

Basic scripting - Fundamentals

Content

Introduction

Fundamentals

Write and edit scripts

Load and run scripts: Generals

Load scripts using W/LAN interface (configuration website)

Load and run scripts using SD memory cards

Interfaces: Using RS232

Interfaces: Using CAN

Interfaces: Using I2C (Slave and Master)

I/O, PWM, Analog Ports

Working with TCP connections

Working with UDP data streams

Store and read user data

Performance

Example of a script

Introduction

The Avisaro 2.0 products have a build-in scripting language. This scripting can be used to define stand alone behavior of the box. The scripts are like little BASIC programs which are stored on the Avisaro 2.0 product and start automatically when the product is powered up.

Typical applications are formatting of data to be stored on a SD card or contacting a server in the network to post collected data. The scripting engine is designed for small applications - use an external controller for large full scale applications.

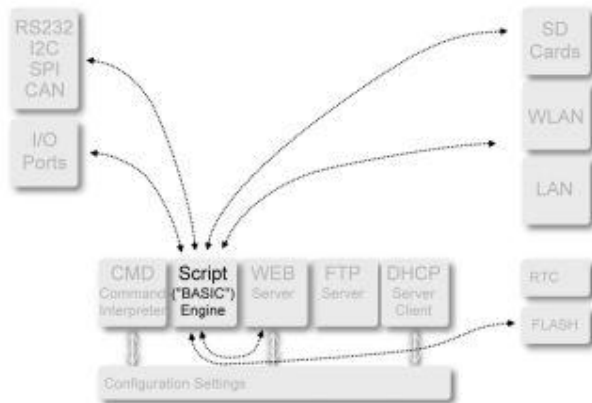
This is a printout of the online documentation. See the online documentation for further details and actualizations. Next to this printout there exists a printout of the list and descriptions of the commands.

Fundamentals

BASIC Scripting Engine

The Scripting Engine allows to:

- Get and send data to a data interface (RS232, CAN,...)
- Control I/O ports, PWM and analog input
- Write and read data from the SD memory card
- Send and receive data through WLAN and LAN
- Store data into internal flash
- Post and get information from the build-in web-server



Commands vrs. Scripts

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

The **Command Interface** allows sending 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** allows having 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, gsub-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. A SPS or micro controller sends or retrieves data to several server. Commands are send to control connections.
Send several parallel datastreams from A to B		✓	
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 additional 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 function or they can be disabled to use the command interface.

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

Limits of scripting in numbers

The scripting capability is designed for smaller and targeted tasks. It is not designed to be a generic, general purpose programmable device. Where the limits are, depends on size of script, speed and amount of data to be stored during operation.

A few limits are (unless otherwise stated, all products have the "regular firmware"):

Resource	Limit (regular Firmware)	Limit (extended Firmware)	Comment
Length of source code	12.288 Bytes	24.576 Bytes	equals the size of Source Code file loaded into the module
Default size of tokenised code.	3.072 Bytes	6.144 Bytes	Can be modified using command memcfg
Number of 'goto' commands	30	60	Including gosub commands
Nested gosubs	26	26	
Number of variable definitions	30	30	This is the number of variables, not the amount of memory they are using.
RAM Size:	12.288 Bytes	24.576 kB	The memory split can be changed by the scripting command MEMCFG
Standard Split:			
Heap size	8.192 Bytes	16.384 Bytes	
Code segment	3.072 Bytes	6.144 Bytes	
Data segment	1.024 Bytes	2.048 Bytes	

Write and edit scripts

Scripts for the Avisaro 2.0 products can be written with any text editor. Windows "Notepad" or "Editor" is sufficient. Typical scripts have less than 200 lines of code.

To start off easily, take an existing script as a basis and start to modify it.

You find the existing script at

<http://www.avisaro.com/tl/script.html>.

See the separate documentation of the commands for their explanation.

```

wrl-13.txt - Editor
Datei Bearbeiten Format Ansicht ?

* WLAN RS232 Device Server
* (C) Avisaro AG 01.12.2008
* Version 1.13

DIM A(500)
DIM B(0)
let n = 0
let t = TIME
let f = 60

REM INIT WEB

let x$ = "connect to (IP, =0 for listen):"
put -100, x$, len(x$)
let x$ = "connect to / Listen (Port):"
put -102, x$, len(x$)
let x$ = "status (Script version 1.13.):"
put -104, x$, len(x$)

load 0, t$
put -101, t$, len(t$)

load 25, u
if {-1 = u} then
  let u = 23
  save 25, u
end if
let x$ = str$(u)
put -103, x$, len(x$)

TRY_CONNECT:
* RS232 leer lesen
mode -3
input A
  
```

Load and run scripts: Generals

Scripts are loaded into the flash memory of the Avisaro module. Thus, they are available even after power cycling the module. Scripts can also be configured to start automatically after power up.

