

# **FT NavVision<sup>®</sup>**

## **Connections and Protocols on the FT NavVision System**

Project:

Project number:

Main title: FT NavVision®

Sub title: Port connections and Protocols on the FT  
NavVision System

Special remark:

Issue: 1.2

Date: October 15, 2012

Total number of pages: 57

Publication code: OM

Author: Vince Kerckhaert

Quality control:

Acknowledge:

Approved:

# 1. Table of contents

	Page no.
2. Figures .....	6
3. Tables.....	6
4. References.....	9
5. Introduction .....	10
6. About the manual .....	10
7. Abbreviations list .....	11
8. Safety instructions .....	11
9. Document revisions .....	12
10. Serial Communication.....	12
10.1 Introduction .....	12
10.2 DTE and DCE .....	13
10.3 RS232 specifications.....	13
10.3.1 RS232 Bit Streams .....	13
10.3.2 Start Bit.....	13
10.3.3 Data Bits .....	14
10.3.4 Parity Bit .....	14
10.3.4.1 Even Parity .....	14
10.3.4.2 Odd Parity.....	14
10.3.4.3 Disadvantages on the Parity System.....	14
10.3.5 Stop Bit(s).....	14
10.3.6 Voltages .....	15
10.3.7 Maximum Cable Lengths.....	15
10.3.8 RS232 Pinout.....	15
10.3.9 RS232 Flow Control and Handshaking.....	16
10.3.9.1 Software flow control.....	16
10.3.9.2 Hardware flow control .....	17
10.3.10 Special connections .....	18
10.3.10.1 RS232 DB25 to DB9 converter.....	18
10.3.10.2 RS232 serial loopback test plugs .....	19
10.3.10.3 RS232 Null Modem Cables .....	19
10.3.10.3.1 Compatibility Issues .....	20
10.3.10.3.2 Null modem with loop back handshaking .....	20
10.3.10.3.3 Compatibility issues .....	21
10.3.10.3.4 Null modem with full handshaking .....	21
10.3.10.3.5 Compatibility issues .....	22
10.4 RS485 Specifications .....	22
10.4.1 Differential signals with RS485: Longer distances and higher bit rates.....	22
10.4.2 Network topology with RS485 .....	23
10.4.3 RS485 functionality .....	24
10.4.4 What Pins Are Needed for 2- and 4- Wire Transmission with RS-485 Serial Communication? .....	24
11. TCP/IP .....	25
11.1.1 Introduction.....	25

11.1.2	Internet Protocol Suite.....	25
11.1.3	Layers in the TCP/IP stack.....	26
11.1.4	The Network Access layer.....	26
11.1.4.1	The physical layer.....	26
11.1.4.2	The data link layer.....	26
11.1.4.3	The Internetwork layer.....	27
11.1.4.4	The transport layer.....	27
11.1.4.5	The application layer.....	28
11.1.5	TCP/IP Relevance.....	28
<b>12.</b>	<b>Modbus.....</b>	<b>29</b>
12.1	Introduction.....	29
12.2	Modbus message structure.....	29
12.3	Modbus serial transmission modes: Modbus/ASCII and Modbus/RTU.....	30
12.4	Modbus addressing.....	31
12.5	Modbus function codes.....	31
12.5.1	Function 01: Read coil status.....	32
12.5.2	Function 02: Read input status.....	32
12.5.3	Function 03: Read holding registers.....	33
12.6	Addressing (0- or 1-based).....	33
12.7	Understanding Raw Data.....	34
12.7.1	How is data stored in Standard Modbus?.....	34
12.7.2	Examples of questions and answers.....	35
12.7.2.1	Example 1.....	35
12.7.2.2	Example 2.....	35
12.7.2.3	Example 3.....	36
12.7.3	Data Types.....	36
12.7.4	Byte and Word ordering.....	37
12.7.5	Modbus Map.....	37
<b>13.</b>	<b>Bus-Protocols.....</b>	<b>38</b>
13.1	Introduction.....	38
13.2	Canbus.....	38
13.3	Data transmission.....	39
13.4	Speed, cable and termination.....	40
<b>14.</b>	<b>Implementation in the FT NavVision® environment.....</b>	<b>41</b>
14.1	Introduction.....	41
14.2	Moxa UC-7110.....	41
14.2.1	Overview.....	41
14.2.2	Wiring requirements.....	42
14.2.3	LED indicators.....	42
14.2.4	Connecting to the network.....	43
14.2.5	Pinouts.....	43
14.2.6	Connecting to a serial device.....	44
14.2.7	Serial LAN server for Moxa.....	44
14.2.7.1	Type (Moxa UC-711X).....	45
14.3	ICP DAS i-7540D.....	46
14.3.1	Power supply connection.....	46
14.3.2	Connection to CAN bus.....	47
14.3.3	Resistance check.....	47
14.3.4	Com Ports.....	48
14.3.5	CAN ports.....	48
14.3.6	Type (ICPdas i7540D).....	49
14.4	COM port assignment.....	50
14.5	485LDRC9.....	54
14.5.1	Operation.....	54



## 2. Figures

Figure 10-1: RS232 DB9 Pinout	16
Figure 10-2: RS232 DB25 to DB9 Convertor	18
Figure 10-3: DB9 RS232 Loopback Plug	19
Figure 10-4: Null modem cable without handshaking	19
Figure 10-5: Null modem with loop back handshaking	20
Figure 10-6: Null modem with full handshaking	21
Figure 10-7: Noise in straight and twisted pair cables	22
Figure 10-8: RS485 network topology	23
Figure 10-9: 422/485 DB9 Pinout	25
Figure 13-1: Canbus topology	40
Figure 14-1: Overview (MOXA)	42
Figure 14-2: 8-pin RJ connector and pinouts	43
Figure 14-3: Pin assignments (DB9 male port)	44
Figure 14-4: Type (Moxa)	44
Figure 14-5: Comm Port Settings	45
Figure 14-6: Power connection	46
Figure 14-7: CAN bus connector	47
Figure 14-8: Electronic circuit CAN bus connector	47
Figure 14-9: Terminator resistor	48
Figure 14-10: Interface	48
Figure 14-11: Standard	49
Figure 14-12: Type (ICPdas i7540D)	50
Figure 14-13: Drop-down menu (device interfaces)	51
Figure 14-14: COM port assignment	52
Figure 14-15: additional configuration	52
Figure 14-16: Comm Port Settings	53
Figure 14-17: 2-wire RS485	55
Figure 14-18: 4-wire RS485	55
Figure 14-19: RS422	56
Figure 14-20: Extend and isolate RS232	56

## 3. Tables

Table 10-1: RS232 Voltages Values	15
Table 10-2: RS232 Cable Length	15
Table 10-3: RS232 DB9 Pinout	16
Table 10-4: DB9 – DB25 conversion	19
Table 10-5: DB9 RS232 Loopback Plug	19
Table 10-6: : Null modem cable without handshaking	20
Table 10-7: Null modem with loop back handshaking	21
Table 10-8: Null modem with full handshaking	22
Table 11-1: Internet Protocol Suite	26
Table 12-1: Modbus Message Structure	30
Table 12-2: Properties of Modbus/ASCII and Modbus/RTU	30
Table 12-3: Device and Modbus address ranges	31
Table 12-4: Common Modbus function codes	32
Table 12-5: Function 01 query structure	32
Table 12-6: Function 01 answer structure	32
Table 12-7: Function 02 query structure	33
Table 12-8: Function 02 answer structure	33

Table 12-9: Function 03 query structure	33
Table 12-10: Hexadecimal values	34
Table 12-11: Modbus Storage	34
Table 14-1: Moxa Led Indicators	42
Table 14-2: Pin assignment (CAN bus)	47
Table 14-3: Communications mode selection	54
Table 14-4: Data rate selection	54

### ***NOTE***

***No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher, Free Technics B.V.***

***This manual is part of the FT NavVision® software-package, version 9.18.3.3214.***

***Free Technics B.V. rejects any responsibility for damage and/or personal and material consequences due to use of the FT NavVision® software package.***

***Because of continuous development of the software, it may occur that the manual does not cover the entire functionality of the delivered software. In no way can the user obtain any rights because of this.***



## 4. References

Not applicable.

## **5. Introduction**

When installing or commissioning the FT NavVision © system you will surely run into diverse serial, Modbus or network issues that you will have to deal with. As the program is developed to work with a wide variety of protocols we would like to give you the basic knowledge to properly check all the wiring and settings. Most of the basic knowledge will be a repetition of what you already know, but do take care in reading this manual . Some settings will be different of what you know because of the structure of the program. Also we will describe the many exceptions we encountered during the 15 years that we are installing FT NavVision © so you'll be prepared.

## **6. About the manual**

In the first chapters we will give some common information about Serial Ports, Network Ports, Protocols etc. This is common knowledge that can be found everywhere on the internet, but this is bundled here to have a quick reference. Later on we will elaborate on the diverse connections that we encountered and the pitfalls for the diverse interfaces. Watch for the version number cause this manual will extend over time.

## 7. Abbreviations list

CPU	Central Processing Unit
CSV	Comma Separated Values
ECR	Engine Control Room
FT	Free Technics
GUI	Graphical User Interface
HMI	Human Machine Interface
I/O	Input/Output
LED	Light Emitting Diode
LPU	Local Processing Unit
RS	Radio Standard
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
API	Application Programming Interface
CAN	Controller Area Network
COM	Communication
CPU	Central Processing Unit
CTS	Clear To Send
DC	Direct Current
DHCP	Dynamic Host Configuration Protocol
DIN	Deutsches Institut für Normung
DIP	Dual In-line Package
DSR	Data Set Ready
ER	Error
GND	Ground
IM	Installation Manual
I/O	Input/Output
INIT	Initialize
LED	Light Emitting Diode
Mbps	Megabit per second
RJ	Registered Jack
RTC	Real Time Clock
RTS	Request To Send
Rx	Receive
RxD	Receive Data
SRAM	Static Random Access Memory
TCP/IP	Transmission Control Protocol/ Internet Protocol
Tx	Transmit
TxD	Transmit Data

## 8. Safety instructions

The indications NOTE, CAUTION and WARNING have the following significance:

**NOTE:**

*An operating procedure, practice or condition etc., which it is essential to emphasize.*

**CAUTION:**

An operating procedure, practise or condition etc., which, if not strictly observed, may damage or destroy equipment.

**WARNING:**

An operating procedure, practise or condition etc., which, if not carefully observed may result in personal injury or loss of life.

## 9. Document revisions

Revision	Date	Written by	Approved by
New	October 15, 2012	VK	

## 10. Serial Communication

### 10.1 Introduction

In computing, a serial port is a serial communication physical interface through which information transfers in or out one bit at a time (in contrast to a parallel port). Throughout most of the history of personal computers, data transfer through serial ports connected the computer to devices such as terminals and various peripherals.

While such interfaces as Ethernet, FireWire, and USB all send data as a serial stream, the term "serial port" usually identifies hardware more or less compliant to the RS-232 standard, intended to interface with a modem or with a similar communication device.

Modern computers without serial ports may require serial-to-USB converters to allow compatibility with RS 232 serial devices. Serial ports are still used in applications such as industrial automation systems, scientific instruments, shop till systems and some industrial and consumer products. Server computers may use a serial port as a control console for diagnostics. Network equipment (such as routers and switches) often use serial console for configuration. Serial ports are still used in these areas as they are simple, cheap and their console functions are highly standardized and widespread. A serial port requires very little supporting software from the host system.

## 10.2 DTE and DCE

The individual signals on a serial port are unidirectional and when connecting two devices the outputs of one device must be connected to the inputs of the other. Devices are divided into two categories "data terminal equipment" (DTE) and "data circuit-terminating equipment" (DCE). A line that is an output on a DTE device is an input on a DCE device and vice-versa so a DCE device can be connected to a DTE device with a straight wired cable.

Conventionally, computers and terminals are DTE while modems and peripherals are DCE.

