Command Interface Guide Avisaro 2.0 Product Series

"API"

Version / Date 2014-05-28

1 Table of Content

2 THIS DOCUMENT

2.1 LINKS

Please check for the most current update of this document here:

German: www.avisaro.de/de/20-Datenlogger-Dokumente.html

English: http://www.avisaro.com/en/20-data-logger-documents.html

2.2 RELATED DOCUMENTS

Please check for other documents here:

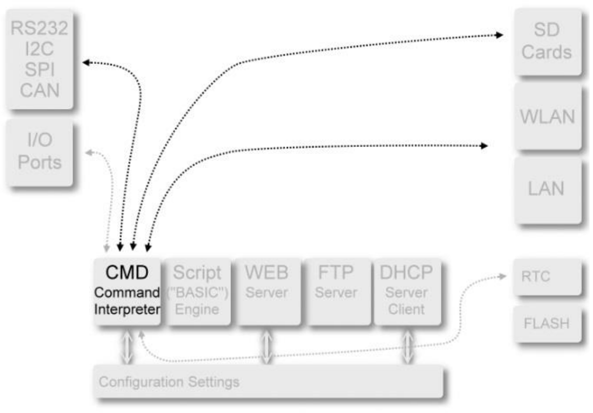
German: www.avisaro.de/de/20-Datenlogger-Dokumente.html

English: http://www.avisaro.com/en/20-data-logger-documents.html

2.3 HISTORY

3 Introduction

Avisaro 2.0 products have a build-in command interface. This command interface allows to control the functions of the product: start a connection, store data into a file and change the configuration. The commands are entered using the Data Interface (RS232, SPI, I2C, ...). The format of the command is either plain ASCII or alternatively a compact binary format.



The Command Interface allows to type and send commands in ASCII format. For example, a RS232 product is connected with a terminal program on a PC, one can communicate by typing in commands and by reading the answer on the terminal. The command interface is available on all physical interfaces (CAN, SPI, I2C, ...).

The Command Interface allows to:

- Open and control TCP or UDP connections
- Send data through those connections
- Open and control files on the SD card
- Store and retreive data from SD card
- Change configuration settings

4 Usage: Command Interface ("API") vrs. Scripting ("Apps")

There are two main methods to operate the Avisaro 2.0 product:

The Command Interface (Mehr more) allows to send commands from an external unit. This unit is typically a SPS control or a micro controller. Most commands can be send in an easy to read ASCII or compact binary format. Commands exists to ...

- Read and write data on SD cards
- Establish connections and share data through WLAN and LAN
- Configure the product and check status

The Scripting Engine (Mehr more) allows to have the Avisaro 2.0 product perform functions on a self sustained basis. The engine is designed to perform small tasks rather than large scale applications. Scripts are stored in internal flash and executed upon power up. Structures exists to ...

- Read and write data on SD cards
- Establish connections and share data through WLAN and LAN
- Parse and reformat data

- Control I/O ports, PWM and analog input
- Do if-then-else, do-loops, for-next, gosub-return,.... structures

Decision matrix:

Application	BASIC Scripting	Command Interface	Details
Send one datastream from A to B through WLAN	√		Ready to use scripts are available. Connection handling is done by Avisaro, user simply sends data.
Send several parallel datastreams from A to B		√	A SPS or micro controller sends or retrieves data to several server. Commands are send to control connections.
Store datastream into file	√		Ready to use scripts are available. File handling is done by Avisaro, user simply sends data.
Poll sensor and store data into file	√		Scripting is powerfull enough to poll a sensor through i.e. rs232, i2c, ..., than format data and store data into file. No aditional controller is needed.
High Speed Data		√	For high performance applications, the "packet commands" in binary form is used.

Note:

Avisaro 2.0 "Box" and "Cube" products are shipped with pre-installed Scripts. Those Scripts can be changed to perform a different funciton or they can be disabled to use the command interface.

Avisaro 2.0 "Modules" are shipped with no Script installed.

5 Interfaces

The Avisaro 2.0 products come with the capability to work with different interfaces and protocolls. Which interface can be used depends widely on the product purchased. Within the "Box" and "Cube" product series, only certain interfaces are routed to the connector.

The Avisaro 2.0 Module has all the interfaces routed to its pins - however due to pin multiplexing, not all interfaces are available at the same time. Be aware, that some interfaces require external components to comply with the signal levels (this is only true for the module, "Box" and "Cube" are "ready-to-use").

Summary:

- RS232: Avisaro supports all the baudrates - up to about 1Mbit - and all the typical settings such as flow control.
- CAN:
- I2C:
- SPI:
- I/O, analog:

5.1 USING RS232, RS485 OR RS422 INTERFACES

The command interface is available on RS232 interface port 1. The second RS232 interface is available within the scripting language. See tl_files/dynamic_dropdown/link.gif here for details on RS232 within scripting.

Setup and configuration

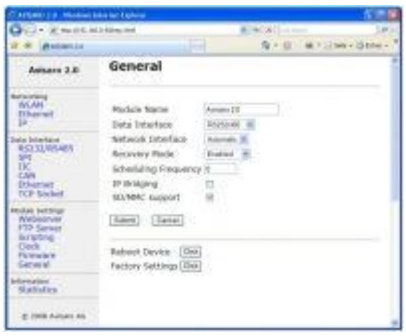
Activating and configuration of RS232 interface (Primary)

The RS232 port 1 is designed to receive user data as well as commands adressed to configure the Avisaro product.

For all Avisaro RS232 Data Logger and Avisaro RS232 WLAN products, the RS232 interface is already activated and operates with default values (see below). Change baudrate and filter settings using the web interface or SD memory card.

If a Avisaro Module product was purchased or after a 'reset to factory settings' command, the RS232 interface is activated by default.

5.1.1 Activating RS232 interface ... via web interface



Module Name: Avisaro 2.0
Data Interface: RS232/485
Network Interface: Automatic
Recovery Mode: Enabled
Scheduling Frequency
IP Bridging
SD/MMC support
Reboot Device
Factory Settings

Module Name

The text entered in this field shows up in the top left corner of the web site. It has no functional meaning, but can be used to distinguish between modules.

Data Interface

Selects the active data interface. This is the main interface for the module. Through this interface the Avisaro device receives commands or sends out messages. Or, through this interface data is send and received to be stored or forwarded wirelessly.

RS232/RS485: Sets primary RS232 or RS485 or RS422 interface

IIC: Interface is set to IIC (= I2C) slave. See Mehr here for I2C master settings

SPI: SPI slave

CAN: Sets primary CAN interface

Ethernet: Sets to raw ethernet interface. For experts only.

TCP Socket: Sets to TCP socket interface

None: No interface is active

File: Outputs are written into a file.

Network Interface

The valid network interface option depend on the type of interface connected to the Avisaro device

WLAN: Only the wireless LAN interface is active (if present)

Ethernet: Only the LAN interface is active (if present)

None: No network interface becomes active (even if present)

Automatic: The network interface is automatically selected. The search is processed in the order 1) WLAN 2) LAN

Both: Both interfaces - WLAN and LAN - are active (if present)

Recovery Mode

There is a special feature to access a Avisaro device through a network interface, even if settings are messed up. See Mehr here for details.

Scheduling Frequency

Avisaro system runs on a multitasking system. This setting defines the time in milliseconds each task is assigned. Setting '0' sets this value to 'dynamic'. The recommended value is '0'.

IP Bridging

If there are two network interfaces, the Avisaro module can work as a bridge between those two. Thus, network traffic is routed from one to the other interface.

SD/MMC Support

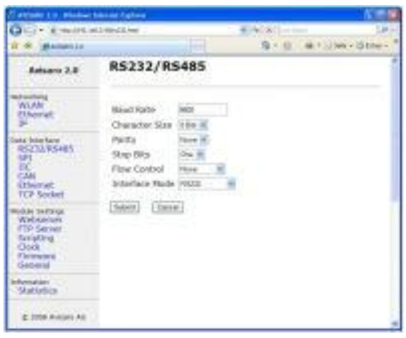
The SD card slot requires periodic processing resources even if no card is inserted. If no SD card slot is present, the 'SD Support' should be disabled.

Reboot Device

This reboots the device. Quite a few changes require a reboot in order to become effective.

Factory Settings

This resets all customer settings to default values. All entries such as selected data interface, WLAN and IP settings are reset. The stored script remains stored, however the automatic execution upon startup is disabled.



RS232/RS485

Baud Rate: 9600
Character Size: 8 Bit
Parity: None
Stop Bits: One
Flow Control: None
Interface Mode: RS232

Configuring RS232 interface

Baud Rate

Valid rates: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800

Character Size

Number of bits: 5, 6, 7, 8

Parity

Parity: odd, even, none

Stop Bits

Stop bits: 1, 2

Flow Control

Flowcontrol: none, Xon/Xoff (=SW), RTS/CTS (=HW)

Interface Mode

RS232: This must be one of the keywords RS485

RS485: In RS485 mode, a transceiver chip must be connected that handles the physical bus. Therefore, the module automatically toggles a control line that most chips need to switch from RX to TX and vice versa. If RS485 is given, the Module drives the DTR control line from LOW to HIGH while sending.

RS485 Inverse: When RS485INV is given, that control line behaves inversely, that is, it goes from HIGH to LOW while sending.

5.1.2 Activating and configuration of RS232 interface 'Port 2' (auxiliary) ... via SD card

The RS232 port 2 is designed to be used in scripting language only. There is no access to the auxiliary port within the command interface.

Working with RS232 data interface

Enter data

The RS232 command interface is well suited to be used with a terminal program. This way it is easy to try commands and see their results. Just as well, a micro controller or a SPS unit can communicate with using the RS232 interface.

Typically, the text (ASCII) version of the commands are used. This makes the commands and the responces readable to the user.

To enter a command, simply type in the command followed by a carriage return and new line:

```
> time?
2009/01/10 15:31:12
>
```

This "time?" command reads out the current time of the module, followed by the prompt. The list of all commands can be found Mehr here.

5.2 Using CAN Interface

Commands are only accepted by the Avisaro device if the script is switched off.

The format is a standard CAN message. As CAN ID use the 49. The command must be written in HEX taking 8 letters per CAN message.

Each command line must be terminated with a CR and Line-feed - in HEX: 0D 0A. It is important that the last line is finalized like that as well. Othwise the command will not be accepted.

5.3 Using I2C interface

The command interface is available on the I2C interface. This interface is not the default interface, thus it needs to be activated once after receiving the module.

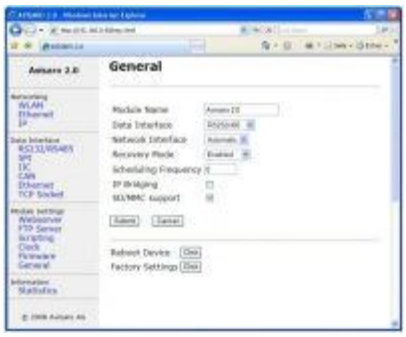
5.3.1 Setup and configuration

Activating and configuration of I2C interface (Slave)

The I2C is designed to receive user data as well as commands adressed to configure the Avisaro product. The interface is working as a slave device.

Activate and configure the I2C interface either through the build-in configuration web site or through SD-Card:

5.3.2 Activating I2C Slave interface ..via Web



Module Name: Avisaro 2.0
Data Interface: RS232/485
Network Interface: Automatic
Recovery Mode: Enabled
Scheduling Frequency: 5
IP Bridging: ☐
SD/MMC support: ☑
Reboot Device
Factory Settings

Module Name

The text entered in this field shows up in the top left corner of the web site. It has no functional meaning, but can be used to distinguish between modules.

Data Interface

Selects the active data interface. This is the main interface for the module. Through this interface the Avisaro device receives commands or sends out messages. Or, through this interface data is send and received to be stored or forwarded wirelessly.

RS232/RS485: Sets primary RS232 or RS485 or RS422 interface

IIC: Interface is set to IIC (= I2C) slave. See Mehr here for I2C master settings

SPI: SPI slave

CAN: Sets primary CAN interface

Ethernet: Sets to raw ethernet interface. For experts only.

TCP Socket: Sets to TCP socket interface

None: No interface is active

File: Outputs are written into a file.

Network Interface

The valid network interface option depend on the type of interface connected to the Avisaro device

WLAN: Only the wireless LAN interface is active (if present)

Ethernet: Only the LAN interface is active (if present)

None: No network interface becomes active (even if present)

Automatic: The network interface is automatically selected. The search is processed in the order 1) WLAN 2) LAN

Both: Both interfaces - WLAN and LAN - are active (if present)

Recovery Mode

There is a special feature to access a Avisaro device through a network interface, even if settings are messed up. See Mehr here for details.

Scheduling Frequency

Avisaro system runs on a multitasking system. This setting defines the time in milliseconds each task is assigned. Setting '0' sets this value to 'dynamic'. The recommended value is '0'.

IP Bridging

If there are two network interfaces, the Avisaro module can work as a bridge between those two. Thus, network traffic is routed from one to the other interface.

SD/MMC Support

The SD card slot requires periodic processing resources even if no card is inserted. If no SD card slot is present, the 'SD Support' should be disabled.

Reboot Device

This reboots the device. Quite a few changes require a reboot in order to become effective.

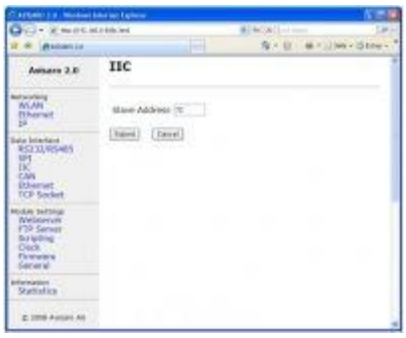
Factory Settings

This resets all customer settings to default values. All entries such as selected data interface, WLAN and IP settings are reset. The stored script remains stored, however the automatic execution upon startup is disabled.

5.3.3 Configuring I2C Slave interface

There is only one setting to be made:

I2C Slave Address: Enter the slave address for the Avisaro unit on the I2C bus. Values are entered in decimal.



Slave Address 70

5.3.4 Activating and configuration of I2C interface (Master) .. via SD Card

The I2C port in Master configuration is only available within Scripting. See Mehr here for details.

Working with I2C data interface

Enter data

For the I2C interface, commands can be entered in text (ASCII) form as well as in packet mode (binary):

Commands send in text (ASCII) mode

To send a command, perform a I2C write to the Avisaro module on the bus (by default it is address 73 decimal). Send the command (like "time?") by sending the ASCII values: 0x74, 0x69, 0x6d, 0x65, 0x3f . The command needs to be terminated with a carriage return: 0x0d, 0x0a . Finish the I2C communication with a I2C 'stop' sequence.

To read the answer, perform a I2C read to the Avisaro module. The module will return 0xFF if no valid answer can be read. Keep on reading until valid text (ASCII) character are read. Finish the I2C read when all characters are read - to be recognized at a 0xFF after the last 0x0d 0x0a.

It is required to read the answer, otherwise the module will be blocked by unread answeres.