In addition, there is an option to have scripts loaded temporarily, leaving the script in flash memory unchanged. This feature is good for 'once in a while' tasks or for testing.

There are three methods to load and run a script:

Build-in Website: If the product is equipped with a WLAN or LAN interface, scripts can be loaded through the build-in configuration web site. This is probably the most easy way during development.

SD Cards: If the product is equipped with a SD card slot, the Scripts can be uploaded using a batch file. This is a good methods for high volume deployment.

Command interface: Particularly using the RS232 interface and a terminal program, scripts can be uploaded this way.

Load scripts using W/LAN interface (configuration website)

To load a script using the build-in configuration website, follow the following steps:
Navigate to the configuration web site of the module using any suitable browser. If default IP settings are used, simply type "192.168.0.74" in the address field of your browser.

Navigate to the "Scripting" section of "Module Settings".

Click on 'search' and select the script file you want to upload.

Click on 'send file' - that's all.

If the Script does not start, there are probably syntax errors. Navigate to the page "cmd" and type in the command "run". If there is an error message, it will be displayed here.

Settings: Scripting

Upload new script:

Select a script file and press 'send file'

Run script on start-up:

No: Script is not executed when devices is powered up

Asynchronous: Script is executed automatically, when devices is powered up.

This option is recommended!

Exclusive: Script is started exclusively - for experts only.

Currently running

Checked: The script is up and running. To manually start a script, select this box and click 'submit'

Not checked: The script is not running. To manually stop a script, uncheck this box and click 'submit'. Notice: after pressing 'submit', the box comes up check again. Press 'refresh' to see whether the script really stopped.

3 x 2 Table

This table can be used by scripts to display data on the web site or the ask user to enter data. Content of this table therefore depends on the loaded script. See script documentation for details.



up, add a 'autorun.txt' file. Enter the command "run temp_run.bas" ([more](#)) to force execution of this script without loading it into internal flash.

Details:

The execution of "temp_run.bas" forces currently running scripts to be stopped.

However, a mechanism exists to avoid that just loaded scripts are stopped and started again when the same SD card is ejected and inserted. This works based on the card ID - a unique ID each SD card has.

Interfaces: Using RS232

There are two RS232 interfaces which can be used on an individual basis. RS232 interface 'Port 1' is the primary one, 'Port 2' is the auxiliary one. The primary port can be used to transmit commands to configure the Avisaro product. Both ports can be used to receive and send RS232 data to be stored or to be forwarded wirelessly over a network.

Activating and configuration of RS232 interface 1 (Primary)

The RS232 port 1 is designed to receive user data as well as commands addressed 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.

... via web interface

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.

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.

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.

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.



... via SD card

To activate the RS232 interface, copy the following commands into a file called 'autorun.txt'. Place this file on a SD card, insert card and power on device.

Example for set-up commands

```
prot rs232
rs232 8 N 1 NONE
```

Activating and configuration of RS232 interface 'Port 2' (auxiliary)

The RS232 port 2 is designed to be used in scripting language only. It is well suited to receive and send RS232 data for data logging or data transfer over wireless networks.

The RS232 port 2 is activated only within BASIC scripts:
Use the command auxopen to activate the second RS232 interface. Example:

```
auxopen -4, 9600, ASC("N"), 1, 8, ASC("N")
```

Most users decide to set baudrate and other settings for this port within the script - since scripts can be loaded easily, changes are easy to do.

Working with RS232 in Scripts

Read RS232 data

RS232 data is presented a stream of bytes. They can be read one-by-one or copied into an array of bytes.

The script needs to define that the RS232 interface should be handled by the scripting engine, rather than by the command engine. Use the INMODE command to do so:

```
inmode -3
```

As an example, define an array with 100 bytes. This array will be used to hold RS232 data

```
dim A(100)
```

Using the get command, a message from RS232 port 1 is retrieved:

```
get -3, A
```

To read a message from RS232 port 2, use '-4' instead of '-3'. Alternatively to the 'get -3' command, the input command can be used.

