01 0x0100100000     -> Dp#0D#01#00#10#00#00#CK
```

<Dc> Drive Configuration Write

This command manipulates general drive Settings.

```
Send:      Dc<CR> <Pid> <value> <CK>
Answer:    Dc<CR> <enDrvErrCode> <CK>

    <Pid>    : Parameter -ID (8-Bit)
    <value>  : Value to set (32-Bit)
    <CK>     : Checksum

Return:    <Echo><enDrvErrCode><CK>   see Return Type Enumeration <enDrvErrCode>
```

Examples

```
Set Control Mode->     Pid=10 Value=1        -> Dc#0D#0A#01#00#00#00#CK
```

Configuration Parameters Standard

Total 32-Byte

Pid	Parameter Name	Unit	Rep/Rights	Description
	Checksum		*UI_8, RO*	*Checksum of Control Settings*
1	Factory Driver Setting	E	BF_8, RO	Factory Driver Main-Setting B0: MotMoveDIR_Inv [0:No /1: Yes] B1: EncoderOnGearAxis [0:No /1: Yes]
2	MotorStepType	E	UI_8, RO	Motor Step Type, Steps/Half Round e.g. 200 = 0.9° // 100 = 1.8°
3	Driver Output Config	E	BF_8, RW	B0: OUT1 - NPN active B1: OUT1 - PNP active B2: OUT2 - NPN active B3: OUT2 - PNP active ..
4	Motor_Current_Max	mA	UI_16, RWP	Max motor current, 100% reference for Torque Settings
5	Motor_Acceleration_Max	r/s^2	UI_16, RWP	Max motor acceleration FixPoint 0xFF.FF
6	Driver Input LoTHR	mV	UI_16, RW	Digital Input Low-Threshold
7	Driver Input HiTHR	mV	UI_16, RW	Digital Input High-Threshold
8	Gear_Ratio	1:N	UI_32, RWP	Gear Ratio FixPoint 24 (0xFF.FFFFFF)
9	Gear_Backlash	0.1° / Steps / um	SI_16, RW	Gear Backlash
10	Control_Mode	0.1° / um	UI_8, RWP	Control mode [0: um / 1: 0.1°]
11	Microsteps	1,2,4,8,16,32,64,128	UI_8, RW	Microstepping mode, depending on driver [0..7] [0:1 uStps / 1:2 uStps / 2:4 uStps … 7: 128 uStps
12	Speed_Min	0.1 rpm	UI_16, RW	Min_Speed, [0.1rpm/E]
13	Speed_Max	0.1 rpm	UI_16, RW	Max_Speed, [0.1rpm/E]
14	Torque_Hold	0.5 %	UI_8, RW	Holding Torque, [0.5%/E] of Motor Current_Max
15	Torque_ACC	0.5 %	UI_8, RW	Holding Torque, [0.5%/E] of Motor Current_Max
16	Torque_Run	0.5 %	UI_8, RW	Holding Torque, [0.5%/E] of Motor Current_Max
17	Torque_DEC	0.5 %	UI_8, RW	Holding Torque, [0.5%/E] of Motor Current_Max
18	Acceleration	0.5 %	UI_8, RW	Acceleration value, in [0.5%/E] of Acceleration Max
19	Deceleration	0.5 %	UI_8, RW	Deceleration value, in [0.5%/E] of Acceleration Max
20	Position Regulator Control	E	UI_8, RW	Position Regulator Control Byte
21	Driver Input Filter Control	E	UI_8, RW	Analog and Digital Inputs Filtering Control Byte

RO: read only (factory setting) RW: read-write RWP: write protected

Configuration Parameters K17a specific

Total 6-Byte

22	K17a_CNTRL	E	UI_8, RWP	Driver specific CONTROL Setting
23	K17a_KVAL	E	UI_8, RWP	Driver specific Motor Setting
24	K17a_Start_SLOPE	E	UI_8, RWP	Driver specific Motor Setting
25	K17a_Final_SLOPE	E	UI_8, RWP	Driver specific Motor Setting
26	K17a_INTSPEED	E	UI_16, RWP	Driver specific Motor Setting

Details see K17a Firmware Documentation

Configuration Parameters K24c specific

Total 10-Byte

22	K24c_CNTRL	E	UI_8, RWP	Driver specific CONTROL Setting
23	K24c_KVAL/TVAL	E	UI_8, RWP	Driver specific Motor Setting
24	K24c_Start_SLOPE/TFAST	E	UI_8, RWP	Driver specific Motor Setting
25	K24c_Final_SLOPE/TON_MIN	E	UI_8, RWP	Driver specific Motor Setting
26	K24c_INTSPEED	E	UI_16, RWP	Driver specific Motor Setting
27	K24c_GATECFG1	E	UI_16, RWP	Driver specific Motor Setting
28	K24c_GATECFG2	E	UI_8, RWP	Driver specific Motor Setting
29	K24c_TOFF_MIN	E	UI_8, RWP	Driver specific Motor Setting (Bit7=Curr Mode)

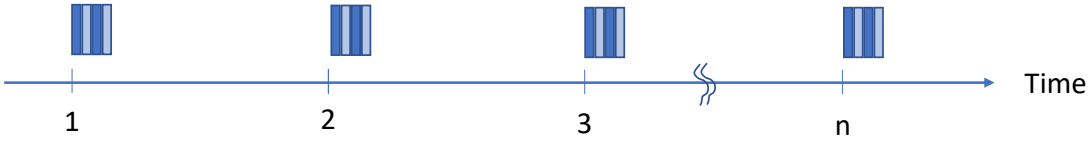
Details see K24c Firmware Documentation

Configuration Parameter <21> Filter Settings Details

Filter Setting parameter enables, filter choice for each digital input, and also for analog input.

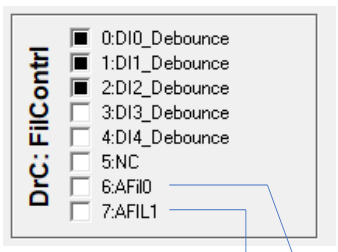
IO-Ain Sampling scheme

Every input is (if possible) sampled as an analogue input, 1 sample is converted for each channel every 1-ms



Without selecting a extra filtering by Parameter 21, every sample is added to a buffer, wich is acting as an low-pass filter of first order. Additionaly you might add a second filter by param 21, as describben below.

Filter Setting / Bit Representation



Digital Input signal Filtering
1-Bit for each Input (Set/Not Set)

Dix_Debounce	
0	Standard low-pass filter, 1. order SR=1ms t=5ms
1	Debounce filter, for mechanical switches SR=1ms t=24ms

analog Input Filter Selection

AFIL1	AFil0	Comment	
0	0	Standard low-pass filter, 1. order	SR=1ms t=5ms
0	1	Mean-Filter (of 8-Values)	SR=5ms t=40ms
1	0	Median-Filter (of 8-Values)	SR=5ms t=40ms
1	1	Reserved	

SR: sample rate

for electromechanical switches, use Debounce filter setting for proper operation.

<DC> Drive Configuration Read

This command returns drive Settings, Pid=0xFF returns all parameters.