Commands send in packet (binary) mode

Typically, the I2C is used in a micro controller environment. The Avisaro module supports a simple packet mode which is well suited for this environment. See Mehr here for more details.

The procedure is to perform a I2C write sending the packet to the Avisaro module. To read the answer (required) perform a I2C read and get the answer - also in packet form.

6 Doing Things – Examples

6.1 Working with TCP connections

This chapter describes how to open and manage a TCP connection using the command interface. For example, a external micro controller opens a TCP channel and sends data. There are a couple of text commands to manage a TCP connection. Whether to use text or binary commands is up to the user - whatever is more comfortable.

See (Mehr here) to get details on how open and manage a TCP connection automatically using scripting. For example, a sensor simply sends data but can't be modified to send commands.

TCP/IP fundamentals

This document assumes you are familiar with TCP/IP fundamentals. For further information on TCP/IP please search the internet or follow the external links:

Wikipedia: TCP/IP (Mehr more)

PDF state diagram (Mehr more)

6.1.1 Open TCP connections

A TCP connection can be established two ways:

1.) Waiting for a incoming connection (Avisaro is TCP server)

Use the text command "LISTEN" (Mehr more) or the binary command "PCMD_NET_LISTENTCP" (Mehr more) to create a 'TCP socket' in listen mode. Specify a internal 'handle' number - this handle number adresses this TCP socket. Also specify a TCP port number.

2.) Connection to a server (Avisaro is TCP client)

Use the text command "CONNECT" (Mehr more) or the binary command "PCMD_NET_CONNTCP" (Mehr more) to actively connect to a TCP server. Specify a internal 'handle' number, the other TCP adress and port number.

Once the TCP socket with its handle number is declared, you can query the module whether the TCP connection was established successfully. There a different strategies to do so:

1) Listen/Connect and Wait: The "LISTEN" and "CONNECT" command offer the optional parameter "WAIT". When this parameter is used, the command returns when the connection was established. Use this option together with the Listen command with care, since the module waits forever with no connection being established. For connect there is a timeout.

2) Checking handle status: Using the text command "SSTAT" (Mehr more) or the binary "PCMD_NET_SSTAT" (Mehr more) one can check the status of the TCP socket. This method is the cleanest, but it requires some parsing of the status responce.

3) Polling with stream command: The command "STREAM" (Mehr more) is usually used to start data transmission. When the connection is not established yet, this command returns an error. It turns out to be pratical to use this command to do two things at a time: start data transmission / receiving when this command returns positive, otherwise wait and try later again.

6.1.2 Sending and receiving data

Once the connection is established, it can be used to send and receive data.

Sending data using text command "STREAM":

The "STREAM" command (Mehr more) redirects inputs to be send to the TCP connection. So all data (RS232, CAN, SPI, I2C, ..) are send to the TCP partner. All data received through the TCP connection can be received using the data connection. It is required to read those data from the Avisaro product - otherwise buffer fill up and the connection stalls.

The STREAM command allows data to be send only over one TCP connection.

To switch back to issue commands, use the Stop Sequence (Mehr more). By default, this is "+++" - but can also be changed. Make sure the stop sequence does not appear in the data stream.

Sending data using binary commands "PCMD_NET_PUTPACKET" and "PCMD_NET_GETPACKET":

The binary send and receive commands are more powerfull since more than one connection can be served and it is easier to switch between commands and data. The drawback is the formatting efford for those commands. Use "PCMD_NET_PUTPACKET" (Mehr more) to send data and the "PCMD_NET_GETPACKET" (Mehr more) to receive data.

6.1.3 Closing TCP/IP connection

When done with sending and receiving data, the TCP connection can be closed.

Closing using text command "CLOSE":

The "CLOSE" command (Mehr more) closes the TCP connection. Use the handle number to specify which connection should be closed.

Closing using binary command "PCMD_FCLOSE":

The binary command "PCMD_FCLOSE" (Mehr more) closes the TCP connection. Use the handle number to specify which connection should be closed.

6.1.4 Example: Listening for incoming TCP connection

This example is using text commands to start listening for an incoming connection, sending some data and finally closing the connection again. This example works for any data interface (RS232, CAN, I2C, SPI, ...):

Send from your application	Send from Avisaro device	Comment
listen 101 23		Set socket with handle number 101 to listen on port 23
	>	
stream 101		Try to establish stream to send data.
	ERR 28 >	This error is expected since connection is not established yet.
stream 101		
	ERR 28 >	... still not established.
stream 101		Oh, good – no error – connection is established
THIS DATA WAS SEND FROM SERVER TO AVISARO		Data is send back ...
	THIS DATA WAS SEND FROM AVISARO TO THE SERVER	... and forth.
+++		Sending stop sequence to return to command mode
	>	
close 101		Close connection
	>	

Some details:

- The ">" is the return prompt of the Avisaro module. Each > is headed by a <cr> <lf>. It can be changed i.e. to "OK" if so desired
- All commands are terminated by <cr> <lf> ("enter") - this is not shown expliciately here

7 DOING THINGS – EXAMPLE FOR FILE HANDLING

Files can be handled just like using a command line shell on a regular computer. There's a DIR command that can be used to list all files in a directory. Simple open, read and write operations are possible by hand or may be invoked by any device that is connected to the I/O interface.

The following is a short tutorial that shows how to read and write files on SD card. If you want to work through it, please create a file on a FAT16/32 formatted card (by using a PC or something like that), name it "hello.txt" and insert it into the SD slot of your Avisaro-Module. The file should contain a single line: "the_quick_brown_fox_jumps_over_the_lazy_dog". You need also a terminal connected to the I/O interface of the Avisaro Module. In case of RS232, you may find this tool useful.

7.1 Reading existing files

Suppose there's a SD-card inserted that has a file named "hello.txt" in its main directory. To open it for reading and get all its content, can simply be done with two commands:

```
open 1 hello.txt
```

```
stream 1
```

After that, the module transmits the entire file content to your terminal:

```
the_quick_brown_fox_jumps_over_the_lazy_dog
```

Please note the 1 before the file name. This is something called a "File Handle". File handles are used to identify and access files after they were opened. You will see that the 1 from here is used in all subsequent file commands.

Often, it is more practical to read files chunk by chunk than getting all that stuff as a whole. This can be done with the READ command. Note that we first must close the file, before we can open it again.

```
close 1
```

```
open 1 hello.txt
```

```
read 1 9
```

After that, you get only the first nine charcters, which are:

```
the_quick
```

To read the next two words, invoke READ again:

```
read 1 10
```

And this is what you get:

```
_brown_fox
```

So to say, calling READ again and again moves the internal "File Pointer" towards the end, until there's nothing more to read. Try it yourself and see what happens. Tip: if ERR33 appears, type ERR?

By the way, the file pointer not only moves automatically on READs. There's a command called POS that you can use to set the file pointer to any position in the file. Try this:

```
pos 1 20
```

```
read 1 10
```

And you will see:

```
jumps_over
```

POS has moved the file pointer to the 20th position, so reading 10 characters from there gives you the output above.

Finally, when we're ready, a file should be closed using the CLOSE command (we already done that before). Closing files frees all ressources associated with them and, in case of writing, ensures that cached data is flushed to disk. So, do it now:

```
close 1
```

7.2 Creating and Writing Files

Now that you know how to open and read files, this part of the tutorial shows you how to create new files and put data into them. First we must create a new file on disk. Let's do this, we create a new file named "myfile.txt":

```
new 1 myfile.txt
```

This creates a new file and also opens it for writing. If you get an ERR27, you possibly forgot to close the file used by the previous section, which also uses file handle #1, type CLOSE 1 and then try again.

Invoke DIR to see what's on your SD card:

```
dir
```

The output should be:

```
myfile.txt 0

hello.txt 43
```

Don't worry if output on your module appears in reverse order, the DIR command reads the file system structures directly and doesn't sort what it finds. It's more essential that you see the new file "myfile.txt" with zero length. That's the file you created just now.

As mentioned before, the file is already open. To write data data into it, simply enter streaming mode:

```
stream 1
```

Everything you type, after this command, is written to the file. Now type the this (only the six characters without return):

```
123456
```

After that type three plus signs:

```
+++
```

As you may notice, immediately after the third + the Module prompt (usually a >) appears to indicate that the module has left streaming mode and entered command mode again. The sequence of three pluses is the module's so-called "stop sequence". It is also used in other situations, where it must be possible to leave data mode.

As a contrast to reading, it is inherently important to close a file after you're done with write operations. If you miss that, it's most likely that not all data is written to disk. So let's do it:

```
close 1
```

Now assume that six bytes, the characters 123456 are written into the file, you may satisfy yourself. Type:

```
dir
```

...and verify that the output looks like:

```
myfile.txt 6

hello.txt 43
```

Myfile.txt contains six bytes, is it true? Ok, then let's append some more data. For this to work, we must open "myfile.txt" again but in this case, we need a special command that opens an existing file for writing and moves the file pointer to the end. The command is named APPD, let's do it:

```
appd 1 myfile.txt
```

The file is opened again. Now we introduce another command that can be used to write data into a file. Differently from streaming mode, this command writes all of its second argument into the filel. Just see how simple it is:

```
write 1 abcdef
```

That's all we need to put the characters "abcdef" at the end of the file. Let's close the file and see the directory:

```
close 1

dir
```

The output now should look like this, "myfile.txt" must be six bytes bigger:

```
myfile.txt 12

hello.txt 43
```

Finally, make sure the file content is really that what you assume. You can easily verify that by showing the entire file (as you learned from the first section):

```
open 1 myfile.txt

stream 1
```

Do you see "123456abcdef", right?

That's all!

8 Doing Things – Example Setting the Cock

How to set the clock

Description

The Avisaro Devices own an integrated clock. For all "Box" and "Cube" products, the RTC is battery backed, thus it keeps the time even if power is disconnected. Delivering the device the time is set. The "Modules" requires external supply to hold time.

With WLAN / LAN

If your device has got an WLAN or LAN interface you can set the time most easily via the administration page.

Enter the adminstration webpage - chose the menu 'clock' and set the time.

Via SD Card

To set the time create a file names autotun.txt and save it at a SD card. The file shall contain the command time in a format as descript here. Press the Enter-key at the end of the line.

TIME requires six arguments separated by spaces in the following order:

Year: 2000...2099

Month: 1...12

Day: 1...31

Hour: 0...23

Minute: 0...59

Second: 0...59

Example

```
TIME 2008 10 20 12 13 14
```

Sets the RTC date to 2008/10/20 and time to 12:13:14

Insert the SD card into the data logger and restart the data logger at the time and second mentioned in the autorun.txt. The data logger will set the clock to the mentioned time.

Please do not forget to remove the file from the SD card afterwards to avoid wrong time setting afterwards.

Via data interface

To set the time via data interface you have to disable the script first. Then the time can be set with the command 'time'.

Afterwards the script must be restarted.

9 DETAILS

9.1 POWER SAVING MODES

Introduction

The Avisaro Modules supports three power saving modes, one for the main controller and two for WLAN unit. All of them can be combined.

Putting the main controller asleep

Send a SLEEP command over the I/O interface to freeze the MCU so that it consumes very little power. The sleep can be time-controlled, this means the MCU awakes after a specific time, or it can sleep forever until it detects a pulse on its wakeup-pin. Please look here for further information.

Switching the WLAN unit off and on

Send WLAN SLEEP NOW over the I/O interface to switch the WLAN unit off. If can later be switched on again by the WLAN AWAKE NOW command. Please look here for further information.

Running WLAN in power save mode

The WLAN unit is able to save energy while being fully functional. To bring the WLAN into power save mode, use either the WLAN command (see here), or activate the checkbox on the web page.

9.2 HANDLES

All resources (files, TCP or UDP connections) are addresses by handles. A handles is a number which is given uniquely to one resource. The numbers are controlled by the user, but have to be from the range:

1..100	file handles
101 …200	TCP handles
201 …300	UDP handles

Example: the command “open 1 text.txt” opens a file for reading. The parameter 1 is the handle. All following operations use the same handle: “close 1” closes the file again. The handle concept is powerful, since parallel operations are possible.

9.3 MULTITASKING

The Avisaro 2.0 products are powered by a multitasking operating system. Thus, functions like scripting and command interface run parallel.

10 Text Commands

10.1 Upload - How it works

In the opposit to scripting the command face is not used in a little program but given directly to the device.

Basically there are three ways of sending the command:

via SD card: Write the commands in the autorun.txt, place it on the SD card and boot the device by connecting it with power

via web browser: call for the page http:\\ [ip-address] \cmd

via interface: if the script is deactivated the commands can be send via interface like RS232, CAN or I2C

10.2 APPD

	APPD
Description	Opens a file for writing and sets the file pointer behind the last byte, therewith, subsequent write operations can append data to the end of the file The first argument is an arbitrary number in the range from 1 to 100, that is used as file handle. The second argument must be the name of an existing file.
Parameters	This command needs 2 arguments 1. A file handle 2. The name of a file
Return value	ERR_OK (0) - if command is accepted ERR_ARGUMENT (4) - if there's a problem with the arguments ERR_ID_USED (27) - if the given file handle is already in use ERR_FILE_OPEN (32) - if the file is already open ERR_FIL_EXHAUSTED (26) - if the system can't allocate a new file control block ERR_FR_XXX (13...25) - on internal file system errors
Example	`APPD 1 hello.txt`
Remarks	This command only exists on modules with storage functionality (such as SD-card or USB frash drive).

10.3 ARP

	ARP
Description	This command can be used to query the internal ARP table and to delete all of its entries. ARP means Address Resolution Protocol, which is a protocol that provides mapping from IP- to Ethernet-Adresses. The ARP? command prints out one or more lines, including the broadcast address. Each line shows six hexadecimal numbers separated by colons, a hyphen and four decimal numbers separated by dots. The part before the hyphen is the Ethernet-Address and that after the hyphen is the corresponding IP-Address. The ARP CLEAR command simply deletes all stored ARP entries, this means, that the module issues an ARP query to get the MAC address of the recipient, before the next IP packet can be sent.
Parameters	There are two version ARP? Lists the ARP table ARP CLEAR Clears the table
Return value	ARP?: The output is always ERR_OK (0). ARP: ERR_OK(0) or ERR_ARGUMENT(4) if the argument is not CLEAR

Example	`>  ARP?`
	`<  ff:ff:ff:ff:ff:ff - 255.255.255.255`
	`   0:c:41:9d:2f:62 - 192.168.0.1`

Remarks	ARP and ARP? are only available on module that have a network interface (such as WLAN or Ethernet)

10.4 BC

	BC
Description	This command can be used to query the last crash information. Like any complex computer system, the Avisaro Module can probably crash due to bugs in the firmware or faulty BASIC scripts. If such breakdown occurs, the module stores crash information into battery powered RAM and re-starts itself. If crash information is available, BC? outputs a line containing seven parts separated by spaces. These are, from left to right: 1. Exception Reason PRE -- Instruction fetch memory abort. The processor tried to execute code at an undefined address. DAT -- Data access memory abort. The processor tries to read from or write to an undefined address. UND -- Undefined instruction. The processor tried to decode an instruction that is not part of the ARM7 instruction set. BRW -- Brown out. Low peak of supply voltage below 2.95V. Processor state when exception happened ARM -- The processor was in ARM state. TMB -- The processor was in THUMB state. 2. Processor mode when exception happened USR -- The processor was in normal "user" mode. FIQ -- The processor was executing a fast interrupt routine. IRQ -- The processor was executing a general interrupt routine. SUP -- The processor was in "supervisor" mode. ABT -- The processor emulates virtual memory. (not used) UND -- The processor emulates code for a co-processor. (not used) SYS -- The processor was in "system" (highest privilege) mode. 3. Date when exception happened 4. Time when exception happened 5. The value of the instruction pointer when exception happend This is a hexadecimal value without leading zeros. 6. The value of the status register when exception happend Also a hex value without leading zeros
Parameters	<none>
Return value	The return value is always ERR_OK (0)

Example	`>  BC?`
	`<  BRW ARM SUP 2008/01/01 22:46:20 13a14 60000013`

Remarks	Exception information is only stored among power on/offs if the module has a battery (usually used to keep the RTC running). When a crash occurs, the module reboots.