The get command is not blocking - if no data were received, the command returns with no data. To verify how much bytes were received, use the 'bytesread' system variable:

```
if BYTESREAD > 28 then
  ' data were received
  ' do something with it
end if
```

Of course, there is always the good old 'input' command:

```
input a$
```

Process RS232 data

The content of the RS232 data can be checked for specific data, reformatted, One way to do so is to check the array byte by byte:

```
for n = 0 to BYTESREAD-1
  ' check with i.e. if a(n) = ASC("g")
  ' modify with let a(n) = a(n) + 5
next n
```

Output RS232 data

The byte array can be send out using the put command. In this case the same amount of data as previously read in is printed out again:

```
put -4, a, BYTESREAD
```

Of course there is always the good old 'print' command:

```
print "Hello world"
```

Interfaces: Using CAN

There are two CAN interfaces which can be used on an individual basis. CAN interface 'Port 1' is the primary one, 'Port 2' is the auxiliary one. The primary port can be used to transmit commands to configure the Avisaro product. Both ports can be used to receive CAN messages to be stored or to be forwarded wirelessly over a network.

Setup and configuration

Activating and configuration of CAN interface 1 (Primary)

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

For all Avisaro CAN Data Logger and Avisaro CAN WLAN products, the CAN interface is already activated and operates with default value. Change baud rate 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 CAN interface must be activated and configured.

Activating and configuration of CAN interface 'Port 2' (auxiliary)

The CAN port 2 is designed to be used in Scripting language only. It is well suited to receive and send CAN messages for data logging or data transfer over wireless networks.

The CAN port 2 is activated only within BASIC scripts:

Use the command `auxopen` to activate the second CAN interface. Example:

```
auxopen -8, 125000, 0, 10000, 0, 0
```

Most users decide to set baudrate and other settings for this port within the script - since scripts can be loaded easily, changes are easy to do.

Working with CAN in Scripts

Principle of operation

CAN messages are stored internally in a fixed 28 Byte long format - no matter how long the original CAN messages was. This message format is documented at the end of this site. All operations with CAN messages are based on the 28 Byte format: To receive a message, a 28 Byte array must be declared; sending CAN messages over TCP is done in 28 Bytes (or multiple of it) chunks.

Read CAN messages from CAN bus

The script needs to define that CAN messages should be handled by the scripting engine, rather than by the command engine. Use the `inmode` command to do so:

```
inmode -3
```

Define an array with 28 bytes. This array will be used to hold a CAN message:

```
dim A(28)
```

Using the `get` command, a message from CAN port 1 is retrieved:

```
get -3, A
```

To read a message from CAN port 2, use '-8' instead of '-3'. Alternatively to the 'get -3' command, the `getcan` command can be used.

The `get` command is not blocking - if no message was received, the command returns with no data. To verify whether or not data were received, use the 'bytesread' system variable:

```
if BYTESREAD = 28 then
  ' message was received
  ' do something with it
end if
```

Process CAN messages

The content of the CAN message is packaged into those 28 bytes. The storage format is described at the end of this page. To avoid bit operations, the command 'caninfo' was introduced to extract specific CAN fields:

```
if caninfo(1) = 5 then
  ' a CAN message with CAN ID 5
  ' was received
end if
```

The data bytes of the can message are accessed by addressing the array: A(13) contains the first byte of the payload ... all the way to A(20) for the 8th byte.

Details

CAN Data Format

CAN messages are mapped into a 28 byte long array:

Byte	Bits	Description
1	7..0	User definable header bytes. Usually all '0'.
2	15...8	
3	23..16	
4	31..23	
5	7..0	Frame descriptor: reserved bits, all '0'
6	7..0	Frame descriptor: reserved bits, all '0'
7	7..4	Frame descriptor: reserved bits
	3..0	Frame descriptor: CAN data length
8	7	Frame descriptor: Frame type (0 = standard, 1 = extended)
	6	Frame descriptor: RTR bit ('1' = RTR bit is set)
	5..0	Frame descriptor: reserved bits
9	7...0	Message ID
10	15...8	Standard message: valid bits 10..0
11	23..16	Extended message: valid bits 28..0
12	31..23	
13	7..0	CAN data bytes in the order 1 ...8 Valid number of bytes defined by CAN data length
14	7..0	
15	7..0	
16	7..0	
17	7..0	
18	7..0	
19	7..0	
20	7..0	
21	7...0	Real time clock time stamp in seconds since 01.01.2007
22	15...8	
23	23..16	
24	31..23	
25	7...0	Millisecond time stamp. Value in ms since module power on.
26	15...8	
27	23..16	
28	31..23	

CAN Message Examples

The following shows some example frames that can be send from the Avisaro module to the CAN bus. In this examples, header and timestamp fields are not used.