*Note: If it is necessary to connect two DTE devices (or two DCE devices but that is more unusual) a special cable known as a null-modem cable (see Chapter ) must be used.*

## 10.3 RS232 specifications

Communication as defined in the RS232 standard is an asynchronous serial communication method. The word serial means, that the information is sent one bit at a time. Asynchronous tells us that the information is not sent in predefined time slots. Data transfer can start at any given time and it is the task of the receiver to detect when a message starts and ends.

### 10.3.1 RS232 Bit Streams

The RS232 standard describes a communication method where information is sent bit by bit on a physical channel. The information must be broken up in data words. The length of a data word is variable. On PC's a length between 5 and 8 bits can be selected. This length is the netto information length of each word. For proper transfer additional bits are added for synchronisation and error checking purposes. It is important, that the transmitter and receiver use the same number of bits. Otherwise, the data word may be misinterpreted, or not recognized at all.

With synchronous communication, a clock or trigger signal must be present which indicates the beginning of each transfer. The absence of a clock signal makes an asynchronous communication channel cheaper to operate. Less lines are necessary in the cable. A disadvantage is, that the receiver can start at the wrong moment receiving the information. Resynchronization is then needed which costs time. All data received in the resynchronization period is lost. Another disadvantage is that extra bits are needed in the data stream to indicate the start and end of useful information. These extra bits take up bandwidth.

Data bits are sent with a predefined frequency, the baud rate. Both the transmitter and receiver must be programmed to use the same bit frequency. After the first bit is received, the receiver calculates at which moments the other data bits will be received. It will check the line voltage levels at those moments.

With RS232, the line voltage level can have two states. The on state is also known as mark, the off state as space. No other line states are possible. When the line is idle, it is kept in the mark state.

### 10.3.2 Start Bit

RS232 defines an asynchronous type of communication. This means, that sending of a data word can start on each moment. If starting at each moment is possible, this can pose some problems for the receiver to know which is the first bit to receive. To overcome this problem, each data word is started with an attention bit. This attention bit, also known as the start bit, is always identified by the space line level. Because the line is in mark state when idle, the start bit is easily recognized by the receiver.

### **10.3.3 Data Bits**

Directly following the start bit, the data bits are sent. A bit value 1 causes the line to go in mark state, the bit value 0 is represented by a space. The least significant bit is always the first bit sent.

### **10.3.4 Parity Bit**

For error detecting purposes, it is possible to add an extra bit to the data word automatically. The transmitter calculates the value of the bit depending on the information sent. The receiver performs the same calculation and checks if the actual parity bit value corresponds to the calculated value. For more advanced error checking there is the use of CRC (Cyclic Redundancy Check) but this goes beyond the scope of this manual.

#### **10.3.4.1 Even Parity**

Basically, the parity bit can be calculated in two ways. When even parity is used, the number of information bits sent will always contain an even number of logical 1's. If the number of high data bits is odd, a high value parity bit is added, otherwise a low bit will be used.

#### **10.3.4.2 Odd Parity**

The odd parity system is quite similar to the even parity system, but in this situation, the number of high bits will always be odd.

#### **10.3.4.3 Disadvantages on the Parity System**

The parity system using one bit for each data word is not capable of finding all errors. Only errors which cause an odd number of bits to flip will be detected. The second problem is, that there is no way to know which bit is false. If necessary, a higher level protocol is necessary to inform the sender that this information must be resent. Therefore, on noisy lines, often other detection systems are used to assure that the sent information is received correctly. These systems mostly do not operate on single data words, but on groups of words.

### **10.3.5 Stop Bit(s)**

Suppose that the receiver has missed the start bit because of noise on the transmission line. It started on the first following data bit with a space value. This causes garbled data to reach the receiver. A mechanism must be present to resynchronize the communication. To do this, framing is introduced. Framing means, that all the data bits and parity bit are contained in a frame of start and stop bits. The period of time lying between the start and stop bits is a constant defined by the baud rate and number of data and parity bits. The start bit has always space value, the stop bit always mark value. If the receiver detects a value other than mark when the stop bit should be present on the line, it knows that there is a synchronization failure. This causes a framing error condition in the receiving UART. The device then tries to resynchronize on new incoming bits.

For resynchronizing, the receiver scans the incoming data for valid start and stop bit pairs. This works, as long as there is enough variation in the bit patterns of the data words. If data value zero is sent repeatedly, resynchronization is not possible for example.

The stop bit identifying the end of a data frame can have different lengths. Actually, it is not a real bit but a minimum period of time the line must be idle (mark state) at the end of each

word. On PC's this period can have three lengths: the time equal to 1, 1.5 or 2 bits. 1.5 bits is only used with data words of 5 bits length and 2 only for longer words. A stop bit length of 1 bit is possible for all data word sizes.

### 10.3.6 Voltages

The signal level of the RS232 pins can have two states. A high bit, or mark state is identified by a negative voltage and a low bit or space state uses a positive value. This might be a bit confusing, because in normal circumstances, high logical values are defined by high voltages also (see **Fout! Verwijzingsbron niet gevonden.**).

Level	Transmitter Capable (V)	Receiver Capable (V)
Space State (0)	+5...+15	+3...+25
Mark State (1)	-5...-15	-3...-25
Undefined	-	-3...+3

Table 10-1: RS232 Voltages Values

The maximum voltage swing the computer can generate on its port can have influence on the maximum cable length and communication speed that is allowed. Also, if the voltage difference is small, data distortion will occur sooner.

### 10.3.7 Maximum Cable Lengths

Cable length is one of the most discussed items in RS232 world. The standard has a clear answer, the maximum cable length is 50 feet, or the cable length equal to a capacitance of 2500 pF. The latter rule is often forgotten. This means that using a cable with low capacitance allows you to span longer distances without going beyond the limitations of the standard. If for example UTP CAT-5 cable is used with a typical capacitance of 17 pF/ft, the maximum allowed cable length is 147 feet.

The cable length mentioned in the standard allows maximum communication speed to occur. If speed is reduced by a factor 2 or 4, the maximum length increases dramatically. Keep in mind, that the RS232 standard was originally developed for 20 kbps. By halving the maximum communication speed, the allowed cable length increases a factor ten!

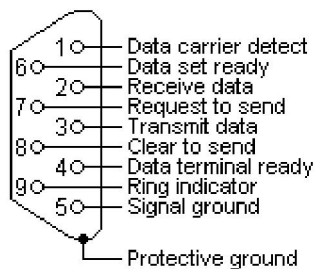
Baud Rate	Maximum Cable Length (ft)
19200	50
9600	500
4800	1000
2400	3000

Table 10-2: RS232 Cable Length

### 10.3.8 RS232 Pinout

The RS232 connector was originally developed to use 25 pins. In this DB25 connector pinout provisions were made for a secondary serial RS232 communication channel. In practice, only one serial communication channel with accompanying handshaking is present. Only very few computers have been manufactured where both serial RS232 channels are

implemented. the smaller DB9 version is more commonly used today, so we will refer to that connector in this manual.



**Figure 10-1: RS232 DB9 Pinout**

Pin	Line	Abbr.	Explanation
1	Data Carrier Detect	DCD	Connected to telephone (obsolete)
2	Transmit Data	TxD	Carries Data from DTE to DCE
3	Receive Data	RxD	Carries Data from DCE to DTE
4	Data Terminal Ready	DTR	Indicates Presence of DTE to DCE
5	Signal Ground	GND	Signal Ground
6	Data Set Ready	DSR	DCE is ready to receive Data
7	Request To Send	RTS	DTE requests DCE Prepare to receive Data
8	Clear To Send	CTS	Indicates DCE is ready to receive
9	Ring Indicator	RI	Detect telephone (obsolete)

**Table 10-3: RS232 DB9 Pinout**

### 10.3.9 RS232 Flow Control and Handshaking

Consider the situation where someone is helping you picking apples from a tree. Your helper climbs in the tree and throws all the apples to you. You have to put them in buckets. In the normal situation, you can easily catch all apples, but when one bucket is full and it has to be replaced by an empty one, this action costs more time than is available between two apples thrown by your helper.

Two different things can occur. Your helper stops until the new bucket is in position, or some apples are damaged because they fall on the ground in the small period you are not able to catch them.

You would probably prefer the first method where your helper stops for a small period. To achieve this, there will be some communication, eye-contact, a yell, or something like that to stop him from throwing new apples. How simple, but is it always this simple? Consider the situation where one computer device sends information to another using a serial connection. Now and then, the receiver needs to do some actions, to write the contents of its buffers to disk for example. In this period of time no new information can be received. Some communication back to the sender is needed to stop the flow of bytes on the line. A method must be present to tell the sender to pause. To do this, both software and hardware protocols have been defined.

#### 10.3.9.1 Software flow control

Both software and hardware flow control need software to perform the handshaking task. This makes the term software flow control somewhat misleading. What is meant is that with hardware flow control, additional lines are present in the communication cable which signal



handshaking conditions. With software flow control, which is also known under the name XON-XOFF flow control, bytes are sent to the sender using the standard communication lines.

Using hardware flow control implies, that more lines must be present between the sender and the receiver, leading to a thicker and more expensive cable. Therefore, software flow control is a good alternative if it is not needed to gain maximum performance in communications. Software flow control makes use of the datachannel between the two devices which reduces the bandwidth. The reduce of bandwidth is in most cases however not so astonishing that it is a reason to not use it.

Two bytes have been predefined in the ASCII character set to be used with software flow control. These bytes are named XOFF and XON, because they can stop and restart transmitting. The bytevalue of XOFF is 19, it can be simulated by pressing Ctrl-S on the keyboard. XON has the value 17 assigned which is equivalent to Ctrl-Q.

Using software flow control is easy. If sending of characters must be postponed, the character XOFF is sent on the line, to restart the communication again XON is used. Sending the XOFF character only stops the communication in the direction of the device which issued the XOFF.

This method has a few disadvantages. One is already discussed: using bytes on the communication channel takes up some bandwidth. One other reason is more severe. Handshaking is mostly used to prevent an overrun of the receiver buffer, the buffer in memory used to store the recently received bytes. If an overrun occurs, this affects the way new coming characters on the communication channel are handled. In the worst case where software has been designed badly, these characters are thrown away without checking them. If such a character is XOFF or XON, the flow of communication can be severely damaged. The sender will continuously supply new information if the XOFF is lost, or never send new information if no XON was received.

This also holds for communication lines where signal quality is bad. What happens if the XOFF or XON message is not received clearly because of noise on the line? Special precaution is also necessary that the information sent does not contain the XON or XOFF characters as information bytes.

Therefore, serial communication using software flow control is only acceptable when communication speeds are not too high, and the probability that buffer overruns or data damage occur are minimal.

#### **10.3.9.2 Hardware flow control**

Hardware flow control is superior compared to software flow control using the XON and XOFF characters. The main problem is, that an extra investment is needed. Extra lines are necessary in the communication cable to carry the handshaking information.

Hardware flow control is sometimes referred to as RTS / CTS flow control. This term mentions the extra input and outputs used on the serial device to perform this type of handshaking. RTS / CTS in its original outlook is used for handshaking between a computer and a device connected to it such as a modem.

First, the computer sets its RTS line to signal the device that some information is present. The device checks if there is room to receive the information and if so, it sets the CTS line to

start the transfer. When using a null modem connection, this is somewhat different. There are two ways to handle this type of handshaking in that situation.

One is, where the RTS of each side is connected with the CTS side of the other. In that way, the communication protocol differs somewhat from the original one. The RTS output of computer A signals computer B that A is capable of receiving information, rather than a request for sending information as in the original configuration. This type of communication can be performed with a null modem cable for full handshaking. Although using this cable is not completely compatible with the original way hardware flow control was designed, if software is properly designed for it it can achieve the highest possible speed because no overhead is present for requesting on the RTS line and answering on the CTS line.