10.5 BIND

	BIND
Description	This command can be used to query the internal ARP table and to delete all of its entries. ARP means Address Resolution Protocol, which is a protocol that provides mapping from IP- to Ethernet-Adresses. The ARP? command prints out one or more lines, including the broadcast address. Each line shows six hexadecimal numbers separated by colons, a hyphen and four decimal numbers separated by dots. The part before the hyphen is the Ethernet-Address and that after the hyphen is the corresponding IP-Address. The ARP CLEAR command simply deletes all stored ARP entries, this means, that the module issues an ARP query to get the MAC address of the recipient, before the next IP packet can be sent.
Parameters	There are two version ARP? Lists the ARP table ARP CLEAR Clears the table
Return value	ARP?: The output is always ERR_OK (0). ARP: ERR_OK(0) or ERR_ARGUMENT(4) if the argument is not CLEAR

Example	`>  ARP?`
	`<  ff:ff:ff:ff:ff:ff - 255.255.255.255`
	`   0:c:41:9d:2f:62 - 192.168.0.1`

Remarks	ARP and ARP? are only available on module that have a network interface (such as WLAN or Ethernet)

10.6 CAN / CAN ERRLOG / CAN?

	CAN
Description	Changes settings of the CAN (Controller Area Network) Interface. Settings are stored in NVRAM and are effective after next reboot.
Parameters	CAN has 5 required and one optional argument 1. Baud Rate (decimal value) 2. RX Message ID (hex value) 3. RX ID is extended ID (binary 0 or 1) 4. TX Message ID (hex value) 5. TX ID is extended (binary 0 or 1) 6. Optional: whether proprietary XON/XOFF flow control shall be used (see Remarks section)
Return value	ERR_OK (0) if command is accepted ERR_ARGUMENT (4) if one or more arguments are wrong ERR_PARAMCOUNT (3) if number of arguments is not 5
Example	`> CAN 125000 1fe 0 2ff 1` Sets the CAN interface to 125,000 bits per second, the Message ID where the module listens to the Standard Message ID 0x1fe and the Message ID that the module uses for transmissions to the Extended Message ID 0x2ff
Remarks	To enable flow control set the sixth argument to ON. In this mode, the Avisaro Module sends CAN frames with a single data byte XOFF (19) if its input buffer is nearly full. When input buffer space becomes sufficient again, the module sends frames containing a single XON (17) data byte. This is similar to RS232 software flow control. To switch off proprietary flow control, set the sixth argument to OFF and reboot the module.

	CAN?
Description	With the CAN? command one can query the actual CAN settings including that concerned to filtering, special modes and handling of external tarnsceiver chips. A single line containing all information (9, 11 or 12 separated by spaces is submitted to the active I/O interface.
Parameters	There are two version ARP? Lists the ARP table ARP CLEAR Clears the table
Return value	ARP?: The output is always ERR_OK (0). ARP: ERR_OK(0) or ERR_ARGUMENT(4) if the argument is not CLEAR

Example	`>   CAN?`
	`<   125000 49 STD 49 STD 3 6 OPEN OFF NORMAL OFF NORMAL`
	
	The meaning of all fields from left to right is:
	1. The baud rate as decimal number.
	2. Own RX ID. Message ID that must be used to address the Avisaro Module. This is a hexadecimal number.
	3. Type of own RX ID. This can be either STD or EXT.
	4. Own TX ID. Message ID of outgoing messages from the Avisaro Module. This is a hexadecimal number.
	5. Type of own TX ID. This can be either STD or EXT.
	6. Filter settings: Start ID of filter. This is a hexadecimal number.
	7. Filter settings: End ID of filter. This is a hexadecimal number.
	8. Filter settings: Filter mode. This can be one of OPEN, CLSD, EXT, STD, BOTH. Please see the CANFLT page for details.
	9. Flow control activity. This can be either ON or OFF
	10. Operation mode: This can be LISTEN or NORMAL, where listen means, that the module only listens on the bus but does not actively drive it (does not send frames, ACKs and so on).
	11. Can be one of: OFF, 11, 10, 01 or 00. If this is OFF, the module does not drive the lines for special transceiver chips. Any other output shows how the lines are driven. The fist bit is for STB/Mode0, the second one is for EN/Mode1. For an explanation of those lines, please see the TJA 1041 and similar user manuals.
	12. Can be either NORMAL or ERRBITS. This info shows the state of the error logging facility (see the remarks section above).

Remarks	The 9th and 11th value are not available prior to firmware version 3.52 The 12th value only exists on version 4.39 and above

	CAN ERRLOG
Description	This command enables or disables logging of special event messages. If ERRLOG in ON, events like bus errors, error warnings, overruns and so on are also stored (beside regular frames) into the receive FIFO. Requires Firmware Version 4.39 or higher.
Parameters	<on/off> ON Enable logging of scecial events OFF Disable logging of special events
Remarks	If error logging is enabled, e.g. with CAN ERRLOG ON, certain events from the internal CAN controller circuit cause extra, synthetical frames, to bestored in the receive FIFO. Those frames can be read like regular frames, but to distinguish regular frames from extra frames, the CAN-ID must be checked. All extra frames have an ID of 0xffffffff (or -1 in signed integer format), that is much bigger than any valid regular ID. All extra frames also have an identifier that is the first byte of the 8-bytes payload field. The other members of the payload field hold additional information. The following list shows all possible extra frame IDs, which can be found in the **first byte** of the CAN payload field: 1. Error Warning frame This frame is generated when a receive or transmit operation failed due to some unsuccessful attempts to repeat the operation. 2. Overrun This frame is generated when all internal RX buffers are full and a new CAN frame appears on the bus that cannot be received. 3. Wakeup This frame is generated while the CAN controller is sleeping and bus activity is detected. 4. Error Passive frame This frame is generated if the CAN controller switches from passive to active mode or vice versa. 5. Arbitration lost This frame is generated if the CAN controller loses bus arbitration while attempting to transmit. 6. Bus Error frame This happens when bus anomalies are detected. The **second byte** of the CAN payload field contains the bit position where, in case of a bus error, the error happens. Here's a list of values and their meanings. A bus error occured ... 1. (not used) 2. .. between ID bits 21 and 28 3. .. at the start of the CAN frame 4. .. at the SRTR bit 5. .. at the IDE bit 6. .. between ID bits 18 and 20 7. .. ID bits 13 and 17

8. .. in the CRC sequence
9. .. at Reserved Bit 0
10. .. in the Data Field
11. .. at the Data Length Code
12. .. at the RTR bit
13. .. at Reserved Bit 1
14. .. between ID bits 0 and 4
15. .. between ID bits 5 and 12
16. (not used)
17. .. in the Active Error Flag
18. .. in the intermission bits
19. .. in the "tolerate dominant bits" section
20. (not used)
21. (not used)
22. .. at the Passive Error Flag
23. .. in the Error Delimiter
24. .. in the CRC delimiter
25. .. in the ACK Slot
26. .. at the End of Frame
27. .. in the ACK Delimiter
28. .. at the Overload Flag

The **third byte** of the CAN payload field contains the direction, that means, RX or TX of the bus error. These two values are possible:

0. Error during transmission
1. Error while receiving

The **fourth byte** of the CAN payload field contains the type of bus error that recently happened, This can be:

0. Bit Error
1. Form Error
2. Stuff Error
3. Other Error

The **fifth byte** of the CAN payload field contains the bit position from beginning of the frame, when the CAN controller loses bus arbitration. Possible values are:

0 ... 10	Arbitration lost in the standard identifier
11	Arbitaration lost in the standard-RTR (or extended-SRR) bit
12	Arbitration lost in IDE bit
13 ... 30	Arbitration lost in the second part of ID (Extended frames only)
31	Arbitration lost in RTR bit of extended frame.

The **sixth byte** contains the current RX error counter.

The **seventh byte** contains the current TX error counter.
Every extra frame also has valid timestamps, just like regular frames. All members not mnetioned here are not used and filled with zeros.

10.7 CANEXT

	CANEXT
Description	This command can be used to change advanced settings of the CAN interface of the Module. CANEXT requires one or three arguments. To take effect, the module must be restarted.
Parameters	CANEXT requires one or three arguments. 1. Operating mode: one of the words LISTEN or NORMAL. In LISTEN mode, the module passively listens on the bus, that is, it can only receive but never sends and never causes any bus activity (including ACKs and error frames). 2. Optional: 0 or 1. Defines how control lines of a special CAN transceiver chip are driven. If this is 0 and argument #3 is 0, both control lines are driven low. If this is 0 and argument #3 is 1, pin 5 (on the Avisro Module) is driven low and pin 6 is driven high. If this is 1 and argument #3 is 0, pin 5 is driven high and pin 6 is driven low. If both are 1, both pins are driven high. r. Optional (but required if argument #2 is present): can be 0 or 1 If the last two arguments are omitted and after a restart, the control lines are not driven furthermore.
Return value	ERR_OK (0) if input contains no errors ERR_ARGUMENT (4) If one or more arguments were rejected

Example	`> CANEXT LISTEN`
	(Swiches the Module to listen mode)
	`> CANEXT NORMAL 0 1`
	(Normal mode (transmit and receive). Pin 6 will be high and pin 5 will be low)

Remarks	When using a standard CAN transceiver and for normal participation on the bus, it is not necessary to use this command. To switch back to normal mode, simply send CANEXT NORMAL and reboot the module.

10.8 CANFLT

	CANFLT
Description	This command exists to change properties of filtering of the CAN (Controller Area Network) interface. CANFLT awaits three or four arguments. If a fourth argument is given, and it is the keyword "NOW", settings are applied immediately but are not stored. Otherwise, if there's no such fourth argument, the settings are stored into internal Flash memory and effective after next reboot. In any case, the meanings of the arguments are...
Parameters	There are two version 1. First Filter ID This is the first message ID that passes the filter. This must be a hexadecimal number. All messages below this ID are discarded. 2. Last Filter ID This is the last message ID that passes the filter. This must be a hexadecimal number. All messages above this ID are discarded. 3. Filter Mode This must be one of the keywords OPEN, CLSD, EXT, STD, BOTH. See the following list for filter modes. • OPEN : The filter is completely open. All CAN messages are accepted regardless of other filter settings. Use this configuration only in low bandwidth environments. All messages are stored in the receive FIFO of the module. • CLSD : The filter is closed. No message IDs are accepted except the one that addresses the module itself. See also the CAN command. • EXT : The first and last filter IDs are valid for extended frames only, that is, all standard messages are discarded (except the own ID) and extended IDs are filtered using the given filter range. • STD : The first and last filter IDs are valid for standard frames only. All extended messages are discarded (except the own ID) and standard IDs are filtered by the given filter range. • BOTH : The first and last filter IDs are valid for both, standard and extended frames. All messages in range pass the filter, regardless if they are extended or standard frames.
Return value	ERR_OK (0) if all arguments were accepted. ERR_ARGUMENT (4) if any argument didn't match or the first filter ID was greater than the last filter ID.
Example	`>    CANFLT 1f0 200 EXT NOW` (Sets the filter range to 0x1f0..0x200 for extended frame, thus, all 17 message ID will be passed by the filter: All standard frames are blocked and the settings will be applied immediately without being stored into Flash memory:)
Remarks	The filters are embedded in hardware. Using filters that only let pass desired CAN-frames reduces the interrupt load of the CPU.

10.9 CLOSE

	CLOSE
Description	This command closes files and network connections. For files that are open for writing, CLOSE flushes all buffers to disk before it closes the them. TCP connections are closed gracefully. UDP channels are inactivated. CLOSE takes one argument that can be either a file/network handle or the keyword ALL. In case of ALL, all open files are closed. Network handles are not affected by ALL.
Parameters	An open handle or ALL to close all files
Return value	ERR_NOT_OPEN (28) if the specified handle is not an open file or network connection. ERR_ARGUMENT (4) if the argument was rejected. File System errors (13...25) if the FS could not close the file. ERR_UNSPEC (7) on internal errors (very unlikely).
Example	> `CLOSE 101` Closes TCP connection with the handle number 101. > `CLOSE 13` Closes an open file with the handle number 13 > `CLOSE ALL` Closes all open file but keeps network connections open
Remarks	An application should read all receive buffers of a network connection until they're empty, before a CLOSE command is performed. Otherwise, all buffered data will be lost.

10.10 CMDS

	CMDS
Description	This commands prints out all available commands (including itself).
Parameters	This command doesn't need any arguments.
Return value	ERR_OK (0) (always)
Example	> `CMDS?`
Remarks	CMDS? exists just for debugging purposes. It is not very useful in normal situations.

10.11 CONNECT

	CONNECT
Description	This command initiates a TCP connection to a remote host. CONNECT needs at least an IP address and a port number to establish a TCP connection. A third argument (tx_delay in milliseconds) can be used for streaming mode to collect data until tx_delay elapses. A fourth argument, which is the keyword WAIT, causes the command machine to block until connection is established or timed out.
Parameters	CONNECT needs at least 3 and can have 1 or 2 additional arguments 1. Handle number This can be any number from 101 to 200. If CONNECT succeeds, that number can be used in subsequent calls to other TCP-related commands. 2. IP Address The IP address of the remote machine. 3. Port number A port number where the remote service is listening. 4. TX timeout (optional) A tx_delay value (time in milliseconds) used for TCP streaming. 5. Blocking / Non Blocking (optional) The keyword WAIT
Return value	ERR_ARGUMENT (4) if one of the arguments is invalid ERR_OK (0) if the command is accepted ERR_NOCONN (38) if a connection could not be established
Example	> `CONNECT 101 192.168.0.233 80` Connects to host 192.168.0.233 on port 80. > `CONNECT 101 192.168.0.233 80 1000` Connects to host 192.168.0.233 on port 80 and sets the time-out for streaming mode to 1 second. > `CONNECT 101 192.168.0.233 80 100 WAIT` Connects to host 192.168.0.233 on port 80, sets the time-out for streaming mode to 100 ms and waits until connection established or timed out. he filters are
Remarks	CONNECT only works with modules that have a network interface.