Extended frame, ID is 0x0301, three bytes of data: 01, 02, 03

00 00 00 00 00 00 03 80 01 03 00 00 01 02 03 00 00 00 00 00 00 00 00 00 00 00 00 00

Standard frame, ID is 0x0301, eight bytes of data: 01, 02, 03, 04, 05, 06, 07, 08

00 00 00 00 00 00 08 00 01 03 00 00 01 02 03 04 05 06 07 08 00 00 00 00 00 00 00 00

Extended frame, ID is 0x11223344, three bytes of data: 01, 02, 03

00 00 00 00 00 00 03 80 14 33 22 11 01 02 03 00 00 00 00 00 00 00 00 00 00 00 00 00

Extended frame, RTR bit is on, ID is 0x11223344, three bytes of data: 01, 02, 03

00 00 00 00 00 00 03 c0 14 33 22 11 01 02 03 00 00 00 00 00 00 00 00 00 00 00 00 00

Legend

- Header bytes
- Frame Descriptor
- ID
- Data Field
- Timestamps

Please note that unused bits of the Frame Descriptor and ID Field *always must be zero*.

Interfaces: Using I2C (Slave and Master)

The I2C interface is available within Scripting in master and in slave mode. To use the slave mode, the module can be either configured as regular data interface or as auxiliary interface. The master mode can be activated only as auxiliary interface in scripts.

Setup and configuration

Activating I2C Slave as primary Data Interface

To operate I2C in slave mode, the regular data interface settings are used.

... via Web interface

If there is a network interface, the build-in configuration website can be used. Navigate to:

General: Enter I2C as active Data Interface

I2C: Enter the I2C adress the module should respond to.

... via SD card

To activate the I2C interface, copy the following commands into a file called 'autorun.txt'. Place this file on a SD card, insert card and power on device. See for details on how to configure Avisaro products using the 'autorun.txt' file.

```
prot i2c
i2c 73
```

Activating I2C Slave as second (auxiliary) interface

The I2C interface can be activated as a second (auxiliary) interface in scripting. If for example, the primary data interface is set to RS232, the Avisaro device still can be connected to a I2C bus as a slave. Data can be send and received from those devices to

be stored on SD card or forwarded wirelessly.

Use the command `auxopen` to activate the I2C interface in slave mode.

Example:

```
auxopen -5, 0, 43, 1, 0, 0
```

This example opens the I2C interface, sets the I2C bus address to communicate on to 43.

Activating I2C Master as second (auxiliary) interface

The I2C interface in master mode is configured in scripting language only. It is well suited to connect sensors or other I2C devices to the Avisaro products. Data can be send and received from those devices to be stored on SD card or forwarded wirelessly.

Use the command `auxopen` to activate the I2C interface in Master mode.

Example:

```
auxopen -5, 100000, 43, 0, 0, 0
```

This example opens the I2C interface, sets the I2C bus address of the client to communicate with to 43.

Working with I2C (Master) in Scripts

Send data to I2C device

To following example sends the string "hello" to a I2C client with the bus address 51 (decimal). :

```
dim a(10)
auxopen -5, 100000 , 51 , 0 , 0, 0

let a(0) = 104
let a(1) = 101
let a(2) = 108
let a(3) = 108
let a(4) = 111

put -5, a, 5
```

The `dim` command is used to define an array of 5 bytes to hold the message to be send.

The `auxopen` command opens the I2C Master interface. The client address is defined with this `auxopen`. Thus before sending data to different clients, the `auxopen` command has to be repeated with the appropriate bus address.

The following `let` commands fill the array with the ASCII codes for "hello".

The `put` command finally sends the data to I2C client. In this example, 5 bytes are send.

This example uses a 8 bit I2C address. See later chapter on how to use 11 bit addresses.

Read data from I2C device

The following example reads data from the I2C device. We assume the I2C is still configured from the above write command. This I2C device requires to write the desired register (here: 0) and then start reading.

```
let a(0) = 0

put -6, A, 1
get -5 , A
```

In order to perform a write and then read, we use the handle "-6" in the put command which means "send no stop condition". Then, the 'GET' follows to read bytes. In this case, 10 bytes are read since the array a() is 10 bytes long.