In the second situation of null modem communication with hardware flow control, the software side looks quite similar to the original use of the handshaking lines. The CTS and RTS lines of one device are connected directly to each other. This means, that the request to send query answers itself. As soon as the RTS output is set, the CTS input will detect a high logical value indicating that sending of information is allowed. This implies, that information will always be sent as soon as sending is requested by a device if no further checking is present. To prevent this from happening, two other pins on the connector are used, the data set ready DSR and the data terminal ready DTR. These two lines indicate if the device attached is working properly and willing to accept data. When these lines are cross-connected (as in most null modem cables) flow control can be performed using these lines. A DTR output is set, if that computer accepts incoming characters.

### 10.3.10 Special connections

#### 10.3.10.1 RS232 DB25 to DB9 converter

Mostly you will come across DB9 connectors. Sometimes you will encounter an interface with a DB25 connector (some printers). You will need to make a convertor. See the example and conversion table below for help.

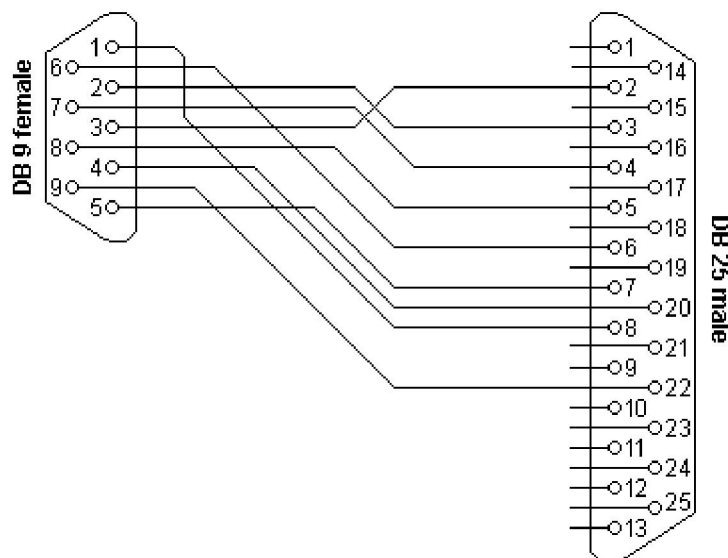


Figure 10-2: RS232 DB25 to DB9 Converter

DB9	DB25	Function
1	8	Data Carrier Detect
2	3	Receive Data
3	2	Transmit Data
4	20	Data Terminal Ready
5	7	Signal Ground
6	6	Data Set Ready
7	4	Request To Send
8	5	Clear To Send
9	22	Ring Indicator

Table 10-4: DB9 – DB25 conversion

### 10.3.10.2 RS232 serial loopback test plugs

The following RS232 connectors can be used to test a serial port on your computer. The data and handshake lines have been linked. In this way all data will be sent back immediately. The PC controls its own handshaking.

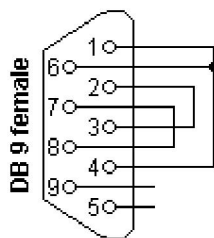


Figure 10-3: DB9 RS232 Loopback Plug

DB9	DB25	Function
1 + 4 + 6	6 + 8 + 20	DTR => CD + DSR
2 + 3	2 + 3	Tx => Rx
7 + 8	4 + 5	RTS => CTS

Table 10-5: DB9 RS232 Loopback Plug

### 10.3.10.3 RS232 Null Modem Cables

How to use the handshaking lines in a null modem configuration? The simplest way is to don't use them at all. In that situation, only the data lines and signal ground are cross connected in the null modem communication cable. All other pins have no connection. An example of such a null modem cable without handshaking can be seen in the figure below.

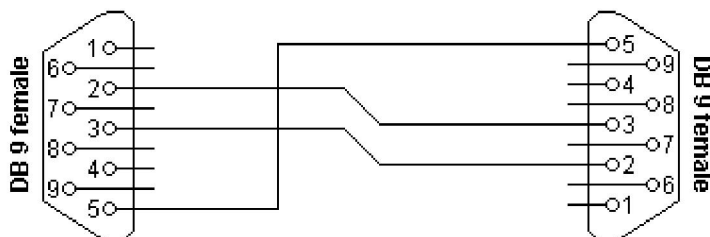


Figure 10-4: Null modem cable without handshaking

Connector 1	Connector 2	Function
2	3	Rx <= Tx

3	2	Tx => Rx
5	5	Signal Ground

**Table 10-6: : Null modem cable without handshaking**

### 10.3.10.3.1 Compatibility Issues

If you read about null modems, this three wire null modem cable is often talked about. Yes, it is simple but can we use it in all circumstances? There is a problem, if either of the two devices checks the DSR or CD inputs. These signals normally define the ability of the other side to communicate. As they are not connected, their signal level will never go high. This might cause a problem.

The same holds for the RTS/CTS handshaking sequence. If the software on both sides is well structured, the RTS output is set high and then a waiting cycle is started until a ready signal is received on the CTS line. This causes the software to hang because no physical connection is present to either CTS line to make this possible. The only type of communication which is allowed on such a null modem line is data-only traffic on the cross connected Rx/Tx lines.

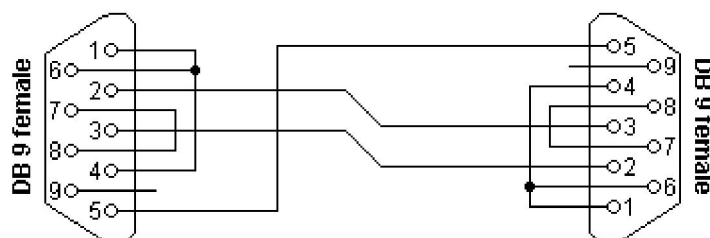
This does however not mean, that this null modem cable is useless. Communication links like present in the Norton Commander program can use this null modem cable. This null modem cable can also be used when communicating with devices which do not have modem control signals like electronic measuring equipment etc.

As you can imagine, with this simple null modem cable no hardware flow control can be implemented. The only way to perform flow control is with software flow control using the XOFF and XON characters.

### 10.3.10.3.2 Null modem with loop back handshaking

The simple null modem cable without handshaking shows incompatibilities with common software. The main problem with this cable is that there is a possibility for the software to hang if it checks the modem signal lines in a proper way. I.e. with this null modem cable, good written programs will perform worse than badly written programs.

To overcome this problem and still be able to use a cheap null modem communication cable with only three lines in it, a fake null modem cable layout has been defined. The null modem cable with loop back handshaking resulted from this.



**Figure 10-5: Null modem with loop back handshaking**

Connector 1	Connector 2	Function
1	7 + 8	RTS2 => CTS2 + CD1
2	3	Rx <= Tx
3	2	Tx => Rx
4	6	DTR => DSR

5	5	Signal Ground
6	4	DSR <= DTR
7 + 8	1	RTS1 => CTS1 + CD2

Table 10-7: Null modem with loop back handshaking

### 10.3.10.3.3 Compatibility issues

This null modem cable is the best of two worlds. There is the possibility of hardware flow control without being incompatible with the original way flow control was used with DTE/DCE communication. Let us first consider the RTS/CTS flow control lines present on pins 7 and 8. As with the loop back null modem cable, these signals are not connected to the other device, but directly looped back on the same connector. This means, that RTS/CTS flow control is allowed to be used in the software, but it has no functional meaning. Only when the software at the other side checks the CD signal at pin 1, the RTS information will reach the other device. This would however be only the case in specifically developed software which uses the CD input for this purpose.

More important however is the cross connection of the DSR (pin 6) and DTR (pin 4) lines. By cross connecting these lines, their original function is simulated pretty well. The DTR output is used to signal the other device that communication is possible. This information is read on the DSR input, the same input used for this purpose with modem communication. Because of this cross connection, the DTR output line can be used for simple flow control. Incoming data is allowed when the output is set, and blocked if the output is not set.

Software using only the RTS/CTS protocol for flow control cannot take advantage of the partial handshaking null modem cable. Most software however will also check the DSR line and in that case—when using the null modem cable with partial handshaking—the best possible hardware flow control can be achieved which is still compatible with the original use with modems.

### 10.3.10.3.4 Null modem with full handshaking

The most expensive null modem cable is the null modem cable suitable for full handshaking. In this null modem cable, seven wires are present. Only the ring indicator RI and carrier detect CD signal are not linked. The cable is shown in the following figure.

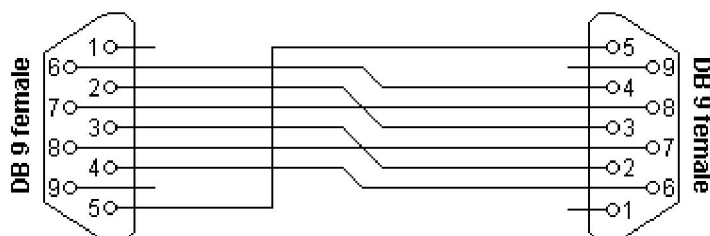


Figure 10-6: Null modem with full handshaking

Connector 1	Connector 2	Function
2	3	Rx <= Tx
3	2	Tx => Rx
4	6	DTR => DSR
5	5	Signal Ground
6	4	DSR <= DTR
7	8	RTS => CTS
8	7	CTS <= RTS

Table 10-8: Null modem with full handshaking

### 10.3.10.3.5 Compatibility issues

The null modem cable with full handshaking does not permit the older way of flow control to take place. The main incompatibility is the cross connection of the RTS and CTS pins. Originally, these pins are used for a question/answer type of flow control. When the full handshaking null modem cable is used, there is no request anymore. The lines are purely used for telling the other side if communication is possible.

The main advantage of this cable is, that there are two signalling lines in each direction. Both the RTS and DTR outputs can be used to send flow control information to the other device. This makes it possible to achieve very high communication speeds with this type of null modem cable, provided that the software has been designed for it.

## 10.4 RS485 Specifications

RS485 is the most versatile communication standard in the standard series defined by the EIA, as it performs well on all four points. That is why RS485 is currently a widely used communication interface in data acquisition and control applications where multiple nodes communicate with each other.

### 10.4.1 Differential signals with RS485: Longer distances and higher bit rates

One of the main problems with RS232 is the lack of immunity for noise on the signal lines. The transmitter and receiver compare the voltages of the data- and handshake lines with one common zero line. Shifts in the ground level can have disastrous effects. Therefore the trigger level of the RS232 interface is set relatively high at  $\pm 3$  Volt. Noise is easily picked up and limits both the maximum distance and communication speed. With RS485 on the contrary there is no such thing as a common zero as a signal reference. Several volts difference in the ground level of the RS485 transmitter and receiver does not cause any problems. The RS485 signals are floating and each signal is transmitted over a Sig+ line and a Sig- line. The RS485 receiver compares the voltage difference between both lines, instead of the absolute voltage level on a signal line. This works well and prevents the existence of ground loops, a common source of communication problems. The best results are achieved if the Sig+ and Sig- lines are twisted. The image below explains why.

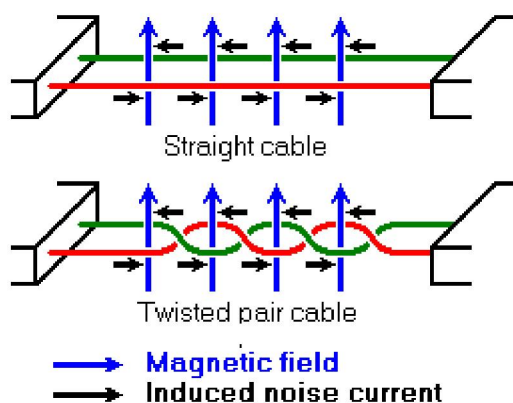


Figure 10-7: Noise in straight and twisted pair cables

In the picture above, noise is generated by magnetic fields from the environment. The picture shows the magnetic field lines and the noise current in the RS485 data lines that is the result of that magnetic field. In the straight cable, all noise current is flowing in the same direction,



practically generating a looping current just like in an ordinary transformer. When the cable is twisted, we see that in some parts of the signal lines the direction of the noise current is the opposite from the current in other parts of the cable. Because of this, the resulting noise current is many factors lower than with an ordinary straight cable. Shielding—which is a common method to prevent noise in RS232 lines—tries to keep hostile magnetic fields away from the signal lines. Twisted pairs in RS485 communication however adds immunity which is a much better way to fight noise. The magnetic fields are allowed to pass, but do no harm. If high noise immunity is needed, often a combination of twisting and shielding is used as for example in STP, shielded twisted pair and FTP, foiled twisted pair networking cables. Differential signals and twisting allows RS485 to communicate over much longer communication distances than achievable with RS232. With RS485 communication distances of 1200 m are possible.