10.12 CSTAT?

	CSTAT?
Description	This command prints out statistics counters of the CAN (Controller Area Network) interface. The output consist of five decimal numbers separated by whitespace (0x20) characters. From left to right: 1. RX_OK: Number of successfully received and buffered CAN frames. 2. RX_LOST: Is incremented when the input FIFO has not enough room to store a received CAN Frame, so that frame must be thrown away. 3. TX_OK: Number of successfully transmitted CAN frames. 4. TX_FAILED: Is incremented when a frame could not be transmitted because the CAN interface is temporarily busy. 5. BUS_ERRORS: Is incremented on bus errors. e.g. there's no other node that acknowledge our retransmissions.
Parameters	CSTAT? doesn't need any arguments
Return value	ERR_OK (0) - Always
Example	`>     CSTAT?` outputs the following: 411 0 17 1 4
Remarks	The CSTAT? command is only meaningful if the CAN interface is active. This command only prints statistics counters for the primary CAN interface

10.13 DEL

	DEL
Description	With the DEL command, existing files can be removed. DEL requires a single argument which must be the full path to the file. A path, in this terms, is a composition of folder names and the file name separated by slashes. To delete a file in the root directory, only the file name is required. The file must not be open for DEL to succeed.
Parameters	Path name and File name
Return value	ERR_OK (0) if the file was deleted successfully. ERR_FILE_OPEN (32) if an attempt was made to delete an open file. ERR_FR_XX (13...25) file system error if something's gone wrong while trying to delete the file.
Example	> `DEL test.txt` Removes the file "test.txt" that is located in the root folder > `DEL foo/bar/test2.txt` Removes the file "test2.txt" that resides in the folder "bar" which is a sub folder of "foo"
Remarks	DEL is only useful on modules with mass storage enabled (SD cards or USB sticks). DEL only works on files that are currently not in use.

10.14 DIR

	DIR
Description	The DIR command can be used to show all files of a folder of the SD card. If no argument is given, DIR prints the files of the root directory. If an argument is attached, it must be the path of the folder which files should be displayed. Nested folder names must be separated by slashes. If the argument is a single slash only, the root directory is displayed as there were no argument. For every entry in the specified directory the output format is: **1.** 14 characters file or directory name in 8.3 FAT format. Long file names are not supported. If a file name is shorter it is filled up with space characters (0x20). **2.** In case of files, a decimal number that is the file size. In case of folders, the five characters. **3.** every entry is terminated by a CR/LF sequence (0x0d, 0x0a).
Parameters	
Return value	ERR_OK (0) if directory was read without error. ER_FS_xx (13...25) file system error if anything's gone wrong while accessing the disk.
Example	DIR Shows the content of the root directory DIR dir1/dir2 Shows the content of dir2 which is a sub directory of dir1 which resides in the root directory
Remarks	DIR only makes sense on modules that have some kind of mass storage like SD-Cards or USB-sticks. The output of DIR is somewhat different from the DOS-equivalent.

10.15 DNS

	DNS
Description	This command issues a name server query. It needs a single argument which is the domain name. For this to work, the Avisrao module must be connected to the internet and the name server must be configured as well.
Parameters	
Return value	ERR_OK (0) if the DNS query succeded. Also the IP address of the given domain name will be printed to the command line. ERR_NO_DATA (8) if no answer arrived.
Example	DNS www.yahoo.com
Remarks	IP address output, if successful, is of the well-known dotted format.

10.16 ECHO

	ECHO
Description	The ECHO command simply sends back its argument. This command might by useful to test the I/O interface connection.
Parameters	
Return value	ERR_OK (0) Always.
Example	ECHO Hello *The module sends "hello" back to the user*
Remarks	Output is always sent to the active I/O interface, also if invoked from the CMD page of the web interface.

10.17 ERR?

	ERR?
Description	With the ERR? command, one can query the global error state, that is, a variable that changes whenever a command is executed. If a command fails, the module sends a message "ERR x" to the I/O interface, where x is the actual error code, indicating that something went wrong.ERR? without an argument shows the textual representation of this error code. When an argument is given, ERR? interprets that argument as error code and shows its textual representation. After ERR? was called, the clobal error state is set to ERR_OK.
Arguments	None or an error number which should be displayed as text.
Return value	ERR_OK(0) Always. ERR? resets the global error variable.
Example	ERR? yields to (usually): I AM OK ERR? 4 yields to: WRONG ARGUMENT
Remarks	None, because this command is much too simple.Wink

10.18 ERRORS?

	ERRORS?
Description	This command prints out all system-known error numbers and their textual representations.
Parameters	None.
Return value	ERR_OK (0) Always.
Example	ERRORS? Outputs the following: (0) I AM OK (1) COMMAND DOES NOT EXIST (2) UNKNOWN FRAME TYPE (3) ARGUMENT COUNT MISMATCH, (4) WRONG ARGUMENT (5) WRONG SIZE (6) CRC CHECK FAILED (7) UNSPECIFIED ERROR, (8) NO DATA (9) NO DISK (10) INVALID HANDLE (11) TRUNCATED (12) REJECTED (13) FS NOT READY (14) FS NO FILE (15) FS NO PATH (16) FS INVALID NAME (17) FS INVALID DRIVE (18) FS ACCESS DENIED (19) FS FILE EXISTS (20) FS R/W ERROR (21) FS WRITE PROTECTED (22) FS NOT ENABLED (23) FS NO FILE SYSTEM (24) FS INVALID OBJECT (25) GENERAL FS ERROR (26) OUT OF RESSOURCES (27) ID IN USE (28) NOT OPEN (29) NO READ ACCESS (30) NO WRITE ACCESS (31) TOO MUCH BYTES (32) ALREADY OPEN (33) END OF FILE (34) DISK FULL (35) NO FW IMAGE (36) TASK ALREADY ALIVE (37) TASK NOT RUNNING

	(38) NET CONNECTION FAILED (39) NET DOWN
Remarks	This command is for informational purpose. The error list might grow in future releases.

10.19 ETH

	ETH
Description	This command sets the Ethernet address of the ENC28J60 chip. ETH requires a single argument which is the new Ethernet address of the ENC28J60. The argument must a 12-digits hexadecimal number. The new address will be effective after next reboot.
Parameters	
Return value	ERR_OK (0) if the argument was accepted. ERR_ARGUMENT (4) if the argument was not a valid 12-digits hex number. ERR_REJECTED (12) if the argument has bit 0 set. Bit 0 is reserved for multicast addressing.
Example	ETH 0c37e3771d66 Sets the local ethernet of the ENC28J60 to 0c37e3771d66
Remarks	This command only makes sense on modules that are equipped with an Ethernet interface. WLAN interfaces have their own factory-assigned addresses which cannot be changed.

10.20 ETH?

	ETH?
Description	With ETH? one can query the current Ethernet address of the ENC28J60 chip and also some statistical values. The output consists of seven values separated by CR/LF (0x0d, 0x0a): 1. The ethernet address of the interface (assigned by the ETH command) 2. Number of received packets 3. Number of unsuccessful RX attempts (dropped packets) 4. Number of transmitted packets 5. Number of TX failures 6. Number of internal resets (rarely needed to keep the net alive) 7. The connection status, can be either NC (not connected) or CONN (connected)
Parameters	
Return value	ERR_OK (0) always.
Example	ETH? Example output can be: 0c37e3771d66 1255 1 6340 0 0 CONN
Remarks	This command is only available if the module has an Ethernet interface.

10.21 FSTAT

	FSTAT
Description	The FSTAT? command can be used to query file- and disk related information. FSTAT? can be called with zero or one argument. If no argument is given, FSTAT? shows the disk parameters, that is, file system, media size and number of free Kbytes.
Parameters	No Argument: FSTAT? shows general disk information File or directory name : FSTAT? shows information related to the given object
Return value	When mass storage device is not available or damaged: NO DISK When called without argument and disk is present: xxx SIZE:yyy FREE:zzz, where xxx is the file system type (FAT12, FAT16, FAT23, RAW), yyy is the total size and zzz is the size of free bytes. When called with a file name: filename size date time, attributes (separated by spaces). The attributes field can contain the characters R(read only), H(hidden), S(system), A(archive bit set) When called with a directory name: dirname, 0, date, time, D in contrast to the file output, the first value is always 0 and the last is always D(directory).
Example	FSTAT?

	Example output might be: FAT16 SIZE: 1981024 FREE:1955232 FSTAT? autorun.txt When autorun.txt is a file, example output is: autorun.txt 17 2010/10/04 18:03:16 A FSTAT? mydir When mydir is a directory, example output is: mydir 0 2010/04/17 13:21:18 D
Remarks	Only modules with some kind of mass storage (SD-Card, USB-Stick) support this command.

10.22 FSYNC

	FSYNC
Description	The FSYNC command can be used to specify a time-out condition in order to flush file caches and update the FAT structure. if FSYNC is used, and a file is open for writing, all cached data will be written to disk, and corresponding FAT entries are written periodically. This feature helps to minimize data loss under rough conditions, where power failures can occur. FSYNC requires one argument that is the "flush time" in milliseconds. If 0 is given, the flush feature is switched off. The argument of FSYNC is stored in internal NVRAM and is effective after next reboot. FSYNC globally applies to all files that are open for writing. There's no per-file FSYNC feature. To retrieve the stored FSYNC argument, simply invoke FSYNC?
Parameters	
Return value	ERR_OK (0) Always. Invalid inputs (such as letters) are interpreted as zero.
Example	FSYNC 1000 All files are flushed within a period of one second FSYNC? Shows current FSYNC value: 1000
Remarks	Only devices with mass storage such as SD-Cards or USB sticks support this command.

10.23 FTP

	FTP
Description	Description This command exists to configure the internal FTP server. FTP requires six arguments in the following order: 1. Protocol Port This is the port number that an FTP client must use to connect to this server. Usually port 21. 2. Data Port This is the port number used for data transfers of this FTP server. 3. User Name Login name to use this FTP service. 4. Password Password needed to use this FTP service. 5. Enabled Use one of the words ON or OFF. ON enables the server, OFF disables it. 6. Timeout A value in seconds while the server keeps the connection open event if it is inactive.
Parameters	
Return value	ERR_LENGTH (5) if username or password is too long. ERR_ARGUMENT (4) if other arguments do not match. ERR_OK (0) if command was accepted.
Example	FTP 21 12345 mama roach ON 120 This enables the FTP server on port 21, using 12345 as data port. Users must logon using mama/roach and the server automatically cancels connections after 2 minutes of inactivity
Remarks	FTP is only possible on devices that have some kind of network interface (Ethernet or WLAN). Per default (factory settings), the FTP server is disabled.

10.24 GSM

	GSM
Description	This command can be used to change settings that are relevant for GSM functionality. The local GSM client can also be startet and stopped by this command and AT and USSD queries can be executed. The command requires always two arguments. The following options are possible: GSM CMD <...> This executes either an USSD query or an AT command. If the first character in the third argument is an asterisk (*), the third argument is interpreted as USSD

	query. Otherwise, if it begins with 'a' ore 'A' followed by 't' or 'T', it is interpreted as AT-command. Any argument that doesn't fit these requirements is rejected. GSM NET <START|STOP> This forces a startup or shutdown of the local GSM client regardless of dial-in policy. However, if dial-in policy is set to PERMANENT, the network will automatically reconnect after a GSM NET STOP. GSM NET START will fail if the module is not configured to use GSM as networking interface. Please see the NET command. GSM APN <...> This sets the APN (Access Point Name) required by GSM/GPRS networks. Contact your GSM provider for your APN or seek for it here. GSM PIN <...> This sets the PIN, usually a 4-digit number, to unlock your SIM card used by the GSM modem. You should get this number from your GSM provider. Often it can be found on the package that contains the SIM card. GSM DIAL <...> This sets "Dial String", that is something like a telephone number to attach to the GSM/GPRS network. Often this string is "*99#" (without the quotes). GSM PASS <...> This sets PPP (Pomt-to-Point Protocol) password used for access to the GSM/GPRS network. Contact your GSM provider for the password or seek for it here. GSM USER <...> This sets PPP (Pomt-to-Point Protocol) user name used for access to the GSM/GPRS network. Contact your GSM provider for the user name or seek for it here. GSM POLICY <AUTO|PERM> This sets the dial-in policy. There are currently two exclusive options defined: PERM and AUTO. If GSM POLICY PERM is given, the module aggressively tries to reconnect whenever the conenction is lost. On GSM POLICY AUTO, the module connects 'on demand', that is, it connects to the GSM network when any local socket is open and disconnects when the last socket is closed.
Parameters	
Return value	ERR_NET_DOWN (39) if GSM NET START failded. ERR_ARGUMENT (4) if one or more arguments do not match. ERR_REJECTED (12) if GSM CMD <...> was called with invalid input. ERR_OK (0) if command was accepted.
Example	GSM CMD AT+CPIN? This shows the status of the modem regarding to a SIM card.
Remarks	This command is available on firmware version 6.05 and above. There must be a GSM modem attached to the module.

10.25 GSM?

	GSM?
Description	This command simpy shows the settings of the local GSM client separated by CR/LFs.
Parameters	
Return value	
Example	pinternet.interkom.de 9778 *99# PPPUSER PPPASS PERM This means, from top to bottom: APN, PIN, Dial-String, PPP Username, PPP Password, Dial-In Policy
Remarks	

10.26 HTTP

	HTTP
Description	This command can be used to set up the internal web server which can be used to communicate with the module and configure the module over any web browser. HTTP requires one or two arguments which are listed below: 1. A single argument, either ON or OFF This enables(ON) or disables(OFF) the HTTP server. 2. Two arguments, USER xxxx This sets the user name for HTTP authentication. You must enter this name in the login box of the browser when connecting to the module. 3. Two arguments, PASS xxxx This sets the password for HTTP authentication. You must enter this name in the login box of the browser when connecting to the module. 4. Two arguments, PORT xxxx This is the port number where the server listens for connections. Usually use PORT 80 for standard HTTP, but you can use any other port as well.
Parameters	
Return value	ERR_LENGTH (5) if username or password is too long. ERR_ARGUMENT (4) if other arguments do not match. ERR_OK (0) if command was accepted.
Example	`HTTP USER paparoach`

	This configures the user name to be "paparoach".
Remarks	The internal HTTP server supplies the web interface for a module's configuration. It is not intended to run user-specific web pages. Per default (factory settings) the server is enabled on port 80, user is admin and password 1234.

10.27 HTTP?

	HTTP?
Description	This command simpy show the settings of the web server as a single line. For example: admin 1234 80 ON This means: the server is enabled on port 80, user is admin and password 1234.
Parameters	
Return value	
Example	
Remarks	

10.28 ICC

	I2C
Description	Changes the I2C slave address of the I2C I/O interface.
Parameters	Just a single decimal value between 1 and 127 (both numbers inclusively).
Return value	ERR_OK (0) if input was valid. ERR_ARGUMENT (4) if input was not in range
Example	I2C 119 This sets the module's I2C slave address to 119
Remarks	The default address (factory settings) of any new module is 73. This command has no meaning if the I2C I/O interface is not in use. This command does not affect the secondary I2C interface. The new address is active after next reboot.