```
Send:       DC<CR> <Pid> <CK>
Answer:     DC<CR> <value> .. <CK>

    <Pid>    : Parameter -ID (8-Bit)    (Pid=0xFF returns all parameters)
    <CK>     : Checksum

PID=0xFF    Return:       <Echo> Data of all DC parameters PID0..PIDnn <CK>
PID=0xFE    Return:       <Echo> calculated Position Limit [um/0.1°] SI_32 <CK>
PID=0..n    Return:       <Echo><Pid><Mode&Size><value32-Bit><CK>
```

<Df> Drive forward Data to User Sequence

This command allows to forward data into User specific Coding segment. See also KannMotion Manager or Application Note 100631.

```
Send:       Df<CR> <data32> <CK>
Answer:     Df<CR> <enDrvErrCode> <CK>

    <data32>    : forwarded data (32-Bit, interpretation User Code specifig
    <CK>        : Checksum

Return:     enDrvErrCode        see Return Type Enumeration <enDrvErrCode>
```

<Data32> ist accesible inside USER code with structure `<stAppCSPS.SPSUserVar.st32_ComData>`. At the End of operation this data structure shall be set to ***<0>*** inside user code otherwise ***<Df>*** command will next time return ***eMS_ERR_CMDnotAccepted***

<Dh> Drive Do Homing

This command executes a Homing travel.