Working with I2C (Slave) in Scripts

Read I2C data (Slave)

I2C data is presented as a stream of bytes. They can be read one-by-one or copied into an array of bytes.

The script needs to define that the I2C interface should be handled by the scripting engine, rather than by the command engine. Use the inmode command to do so:

```
inmode -3
```

As an example, define an array with 100 bytes. This array will be used to hold I2C data

```
dim A(100)
```

Using the get command, a message from I2C interface is retrieved:

```
get -3, A
```

The get command is not blocking - if no data were received, the command returns with no data. To verify how much bytes were received, use the 'bytesread' system variable:

```
if BYTESREAD > 28 then
  ' data were received
  ' do something with it
end if
```

Of course, there is always the good old 'input' command:

```
input a$
```

I/O, PWM, Analog Ports

If a pin is not used for data transmission, it can be used as an input, output, pulse width modulation (PWM) or analog pin. This feature is available for the WLAN and Logger Module.

Command Interface vrs. Basic Scripting

Those pins can be controlled using Basic Scripting (described in this article) or the Command Interface. Choose basic scripting option if the Avisaro Module should perform stand alone functions.

Setting and querying I/O ports

To set an I/O port, use the 'put' command. To read an I/O port, use the 'get' command. There is also the 'setleds' and the 'key' command. Do not use those commands- they are there for compability reason only.

There is no need to declare a port as input or output port. By performing a put command, this port is declared as an output port. By performing a get, this port is declared as an input port. By default, a port is an input port.

This script listing is a simple example of how to use I/O ports within Basic. The script turns and off Pin 2 . It reads in Pin4 and sets Pin 3 the same :

```
' Script to set and read ports
' for demonstration purpose

let in = 0      ' declare variable in
do
  put -202, #1  'pin 2: on
                'pin 2 is often the red led
                'sleep 1 second
  sleep 1000
  put -202, #0  'pin 2: off
  sleep 1000
  get -204, in
  put -203, in  ' set pin 3 same as
                ' input pin 4
loop
```

Details

There is no need to declare a port as I/O port. Simply use the port. A new port command overwrites a previous configuration. Thus, if a pin was used for data communication, a port command can interrupt this functionality. Make sure to follow the electrical rules pins

Working with TCP connections

This chapter describes how to open and manage a TCP connection using the BASIC scripting. For example, an external sensor only sends data but can't be modified to send commands to handle a connection.

See to get details on how open and manage a TCP connection using commands send from an external device such as a micro controller.

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](#)

PDF state diagram

Open TCP connections

A TCP connection can be established two ways:

- 1.) Waiting for a incoming connection (Avisaro is TCP server)
Use the command "LISTEN" to create a 'TCP socket' in listen mode. Specify an internal 'handle' number - this handle number addresses this TCP socket. Also specify a TCP port number.
- 2.) Connection to a server (Avisaro is TCP client)
Use the text command "CONNECT" to actively connect to a TCP server. Specify an 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:

Control TCP connections

Using the "STATUS" command it can be verified whether or not a connection was established. A return value of '9' for example means a successfully established connection.

Sending and receiving data

Once the connection is established, data can be exchanged using the "PUT" and "GET" command. One TCP packet can hold 1452 bytes of data. Ideally, an array with this length should be used for sending and receiving data.

Sending data with PUT

The PUT command is used to send data. The handle number specifies which connection to use since more than one TCP connection can be handled. As data to send, a string variable or an array can be given as a parameter. Remember to check the dummy variable 'LASTERR' after sending since transmitting data can fail. See command description for details.

Receiving data with GET

The GET command is used to receive data. It is advised to define an array with 1500 Bytes as the receiving array - otherwise a received tcp/ip packet does not fit. See command details.

Closing TCP/IP connection

When done with sending or receiving data, the connection can be closed. Use the "CLOSE" command to perform this function.

Monitoring TCP/IP connection

While receiving and transmitting data, the TCP/IP connection can be interrupted. Reasons could be the loss of wireless connection or the other party was switched off or terminated. The command "STATUS" allows to monitor the connection and reaction properly to a change in connection status.

Working with UDP data streams

UDP is a "connectionless" way to exchange data. It is very simple to set up and operate, however data are not protected by acknowledgements.