10.29 I2C?

	I2C
Description	Simply prints out the I2C settings. This is currently a single number (only the slave address).
Parameters	
Return value	
Example	
Remarks	

10.30 IP

	IP
Description	The IP command exists to change settings of the integrated IP protocol stack. IP requires two or three arguments. The first argument is a refinement, that is, "what" should be done and the second argument is the actual value.The following list shows the capabilities of the IP command: IP LOCAL xxx.xxx.xxx.xxx This sets the local IP address (the address of the module itself). The second argument must be a valid IP address in standard dotted-decimal notation. IP GW xxx.xxx.xxx.xxx This tells the module where the IP gateway can be reached. All IP packets that doesn't match the subnet mask are send to the gateway which (hopefully) routes them into another net. The second argument must be a valid IP address in standard dotted-decimal notation. IP MASK xxx.xxx.xxx.xxx This sets the subnet mask that the module uses to decide if packets must be sent to the gateway machine. The second argument must be a valid IP address in standard dotted-decimal notation. IP DNS xxx.xxx.xxx.xxx To map domain names to IP addresses, the module needs the address of a machine that is able to respond to DNS queries. The second argument must be a valid IP address in standard dotted-decimal notation. IP DHCP OFF | AUTO | AUTOCL | CLIENT | SERVER Use IP DHCP CLIENT to enable automatic IP address assignment of the Avisaro module by a DHCP server.

	If you want the Avisaro Module to be itself a DHCP server, invoke IP DHCP SERVER. DHCP server functionality of Avisaro Modules is limited to some extend, because it only offers addresses in the 192.168.0.x range without keeping any track of the clients. The IP DHCP AUTO function can be used for easy auto-configuration of unconfigured modules. In this mode the module enables its very simple DHCP server, when it is in default mode. Default mode means that the SSID must be "avisaro" and the WLAN must be in ad-hoc mode. Any other mode disables the DHCP server automatically until the module is brought back into default mode. IP DHCP AUTOCL is a mix mode of AUTO and CLIENT, because the module behaves like IP DHCP AUTO is active in default mode, but enables its DHCP client in configured mode. To disable anly DHCP functionallity use IP DHCP OFF. IP ALIVE xxx This sets or resets the global keep-alive timeout value which counts seconds. If the second argument is a decimal value greater than zero, all connected TCP sockets on all network interfaces will send keep-alive packets if they are inactive for the given time. If the second argument is zero, keep-alives are globally switched off.
Parameters	
Return value	ERR_OK (0) if the command was accepted. ERR_ARGUMENT (4) if one or more arguments didn't match. ERR_REJECTED (12) if both network interfaces are enabled but the third argument was missing. See (Mehr here) for complete list of error codes.
Example	IP LOCAL 192.168.0.233 IP GW 192.168.0.1 IP MASK 255.255.255.0 IP DNS 192.168.0.1 IP DHCP OFF IP ALIVE 10 This is a complete configuration sequence for interface 0 (WLAN), if Ethernet is also enabled. In addition, the global keep-alive timeout is set to 10 seconds
Remarks	If only one network interface is active, either WLAN or ethernet, the above commands change settings for only that interface. If both interfaces, WLAN and Ethernet, are in use simultaneously a third argument is needed which must be 0 (for WLAN) and 1 (for Ethernet). The only exception is the IP ALIVE command which never needs a third argument because IP ALIVE affects all network interfaces.

10.31 IP?

	IP?
Description	With the IP? command, all IP-related settings and values can be requested. IP? requires either zero or one argument which must be either 0 or 1. There are two sets of configuration entries, one for WLAN and one for Ethernet that can be queried. If 0 is given as argument, IP? shows all WLAN related settings. If 1 is given, then the Ethernet-related IP settings will be shown. Without an argument, IP? prints the settings of the currently active network interface. An argument is mandatory if both network interfaces are used simultaneously. Each output shows two blocks of IP addresses. The first block contains values stored in Flash memory while the second one shows the actually used values. These can differ from the stored settings if dynamic configuration over DHCP is enabled. IP? prints out IP settings line by line in the following order. 1. NVRAM-stored: Local IP address. 2. NVRAM-stored: Subnet mask. 3. NVRAM-stored: Gateway address. 4. NVRAM-stored: Nameserver address. 5. DHCP flag. This can be of of OFF, AUTO, AUTOCL, CLIENT, SERVER, indicating if dynamic configuration with DHCP is enabled or not. 6. Currently active: Local IP address. 7. Currently active: Subnet mask. 8. Currently active: Gateway address. 9. Currently active: Nameserver address. 10. Keep-alive timeout. This is the global TCP keep-alive timeout value.
Parameters	
Return value	
Example	IP? 0 Prints out something like that 192.168.0.133 255.255.255.0 192.168.0.1 192.168.0.1 OFF 192.168.0.133 255.255.255.0 192.168.0.1 192.168.0.1 10
Remarks	

10.32 LIST

	LIST
Description	The LIST command can be used to print out a stored BASIC script. Simply type LIST at the command line (without any arguments) and the BASIC script is sent to the currently selected I/O interface.
Parameters	
Return value	ERR_OK (0) If the script source code is visible. ERR_REJECTED (12) If the module contains a pre-compiled code.
Example	LIST Shows what is stored as BASIC script
Remarks	There's no way to make pre-compiled code visible. This behaviour is by design.

10.33 LISTEN

	LISTEN
Description	The LISTEN command is used to open a TCP port for incoming connections. This allows remote clients to contact am Avisaro Module over TCP/IP. Listen requires two, three or four arguments.
Parameters	The arguments are in the following order: 1. Handle number This can be any number from 101 to 200. If LISTEN succeeds, that number can be used in subsequent calls to other TCP-related commands. 2. Listen Port A port number that is armed to satisfy connection requests. Can be any number in the range from 0 to 65535. 3. TX Delay (Optional) A value in milliseconds that specifies the delay for outgoing packets. This argument is only valid in streaming mode. Packets are kept as long as they're too small or tx delay expires. 4. Wait until connection established (Optional) If this argument is given, and it is exactly the word "WAIT", the command interface will block until a client has successfully established a connection.
Return value	ERR_OK (0) if everything works as expected. The socket is then in listen state. ERR_ARGUMENT (4) if one ore more arguments failed validation. ERR_NOCONN (38) if, for any reason, the socket could not enter listen state.
Example	LISTEN 101 12345 Puts socket #101 into listen mode on TCP port 12345. Arguments 3 and 4 are omitted.

Remarks	If the 4.th argument (WAIT) is used and no client connects to the module, the command interface remains frozen until the module restarts or the socket is closed by other means (e.g. over the web interface).

10.34 LOAD

	LOAD
Description	LOAD loads a new BASIC skript into Flash memory of the module. LOAD can be called with zero or one argument. If no argument is given, LOAD reads characters from the currently selected I/O interface and stores them into BASIC's Flash memory area. In this case, if LOAD founds a Stop Sequence (usually three successive pluses), transfer is complete. If an argument is given, it must be the name of an existing file on SD card which contains the source code of a BASIC script.
Parameters	None or an existing file on disk (SD-Card or USB stick)
Return value	ERR_OK (0) on success. ERR_FR_xx (13...25) if something's gone wrong while accessing the media. ERR_TOO_MUCH (31) if the the source code is too big.
Example	LOAD hello.bas Loads the file "hello.bas" into flash memory for execution by the scripting subsystem
Remarks	Any type of file can loaded by LOAD, LOAD does no validation except for the file size. Actually supported files are BASIC skripts as ASCII text and pre-compiled BASIC programs.

10.35 LOADFW

	LOADFW
Description	This command can be used to tranfer a new firmware image from mass storage card into Flash memory which can then be used to re-program the MCU. The command needs one or two arguments. The first argument is the file name (complete path) of a firmware image that resides on the media. The second argument is an optional argument for internal use only. It can be MV1 or MV2. If the second argument is given, certain parts of the firmware can be changed without affecting the main code.
Parameters	1. File name of the new firmware 2. (Optional) MV1 or MV2, not intended for customer use. Please do not use!
Return value	ERR_OK (0) if image is transferred successfully into Flash memory. ERR_TOO_MUCH (31) if image file is too big. ERR_FR_NOT_READY(13) .... ERR_FS_UNKNOWN(25) file system error if something's gone wrong during access of the file
Example	LOADFW v3_32.bin Loads the file v3_32.bin into Flash memory for later re-programming of the MCU
Remarks	Never invoke PROGFW if LOADFW fails. Otherwise the module might be seriously damaged!

10.36 MKDIR

	MKDIR
Description	The MKDIR command's purpose is to create new folders on the SD card. MKDIR requires one argument that is the full path to the new folder. Nested folder names must be separated by slashes. A folder in the root directory does not need any slashes
Parameters	Name and path of new folder that should be created.
Return value	ERR_OK (0) if a new folder was created successfully. ERR_FR_XX (13...25) file system error if anything's gone wrong while trying to create the folder.
Example	MKDIR foo Creates a new folder named "foo" in the root directory MKDIR foo/bar

	Creates a new folder named "bar" that resides in folder "foo"
Remarks	This command only exists on devices that have some kind of mass storage (such as SD-Card or USB stick).

10.37 MOVE

	MOVE
Description	The MOVE command can be used to move files between directories and also to rename files.
Parameters	MOVE requires two arguments, the first one is the current path and name of the file and the second one is the new path/name. MOVE is also able to move or rename directories.
Return value	ERR_OK (0) f the file or directory was moved or renamed successfully. ERR_FR_XX (13...25) File system error if anything's gone wrong while trying to move or rename a file or directory. ERR_FILE_OPEN (32) If an attempt was made to move or rename a file that is open for reading or writing.
Example	MOVE subdir/test.txt hello.txt Moves the file "test.txt" from the directory "subdir" to the root directory and renames it to "hello.txt" MOVE before after Renames a file or directory in the root directory from "before" to "after"
Remarks	New since version 4.57! Files that are currently open cannot be moved. Do not move or rename directories which contain files that are open for reading or writing.

10.38 NAME

	NAME
Description	Sets a new module name. By default the name is "Avisaro 2.0". Changing this name might be useful in some situations e.g. identifying different modules easily by looking at their web pages.
Parameters	A single word that is the new module name.
Return value	ERR_OK (0) if the new name was accepted. ERR_LENGTH (5) if name was too long.
Example	NAME Andromeda Assigns the new name "Andromeda"
Remarks	The name is stored into NVRAM and changed immediately. No reboot is necessary.

10.39 NET

	NET
Description	This command can be used to set the global network configuration, that is, if one or two network interfaces will be used and how they work together.
Parameters	NET requires one or two arguments. The first one is the network selector and the second, optional one, enables or disables network bridging mode. The first argument must be one of the following: GSM Since firmware version 6.05. Only GSM is enabled. This requires a GSM modem attached to the AVISARO module. WLAN Only WLAN is enabled. This is an absolute setting that does not probe for other network components. ETH Only Ethernet is enabled. Same as WLAN but requires an ENC28J60-chip. NONE All network interfaces are disabled. Networking is not possible even if a network interface exists. AUTO The module first probes if a WLAN interface exists. If that succeeds, WLAN will be usedas network interface. If no WLAN interface is found, the module then checks for the presence of Ethernet interface. BOTH Both, WLAN an Ethernet are used simultaneously. This is mandatory for bridging mode. The second argument can be:

	BRIDGE Bridging mode is enabled. In bridging mode, Ethernet packets are transparently routed over WLAN and vice versa. Also the first argument of NET must be BOTH for this to work. NOBR Bridging mode is disabled. Like the most configuration commands, NET settings are effective after next reboot.
Return value	ERR_OK (0) if command was accepted ERR_ARGUMENT (4) if one or more arguments don't match
Example	NET WLAN NOBR Uses WLAN-only mode and switches previous enabled bridging off
Remarks	This command has no effect, if the module has neither a WLAN nor an Ethernet interface.

10.40 NET?

	NET?
Description	With NET? the current network configuration can be queried and printed to the I/O interface. The output consists of two or three words that reflect the network configuration status. The first word is always the setting that has been made by using the NET command. The second word is the actual network configuration. For example: If NET was invoked with AUTO and the firmware found a WLAN module, then NET? will give AUTO WLAN. If the network was configured to use bridging mode (that is: NET BOTH BRIDGE), NET? will give BOTH BOTH BRIDGE, if both network devices are functional so that bridging can take place. Here are some example outputs: AUTO WLAN When AUTO was selected and WLAN is found BOTH WLAN When BOTH was selected but only WLAN is found. Potentially enabled bridging is OFF in this case, because bridging requires both interfaces but only one is active. BOTH BOTH When BOTH was selected without bridging and both network interfaces was found. WLAN NONE When WLAN was selected but can't be activated. BOTH BOTH BRIDGE When BOTH was selected with bridging and both interfaces are active so that bridging can be performed. Some more combinations are possible, such as NONE NONE etc.
Parameters	

Return value	
Example	
Remarks	

10.41 NEW

	NEW
Description	This command creates a new file on disk and opens it for writing. If another file with the same name (or path) aready exists, the command is rejected by the module.
Parameters	NEW requires two arguments. The first one is a handle number in the range from 0 to 100 which must be used in subsequent operations on that file, if NEW succeeds. The second and last argument is the name (or full path) of the new file.
Return value	ERR_OK (0) if a new file was created and openend for writing. ERR_ID_USED (27) if the supplied handle number is already in use. ERR_FILE_OPEN (32) if a file with the same name is already open. ERR_FIL_EXHAUSTED (26) if the file system has not enough memory to allocate file management information. ERR_FR_... (13...25) file system errors if something's going wrong while creating or accessing the file.
Example	NEW 1 hello.txt Creates a new file named "hello.txt" in the root directory and opens it as handle #1
Remarks	This command only exists on modules with some kind of mass storage (SD-Card or USB stick).

10.42 OPEN

	OPEN
Description	This command can be used to open an existing file for reading.
Parameters	The command needs two arguments. First a handle number in the range from 0 to 100 and next the name (or full path) of a file. If open succeeds, the given handle number must be used in subsequent calls to other operations on that file.

Return value	ERR_OK (0) if the file was opened successfully. ERR_ID_USED (27) if the handle is already in use (e.g. by another file). ERR_FILE_OPEN (32) if the file is aready in use (opened for reading or writing) ERR_FIL_EXHAUSTED (26) if the system couldn't allocate a file descriptor for that file. ERR_FR_... (13...25) file system errors if the file system encountered an error.
Example	`OPEN 1 hello.txt` The file hello.txt is opened for reading as handle #1OPEN 1 hello.txt The file hello.txt is opened for reading as handle #1
Remarks	Before an existing file can be read ist must be opened using OPEN. When done, use the CLOSE command to close the file and free the file handle.