```
Send:      Dh<CR> <mode> <timeout> <DirSpeed>  <CK>
Answer:    Dh<CR> <enDrvErrCode> <CK>

    <mode>      : Homing Mode (8-Bit)
    <timeout>   : Timeout of Homing Movement in [ms] (16-Bit)
    <DirSpeed>  : direction and Speed of Homing Movement rpm (8-Bit)[-:CW] /[+:CCW]
                  [-128..127rpm]
      <CK>      : Checksum

Return:     enDrvErrCode        see Return Type Enumeration <enDrvErrCode>
```

Examples

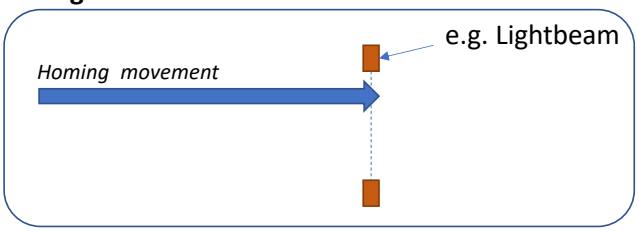
```
DoubleMove on Input 3 = Hi, -100rpm, Timeout=5000ms  -> Dh#0D#53#XL2[5000]#XL1[-100]#CK
```

Homing modes

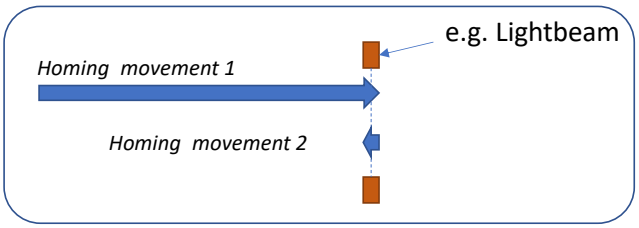
Mode	Type	Description	Timeout	Accuracy
0x00	Store Home	Save actual Position as Home (0)	na	na
0x01	Store 'Stall-Mark' as Home	Saves during Homing generated Mark as 0, this means actual Posiion might be different to Mark.. Use this command after a Homing on Stall, to get better precision	na	na
0x1m	Multi move	Move until **stall** <m> is Torque Setting (TorqRunACCDEC * m / 15)	YES	Lo
0x2n	Single move	Move until **Input <n>, signal goes <Lo>**	YES	Med
0x3n	Single move	Move until **Input <n>, signal goes <Hi>**	YES	Med
0x4n	Double move	Move until **Input <n>, signal goes <Lo>** than direction change and 1/10 speed until **signal goes <Hi>**	YES	Hi
0x5n	Double move	Move until **Input <n>, signal goes <Hi>** than direction change and 1/8 of speed until **signal goes <Lo>**	YES	Hi

n: defines Digital Input number *m: defines Running Torque (reduction) [0..15]*

single move



Double move



Homing Functionality could be also used for driving to a sensor mark, or to detect the driving boarders. Caused by this, if you would storee reached position as Zero (Homing Mark) it's needed to send a <Store-Home> frame after homing mevements.

Homing movements will take some time, you can check progress by polling the <App-state> or you Enable <PosReach> Event.

<Ds> Drive Sequence Write

This command enables sequence writing into Flash.

Send: **Ds<CR> <Did> <data16> <CK>**
Answer: **Ds<CR> <enDrvErrCode> <CK>**

```
<Did>    : Data-Block -ID (16-Bit Data Adress)
<data16> : 16 Bytes of data to write
<CK>     : Checksum
```

Return: enDrvErrCode see Return Type Enumeration <enDrvErrCode>

Note: Did = 0xFFFF, is an not valid Adress and is used for special CMDs

Did= 0xFFFF / data16[0]=0xA0 **Erease Sequence Flash Start (may Take e.g. 100ms!)**
Did= 0xFFFF / data16[0]=0x11 **Finish Programming, Lock Flash**

Examples COMWatch Terminal:

```
Ds#0D#XL2[0xFFFF]#XL1[0xA0]#XL3[0]#XL4[0]#XL4[0]#XL4[0]#CK        // Erase User Section ->Start
Ds#0D#XL2[0]#01#02#03#04#05#06#07#08#09#10#11#12#13#14#15#16#CK   // Write Flash ab 0...
Ds#0D#XL2[16]#01#02#03#04#05#06#07#08#09#10#11#12#13#14#15#16#CK  // Write Flash ab 16...
Ds#0D#XL2[0xFFFF]#XL1[0x11]#XL3[0]#XL4[0]#XL4[0]#XL4[0]#CK        // Finish Flash write Operation
```

<DS> Drive Sequence Read

This command enables sequence reading from Flash.

Send: **DS<CR> <Did> <CK>**
Answer: **DS<CR> <enDrvErrCode> <data16> <CK>**