Using UDP sockets

A UDP socket is opened using the "UDPOPEN" command.

UDP is a packet oriented format. One UDP packet can hold 1460 bytes. Thus an array with this length should be ideally defined for receiving and sending data.

Some details:

If an array shorter than the max length is used during a 'get' command, the left over data from that packet are thrown away.

Store and read user data

Some applications require to store data i.e. for configuring an attached device, print messages to a screen or to recall ip addresses to connect to. The simplest way to store data within a BASIC script is to do a hardcoded line such as ' let ip\$ = "192.168.0.3" '. If there is more data to store, or data needs to be changed without modifying the script, there other ways:

Command: DATA - READ - RESTORE

These set of commands allow to store a number of data within a BASIC script. This is particular useful if i.e. config data needs to be send to device or messages need to be printed to a display. Data stored using this method are more or less 'hardcoded' and always come with the script itself (it depends on the application whether or not this is of advantage or not).

See DATA - READ - RESTORE for details on those commands.

The number of data which can be stored using the data command can be changed. By default it is 1kByte. See MEMCFG on how to modify memory allocation.

Command: SAVE - LOAD

These set of commands allow to store a number of data permanently in the Avisaro device. This is particular useful for configuration data (i.e. IP address to connect to) or temporary user data such as a history of events.

This storage can be used also to store data to be send to a connected device for configuration purpose. A little 'helper' script or a special function (... if file exist ... copy

content to this store area ...) can be used to preload this storage area. Once loaded, the script can access the data within this area. This storage area is non-volatile. Its size is 4kByte - so quite some data fit into this area. Unlike the other method, this area is fixed in size. See SAVE - LOAD for details on those commands.

Files: Store data on SD card

Last but not least, data can be stored on SD card - if a memory card slot is available. Storage space is almost unlimited, however data are only available if a SD card is inserted. See for details.

Performance

General thoughts

"Performance" depends on the type of application. In general, performance is separated into "throughput" and "delay". Applications like data logging tend to require high throughput, whereas voice type application a short delay time require.

Delay considerations

The Avisaro 2.0 system contains a multitasking architecture. The TCP/IP Stack, WLAN driver and BASIC Scripting run in separate tasks. The multitasking has two different modes:

Time sliced mode: Each task gets a defined timeslot assigned. That is independent of whether the task has something to do or not. As a result, the system is very predictable how it reacts to input.

Work load mode: Each task gives up computing resources as soon as there is nothing to do. To avoid a blocking system, there is a maximum time of 20ms for each task. As a result, the system reacts more dynamically to the inputs but is less predictable.

Time Slice Mode

Use the command "sched" to configure the length of the timeslot each task should be given. A value of 20 ms ("sched 50" : $1\text{sec}/50 = 20\text{ms}$) is a good starting point.

Work Load Mode

Use the command "sched" to switch on the Work Load Mode ("sched 0").

It is advised to tell the BASIC Script when to give up computing resources. This is done with the command "sleep 0". Typically this command is inserted after sending a data packet :

```
repeat_put:
put 201, C, BYTESREAD
if LASTERR <> 0 then
sleep 0
goto repeat_put
end if
```

There is a 'safety net' - if no sleep 0 is used, every about 20ms the BASIC Scripting task is switched anyway. This is to avoid a 'hanging' system: If BASIC would not free computing resources to the TCP/IP task, no network traffic could be done.

The overall result is a system with short delay times.

Windows PC: Delayed Ack

When running a Windows PC, there is problem called "Delayed Ack".

The data throughput and response time with the Avisaro wireless module can be significantly improved if the so called "Delayed Ack" feature of Microsoft Windows is turned off. (It unfortunately turns on by default.) This feature has almost all modern TCP stacks. While it is an advantage of Internet surfing, it has more disadvantages in the embedded area. In order to eliminate "Delayed Ack" on a Windows based PC, as you can go to the website:

German: <http://support.microsoft.com/default.aspx?scid=kb;de;328890>

English: <http://support.microsoft.com/kb/328890>

Configuration Tools

Turning of the 'delayed ack' can be done manually or by using these tools. Operating the tools is easy:

1) Use "Install.vbs" to turn of the delayed ack (recommended)

2) Use "Remove.vbs" to enable delayed ack again.

The tool is provided by www.leatrix.com. A manual can be found in the 'gaming community' (gamers also have a need for short network delays).