10.43 OPEN?

	OPEN?
Description	OPEN? can be used to print all file handles that are currently open. A file is in open state, when its handle is in use (activated by OPEN, NEW oer APPD). OPEN? does not require any arguments. The output is a list of all open files separated by comma.
Parameters	
Return value	
Example	OPEN? If e.g handles 1,4 and 5 are open, OPEN? prints: 1,4,5
Remarks	

10.44 PING

	PING
Description	This command sends ICMP echo request messages to a remote host. It works similar to the "ping" utility on modern operation systems.
Parameters	PING needs a single argument that is the IP address of the remote host in standard dotted-decimal format.
Return value	ERR_ARGUMENT (4) if the IP address is not valid. ERR_OK (0) If remote host answered the request.

	ERR_NET_DOWN (39) if echo requests could not be sent. ERR_NO_DATA (8) if remote host didn't answer.
Example	PING 192.168.0.1 Hello 192.168.0.1, are you there? Prints: OK if 192.168.0.1 answers or ERR8 if 192.168.0.1 cannot be reached.
Remarks	PING does not understand host or domain names. You could call DNS before if the hostname is known.

10.45 PORT

	PORT
Description	The PORT command can be used to use a subset of the Module's pins as GPIOs or as analog or digital signal lines. PORT commands overwrite previous configurations. For example: If you're using the RS232 interface, a PORT 9 GET command immediately disables the TXD line.
Parameters	The general syntax of the PORT command is: PORT n cmd arg1 arg2 Where n is the pin number and cmd is the subcommand. The additonal arguments arg1 and arg2 are optional and currently only used for square wave generation (PWM). The following subcommands are available: PWM Generates square waves. arg1 and arg2 are used to set pulse/pause lengths. The first one (arg1) determines the pause length in units of 0.5 µs and the latter one (arg2) is the pulse length, also in 0.5 µs units. Pin 5, 6, 7, 9, 10 and 11 can be used as PWM outputs simultaneously. Since all outputs share a common timer, the frequency of each signal must be equal to the the others. To follow this rule, simply make sure that the sum of arg1 and arg2 is the same for all outputs. SET Drives pin HIGH if it is an output pin CLR Drives pin LOW if it is an output pin GET Reads digital value (0 or 1) from that pin ANA Reads analog value (0...1023) from that pin
Return value	ERR_OK (0) if the action was performed without error. ERR_ARGUMENT (4) if input was rejected.
Example	PORT 2 SET Makes one LED on the trailer board glow

	PORT 7 PWM 1000 1000 Generates symmetric square waves with a period of 1ms
Remarks	This command exists since version 3.48. The Avisaro module pin numbering scheme can be found here: click. The following list shows all possible pin functions: Pin1 Can't be used (VBAT) Pin2 Can be used as GPIO. Pin2 is connected to one LED on the trailer board. Pin3 Can be used as GPIO. Pin3 is connected to the other LED on the trailer board. Pin4 Can be used as GPIO. Pin 4 is connected to the key on the trailer board. Pin5 Can be used as GPIO and PWM output. Pin 5 is connected to DCD on the RS232/RS485 socket. Pin6 Can be used as GPIO and PWM output. Pin 6 is connected to DSR on the RS232/RS485 socket. Pin7 Can be used as GPIO and PWM output. Pin7 is connected to DTR on the RS232/RS485 socket. Pin8 Can be used as GPIO and analog input. Pin 8 is connected to RING on the RS232/RS485 socket. Pin9 Can be used as GPIO, analog input and PWM output. Pin 9 is connected to TXD on the RS232/RS485 socket. Pin10 Can be used as GPIO and PWM output. Pin 10 is connected to RXD on the RS232/RS485 socket. Pin11 Can be used as GPIO and PWM output. Pin 11 is connected to CTS on the RS232/RS485 socket. Pin12 Can be used as GPIO. Pin 11 is connected to RTS on the RS232/RS485 socket. Pin13 Can't be used. (GND) Pin14 Can't be used (RESET) Pin15 Can be used as input or open-drain output. Pin15 is also used as SCL of the IIC interface. Pin16 Can be used as input or open-drain output. Pin 16 is also used as SDA of the IIC interface. Pin17 Can't be used. (Internal SPI) Pin18 Can't be used. (Internal SPI)

	Pin19 Can't be used. (Internal SPI) Pin20 Can't be used. (Internal SPI) Pin21 Can't be used. (Internal SPI) Pin22 Can't be used. (Internal SPI) Pin23 Can't be used. (Internal SPI) Pin24 Can't be used. (VCC)

10.46 POS

	POS
Description	The POS command can be used to move the file pointer of an open file. It can also be used to extend the file size through cluster pre-allocation.
Parameters	POS needs two arguments. The first one is a file handle that must previously be opened by any means such as the OPEN, NEW or APPD commands. The second argument is the new zero-based absolute position of the file pointer.
Return value	ERR_OK (0) if the file pointer is moved successfully. ERR_NOT_OPEN (32) if the given handle doesn't represent an open file. ERR_FR_XX (13...25) file system error if the file pointer could not be moved for any reason.
Example	POS 1 200 Moves the file pointer of a file that is open as handle #1 to position 200 (zero-based)
Remarks	If the file is open for reading or writing, the next read or write access will continue from the new position. If a file is open for writing and the file pointer is moved beyond the file size, that file is enlarged to the new file pointer position. In this case, content between the old and the new position is unpredictable.

10.47 PROGFW

	PROGFW
Description	This command re-programs the MCU with a firmware image that is already stored in Flash memory. While programming a bunch of dots is printed to the current I/O interface.
Parameters	If the single argument "RUN" is given, the module restarts after programming completes. Without RUN, the module must be restarted manually.
Return value	There's no return value because the old firmware is overwritten.
Example	PROGFW RUN Re-programs the MCU and reboot
Remarks	Never invoke PROGFW if you are not sure that a valid firmware image is loaded. See LOADFW also.

10.48 PROMPT

	PROMPT
Description	When entering a command, the interface answers with a prompt. The command "prompt" allows to change this "Prompt". A Prompt is a sequence of characters that the module sends to the currently selected I/O interface when input of new commands is possiple. The default Prompt is the sequence {0x0d, 0x0a, 0x3e}, which is a carriage-return, line-feed and a > character.
Parameters	Just a string used as new prompt. The maximum length of a Prompt is 15. Longer inputs are truncated. To enter a special character, use the "\xxx" format - with xxx being the decimal value for the character. Example: \ 010 \ 13 for a carriage return (omitt the spaces) and a line feed
Return value	ERR_OK (0) Always.
Example	PROMPT \ 010 \ 013hello Sets the Prompt to "hello" and outputs a carriage return/line feed before.
Remarks	The new prompt is active after next reboot.

10.49 PROT

	PROT
Description	Sets the active I/O interface. PROT must be called with one or two arguments. The second argument is optional. Please see below.
Parameters	RS232 (RS232 Interface) I2C (I2C bus - Avisaro is slave device) SPI (SPI bus - Avisaro is slave device) CAN (CAN bus) SOCK (TCP/IP socket - Avisaro is listening) NONE (I/O is disabled) FILE (Output is written to file 'outfile.log' " (MehrDetails) 2) Second argument (Optional) NOW Optional argument to make changes immediately, but not permanently. With "NOW" the module immediately switches over to the new protocol without storing anything in Flash Memory. Without "NOW" only a DataFlash entry is changed. A restart is necessary to activate the new protocol.
Return value	ERR_OK (0) if command was accepted ERR_ARGUMENT (4) if one or more arguments didn't match.
Example	PROT SPI Sets the active I/O interface to SPI, which will be used after next reboot
Remarks	The secondary I/O channels, such as CAN#2 and RS232#2 cannot be used as primary I/O interface. They can only be used from the scripting language.

10.50 READ

	READ
Description	The READ command can be used to read chunks of data from open files. After reading a chunk, the file pointer is moved behind that chunk so that a subsequent READ can get the next few bytes instantly. Read data is sent to the currently selected I/O interface.
Parameters	READ requires two arguments. The first one is a file handle that must already be opened for reading, and the second, last argument is the number of bytes that should be read from the file.
Return value	ERR_OK (0) if everything's gone well. ERR_ARGUMENT (4) if the supplied handle is not a valid handle from the file handle space (0...100). ERR_NOT_OPEN (28) if the given handle is not assigned to an open file. ERR_NO_READ (29) if the file is open but has no read access. ERR_FS_... (13...25) file system errors if the file system runs into trouble while reading from the file.
Example	READ 1 100 Reads the next 100 byte from a file that is open as handle #1
Remarks	If the file pointer has advanced behind the last byte, READ returns ERR 33 (End of File). Before READ can be used, the file must be opened. See also OPEN.

10.51 RECM

	RECM
Description	This command can change settings regarding to the "Recovery Mode". Recovery Mode provides a way to capture modules over the network if they were misconfigured or some settings are forgotten. For this purpose, the module starts with default settings when powered on and remains there for a few seconds. While beeing in recovery mode, the module listens for certain UDP messages. If it receives one, it remains in recovery mode and can be re-configured over the network.
Parameters	The RECM command requires one argument which is one of the keywords ON, OFF or LISTEN: ON Recovery Mode is fully enabled. This means that the module switches to recovery mode on startup and sends recovery beacons that can be detected by a listener program to lock the module. The module also listens for UDP messages that can cause it to remain in recovery mode. LISTEN

	This is the listen-only recovery mode. The module doesn't send recovey beacons but is still sensitive to UDP messages. OFF For maximum security, recovery mode is completely switched off. The module starts directly with the stored settings.
Return value	ERR_OK (0) if command was accepted ERR_ARGUMENT (4) if argument was neither ON, OFF nor LISTEN
Example	RECM OFF No recovery mode.
Remarks	Switching off recovery mode might be risky because misonfigured module are locked out forever. On the otehr hand, the simplest way to prevent hijacking is to disable recovery mode. Also active recovery mode increases boot-up time. In the default configuration (factory settings), recovery mode is enabled.

10.52 RECM?

	RECM?
Description	The command RECM? can be used to query the actual Recovery Mode setting. The output is one of the keywords ON, LISTEN or OFF. Please see the description above.
Parameters	
Return value	
Example	
Remarks	

10.53 RESTART

	RESTART
Description	The module can be rebooted with the RESTART command. Open files and TCP connections are closed, all components are shut down and the module performs a "warm start".
Parameters	If RESTART is trailed with the optional argument CLEAR, all stored configuration entries are overwritten with factory settings. Therefore, use RESTART CLEAR only when your module is extremely misconfigured.
Return value	None, because the module performs a restart.
Example	RESTART Immediately restarts the module
Remarks	The module does all the things as it was powered up, including execution of autostart files and running a BASIC skripts.

10.54 RS

	RS
Description	The RS command allows random read access of raw sectors of an SD card or USB stick. The storage device does not need to be formatted.
Parameters	RS requires a single argument that is the sector number. After the command was invoked, the module sends back 512 bytes which is the content of the requested sector.
Return value	ERR_OK (0) if sector was read successfully. ERR_FR_NOT_READY (13) if disk read operation failed.
Example	RS 1234 Read sector 1234
Remarks	Sector contents are delived as raw binary data.

10.55 RS232

	RS232
Description	Change settings of the primary RS232 interface
Parameters	Five arguments are required. A sixth, optional argument can be used to change the operating mode. Baudrate Valid rates: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800 Bits Number of bits of one data words: 5, 6, 7, 8 Parity O = odd, E = even, N = none Stopbits 1 or 2 Flow control N = none, SW = Xon/Xoff, HW = RTS/CTS Mode (Optional) This can be omitted or must be one of the keywords RS485 or RS485INV to switch the interface into RS485 mode. In RS485 mode, a transceiver chip must be connected that handles the physical bus. Therefore, the module automatically toggles a control line that most chips need to switch from RX to TX and vice versa. If RS485 is given, the Module drives the DTR control line from LOW to HIGH while sending. When RS485INV is given, that control line behaves inversely, that is, it goes from HIGH to LOW while sending. Please Note: If the sixth argument is missing, the interface is switched to standard RS232 mode. All arguments must be upper case.
Return value	ERR_OK (0) if command was accepted ERR_ARGUMENT (4) if one or more arguments are wrong
Example	RS232 115200 8 N 1 N This configures the RS232 interface to use a baudrate of 115200bps, using 8 bits pet character, no parity, one stop bit and now flow control. Because there's no sixth argument, the inetrface runs in standard RS232 mode.
Remarks	All new settings are effective after next reboot. This command only changes settings for the primary RS232 interface.

10.56 RS232 ERRLOG

	RS232 ERRLOG
Description	This command enables or disables logging of additional RS232 frame information.
Parameters	The argument to RS232 ERRLOG can be either ON or OFF. If ERRLOG ON is given, the RS232 will also log RS232 framing information on input. Every received character causes a corresponding status byte to be stored in the input buffer that contains additional information. The status byte consists of eight bits which mean the following (LSB first): Bit 0 Not used, Always zero Bit 1 Set on Overrun error. Overrun errors occur when the input FIFO is full and a new character arrives Bit 2 Value of the parity bit (see remarks). Bit 3 Set on framing error. Incoming serial data does not seem to be valid RS232 frames Bit 4 Break detected. A Break means that the input line remains zero for the time of a full frame Bit 5 Not used, Always zero Bit 6 Not used, Always zero Bit 7 Framing error or Parity error
Return value	
Example	RS232 ERRLOG ON Enables logging of error bits.
Remarks	The RS232 ERRLOG command first appears in version 4.38. If the value of the parity bit is needed, the Module must be configured for either odd or even parity. If the module is not configured for any parity, bit 2 of the status byte will always be zero. All new settings are effective after next reboot.

10.57 RS232?

	RS232?
Description	This command prints out all RS232 settings in one single line. The Output reflects the arguments of the *RS232* command (see above) in the same order. All items are separted by spaces. The second to the last item is either RS232, RS485 or RS485INV. This depends on the last argument of the *RS232* command when using 5 or 6 arguments. The very last item is either NORMAL or ERRLOG. This depends on the settings made by RS232 ERRLOG (see above).
Parameters	
Return value	
Example	RS232? Prints out e.g: 115200 8 N 1 N RS232 NORMAL
Remarks	

10.58 RUN

	RUN
Description	The RUN command starts a BASIC script or can be used to configure BASIC auto-run settings. RUN can have zero or one argument. Without an argument, the script is executed immediately.
Parameters	No Argument Starts the currently loaded script File Name Starts the script which is stored in a without loading it into the internal flash memory. Thus, the script is executed temporarily. Remember that file names must be in the 8.3 format. See the particular meaning of the file name "temp_run.bas" Mehr here. WAIT The BASIC script is executed immediately and the command interface is locked while the script is running. Commands can not be entered before the the BASIC script has terminated. AUTO The BASIC script is not executed immediately. Instead, the internal non-volatile auto-run flag is set. If the module is powered off and on or rebooted, the script is executed and runs in the background. AUTOWAIT The BASIC script is not executed immediately. Instead, the internal non-volatile auto-run flag is set. If the module is powered off and on or rebooted, the script is

	executed in exclusive mode, that is, the command interface is locked while the script is running. MANUAL The BASIC script is not executed, but the internal auto-run flag is cleared. This revokes any previous RUN AUTO or RUN AUTOWAIT commands.
Return value	ERR_OK (0) if the command was accepted. ERR_ALREADY_RUNNING (36) if the BASIC script is already active. ERR_ARGUMENT (4) if the argument was none of the above keywords.
Example	RUN MANUAL Clears the autostart flag RUN Executes the script immediately
Remarks	With exception of running a skript directly from a file (see above), a valid BASIC script must exist in Flash memory in order to be executed. See also the LOAD (Mehr[more) command.