Differential signal lines also allow higher bit rates than possible with non-differential connections. Therefore RS485 can overcome the practical communication speed limit of RS232. Currently RS485 drivers are produced that can achieve a bit rate of 35 mbps.

#### 10.4.2 Network topology with RS485

Network topology is probably the reason why RS485 is now the favorite of the four mentioned interfaces in data acquisition and control applications. RS485 is the only of the interfaces capable of internetworking multiple transmitters and receivers in the same network. When using the default RS485 receivers with an input resistance of 12 k $\Omega$  it is possible to connect 32 devices to the network. Currently available high-resistance RS485 inputs allow this number to be expanded to 256. RS485 repeaters are also available which make it possible to increase the number of nodes to several thousands, spanning multiple kilometers. And that with an interface which does not require intelligent network hardware: the implementation on the software side is not much more difficult than with RS232. It is the reason why RS485 is so popular with computers, PLCs, micro controllers and intelligent sensors in scientific and technical applications.

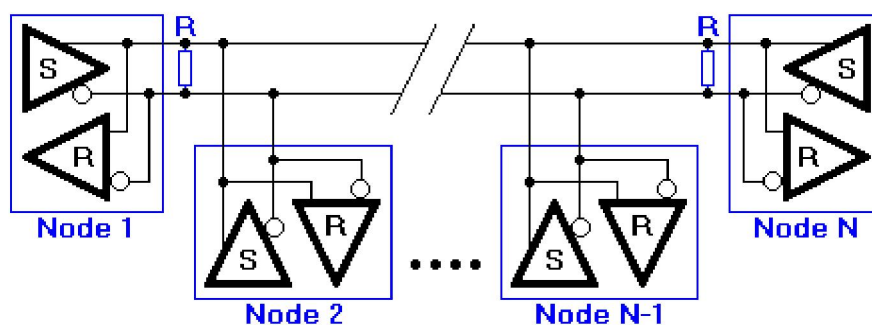


Figure 10-8: RS485 network topology

In the picture above, the general network topology of RS485 is shown. N nodes are connected in a multipoint RS485 network. For higher speeds and longer lines, the termination resistances are necessary on both ends of the line to eliminate reflections. Use 100  $\Omega$  resistors on both ends. The RS485 network must be designed as one line with multiple drops, not as a star. Although total cable length maybe shorter in a star configuration, adequate termination is not possible anymore and signal quality may degrade significantly.

#### 10.4.3 RS485 functionality

And now the most important question, how does RS485 function in practice? Default, all the senders on the RS485 bus are in tri-state with high impedance. In most higher level protocols, one of the nodes is defined as a master which sends queries or commands over the RS485 bus. All other nodes receive these data. Depending of the information in the sent data, zero or more nodes on the line respond to the master. In this situation, bandwidth can be used for almost 100%. There are other implementations of RS485 networks where every node can start a data session on its own. This is comparable with the way Ethernet networks function. Because there is a chance of data collision with this implementation, theory tells us that in this case only 37% of the bandwidth will be effectively used. With such an implementation of a RS485 network it is necessary that there is error detection implemented in the higher level protocol to detect the data corruption and resend the information at a later time.

There is no need for the senders to explicitly turn the RS485 driver on or off. RS485 drivers automatically return to their high impedance tri-state within a few microseconds after the data has been sent. Therefore it is not needed to have delays between the data packets on the RS485 bus.

RS485 is used as the electrical layer for many well-known interface standards, including Profibus and Modbus.

#### 10.4.4 What Pins Are Needed for 2- and 4- Wire Transmission with RS-485 Serial Communication?

For 2- wire and 4- wire transmission, TXD+, TXD-, RXD+, and RXD- are used.

In 4-wire transmission (full duplex), 4 wires run from TXD or RXD on the master to the opposite (TXD or RXD) on the slave(s).

TXD+  $\Leftrightarrow$  RXD+

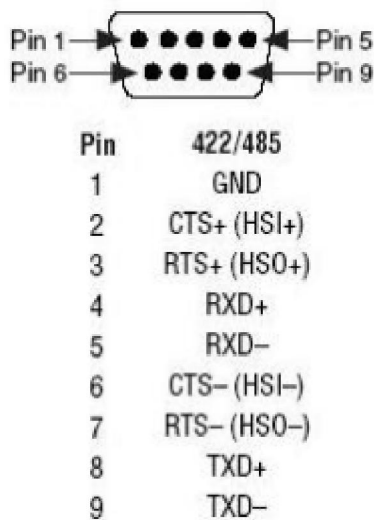
TXD-  $\Leftrightarrow$  RXD-

RXD+  $\Leftrightarrow$  TXD+

RXD-  $\Leftrightarrow$  TXD-

In 2-wire transmission (half- duplex), TXD+ and RXD+ on the master are wired together to TXD+ and RXD+ on the slave(s). TXD- and RXD- on the master are wired together to TXD- and RXD- on the slave(s).





**Figure 10-9: 422/485 DB9 Pinout**

*Note: While there is no handshaking protocol in RS485, please refer to the interface you are working on to see which pins are in use. Flowcontrol is done by the software by keeping gaps up to 3,5 characters after a transmittal. See chapter for additional FT information.*

## 11. TCP/IP

### 11.1.1 Introduction

Everyone will have heard of the TCP/IP protocol. It is used in our well known Internet, browsing on your explorer. This is not the only way it is used. TCP/IP is also used in various communications between peripherals. That is why we will elaborate a little bit on the subject, to get a good understanding of the protocol.

### 11.1.2 Internet Protocol Suite

The Internet protocol suite is the set of communications protocols that implement the protocol stack on which the Internet runs. It is sometimes called the TCP/IP protocol suite, after the two most important protocols in it: the Transmission Control Protocol (TCP) and the Internet Protocol (IP), which were also the first two defined.

The Internet protocol suite can be described by analogy with the OSI model, which describes the layers of a protocol stack, not all of which correspond well with internet practice. In a protocol stack, each layer solves a set of problems involving the transmission of data, and provides a well-defined service to the higher layers. Higher layers are logically closer to the user and deal with more abstract data, relying on lower layers to translate data into forms that can eventually be physically manipulated.

The Internet model was produced as the solution to a practical engineering problem. The OSI model, on the other hand, was a more theoretical approach, and was also produced at an earlier stage in the evolution of networks. Therefore, the OSI model is easier to understand, but the TCP/IP model is the one in actual use. It is helpful to have an understanding of the OSI model before learning TCP/IP, as the same principles apply, but are easier to understand in the OSI model.

Layer	Protocols
Application	FTP, HTTP, HTTPS, IMAP, IRC, NNTP, POP3, SIP, SMTP, SNMP, SSH, Telnet, BitTorrent, Websphere.....
Transport	DCCP, SCTP, TCP, RTP, UDP, IL, RUDP.....
Network	IPv4, IPv6.....
Datalink	Ethernet, WiFi, Token Ring, FDDI, PPP.....
Physical	RS-232, EIA-422, RS-449, EIA-485, 10BASE2, 10BASE-T, ...

**Table 11-1: Internet Protocol Suite**

### 11.1.3 Layers in the TCP/IP stack

There is some discussion about how to map the TCP/IP model onto the OSI model. Since the TCP/IP and OSI protocol suites do not match precisely, there is no one correct answer.

In addition, the OSI model is not really rich enough at the lower layers to capture the true layering; there needs to be an extra layer (the Internetworking layer) between the Transport and Network layers. Protocols specific to a particular network type, but which are run on top of the basic hardware framing, ought to be at the Network layer. Examples of such protocols are ARP, and the Spanning Tree Protocol (used to keep redundant bridges idle until they are needed). However, they are local protocols, and operate beneath the internetwork functionality. Admittedly, placing both groups (not to mention protocols which are logically part of the internetwork layer, but run on top of the internetwork protocol, such as ICMP) all at the same layer can be confusing, but the OSI model is not complex enough to do a better job.

The following diagram attempts to show where various TCP/IP and other protocols would reside in the original OSI model:

### 11.1.4 The Network Access layer

#### 11.1.4.1 The physical layer

The Physical layer describes the physical characteristics of the communication, such as conventions about the nature of the medium used for communication (such as wires, fiber optic links or radio links), and all related details such as connectors, channel codes and modulation, signal strengths, wavelength, low-level synchronization and timing and maximum distances. The Internet protocol suite does not cover the physical layer of any network; see the articles on specific network technologies for detail on the physical layer of each particular technology.

#### 11.1.4.2 The data link layer

The data link layer specifies how packets are transported over the physical layer, including the framing (i.e. the special bit patterns which mark the start and end of packets). Ethernet, for example, includes fields in the packet header which specify which machine or machines on the network a packet is destined for. Examples of Data link layer protocols are Ethernet, Wireless Ethernet, SLIP, Token Ring and ATM.

PPP is a little more complex, as it was originally specified as a separate protocol which ran on top of another data link layer, HDLC/SDLC.

This layer is sometimes further subdivided into Logical Link Control and Media Access Control.

#### **11.1.4.3 The Internetwork layer**

As originally defined, the Network layer solves the problem of getting packets across a single network. Examples of such protocols are X.25, and the ARPANET's Host/IMP Protocol.

With the advent of the concept of internetworking, additional functionality was added to this layer, namely getting data from the source network to the destination network. This generally involves routing the packet across a network of networks, known as an internet.

In the internet protocol suite, IP performs the basic task of getting packets of data from source to destination. IP can carry data for a number of different higher level protocols; these protocols are each identified by a unique IP Protocol Number. ICMP and IGMP are protocols 1 and 2, respectively.

Some of the protocols carried by IP, such as ICMP (used to transmit diagnostic information about IP transmission) and IGMP (used to manage multicast data) are layered on top of IP but perform network layer functions, illustrating an incompatibility between the internet and OSI models. All routing protocols, such as BGP, OSPF, and RIP are also really part of the network layer, although they might seem to belong higher in the stack.

#### **11.1.4.4 The transport layer**

The protocols at the Transport layer can solve problems like reliability ("did the data reach the destination?") and ensure that data arrives in the correct order. In the TCP/IP protocol suite, transport protocols also determine which application any given data is intended for.

The dynamic routing protocols which technically fit at this layer in the TCP/IP Protocol Suite (since they run over IP) are generally considered to be part of the Network layer; an example is OSPF (IP protocol number 89).

TCP (IP protocol number 6) is a "reliable", connection-oriented, transport mechanism providing a reliable byte stream, which makes sure data arrives complete, undamaged, and in order. TCP tries to continuously measure how loaded the network is and throttles its sending rate in order to avoid overloading the network. Furthermore, TCP will attempt to deliver all data correctly in the specified sequence. These are its main differences from UDP, and can become disadvantageous in real-time streaming or routing applications with high internetwork layer loss rates.

The newer SCTP is also a "reliable", connection-oriented, transport mechanism. It is record rather than byte oriented, and provides multiple sub-streams multiplexed over a single connection. It also provides multi-homing support, in which a connection end can be represented by multiple IP addresses (representing multiple physical interfaces), such that if one fails the connection is not interrupted. It was developed initially for telephony applications (to transport SS7 over IP), but can also be used for other applications.

UDP (IP protocol number 17) is a connectionless datagram protocol. It is a "best effort" or "unreliable" protocol - not because it is particularly unreliable, but because it does not verify

that packets have reached their destination, and gives no guarantee that they will arrive in order. If an Application requires these characteristics, it must provide them itself, or use TCP.