Test Tools

Avisaro provides a test tool to check network performance particularly with embedded WLAN devices. For this test to run, follow the steps:

1) Install the script "av_4-1.txt" on the Avisaro WLAN. Configure on page "Scripting" a '0' for listening.

2) Run the Java Tool "Avisaro Performance Test". Enter IP address of Avisaro device and click 'run'.

This gives a clear picture of whether the performance is sufficient.

Example of a script

Script "MR3" with comments

This script sample serves two RS232 interfaces.

Comments are marked by ' or with REM:

```
'  
' Datenl ogger Rev 1.8 (c) Avi saro AG, 14.10.2009  
'
```

With DIM you define an array and the maximum size.

```
DIM A(512)
```

If the internal time is not buffered by a battery, use the following to set the time to default. With Avisaro boxes and cubes it should not be done due to the real-time clock.

```
' i f no battery  
i f time < 10000 then  
    exec "time 2009 01 01 00 00 01"  
end i f  
  
exec "fsync 1000"  
exec "sched 0 fi x"
```

The script is using both RS232 interfaces. Therefore the second interface is now activated. With INMODE and OUTMODE the input- and output-mode is defined to make sure that the data are not understood as commands.

```
' 2. rs232 i nterface  
auxopen -4, 9600, asc("N"), 1, 8, asc("N")  
  
i nmode -3  
outmode -2
```

for any easy starting message the command PRINT can be used.

```
pri nt "Avi saro Logger Rev 1.8 (c) 2009 Avi saro AG"
```

Anchor to go to the beginning of the script later on:

```
BEGI N:
```

Normally there is a green LED at the IO pin 3. This will now be switched on:

```
put -203, #1
```

If the button at IO pin 4 is pushed the script will start all over again: Later the button stops the process. Therefore this function now will make sure that the button is not used now.

```
i f (KEYS & 1) = 1 then  
    goto BEGI N  
end i f
```

Now the script will check if a memory card is in the slot:

```
if l of(0) = 0 then
  sleep 100
  goto BEGIN
end if
```

If the sd-card is working, the script opens a file log-1.txt to add further data. If it the file is not existing ("LASTERR") a new file will be created. For the second rs232 interface a file log-2.txt will be opened.

```
open "AB", 1, "log-1.txt"
if LASTERR <> 0 then
  open "WB", 1, "log-1.txt"
  if LASTERR <> 0 then
    close 1
    goto BEGIN
  end if
end if
```

```
open "AB", 2, "log-2.txt"
if LASTERR <> 0 then
  open "WB", 2, "log-2.txt"
  if LASTERR <> 0 then
    close 2
    goto BEGIN
  end if
end if
```

In case of success the LED at I/O pin 2 shall light up. The time is saved. A 'DO - LOOP' starts as main loop.

```
put -202, #1

do
  let t = time
```

INPUT collects the first data. All data will be included in the array A. BYTESREAD shows the amount of data. If the data are read the light turns off.

```
INPUT A
if BYTESREAD > 0 then
  put -202, #0
  put 1, A, BYTESREAD
end if
```

The same will happen to the second interface:

```
get -4, A
if BYTESREAD > 0 then
  put -202, #0
  put 2, A, BYTESREAD
end if
```

In the following the script checks whether the memory card remains in the slot.

```
if l of(0) = 0 or status(1) <> 2 then
  close 1
  close 2
  put -202, #0
  goto BEGIN
end if
```

By pressing the button the logger stops recording.

```
if (KEYS & 1) = 1 then
  close 1
  close 2
  goto FIN_KEY
end if
```

With a small time lack the red LED turns on. (the time lack has pure optical reasons).

```
if t < time then
  let t = time
  put -202, #1
end if
```

End of the main loop:

```
loop
```

FIN_KEY checks that the button is released. FINISH waits for further commands.

```
FIN_KEY:
  if (KEYS & 1) = 1 then
    goto FIN_KEY
  end if

FINISH:
  put -202, #0
  let x = l of(0)
  if (x = 0) or ((KEYS & 1) = 1) then
    goto BEGIN
  else
    goto FINISH
  end if
```

' +++ signals the end of the script if the script is loaded via interface.

```
goto BEGIN
```

```
' +++
```

Contact

Avisaro AG
Grosser Kolonnenweg 18 /D1
30163 Hannover, Germany
Tel.: +49 (0)511 780 93 90
Fax,: +49 (0)511 353 196 24
E-Mail: info@avisaro.com
Web: www.avisaro.com