10.59 SCAN

	SCAN
Description	The SCAN command can be used to seek the air for nearby WLAN nets. SCAN actively searches all 14 WLAN channels beginning from CH 1 up to CH 14. SCAN requires one argument and can have a second, optional one. When SCAN has finished, a list of all found WLAN nets is sent to the active I/O interface. The output begins with a single-line number that is the count of the following entries. Each entry contains space-separated information in the following order: 1. The BSSID, six bytes, 12 hex digits 2. RSSI: A decimal number that is the power of the received radio signal. The higher this value is, the greater is the probability that the sending station is nearer than stations with a smaller value. 3. The network type, either B or I, where I stands for IBSS (ad-hoc) and B stands for BSS (infrastructure) networks. 4. The encryption type, either 0,1,2 or 3. 0 means "no encryption", 1 means WEP, 2 means WPA and 3 means WPA2. 5. The SSID, a variable-length network name of up to 32 characters. Networks with a hidden SSID are labeled "[no name]".
Parameters	The first argument is one of the keywords BSS, IBSS or ANY. If BSS is given, SCAN only seeks for Access Points. In contrast, IBSS only seeks for ad-hoc networks. If ANY is given, both Access Points and ad-hoc networks are returned in the list. The second optional argument determines how many milliseconds SCAN shall remain in one channel before it switches over to the next channel. The smaller this value is, the faster the scan finishes but scan results might be incomplete. If this argument is missing, a default value of 100 is taken.
Return value	ERR_REJECTED (12) if SCAN could not start because of low ressources. ERR_ARGUMENT (4) if one of the arguments was wrong. ERR_NET_DOWN (39) If the WLAN interface is not active.
Example	SCAN ANY

	Might produce the following output 6 00095bb13202 55 B 0 [no name] 001a4fdc3cb3 67 B 3 Toshiba_APx 001cf084d376 75 B 2 Nagasaki_Medien 001b11fea730 90 B 2 dlink 000c419d2f64 50 B 1 Toshiba_AP 0019700213a3 51 I 0 avisaro
Remarks	This feature is new since version 3.48

10.60 SLEEP

	SLEEP
Description	The SLEEP command puts the module asleep for x seconds. In sleep-mode, most components of the MCU are suspended to save power. To force a premature wakeup, one can pull the EINT0 pin low.
Parameters	SLEEP can be called with zero or one argument, that is the time in seconds the module should sleep. After time is up the module awakes and continues to work. If no argument is given, the module does not wake up automatically. In this case, the only way to wake it up is a low pulse on the EINT0 pin.
Return value	Always ERR_OK (0)
Example	SLEEP 10 Let the module sleep for ten seconds
Remarks	SLEEP only affects the MCU and its components. To put a WLAN device into sleep mode, please see the WLAN command, here.

10.61 SMS

	SMS
Description	The SMS command exists to send and receive SMS messages over a GSM network.
Parameters	The SMS command requires 1 or 3 arguments. If the only argument is GET, the output will be the youngest SMS received by the GSM modem. If there are no messages stored, ERR_REJECTED(8) is returned. Otherwise a SMS is pulled off the modem and printed out as header an message body separated by CR/LFs. An SMS can only be read once.

	If the first argument is SEND, then the second one must be the receipient's mobile phone number and the third one is the message text. Spaces inside the message must be coded as 32, because the command interface does not allow spaces inside arguments.
Return value	ERR_OK (0) - if command is accepted ERR_ARGUMENT (4) - if there's aproblem with the arguments ERR_REJECTED (12) - if READ or GET fails for any reason ERR_PARAMCOUNT (3) - If number of argument didn't match
Example	SMS SEND 017636175395 Hello\032World!
Remarks	This command only exists on modules with FW version 6.05 and above. Also a GSM modem must be attached to the module.

10.62 SOCKIO

	SOCKIO
Description	This command changes the settings of the socket I/O interface. Using the socket I/O interface, one can talk to the module over a TCP/IP connection. This works similar to a telnet session. The socket I/O interface is sometimes useful as a replacement for hardware interfaces such as IIC and RS232.
Parameters	The only setting currently can be made is the port number.
Return value	ERR_OK (0) always. If the argument is greater than 65535, it is wrapped around to zero (mod 65536).
Example	SOCKIO 15524 This command tells the socket I/O interface to listen on TCP port 15524
Remarks	Socket I/O can be enabled using the PROT command.

10.63 SOCKIO?

	SOCKIO?
Description	This command can be used to query the actual SOCKIO settings. SOCKIO? simply prints out the port number.
Parameters	
Return value	
Example	
Remarks	

10.64 SPI

	SPI
Description	This command can be used to change the SPI slave settings of the module.
Parameters	Two arguments are required. The first one is the clock polarity and the second one is the clock phase. Both arguments can be either 1 or 0. The clock polarity determines if the clock signal is active high or active low. A 0 as clock polarity means, that the high clock pulses (the default SPI mode) are used while a 1 means low clock pulses are used. The clock phase determines the relationship between the data lines and the clock line in SPI transfers. If 0, data is sampled on the first clock edge and a transfer must begin and end with with activation and deactivation of the chip select input. If 1, data is sampled on the second clock edge. A transfer starts with the first clock edge and ends with the last sampling edge while the chip select input is active. In this mode it is possible too keep the chip select input constantly active while using the SPI. Here is a list of all possible SPI modes on the Avisaro Module: Mode 0 - Clock Polarity = 0, Clock Phase = 0, Chip Select must be activated and deactivated between transfers Mode 1 - Clock Polarity = 0, Clock Phase = 1, Chip Select must be activated and deactivated between transfers Mode 2 - Clock Polarity = 1, Clock Phase = 0, Chip Select can remain active while transferring multiple bytes Mode 3 - Clock Polarity = 1, Clock phase = 1, Chip Select can remain active while transferring multiple bytes

Return value	ERR_ARGUMENT (4) if one or all of the arguments are neither 0 nor 1 ERR_OK (0) if the command was accepted
Example	SPI 0 0 Sets the SPI to Mode 0
Remarks	To activate the SPI as I/O protocol use the PROT command.

10.65 SPI?

	SPI?
Description	Reads back SPI settings made by the SPI command. SPI? always prints out two binary numbers. For details see the description above.
Parameters	
Return value	
Example	
Remarks	

10.66 SSTAT

	SSTAT
Description	This command can be used to query socket information. It sends a list of all sockets (a snaphshot) to the I/O interface. Also the number of free system-wide packet buffers is displayed.
Parameters	SSTAT? does not require any arguments.
Return value	ERR_OK (0) if the command succeeded. ERR_NET_DOWN (39) if the network is not functioning or disabled
Example	SSTAT? Could produce the following output TCP: 200 LSTN 0 0 1000 80 0 - TCP: 198 LSTN 0 0 1000 21 0 - TCP: 197 LSTN 0 0 1000 21 0 - TCP: 0 CLSD 0 0 1000 80 2663 - TCP: 0 CLSD 0 0 0 0 0 - TCP: 0 CLSD 0 0 0 0 0 - TCP: 0 CLSD 0 0 0 0 0 - TCP: 0 CLSD 0 0 0 0 0 - TCP: 0 CLSD 0 0 0 0 0 -

	TCP: 0 CLSD 0 0 0 0 0 - TCP: 0 CLSD 0 0 0 0 0 - TCP: 0 CLSD 0 0 0 0 0 - UDP: 0 OPEN 0 0 0 53 0 - UDP: 0 CLSD 0 0 0 22122 0 - UDP: 0 FREE 0 0 0 0 0 - UDP: 0 FREE 0 0 0 0 0 - UDP: 0 FREE 0 0 0 0 0 - UDP: 0 FREE 0 0 0 0 0 - POOL: 8 The list contains the following information: 1. The number of lines is the total amount of UDP and TCP sockets in the system. 2. The first entry per line is either UDP or TCP, that is the type of the socket. In the example above, there are 12 TCP and 6 UDP sockets. 3. The number behind the socket type is a handle number if the socket is open. If its zero, the socket is free. 4. The next entry is the socket state. For TCP sockets, this can be: RSRVD - Reserved - The socket is allocated but not yet opened. CLSD - Closed - The socket is free. LSTN - Listening - The socket listens for incoming connection attempts. SYNRC - SYN received - The socket is about to connect. SYNSN - SYN sent - The socket tries to connect to a remote station. FW1 - FIN-Wait1 - The socket waits to complete TCP closing sequence. FW2 - FIN-Wait2 - The socket waits to complete TCP closing sequence. CLSNG - Closing - The socket is about to close. LASTACK - Last Ack - The socket waits for the last ack befor it is closing. TMDWT - Timed wait - The socket is about to close (handles FIN retries) CONN - Connected - The socket is connected. That is the only state where data transfer is possible. For UDP sockets the state can be: FREE - The socket is unused, was never used. CLSD - The socket is closed. OPEN - The socket is currently in use. 5. The next two numbers are the number of packet buffer that the socket currently holds. The first number shows how many packets are in the socket's receive list, while the second number is the amount of packets in the transmit list. 6. The next two numbers are the port numbers for that socket. First comes the local port and the second number is the remote port number 7. The last information in each line (shown as - in the example above) indicates the binding. If the socket is bound to a specific interface, either WLAN or ETH is displayed. If the socket is not bound, a minus sign is shown. 8. Finally, after all socket entries, the output ends with a single line that shows how many free packet buffers the memory pool has.
Remarks	SSTAT? is only useful on modules that have a network interface.

	STOP
Description	The STOP command can be used to halt a running BASIC script. But this only works if the script does not control the current I/O interface's input.
Parameters	STOP can be called without or with a single argument. If there's no argument, STOP sends a signal to the scripting engine subsystem and returns immediately. As soon as possible, the scripting engine then terminates the running BASIC program. If an argument is given, it must be the keyword WAIT. STOP WAIT waits up to ten seconds until the script really terminated.
Return value	ERR_OK (0) if the command was accepted. ERR_ARGUMENT (4) if the optional argument was not the keyword "WAIT". ERR_NOT_RUNNING (37) if STOP was invoked while there was no BASIC program running. ERR_ALREADY_RUNNING (36) if STOP WAIT was invoked but the program didn't terminate within ten seconds.
Example	STOP WAIT Try to stop a running BASIC script and wait until that script terminates
Remarks	If the I/O interface is not available or controlled by a running script, it is possible to stop the script by using the command page http://moduleaddress/cmd of the web interface.

	STOREAGE
Description	This command can be used to change the storage device that the module uses. Currently, there are two media types supported, SD-Cards and USB Flash Drives (USB Sticks).
Parameters	STORAGE requires one of two arguments: SD Selects SD-Cards. USB Selects USB flash drives.
Return value	ERR_ARGUMENT (4) if the arguments is not invalid ERR_OK (0) if the command is accepted See (Mehr here) for complete list of error codes.
Example	STORAGE USB Selects USB Flash Drive as storage media.

Remarks	This command exists since version 4.49. The module must be rebooted for this command to take effect. STORAGE command only makes sense on modules that are equipped with appropriate hardware to access the selected media. CAUTION: Because there is no defined default storage media, changes made by STORAGE cannot be reverted using RESTART CLEAR or other ways to restore factory settings.

10.67 STPSEQ

	STPSEQ
Description	The STPSEQ command can be used to define a new Stop Sequence. A Stop Sequence is a consecution of characters that is used to cancel various operations such as streaming mode. If the module finds the Stop Sequence in the data stream, the current operation is cancelled. If STPSEQ is never invoked, the module reacts to the default Stop Sequence "+++" (without quotation marks). The Stop Sequence's maximum length is 15. Longer strings are truncated.
Parameters	The new stop sequence.
Return value	ERR_OK (0) Always.
Example	STPSEQ 1234 Sets a new stop sequence. "1234" in this case
Remarks	You can enter binary values into the stopsequence by using the \abc notation, where "abc" is a three-digit decimal value from 000 to 255.

	STREAM
Description	This command can be used to enable so-called "streaming" to and from an open file, TCP connection, UDP channel, standard or auxiliary I/O interface.
Parameters	STREAM needs one or two arguments. The first argument is the object (e.g. a file handle) that should be streamed. The streaming direction is determined by the capabilities of that object and how it was opened. The second, optional argument is the other end of the stream. If it is missing, all streams use the current selected I/O interface as default source or destination. While a stream is active, the command interface is blocked until the stream ends because all data has been transmitted, the stop sequence was found, or an error occurs that breaks the stream. Not all possible permutations are yet implemented. Those currently available are listed below:

	Streaming from the current I/O interface into a file If a file is opened for writing and after the STREAM command was invoked on this file, all data from the current I/O interface is streamed into the file. Streaming can only be cancelled if the stop sequence is inserted into the stream. Streaming from a file to the current I/O interface If a file is open for reading and after the STREAM command was invoked on this file, all data from the file is streamed to the I/O interface. Streaming is cancelled automatically if the last byte of the file was sent. Bi-directional streaming to and from a TCP connection If a TCP connection exists and after the STREAM command was invoked on this TCP connection, all incoming data from the TCP connection is routed to the I/O interface. Likewise, data from the I/O interface is sent simultaneously over the TCP connection. Streaming is cancelled automatically, when the TCP connection ends in any way or when a stop sequence is inserted on the I/O interface. If the socket is configured for TX delay, data is retarded for the specified time to collect bigger packets for better bandwidth utilization. Bi-directional streaming to and from a UDP channel If a UDP channel is open and after the STREAM command was invoked on this channel, all incoming data from the UDP channel is routed to the I/O interface. Data from the I/O interface can be sent simultaneously over the UDP channel. Because UDP is not connection oriented, the stream kan only be broken when a stop sequence is inserted on the I/O interface. If the socket is configured for TX delay, data is retarded for the specified time to collect bigger packets for better bandwidth utilization. Streaming from the auxiliary RS232 interface into a file If a file is opened for writing and after the STREAM command was invoked on this file using -4 as the second argument, all data from the auxiliary RS232 interface is streamed into the file. Streaming can only be cancelled if the stop sequence is inserted into the stream. Streaming from the auxiliary IIC interface into a file If a file is opened for writing and after the STREAM command was invoked on this file using -5 as the second argument, all data from the auxiliary IIC interface is streamed into the file. Streaming can only be cancelled if the stop sequence is inserted into the stream. Streaming from the auxiliary IIC interface into a file but omit IIC stop conditions If a file is opened for writing and after the STREAM command was invoked on this file using -6 as the second argument, all data from the auxiliary IIC interface is streamed into the file. Streaming can only be cancelled if the stop sequence is inserted into the stream.
Return value	ERR_OK (0): if a stop sequence was used to cancel the stream, a TCP connection was gracefully closed or a file has streamed its last byte. Any other value not equal to zero: An error occured. The meaning depends on which channels are used for streaming.
Example	STREAM 1 Switch on streaming to or from a file that was opened as handle #1
Remarks	A stream can also be interrupted by closing the socket (see CLOSE) using the cmd page of the web interface: http://moduleaddress/cmd

10.68 SHED

	SHED
Description	This command can be used to manually alter the scheduling frequency of the internal RTOS, that is, the time interval when the current task is stopped and another task gets the processor. SCHED requires one or two arguments. The default frequency of the scheduler is 50Hz, which means that every task is allowed to run 20ms before it is stopped and another task is activated. The RTOS switches tasks in a round-robin manner all tasks get the same time slice.
Parameters	. The first argument is the frequency in Hz of the scheduler. Frequencies from 1 Hz to 27 kHz and the magic value 0 are allowed. If 0 is given, time-controlled task switching is switched off and the RTOS uses cooperative multitasking. If only the first argument exists, the new frequency is applied immediately after the current running task's time quantum is exhausted. The scheduler keeps this frequency until the module is powered off or another SCHED command is invoked. The second argument, if given, must be the word "FIX". If this argument exists, the scheduler is not immediately re-configured but rather the frequency is stored into Flash memory and effective on next reboot.
Return value	ERR_OK (0) if command was accepted. ERR_ARGUMENT (4) if command was rejected due to wrong input
Example	SCHED 0 FIX This sets the configuration entry in Flash memory to "use coorerative multitasking"
Remarks	

10.69 SHED?

	SHED?
Description	This command can be used to query the scheduler settings. SCHED? does not need any arguments. The output is like this: 200 200 The first number is actual scheduling frequency in Hz. The second number is the stored scheduling frequency that will be used when the module starts.
Parameters	
Return value	
Example	

Remarks	.