UDP is typically used for applications such as streaming media (audio and video, etc) where on-time arrival is more important than reliability, or for simple query/response applications like DNS lookups, where the overhead of setting up a reliable connection is disproportionately large.

DCCP is currently under development by IETF. It provides TCP's flow control semantics, while keeping UDP's datagram service model visible to the user.

Both TCP and UDP are used to carry a number of higher-level applications. The applications at any given network address are distinguished by their TCP or UDP Port Number. By convention certain well known ports are associated with specific applications.

RTP is a datagram protocol that is designed for real-time data such as streaming audio and video. Although RTP uses the UDP packet format as a basis, it provides a function that is at the same protocol layer.

#### **11.1.4.5 The application layer**

The Application layer is the layer that most common network-aware programs use in order to communicate across a network with other programs. Processes that occur in this layer are application specific; data is passed from the network-aware program, in the format used internally by this application, and is encoded into a standard protocol.

Some specific programs are considered to run in this layer. They provide services that directly support user applications. These programs and their corresponding protocols include HTTP (The World Wide Web), FTP (File transport), SMTP (Email), SSH (Secure remote login), DNS (Name <-> IP Address lookups) and many others.

Once the data from an application has been encoded into a standard application layer protocol it will be passed down to the next layer of the IP stack.

At the Transport Layer, applications will most commonly make use of TCP or UDP, and are often associated with a well-known port number. Ports were originally allocated by the Internet Assigned Numbers Authority (IANA).

#### **11.1.5 TCP/IP Relevance**

As you might know FT NavVision © is a network based AMS system. For the interconnection between all the workstations and all the peripherals a good understanding of the above is necessary. Not only the workstations are connected through the TCP protocol also a lot of other devices are connected throughout some sort of Ethernet protocol. Think of IP-cameras, serial to Ethernet converters, modbus over TCP/IP and much more. It will take too far to discuss all the possibilities. We will, however, discuss a few of the more difficult features.

In the "hardware installation and commissioning manual" under "performance" we will share some in depth information on the problems we encountered during our testing, to help you solve these connections quickly.

## 12. Modbus

### 12.1 Introduction

Some communication standards just emerge. Not because they are pushed by a large group of vendors or a special standards organisation. These standards—like the Modbus interface—emerge because they are good, simple to implement and are therefore adapted by many manufacturers. Because of this, Modbus became the first widely accepted fieldbus standard.

Modbus has its roots in the late seventies of the previous century. It is 1979 when PLC manufacturer Modicon—now a brand of Schneider Electric's Telemecanique—published the Modbus communication interface for a multidrop network based on a master/client architecture. Communication between the Modbus nodes was achieved with messages. It was an open standard that described the messaging structure. The physical layer of the Modbus interface was free to choose. The original Modbus interface ran on RS-232, but most later Modbus implementations used RS-485 because it allowed longer distances, higher speeds and the possibility of a true multi-drop network. In a short time hundreds of vendors implemented the Modbus messaging system in their devices and Modbus became the de facto standard for industrial communication networks.

The nice thing of the Modbus standard is the flexibility, but at the same time the easy implementation of it. Not only intelligent devices like microcontrollers, PLCs etc. are able to communicate with Modbus, also many intelligent sensors are equipped with a Modbus interface to send their data to host systems. While Modbus was previously mainly used on wired serial communication lines, there are also extensions to the standard for wireless communications and TCP/IP networks.

### 12.2 Modbus message structure

The Modbus communication interface is built around messages. The format of these Modbus messages is independent of the type of physical interface used. On plain old RS232 are the same messages used as on Modbus/TCP over ethernet. This gives the Modbus interface definition a very long lifetime. The same protocol can be used regardless of the connection type. Because of this, Modbus gives the possibility to easily upgrade the hardware structure of an industrial network, without the need for large changes in the software. A device can also communicate with several Modbus nodes at once, even if they are connected with different interface types, without the need to use a different protocol for every connection.

On simple interfaces like RS485 or RS232, the Modbus messages are sent in plain form over the network. In this case the network is dedicated to Modbus. When using more versatile network systems like TCP/IP over ethernet, the Modbus messages are embedded in packets with the format necessary for the physical interface. In that case Modbus and other types of connections can co-exist at the same physical interface at the same time. Although the main Modbus message structure is peer-to-peer, Modbus is able to function on both point-to-point and multidrop networks.

Each Modbus message has the same structure. Four basic elements are present in each message. The sequence of these elements is the same for all messages, to make it easy to parse the content of the Modbus message. A conversation is always started by a master in the Modbus network. A Modbus master sends a message and—depending of the contents of the message—a slave takes action and responds to it. There can be more masters in a Modbus network. Addressing in the message header is used to define which device should

respond to a message. All other nodes on the Modbus network ignore the message if the address field doesn't match their own address.

Field	Description
Device Address	Address of the receiver
Function Code	Code defining message type
Data	Data block with additional information
Error Check	Numeric check value to test for communication errors

**Table 12-1: Modbus Message Structure**

### 12.3 Modbus serial transmission modes: Modbus/ASCII and Modbus/RTU

Serial Modbus connections can use two basic transmission modes, ASCII or RTU, remote terminal unit. The transmission mode in serial communications defines the way the Modbus messages are coded. With Modbus/ASCII, the messages are in a readable ASCII format. The Modbus/RTU format uses binary coding which makes the message unreadable when monitoring, but reduces the size of each message which allows for more data exchange in the same time span. All nodes on one Modbus network segment must use the same serial transmission mode. A device configured to use Modbus/ASCII cannot understand messages in Modbus/RTU and vice versa.

When using Modbus/ASCII, all messages are coded in hexadecimal values, represented with readable ASCII characters. Only the characters 0...9 and A...F are used for coding. For every byte of information, two communication-bytes are needed, because every communication-byte can only define 4 bits in the hexadecimal system. With Modbus/RTU the data is exchanged in a binary format, where each byte of information is coded in one communication-byte.

Modbus messages on serial connections are not sent in a plain format. They are framed to give receivers an easy way to detect the beginning and end of a message. When using Modbus/ASCII, characters are used to start and end a frame. The colon ':' is used to flag the start of a message and each message is ended with a CR/LF combination. Modbus/RTU on the other hand uses time gaps of silence on the communication line for the framing. Each message must be preceded by a time gap with a minimum length of 3.5 characters. If a receiver detects a gap of at least 1.5 characters, it assumes that a new message is coming and the receive buffer is cleared. The main advantage of Modbus/ASCII is, that it allows gaps between the bytes of a message with a maximum length of 1 second. With Modbus/RTU it is necessary to send each message as a continuous stream.

Connector 1	Modbus/Ascii	Modbus/RTU
Characters	ASCII 0...9 And A...F	Binary 0...255
Error Check	LRC	CRC
Frame Start	Character ":"	3.5 Chars Silence
Frame End	Characters CR/LF	3.5 Chars Silence
Gaps in Message	1 Sec.	1.5 times CharLength
Start Bit	1	1
Data Bits	7	8
Parity	Even/Odd    None	Even/Odd    None
Stop Bits	1            2	1            2

**Table 12-2: Properties of Modbus/ASCII and Modbus/RTU**



## 12.4 Modbus addressing

The first information in each Modbus message is the address of the receiver. This parameter contains one byte of information. In Modbus/ASCII it is coded with two hexadecimal characters, in Modbus/RTU one byte is used. Valid addresses are in the range 0..247. The values 1..247 are assigned to individual Modbus devices and 0 is used as a broadcast address. Messages sent to the latter address will be accepted by all slaves. A slave always responds to a Modbus message. When responding it uses the same address as the master in the request. In this way the master can see that the device is actually responding to the request.

Within a Modbus device, the holding registers, inputs and outputs are assigned a number between 1 and 10000. One would expect, that the same addresses are used in the Modbus messages to read or set values. Unfortunately this is not the case. In the Modbus messages addresses are used with a value between 0 and 9999. If you want to read the value of output (coil) 18 for example, you have to specify the value 17 in the Modbus query message. More confusing is even, that for input and holding registers an offset must be subtracted from the device address to get the proper address to put in the Modbus message structure. This leads to common mistakes and should be taken care of when designing applications with Modbus. The following table shows the address ranges for coils, inputs and holding registers and the way the address in the Modbus message is calculated given the actual address of the item in the slave device.

Device Address	Modbus Address	Description
1...10000	Address -1	Coil (outputs)
10001...20000	Address -10001	Inputs
40001...50000	Address -40001	Holding Registers

Table 12-3: Device and Modbus address ranges

## 12.5 Modbus function codes

The second parameter in each Modbus message is the function code. This defines the message type and the type of action required by the slave. The parameter contains one byte of information. In Modbus/ASCII this is coded with two hexadecimal characters, in Modbus/RTU one byte is used. Valid function codes are in the range 1..255. Not all Modbus devices recognize the same set of function codes. The most common codes are discussed here.

Normally, when a Modbus slave answers a response, it uses the same function code as in the request. However, when an error is detected, the highest bit of the function code is turned on. In that way the master can see the difference between success and failure responses.

Code	Description
01	Read coil status
02	Read input status
03	Read holding registers
04	Read input registers
05	Force single coil
06	Preset single register
07	Read exception status
15	Force multiple coils
16	Preset multiple registers
17	Report slave ID

Table 12-4: Common Modbus function codes

### 12.5.1 Function 01: Read coil status

In Modbus language, a coil is a discrete output value. Modbus function **01** can be used to read the status of such an output. It is only possible to query one device at a time. Broadcast addressing is not supported with this Modbus function. The function can be used to request the status of various coils at once. This is done by defining an output range in the data field of the message.

Byte	Value	Description
1	1...247	Slave Device Address
2	1	Function Code
3	0...255	Starting Address, High Byte
4	0...255	Starting Address, Low Byte
5	0...255	Number of Coils, High Byte
6	0...255	Number of Coils, Low Byte
7 (...8)	LRC/CRC	Error Check Value

Table 12-5: Function 01 query structure

When receiving a Modbus query message with function **01**, the slave collects the necessary output values and constructs an answer message. The length of this message is dependent on the number of values that have to be returned. In general, when **N** values are requested, a number of  $((N+7) \bmod 8)$  bytes are necessary to store these values. The actual number of databytes in the datablock is put in the first byte of the data field. Therefore the general structure of an answer to a Modbus function **01** query is

:

Byte	Value	Description
1	1...247	Slave Device Address
2	1	Function Code
3	0...255	Number of Data Bytes <b>N</b>
4...N+3	0...255	Bit Pattern of Coil Values
N+4 (...N+5)	LRC/CRC	Error Check Value

Table 12-6: Function 01 answer structure

### 12.5.2 Function 02: Read input status

Reading input values with Modbus is done in the same way as reading the status of coils. The only difference is that for inputs Modbus function **02** is used. Broadcast addressing mode is not supported. You can only query the value of inputs of one device at a time. Like with coils, the address of the first input, and the number of inputs to read must be put in the data field of the query message. Inputs on devices start numbering at **10001**. This address value is equivalent to address **0** in the Modbus message.

Byte	Value	Description
1	1...247	Slave Device Address
2	2	Function Code
3	0...255	Starting Address, High Byte
4	0...255	Starting Address, Low Byte
5	0...255	Number of Inputs, High Byte
6	0...255	Number of Inputs, Low Byte
7 (...8)	LRC/CRC	Error Check Value



**Table 12-7: Function 02 query structure**

After receiving a query message with Modbus function **02**, the slave puts the requested input values in a message structure and sends this message back to the Modbus master. The length of the message depends on the number of input values returned. This causes the length of the output message to vary. The number of databytes in the data field that contain the input values is passed as the first byte in the data field. Each Modbus answering message has the following general structure.