```
<Did>    : Data-Block -ID (16-Bit Data Adress)
<data16> : 16 Bytes of data read
<CK>     : Checksum
```

Return: enDrvErrCode an Data see Return Type Enumeration <enDrvErrCode>

Examples COMWatch Terminal:

```
DS#0D#XL2[0]#CK     // Read User Section Starting @ 0
DS#0D#XL2[16]#CK    // Read User Section Starting @ 16
DS#0D#XL2[32]#CK    // Read User Section Starting @ 32
```

XML – Communication- Device Descripition - FILE

Interpretation and calculation of values and commands are also described in a XML- File format.
This chapter will give an overview how to work- interpret an XML-File.
If your are not familiar w. XML please refer also to www or dedicated literature.

http://en.wikipedia.org/wiki/XML

XML definition File *.dtd

This file is describing filed elements and types…
Note: for a data set not every element needs to be defined…

ADLOS_DIAGNOSE.dtd

```
<!ELEMENT OnlineData (DataTable)+>
<!ATTLIST OnlineData IddNr CDATA #REQUIRED>

<!ELEMENT DataTable (DataRec)+>
<!ATTLIST DataTable
   CMD       CDATA            #REQUIRED
   Info      CDATA            #IMPLIED
   Timeout    CDATA            #IMPLIED
   Intervall  CDATA            #IMPLIED
>
<!-- Definition DatenRecord -->
<!ELEMENT DataRec EMPTY>
<!ATTLIST DataRec
   name     CDATA              #REQUIRED
   type     (SI_8|UI_8|UI_16|SI_16|UI_32|SI_32|ENU|BF_8|STR_X|UI_16e|SI_16e|UI_32e|SI_32e) #REQUIRED
   factor   CDATA              #REQUIRED
   unit     CDATA              #REQUIRED
   formula  CDATA              #IMPLIED
   digits   CDATA              #IMPLIED
   show     (true|false)        #IMPLIED
   size     CDATA              #IMPLIED
   list     CDATA              #IMPLIED
   cParamID CDATA              #IMPLIED
>
<!-- Definition VersionsRecord -->
<!ELEMENT VerTable EMPTY>
<!ATTLIST VerTable
   VersionHex   CDATA             #REQUIRED
   IdtNr       CDATA             #REQUIRED
   IddNr       CDATA             #REQUIRED
   IdCmdNr      CDATA             #REQUIRED
   IdCofNr      CDATA             #REQUIRED
   Info        CDATA             #IMPLIED
>
<!-- diverse Felddefinitionen -->
<!ELEMENT Device  (#PCDATA)>
<!ELEMENT Autor   (#PCDATA)>
<!ELEMENT Date    (#PCDATA)>

<!-- 2.Teil Controls und Config -->
<!ELEMENT ControlsData  (CommandData, ConfigData)>

<!ELEMENT CommandData  (CommandTable)+>
<!ATTLIST CommandData  IdCmdNr CDATA #REQUIRED>

<!ELEMENT CommandTable EMPTY>
<!ATTLIST CommandTable
   CMD       CDATA            #REQUIRED
   Caption    CDATA            #REQUIRED
   Delay     CDATA            #IMPLIED
   Hint      CDATA            #IMPLIED
   Warning    CDATA            #IMPLIED
>
<!ELEMENT ConfigData   (DataTable)+>
<!ATTLIST ConfigData   IdCofNr CDATA #REQUIRED>
```

XML File Details

Data types

Standard integer types

```
// ******* Unsigned Integer Types *****************************
typedef unsigned char               UI_8;                   //!<  8 Bit -> [0..255]
typedef unsigned int                UI_16;                  //!< 16 Bit -> [0..65535]
typedef unsigned long int           UI_32;                  //!< 32 Bit -> [0..4'294'967'295]