TIM E	
Description	With the TIME command one can assign new values to the RTC (Real Time Clock). For all "Box" and "Cube" products, the RTC is battery backed, thus it keeps the time even if power is disconnected. The "Modules" requires external supply to hold time.
Parameters	TIME requires six arguments separated by spaces in the following order: Year: 2000...2099 Month: 1...12 Day: 1...31 Hour: 0...23 Minute: 0...59 Second: 0...59
Return value	ERR_OK (0) if the command was accepted. ERR_ARGUMENT (4) if one of the arguments is out of range.
Example	TIME 2008 10 20 12 13 14 Sets the RTC date to 2008/10/20 and time to 12:13:14
Remarks	On Modules without battery or permanent power supply the time is reset to 2000/01/01 00:00:00 at power-on.

TIME?	
Description	This command can be used to query the current RTC time of the module. For a format of this output see the Example below.
Parameters	
Return value	
Example	TIME? Might produce these output: 2008/12/10 08:23:44

Remarks	.

10.70 UDP

	UDP
Description	UDP opens a UDP channel, for both, transmission and reception.
Parameters	UDP requires at least 4 and can have one or two optional arguments. The arguments are as following: 1. Handle number This is the handle (or socket) number of the new UDP channel. Any decimal value from 201 to 300 is allowed. If UDP succeeds, this number can be used in subsequent calls where a UDP socket number is required. 2. Outgoing IP address This the IP address that is used as destination address for outgoing packets. 3. Outgoing port number This is the port number of the remote socket. A remote UDP must listen on that socket top receive packets from this module. 4. Incoming port number This is our port number. A remote UDP must send packets to this port number so that we can receive them. 5. TX delay value (optional) This value only affects streaming mode. In streaming mode, small TX packets are delayed until more data arrives or time elapses. This is a little trick to make bigger packets and, therefore, better use of network bandwidth. 6. Use checksums or not (optional) This argument, if given. must be one of the words "ON" or "OFF". ON means that checksums are generated for outgoing packets, whereas OFF means that no checksums will be used.
Return value	ERR_OK (0) if command was accepted and the new channel is created. ERR_ARGUMENT (4) if one or more arguments don't match. ERR_NET_DOWN (39) if command was rejected because the network is just not functional. ERR_FILE_OPEN (32) if command was rejected because that handle is already open by another socket
Example	UDP 201 192.168.0.1 111 222 1000 OFF Open an UDP channel with handle number 201 that listens on port 222. Transmissions over this channel will be send to 192.168.0.1,port 111. If using streaming mode, this socket collects outgoing data for max. 1 second.
Remarks	This command only makes sense on modules with a network interface (Ethernet or WLAN).

10.71 UPTIM?

	UPTIM?
Description	This command can be used to get the time since the module is switched on (up time). The output is a simple decimal value that is the uptime in seconds.
Parameters	This command does not require any arguments.
Return value	ERR_OK(0) Always.
Example	UPTIM? Could print out: 12545
Remarks	UPTIM? uses the RTC but a battery is not required. UPTIM? keeps running when the module is sleeping. It just counts from the point where the module was powered up.

10.72 VER?

	VER?
Description	This simple command prints out the firmware version.
Parameters	No arguments required.
Return value	ERR_OK (0) Always.
Example	VER? Prints out e.g: 3.35
Remarks	This command works on all type of modules.

10.73 WEB

	WEB
Description	WEB can be used to customize the web pages that the module deliveres to the browser of the user.
Parameters	WEB requires a single argument as decimal number which contains 32 bits that switch some parts of the web pages on and off. The following list shows the meaning of those bits: BIT 0: Show Ethernet page

	BIT 1: Show WLAN page BIT 2: Show Ethernet IP settings BIT 3: Show WLAN IP settings BIT 4: Show data interface pages BIT 5: Webserver expert mode BIT 6: Show FTP server page BIT 7: Scripting expert mode BIT 8: General expert mode BIT 9: Show FW download page BIT 10: Show copyright label BIT 11 ... 31: Not used
Return value	ERR_OK (0) If a single argument was given. ERR_PARAMCOUNT (3) If zero or more than one arguments were given.
Example	WEB 10 Switches on bit 2 and bit 8 to enable only the WLAN page and the WLAN IP settings
Remarks	By default (factory settings) all pages are visible.

10.74 WLAN

	WLAN
Description	The WLAN command can be used to change all of the WLAN settings.
Parameters	For each setting, WLAN must be invoked with two arguments, the setting name and the new value. Here is a list that shows the WLAN command's capabilities in detail: WLAN PS ON | OFF Enables or disables power saving mode. The second argument must be one of the words ON or OFF. ON switches to power saving mode, whereas OFF switches the WLAN device to full power mode. In power saving mode, most of the internal components of the WLAN device are switched on and off repeatedly to reduce power consumption. WLAN PASS xxx Sets the WPA pass phrase. This is a human-readable text that is used to calculate the master key for the WPA and WPA2 WLAN encryption schemes. The pass phrase must not exceed 63 characters. WLAN SSID xxx Sets the new SSID (network name) for WLAN infrastructure mode. The SSID must be the same SSID as that of the access point where the module should connected or the same SSID that is used in the ad-hoc network. The SSID must not exceed 32 characters. New since firmware version 4.24: Avisaro Modules can use a special notation *abcde# beginning with an asterisk, a number sign at the end and where a,b,c,d,e

are any characters. This can be used to set the 802.11 Network-ID (BSSID) to 0abcde (for Ad-Hoc networks only). In this case, all ad-hoc nodes that have the same SSID are forcibly brought together. A so configured module does not need to scan the air for ad-hoc partners and, consequently, can't communicate with others that use a variable BSSID.

WLAN MODE INFRA | ADHOC

This sets the connection mode to either Ad-Hoc (IBSS) or infrastructure mode. The second argument must be one of the words INFRA or ADHOC. In infrastructure mode, WLAN nodes require a master station, the so-called "access point". In ad-hoc mode, WLAN nodes can interlink without the need for an access point.

WLAN CHANNEL xxx

This sets the WLAN channel, that means the set of frequencies which are used on the air. Allowed values are 1...14.

WLAN SECURITY WEP40 | WEP104 | WPAPSK | WPA2PSK | NONE

With this command one can change the encryption scheme that is used to secure the communication. WEP40 means 40-bit WEP encryption. WEP104 means 104-bit WEP encryption. WPAPSK means that WPA-TKIP with pre-shared key should be used and WPA2PSK is WPA-AES with pre-shared key. If you supply NONE no encryption will be used thus, communication is visible for everyone.

WLAN WEP xxxxxxxxxxxxxxxxxxxxxxxxxx

This sets a new WEP key. A WEP key is a 26-digit long hexadecimal value (104 bits) that is used as key for WEP encryption, if WEP104 is enabled. If WEP40 is enabled, only the first ten digits (== 40 bits) are valid.

WLAN SLEEP

Puts the WLAN component immediately into "deep sleep" mode, to consume as little power as possible. In contrast to WLAN PS, this mode does not allow to send or receive data. Before the component goes to sleep, it is disconnected from the AP, but all TCP connections are kept open. Deep sleep is a temporary mode, that means the Avisaro module always has an active WLAN after reboot.

WLAN AWAKE

Wakes up the WLAN from deep sleep mode immediately. After it woke up, the WLAN reconnects itself to the AP and continues to work. WLAN AWAKE introduces a one-second delay which is necessary for the WLAN component to settle down.

WLAN BSSID CLEAR | PIN | xxxxxxxxxxxx

(NEW since version 4.45) Sets a constraint on association. If the BSSID filter is active, the module only associates to an AP having exactly the same BSSID that was entered. For example, if you invoke BSSID 001977021385 then the module only seeks for APs with this BSSID when trying to connect. To remove the constraint, call BSSID CLEAR. BSSID can be used with the the following arguments:

A 12-digits hexadecimal number

Set the BSSID filter manually. If 000000000000 is used, the filter is switched off and the modules is able to connect to any AP.

CLEAR

For convenience, same as BSSID 000000000000. Use this to remove the BSSID filter.

PIN

BSSID PIN can be used to set the BSSID filter to the BSSID of the currently connected AP. In order to pin your Avisaro Module to an AP, follow these steps:

1. Invoke BSSID CLEAR to disable any previous filtering

	2. Restart the module and let it associate to the desired AP.You may invoke the WLAN? command to verify that the module is connected to the right AP 3. Invoke BSSID PIN.
Return value	ERR_OK (0) if the command was accepted. ERR_ARGUMENT (4) if one ore more arguments didn't match. ERR_LENGTH (5) if one or more arguments had a wrong length.
Example	WLAN SSID TestAP WLAN MODE INFRA WLAN CHANNEL 3 WLAN SECURITY WEP104 WLAN WEP 12345678901234567890aabbcc This configures the WLAN device to connect to an AP named TestAP on channel 3, using WEP104 as encryption scheme with the key 0x12345678901234567890aabbcc
Remarks	All WLAN settings are effective after next reboot, or if WLAN ist stoppend and started again.

10.75 WLAN?

	WLAN?
Description	Prints WLAN settings and information line-by-line in the following order: 1. The network name, SSID. 2. WLAN mode. This can be either INFRA or ADHOC. 3. Preferred WLAN channel (a scan seeks all channels). 4. WLAN encryption scheme. This can be one of WEP40, WEP104, WPAPSK, WPA2PSK, NONE. 5. The WEP key as hexadecimal value. 6. Power saving mode. Either ON or OFF. 7. The pass phrase for WPA and WPA2. Encryption keys are based on this. 8. Primary Master Key for WPA or WPA2. This value is calculated automatically by the module when SSID or pass phrase has changed. 9. Connection state (CONN == Connected, NC == Not Connected). 10. Number of successfully received packets. 11. Number of receive failures (Dropped packets because of errors) 12. Number of successfully transmitted packets. 13. Number of transmit failures. 14. Signal strength of last received packet. 15. Own MAC address. This is a 12-digits hexadecimal number. 16. BSSID of the WLAN where the module currently is attached to. This is a 12-digits hexadecimal number.

	17. Also a 12-digits hex number. This is the filter BSSID that can be set with the WLAN BSSID command. If all digits are zero, the filter is disabled. (NEW since version 4.45)
Parameters	
Return value	
Example	WLAN? Could produce the following output: Andromeda_AP INFRA 11 WEP104 12345678901234567890a1b2d3 OFF IEEE a687b2429193c66edc7cc10f0e9d3facc0e8ba1d5ed3e8eebe26161ea0ffd2bf CONN 5169 0 41 0 66 00197002121c 000c419d2f64 000000000000
Remarks	.

10.76 WPS

	WPS
Description	The WPS command starts the process to automatically receive Wi-Fi configuration data. WPS is a standard automatic configuration procedure supported by many Access Point.
Parameters	There is only one parameter: WPS START Starts the WPS procedure. Usually, a button has to be pushed on the Access Point to enable WPS mode for 2 minutes. Afterwards, the 'WPS START' command has to be issued on the Avisaro WLAN Device. The command is blocking. It takes about 15 seconds for the Avisaro Module to negotiate the parameters.
Return value	ERR_OK (0) if the command was accepted. ERR_NET(39) net is down = no WPS Access Point in sight.

Example	WPS START This starts the configuration process.
Remarks	When using WPS, it usually makes sense to use also the "DHCP CLIENT" setting.

10.77 WRITE

	WRITE
Description	Writes a chunk of data into an open file and advances the write pointer so that another write operation can append new data.
Parameters	WRITE requires two arguments. The first one is a handle number from 0 to 100, that must refer to a file which is already open. The second and last argument is the data itself, that should be written into the file.
Return value	ERR_OK (0) If everything worked as expected. ERR_ARGUMENT (4) if the handle number was out of range. ERR_NO_WRITE (30) if the file is open but has read access. ERR_DISK_FULL (24) if there's not enough room on the disk ERR_FR_... (13...25) general file system errors if the file system encounters a problem.
Example	WRITE 1 hello_world Writes the string "hello_world" into a file that is open as file handle #1
Remarks	This command only works modules that have some kind of mass storage (USB or SD-Card)

10.78 WS

	WS
Description	This command can be used to bypass the file system and write directly to a sector on the SD card or USB stick. The disk does not need to be formatted. WS can also be used to implement custom file systems.
Parameters	This command must be invoked with a single argument that is the sector number to write, followed by a CR/LF. After that, 512 bytes of arbitrary content must be

	sent. The command interface is blocked until those 512 bytes are completely received. Then, data is written to the specified sector and the command interface is available again.
Return value	ERR_OK (0) if data is successfully written to the card. ERR_FR_NOT_READY (13) if write operation failed.
Example	WS 1000 aaaaaaa.... 512 bytes of 'a' in total, this fills sector 1000 with the character 'a'
Remarks	Be carful when writing with this command to formatted disks. It can damage or completely destroy the filesystem.