Byte	Value	Description
1	1...247	Slave Device Address
2	2	Function Code
3	0...255	Number of Data Bytes <b>N</b>
4...N+3	0...255	Bit Pattern of Input Values
N+4 (...N+5)	LRC/CRC	Error Check Value

**Table 12-8: Function 02 answer structure**

### 12.5.3 Function 03: Read holding registers

Internal values in a Modbus device are stored in holding registers. These registers are two bytes wide and can be used for various purposes. Some registers contain configuration parameters where others are used to return measured values (temperatures etc.) to a host. Registers in a Modbus compatible device start counting at **40001**. They are addressed in the Modbus message structure with addresses starting at **0**. Modbus function **03** is used to request one or more holding register values from a device. Only one slave device can be addressed in a single query. Broadcast queries with function **03** are not supported.

Byte	Value	Description
1	1...247	Slave Device Address
2	3	Function Code
3	0...255	Starting Address, High Byte
4	0...255	Starting Address, Low Byte
5	0...255	Number of Registers, High Byte
6	0...255	Number of Registers, Low Byte
7 (...8)	LRC/CRC	Error Check Value

**Table 12-9: Function 03 query structure**

After processing the query, the Modbus slave returns the 16 bit values of the requested holding registers. Because of the size of the holding registers, every register is coded with two bytes in the answering message. The first data byte contains the high byte, and the second the low byte of the register. The Modbus answer message starts with the slave device address and the function code **03**. The next byte is the number of data bytes that follow. This value is two times the number of registers returned. An error check is appended for the host to check if a communication error occurred

## 12.6 Addressing (0- or 1-based)

The addressing within the Modbus/TCP protocol (that is, the data within the physical packet) is 0-based, meaning the first element/item to be accessed is referenced by address 0. The Modbus standard for handling and displaying the data is 1-based, meaning the first element/data item to be access is referenced by address 1.

Most client applications handle this by having the user enter the 1-based number, and

then subtract 1 to revert to the 0-based addressing required at the protocol level. Some client applications allow the user to enter the 0-based number, or a combination, depending on how it is configured. The addresses defined within the following table are 1-based, as the majority of the client applications work with this method.

Assumptions: When looking at a client's Modbus-layout you can note a few things. For example the register starts with 40017. In this case it will be likely that you can subtract 40001 and you will have the address to put in the sensorlist. Also when you get an excel-sheet with a range like 1, 17, 538, etc. it is likely that you can add that address as the right Modbus address assuming you use the right function code. If however you see 0-based, a start address of 30000 or something, it will be most likely you will have to subtract 1 from the address. Nonetheless, always check with the manufacturer!

## 12.7 Understanding Raw Data

When troubleshooting problems, it can be helpful to see the actual raw data being transmitted. Long strings of ones and zeroes are difficult to read, so the bits are combined and shown in hexadecimal. Each block of 4 bits is represented by one of the sixteen characters from 0 to F.

Each block of 8 bits (called a byte) is represented by one of the 256 character pairs from 00 to FF.

0000=0	0001=1	0010=2	0011=3
0100=4	0101=5	0110=6	0111=7
1000=8	1001=9	1010=A	1011=B
1100=C	1101=D	1110=E	1111=F

Table 12-10: Hexadecimal values

### 12.7.1 How is data stored in Standard Modbus?

Information is stored in the Slave device in four different tables. Two tables store on/off discrete values (coils) and two store numerical values (registers). The coils and registers each have a read-only table and read-write table.

Each table has 9999 values.

Each coil or contact is 1 bit and assigned a data address between 0000 and 270E.

Each register is 1 word = 16 bits = 2 bytes and also has data address between 0000 and 270E.

Coil/Reg. Numbers	Data Addresses	Type	Table Name
1-9999	0000-270E	Read/Write	Discrete Output Coil
10001-19999	0000-270E	Read-Only	Discrete Input Contacts
30001-39999	0000-270E	Read-Only	Analog Input Registers
40001-49999	0000-270E	Read/Write	Analog Output Holding Registers

Table 12-11: Modbus Storage

Coil/Register Numbers can be thought of as location names since they do not appear in the actual messages. The Data Addresses are used in the messages.

For example, the first Holding Register, number 40001, has the Data Address 0000. The difference between these two values is the **offset**. Each table has a different offset. 1, 10001, 30001 and 40001.

### 12.7.2 Examples of questions and answers

To understand how the raw data has to be interpreted here will follow some examples within the different function codes:

#### 12.7.2.1 Example 1

Request

This command is requesting the ON/OFF status of discrete inputs # 10197 to 10218 from the slave device with address 17.

11 02 00C4 0016 BAA9

11: The Slave Address (17 = 11 hex)

02: The Function Code (read Input Status)

00C4: The Data Address of the first input to read. ( $10197 - 10001 = 196 = C4$  hex)

0016: The total number of coils requested. ( $197 - 17 = 180 = 16$  hex)

BAA9: The CRC (cyclic redundancy check) for error checking.

Response

11 02 03 ACDB35 2018

11: The Slave Address (17 = 11 hex)

02: The Function Code (read Input Status)

03: The number of data bytes to follow ( $22 \text{ Inputs} / 8 \text{ bits per byte} = 3 \text{ bytes}$ )

AC: Discrete Inputs 10204 - 10197 (1010 1100)

DB: Discrete Inputs 10212 - 10205 (1101 1011)

35: 2 space holders & Discrete Inputs 10218 - 10213 (0011 0101)

2018: The CRC (cyclic redundancy check).

The more significant bits contain the higher Discrete inputs. This shows that input 10197 is off (0) and 10204 is on (1). Due to the number of inputs requested, the last data field 35 contains the status of only 6 inputs. The two most significant bits in this data field are filled in with zeroes.

#### 12.7.2.2 Example 2

Request

This command is requesting the content of analog output holding registers # 40108 to 40110 from the slave device with address 17.

11 03 006B 0003 7687

11: The Slave Address (17 = 11 hex)

03: The Function Code (read Analog Output Holding Registers)

006B: The Data Address of the first register requested. (40108-40001 = 107 = 6B hex)  
0003: The total number of registers requested. (read 3 registers 40108 to 40110)  
7687: The CRC (cyclic redundancy check) for error checking.

#### Response

11 03 06 AE41 5652 4340 49AD

11: The Slave Address (17 = 11 hex)  
03: The Function Code (read Analog Output Holding Registers)  
06: The number of data bytes to follow (3 registers x 2 bytes each = 6 bytes)  
AE41: The contents of register 40108  
5652: The contents of register 40109  
4340: The contents of register 40110  
49AD: The CRC (cyclic redundancy check).

### 12.7.2.3 Example 3

#### Request

This command is requesting the content of analog input register # 30009 from the slave device with address 17.

11 04 0008 0001 B298

11: The Slave Address (17 = 11 hex)  
04: The Function Code (read Analog Input Registers)  
0008: The Data Address of the first register requested. (30009-30001 = 8)  
0001: The total number of registers requested. (read 1 register)  
B298: The CRC (cyclic redundancy check) for error checking.

#### Response

11 04 02 000A F8F4

11: The Slave Address (17 = 11 hex)  
04: The Function Code (read Analog Input Registers)  
02: The number of data bytes to follow (1 registers x 2 bytes each = 2 bytes)  
000A: The contents of register 30009  
F8F4: The CRC (cyclic redundancy check).

### 12.7.3 Data Types

The example #2 shows that register 40108 contains AE41 which converts to the 16 bits 1010 1110 0100 0001. Great! But what does it mean? Well, it could mean a few things.

Register 40108 could be defined as any of these 16-bit data types:

A 16-bit unsigned integer (a whole number between 0 and 65535)  
register 40108 contains AE41 = 44,609 (hex to decimal conversion)

A 16-bit signed integer (a whole number between -32768 and 32767)  
AE41 = -20,927 (hex to decimal conversion that wraps, if its over 32767 then subtract 65536)

A two character ASCII string (2 typed letters)  
 AE41 = ® A

A discrete on/off value (this works the same as 16-bit integers with a value of 0 or 1. The hex data would be 0000 or 0001)

Register 40108 could also be combined with 40109 to form any of these 32-bit data types:

A 32-bit unsigned integer (a number between 0 and 4,294,967,295)  
 40108,40109 = AE41 5652 = 2,923,517,522

A 32-bit signed integer (a number between -2,147,483,648 and 2,147,483,647)  
 AE41 5652 = -1,371,449,774

A 32-bit double precision IEEE floating point number. This is a mathematical formula that allows any real number (a number with decimal points) to be represented by 32 bits with an accuracy of about seven digits.  
 AE41 5652 = -4.395978 E-11

A four character ASCII string (4 typed letters)  
 AE41 5652 = ® A V R

More registers can be combined to form longer ASCII strings. Each register being used to store two ASCII characters (two bytes).

#### 12.7.4 Byte and Word ordering

The Modbus specification doesn't define exactly how the data is stored in the registers. Therefore, some manufacturers implemented modbus in their equipment to store and transmit the higher byte first followed by the lower byte. (AE before 41). Alternatively, others store and transmit the lower byte first (41 before AE).

Similarly, when registers are combined to represent 32-bit data types, Some devices store the higher 16 bits (high word) in the first register and the remaining low word in the second (AE41 before 5652) while others do the opposite (5652 before AE41).

It doesn't matter which order the bytes or words are sent in, as long as the receiving device knows which way to expect it. For example, if the number 29,235,175,522 was to be sent as a 32 bit unsigned integer, it could be arranged any of these four ways.

AE41 5652	high byte first	high word first
5652 AE41	high byte first	low word first
41AE 5256	low byte first	high word first
5256 41AE	low byte first	low word first

#### 12.7.5 Modbus Map

A modbus map is simply a list for an individual slave device that defines:

- what the data is (eg. pressure or temperature readings)
- where the data is stored (which tables and data addresses)

how the data is stored (data types, byte and word ordering)

Some devices are built with a fixed map that is defined by the manufacturer. While other devices allow the operator to configure or program a custom map to fit their needs.

## 13. Bus-Protocols

### 13.1 Introduction

There are many formats in Bus-Protocols. Bus-Protocols are developed to simplify communication between microcontrollers and devices without the need of a host computer.

In our branch we find several of these protocols. While the Canbus-protocol is the most widely used one, we will stick to that explanation here.

### 13.2 Canbus

CAN is a multi-master broadcast serial bus standard for connecting electronic control units (ECUs).

Each node is able to send and receive messages, but not simultaneously. A message consists primarily of an ID (identifier), which represents the priority of the message, and up to eight data bytes. It is transmitted serially onto the bus. This signal pattern is encoded in non-return-to-zero (NRZ) and is sensed by all nodes.

The devices that are connected by a CAN network are typically sensors, actuators, and other control devices. These devices are not connected directly to the bus, but through a host processor and a CAN controller.

If the bus is free, any node may begin to transmit. If two or more nodes begin sending messages at the same time, the message with the more dominant ID (which has more dominant bits, i.e., zeroes) will overwrite other nodes' less dominant IDs, so that eventually (after this arbitration on the ID.) only the dominant message remains and is received by all nodes. This mechanism is referred to as priority based bus arbitration. Messages with numerically smaller values of IDs have higher priority and are transmitted first.

Each node requires a

#### Host processor

- The host processor decides what received messages mean and which messages it wants to transmit itself.
- Sensors, actuators and control devices can be connected to the host processor.

#### CAN controller (hardware with a synchronous clock).

- Receiving: the CAN controller stores received bits serially from the bus until an entire message is available, which can then be fetched by the host processor (usually after the CAN controller has triggered an interrupt).
- Sending: the host processor stores its transmit messages to a CAN controller, which transmits the bits serially onto the bus.

#### Transceiver

- Receiving: it adapts signal levels from the bus to levels that the CAN controller expects and has protective circuitry that protects the CAN controller.

- Transmitting: it converts the transmit-bit signal received from the CAN controller into a signal that is sent onto the bus.

Bit rates up to 1 Mbit/s are possible at network lengths below 40 m. Decreasing the bit rate allows longer network distances (e.g., 500 m at 125 kbit/s).

### 13.3 Data transmission

CAN features an automatic arbitration-free transmission. A CAN message that is transmitted with highest priority will succeed, and the node transmitting the lower priority message will sense this and back off and wait.