// ******* signed Integer Types *******************************
typedef signed char                 SI_8;                   //!<  8 Bit -> [ -128  .. +127]
typedef signed int                  SI_16;                  //!< 16 Bit -> [-32768 .. +32767]
typedef signed long int             SI_32;                  //!< 32 Bit -> [-2147483648 .. +2147483647]
```

Special integer types

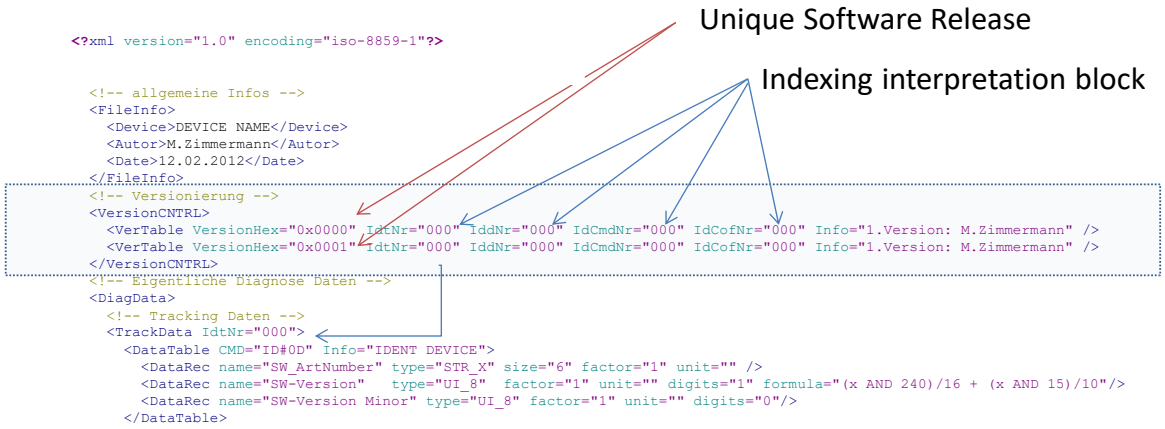
```
ENU;                    //!<  8 Bit Enumeration, element list has to be filled in
BF_8;                   //!<  8 Bit Bitfield,    element list has to be filled in
SI_16e, SI_32e;           //!<  same as SI_16, and SI_32, but invers endian interpretation
UI_16e, UI_32e;           //!<  same as UI_16, and UI_32, but invers endian interpretation
```

Special types

```
STR_X;                  //!<  ASCII String, variable length
```

XML File Firmware Version Handling

Every device has its unique XML-File. An XML file of a certain device might cover several different Firmware Releases, where different data interpretation might be needed. For that purpose, there is a version block included.



XML File 'factor' field

The factor field is the easiest way, to describe a 'real value' calculation out of the raw data.

Example:
UI_16 16 Bit data 1000 => 1.000V

The data the interface gets will be 1000, the Unit is defined as 'V', so we want that the GUI will show 1.000V, to get this we do:

Factor="0.001" digits="3"

XML File 'formula' field

For more complex 'real value' calculation we can also use the formula field. For that purpose it's recommended to set ***factor="1.0"*** due to the fact, that factor operation is always done….

Form of Formula:e.g. 5*x+1 (where x is representing raw data)

Example:
UI_16 16 Bit data value = 2000 -> 1.3A (Offset Correction 700)

Factor="1.0" formula="(x-700) / 1000" digits="1"

XML File Data type STR_X

This data type might handle a various length of answers. You might define the string length to a defined number of chars, while using the additional information size.

e.g. ***size="6"*** ***means, there will be 6 bytes answer expected***

Note:
If you do not know the exact answer size, write at size the max number of bytes you expect and set @ the command descripition line a dedicated timeout in ms ! ***timeout="100"***

XML File Blocks- clusters

The content of the XML file is clustered in separate blocks. The blocks are separated in it's use-case.

Case: 'Online data' block ***<OnlineData IddNr="000">***
This blocks contains often changing data like actual current, voltage…
So this block normally is used to show actual state, mode, currents … of the device.

Case: 'Track data' block ***<TrackData IdtNr="000">***
This blocks contains recording data like Lifetime, max Temperature, RunTime, Serial Number …
So this block normally is used to read out the history of the device

Case: 'Controlls data' block ***<ControlsData>***
This blocks contains special commands of the device, like Reset, Restart, ….
So this block is used to generate a flexible GUI.
There might be also some Hint fields or warning fields defined, where the USER has to accept an warning dialog before chosen instruction might be executed.

Example Data Read CMD R0

Example: Read Structure '0' -> COMMAND 'R0'

Send: **R0<CR>**

Answer: **R0<CR> 0x00 0x11 0x33 … (size has to be evaluated with XML-file)**

Example of CMD <R0> data interpretation

Returns 9 Bytes + CheckSum:

Interpretation of these Bytes according XML-File:

```
<DataTable CMD="R0#0D" Intervall="5" Info="ADC-Daten">
  <DataRec name="Analog Input"      type="UI_16" factor="0.0088644" unit="V" digits="3"  />
  <DataRec name="Digital Input 2"   type="UI_16" factor="0.0088644" unit="V" digits="3"  />
  <DataRec name="Digital Input 1"   type="UI_16" factor="0.0088644" unit="V" digits="3"  />
  <DataRec name="Digital Input 4"   type="UI_16" factor="0.0088644" unit="V" digits="3"  />
  <DataRec name="Digital Input 3"   type="UI_16" factor="0.0088644" unit="V" digits="3"  />
  <DataRec name="Temperature"       type="UI_16" factor="0.000806"  unit="V" digits="3"  />
  <DataRec name="VDDA"              type="UI_16" factor="0.001"     unit="V" digits="3"  />
  <DataRec name="CheckSum"          type="UI_8"  factor="1"        unit="E" digits="0" show="false"/>
</DataTable>
```

Structure definition in C/H-file

```
//! ADC-Results Structure, 5/4/1 Sample Addiert !!, nicht korrigiert mit Abgleichdaten ! RR=1ms
typedef union __attribute__((__packed__))
{
  uint16_t iADCn[ucCHN_COUNT];
  struct  __attribute__((__packed__))
  {
    uint16_t iAin;                  //!< R: Analog Input    [8.8644 mV/E]
    uint16_t iDin2;                 //!< R: DIN2 analog     [8.8644 mV/E]
    uint16_t iDin1;                 //!< R: DIN1 analog     [8.8644 mV/E]
    uint16_t iDin4;                 //!< R: DIN4 analog     [8.8644 mV/E]
    uint16_t iDin3;                 //!< R: DIN3 analog     [8.8644 mV/E]

    uint16_t iTmpS;                 //!< R: Temp-Voltage
    uint16_t iRefV;                 //!< R: VDDA
  }st;
}
ST_ADC;
```

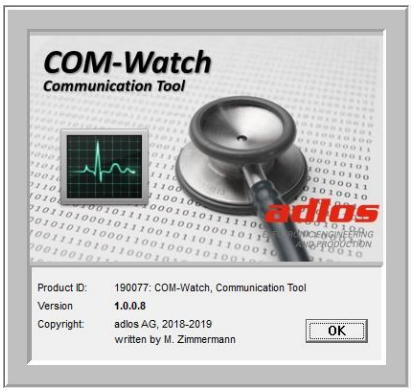
Send call in Firmware

```
    RS232A_SendData((UI_8 *) & stHWCntrl.stADC, sizeof(stHWCntrl.stADC));
    Rs232A_SendTxCS();                                          // Checksumme von Antwort senden
```

Adlos Win32-APPs

adlos offers for it's customers some Helping and Design-In Tools.

3.1 ComWatch Communication Tool (190077)



ComWatch is a helping tool for engineers and technicians to explore device specific parametes, read out tracking data and settings and doing firmware updates.
The software is as it is, and in principle for free for adlos customers, the software is not made for a broad range of standard users, it's made in principle for technical engineers which are used in working w. windows based software and have some understanding of technical things.

http://kannmotion.adlos.com/download/comwatchtool/ComWatchSetup.zip

3.2 KannMOTION API

Adlos offers a windows API (Library) to communicate with our drives. The API enables much shorter implementation of KannMOTION communication with your own Windows based Toolset.

Part number	Short / level	Description	
190073	LEVEL1 API-LLL	LowLevelAbstraction offers RD/WR functions to Com, organizes Checksum and protocol Itself	
190074	LEVEL2 API-HAL	Hardwareabstraction offers data object medeling, means it will take care bout device specifc XML-files	
190080	LEVEL3 API-BAL	BusAbstraction Offers bus data support like CAN	

Download Link coming 2Q/2019