This is achieved by CAN transmitting data through a binary model of "dominant" bits and "recessive" bits where dominant is a logical 0 and recessive is a logical 1. This means open collector, or wired or physical implementation of the bus (but since dominant is 0 this is sometimes referred to as wired and). If one node transmits a dominant bit and another node transmits a recessive bit then the dominant bit "wins" (a logical AND between the two).

So, if a recessive bit is being transmitted while a dominant bit is sent, the dominant bit is displayed, evidence of a collision. (All other collisions are invisible.) A dominant bit is asserted by creating a voltage across the wires while a recessive bit is simply not asserted on the bus. If any node sets a voltage difference, all nodes will see it. Thus there is no delay to the higher priority messages, and the node transmitting the lower priority message automatically attempts to re-transmit six bit clocks after the end of the dominant message.

When used with a differential bus, a carrier sense multiple access/bitwise arbitration (CSMA/BA) scheme is often implemented: if two or more devices start transmitting at the same time, there is a priority based arbitration scheme to decide which one will be granted permission to continue transmitting. The CAN solution to this is prioritized arbitration (and for the dominant message delay free), making CAN very suitable for real time prioritised communications systems.

During arbitration, each transmitting node monitors the bus state and compares the received bit with the transmitted bit. If a dominant bit is received when a recessive bit is transmitted then the node stops transmitting (i.e., it lost arbitration). Arbitration is performed during the transmission of the identifier field. Each node starting to transmit at the same time sends an ID with dominant as binary 0, starting from the high bit. As soon as their ID is a larger number (lower priority) they will be sending 1 (recessive) and see 0 (dominant), so they back off. At the end of ID transmission, all nodes but one have backed off, and the highest priority message gets through unimpeded.

For example, consider an 11-bit ID CAN network, with two nodes with IDs of 15 (binary representation, 00000001111) and 16 (binary representation, 00000010000). If these two nodes transmit at the same time, each will transmit the first six zeros of their ID with no arbitration decision being made. When the 7th bit is transmitted, the node with the ID of 16 transmits a 1 (recessive) for its ID, and the node with the ID of 15 transmits a 0 (dominant) for its ID. When this happens, the node with the ID of 16 will realize that it lost its arbitration, and allow the node with ID of 15 to continue its transmission. This ensures that the node with the lower bit value will always win the arbitration. The ID with the smaller number will win the right to use.



### 13.4 Speed, cable and termination

While we mostly use the j1939-protocol on the Canbus we can acclaim the following speed, cable and termination assumptions:

- Speed 250 kbit/s
- Cable shielded twisted pair (STP)
- Termination 120 Ohm on both sides of the bus

Normally, while the sensors are already powered we use the CAN-high and CAN-low of a shielded twisted pair cable to connect the Canbus to the interface. If, however, for some reason it needs to be connected to a serialport, the following applies for a DB9 connector:

- pin 2: CAN-Low (CAN-)
- pin 3: GND (Ground)
- pin 7: CAN-High (CAN+)
- pin 9: CAN V+ (Power)

Noise immunity is achieved by maintaining the differential impedance of the bus at a low level with low-value resistors (120 ohms) at each end of the bus. However, when dormant, a low-impedance bus such as CAN draws more current (and power) than other voltage-based signaling busses. On CAN bus systems, balanced line operation, where current in one signal line is exactly balanced by current in the opposite direction in the other signal provides an independent, stable 0 V reference for the receivers. Best practice determines that CAN bus balanced pair signals be carried in twisted pair wires in a shielded cable to minimize RF emission and reduce interference susceptibility in the already noisy RF environment.

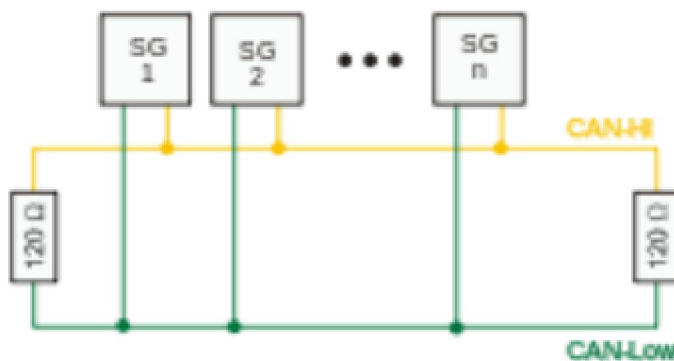


Figure 13-1: Canbus topology



## **14. Implementation in the FT NavVision© environment**

### **14.1 Introduction**

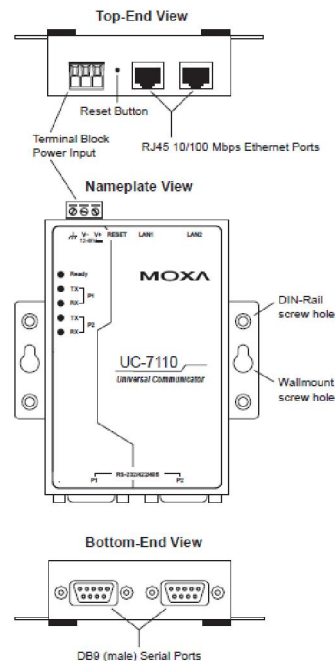
In this chapter we will discuss the above matter more in depth to the FT NavVision© system. How can you use it with the several interfaces that we use, what are the exceptions, how do I make the right settings in the program.

This will all be just an excerpt of the specific manuals on these subjects. It is merely meant to give you an insight to the abovementioned scenarios. For the full details on these subjects we refer you to the specific manuals.

### **14.2 Moxa UC-7110**

#### **14.2.1 Overview**

The Moxa UC-7110 series of RISC-based communication platforms (see Figure 14-1) are ideal for your embedded applications. UC-7110 comes with two RS-232/422/485 serial ports and dual 10/100 Mbps Ethernet LAN ports to provide users with a versatile communication platform.



**Figure 14-1: Overview (MOXA)**

### 14.2.2 Wiring requirements

Please observe the following common safety precautions, before proceeding with the installation of any electronic device (see chapter 2.3.1 Moxa hardware manual).

**NOTE:**

*Do not run signal or communication wiring and power wiring in the same wire conduit. To avoid interference, wires with different signal characteristics should be routed separately.*

- Use separate paths to route wiring for power and devices. If power wiring and device wiring paths must cross make sure the wires are perpendicular at the intersection point.
- Use the type of signal transmitted through a wire to determine which wires should be kept separate. The rule of thumb is that wiring that shares similar electrical characteristics can be bundled together.
- Keep input wiring and output wiring separate.

It is advisable to label the wiring to all devices in the system.

### 14.2.3 LED indicators

LED name	LED colour	LED function
Ready	Green	Power is on and functioning normally
P1/P2 (Tx)	Green	Serial port 1 or 2 is transmitting data
	Off	Serial port 1 or 2 is not transmitting data
P1/P2 (Rx)	Yellow	Serial port 1 or 2 is receiving data
	Off	Serial port 1 or 2 is not receiving data

**Table 14-1: Moxa Led Indicators**

#### 14.2.4 Connecting to the network

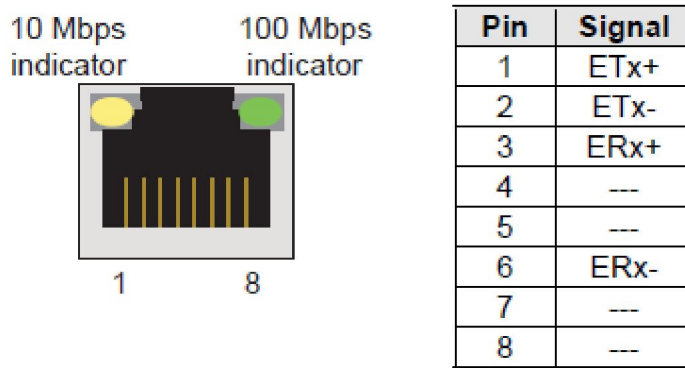
Connect one end of the Ethernet cable to UC-7110's 10/100M LAN1/LAN2 Ethernet port (see Figure 14-1) and the other end of the cable to the Ethernet network.

If the cable is properly connected, UC-7110 will indicate a valid connection to the Ethernet in the following ways:

- The top-right LED on the connector maintains a solid green colour when connected to a 100 Mbps Ethernet network (see Figure 14-2)
- The top-left LED on the connector maintains a solid orange color when connected to a 10 Mbps Ethernet network
- The LEDs will flash when Ethernet packets are being transmitted or received.

#### 14.2.5 Pinouts

The 10/100 Mbps Ethernet LAN 1 and LAN 2 ports use 8-pin RJ45 connectors. Pinouts for these ports are given in the following diagram (see Figure 14-2).



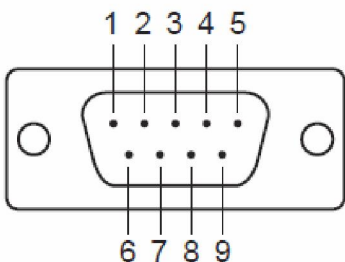
**Figure 14-2: 8-pin RJ connector and pinouts**

### 14.2.6 Connecting to a serial device

Connect the serial cable between UC-7110 and the serial device(s).

Serial ports P1 and P2 use male DB9 connectors, and can be configured for RS-232/422/485 by software. The pin assignments are shown in the following table:

DB9 Male Port



RS-232/422/485 Pinouts

Pin	RS-232	RS-422	RS-485 (4-wire)	RS-485 (2-wire)
1	DCD	TxDA(-)	TxDA(-)	---
2	RxD	TxDB(+)	TxDB(+)	---
3	TxD	RxDB(+)	RxDB(+)	DataB(+)
4	DTR	RxDA(-)	RxDA(-)	DataA(-)
5	GND	GND	GND	GND
6	DSR	---	---	---
7	RTS	---	---	---
8	CTS	---	---	---

Figure 14-3: Pin assignments (DB9 male port)

### 14.2.7 Serial LAN server for Moxa

Under “Serial LAN ports > Serial LAN server” (see Figure 14-4) the server to be assigned can be selected. In addition under “Type” the LAN server type can be selected.

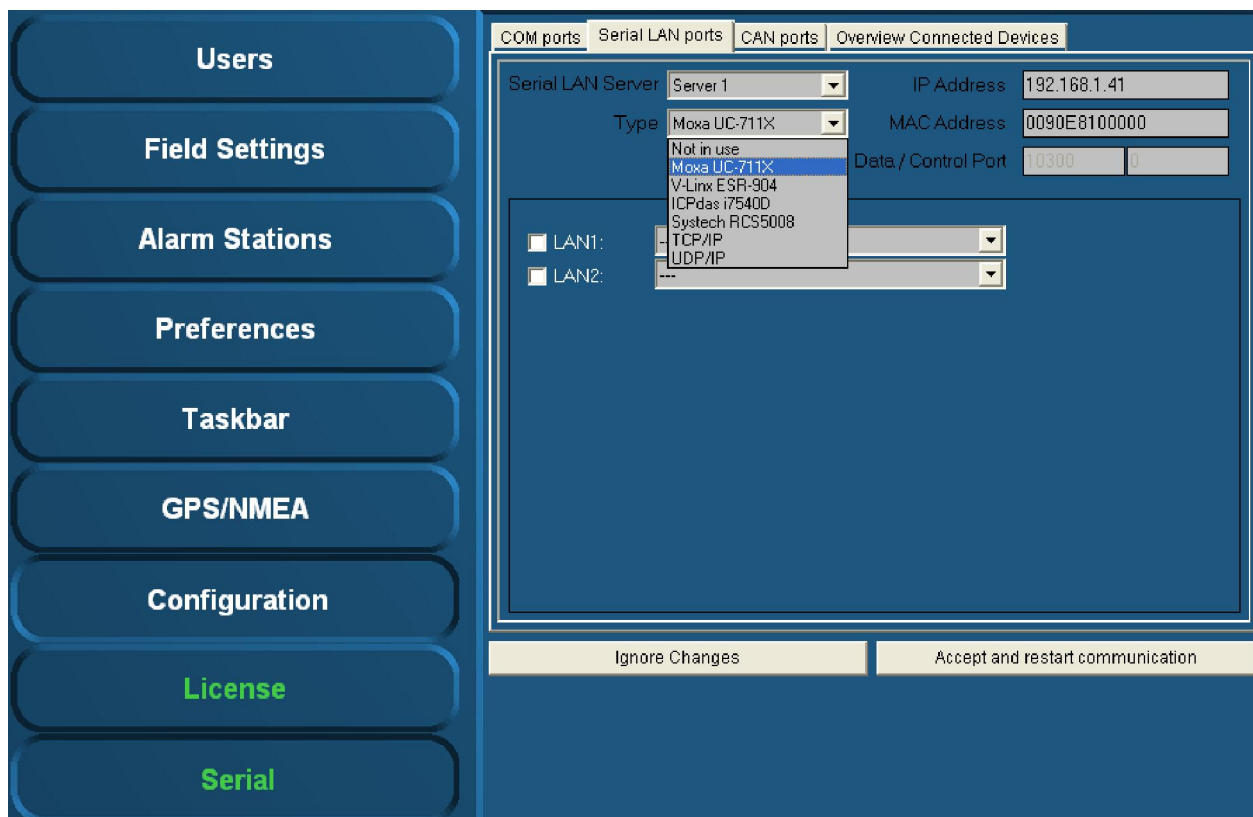


Figure 14-4: Type (Moxa)

#### 14.2.7.1 Type (Moxa UC-711X)

The Moxa is found under “Type” > “Moxa UC-711X” (see Figure 14-4).

Fill in the IP address of the Moxa unit under “IP Address” (use same range as the PC i.e. 172.16.x.x, for Moxa the last digits are in the 40 range).

The very first connected Moxa unit is set to IP address 172.16.1.41 and the next available to 172.16.1.42 etc.

*NOTE:*

*The MAC address can be found on the sticker underneath the unit.*

For the Moxa unit it is necessary to use a MAC address specified under “MAC Address”.

If necessary, verify the LAN1 and/or LAN2 settings and choose the appropriate device interface / protocol (see chapter 14.4 Installation and commissioning manual).

To confirm the settings, click “Accept and restart communication” and verify if the serial data is working within FT NavVision®.

Additional information on the selected port can be configured by clicking on the sign behind the drop-down menu. A new box will open.

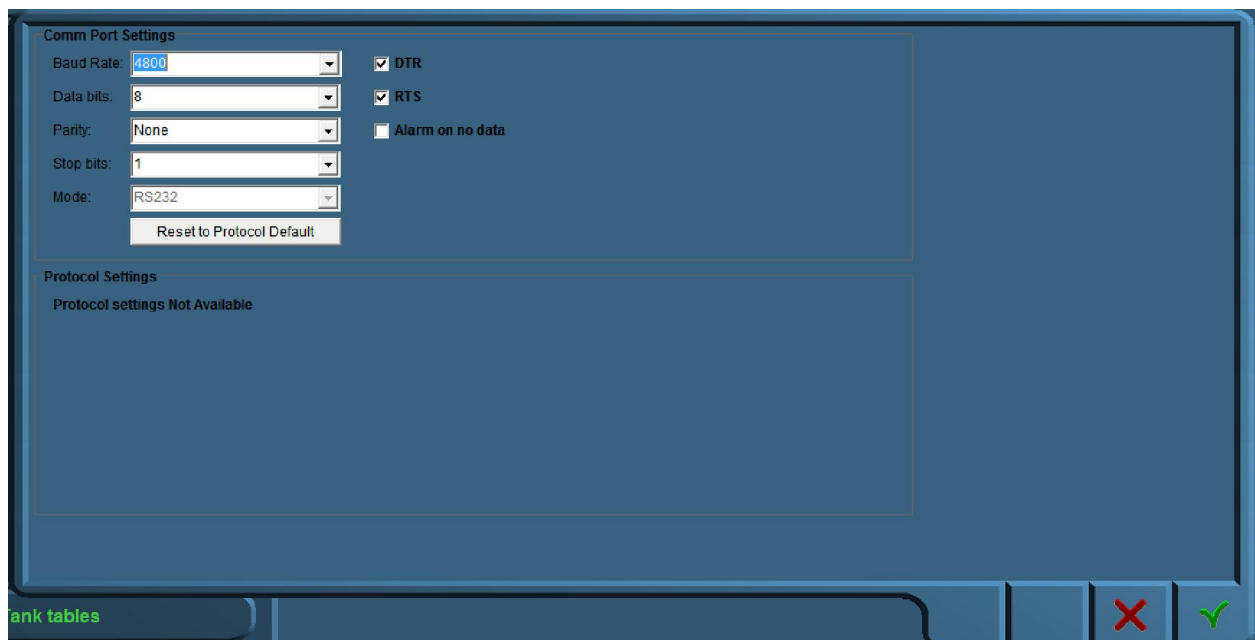


Figure 14-5: Comm Port Settings

In this additional configuration menu (see Figure 14-5) you can force all the settings for the regarding Comm port. The following fields apply:

- **Baud Rate:** Set the appropriate baudrate (see manual attached device)
- **Data Bits:** The number of data bits in each character can be 5 (for Baudot code), 6 (rarely used), 7 (for true ASCII), 8 (for any kind of data, as this matches the size of a byte), or 9 (rarely used). 8 data bits are almost universally used in newer applications. 5 or 7 bits generally only make sense with older equipment such as teleprinters.

- **Parity:** The parity bit in each character can be set to none (N), odd (O), even (E), mark (M), or space (S). None means that no parity bit is sent at all. Mark parity means that the parity bit is always set to the mark signal condition (logical 1) and likewise space parity always sends the parity bit in the space signal condition. Aside from uncommon applications that use the 9th (parity) bit for some form of addressing or special signalling, mark or space parity is uncommon, as it adds no error detection information. Odd parity is more common than even, since it ensures that at least one state transition occurs in each character, which makes it more reliable. The most common parity setting, however, is "none", with error detection handled by a communication protocol.
- **Stop Bits:** Stop bits sent at the end of every character allow the receiving signal hardware to detect the end of a character and to resynchronise with the character stream. Electronic devices usually use one stop bit.
- **Mode:** In mode you can set the protocol that the serial port is using to communicate. Refer to your device for the proper protocol. You can choose between RS232, RS422 and RS485. In some occasions you can't choose Mode cause the interface protocol can only work in a predefined Mode (i.e NMEA is always RS232).
- **DTR:** Data Terminal Ready, indicates presence of DTE to DCE (set high or low)
- **RTS:** Request to send, DTE requests the DCE prepare to receive data (set high or low)
- **Alarm on no data:** Gives an alarm when there is no data on the Comm port
- **Reset to protocol default:** Resets standard configuration for chosen protocol

## 14.3 ICP DAS i-7540D

### 14.3.1 Power supply connection

The ICP DAS i-7540D must be supplied with 10 – 30 VDC electrical power. The VS+ and GND are easily recognizable on the lower front of the gateway housing (see Figure 14-6).

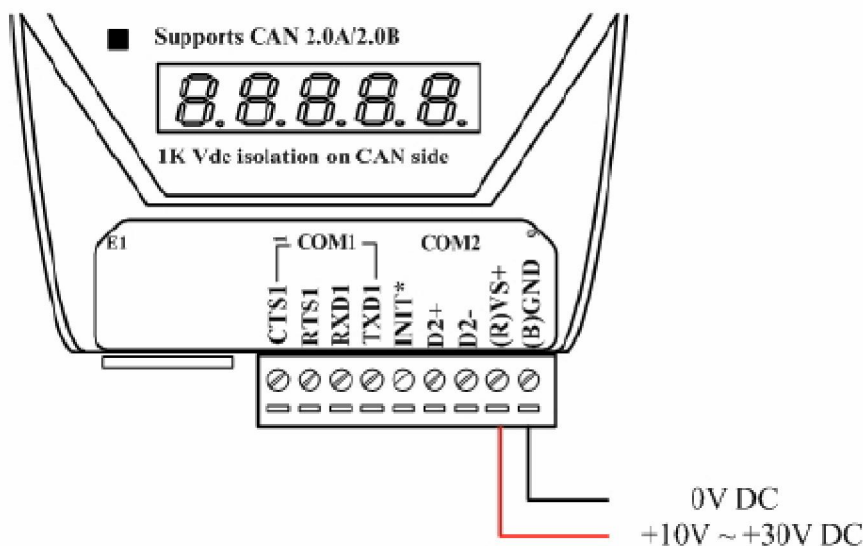


Figure 14-6: Power connection

### 14.3.2 Connection to CAN bus

In order to provide an easy CAN bus wiring, the I-7540D supplies one CAN port with two CAN bus connector interfaces. Each connector built on the I-7540D looks like Figure 14-7.

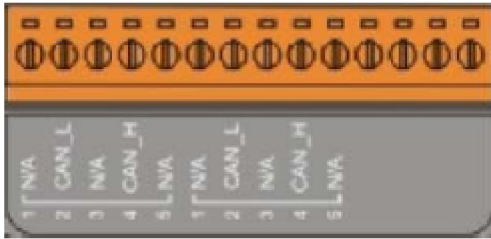


Figure 14-7: CAN bus connector

Pin	Signal	Description
1	N/A	Not connected
2	CAN_L	CAN_L bus line (dominant low)
3	N/A	Not connected
4	CAN_H	CAN_H bus line (dominant high)
5	N/A	Not connected

Table 14-2: Pin assignment (CAN bus)

**NOTE:**

*The electronic circuit comprises of a 120  $\Omega$  resistor.*

Note that the bypass CAN bus wiring, the I-7540D supplies one CAN port with two CAN bus connector interfaces. Each connector built on the I-7540D and looks like Figure 14-8.

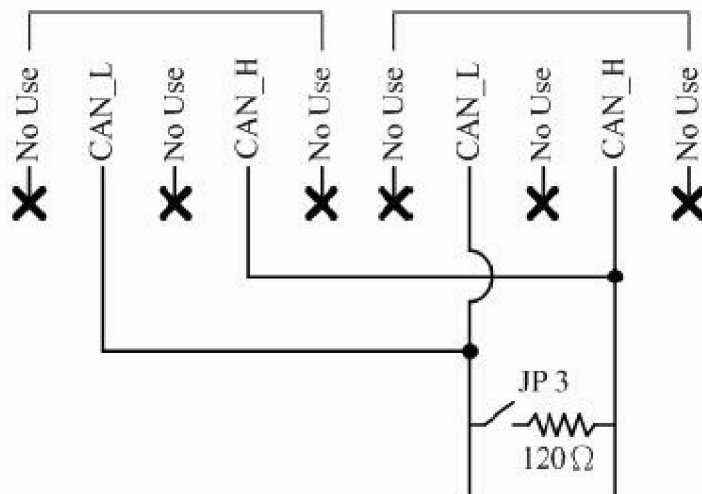
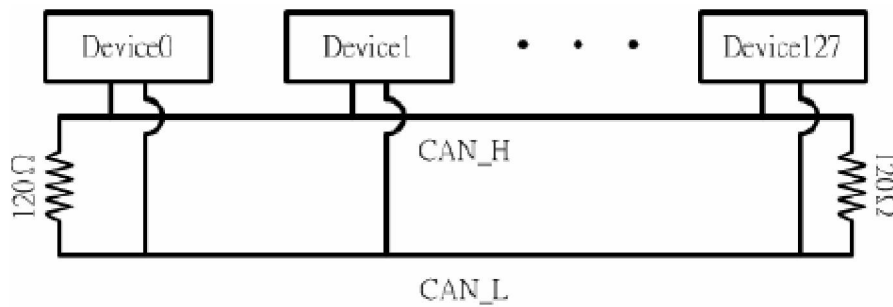


Figure 14-8: Electronic circuit CAN bus connector

### 14.3.3 Resistance check

Users should check the resistances of their CAN bus, before installing a new network as shown in Figure 14-9.



**Figure 14-9: Terminator resistor**

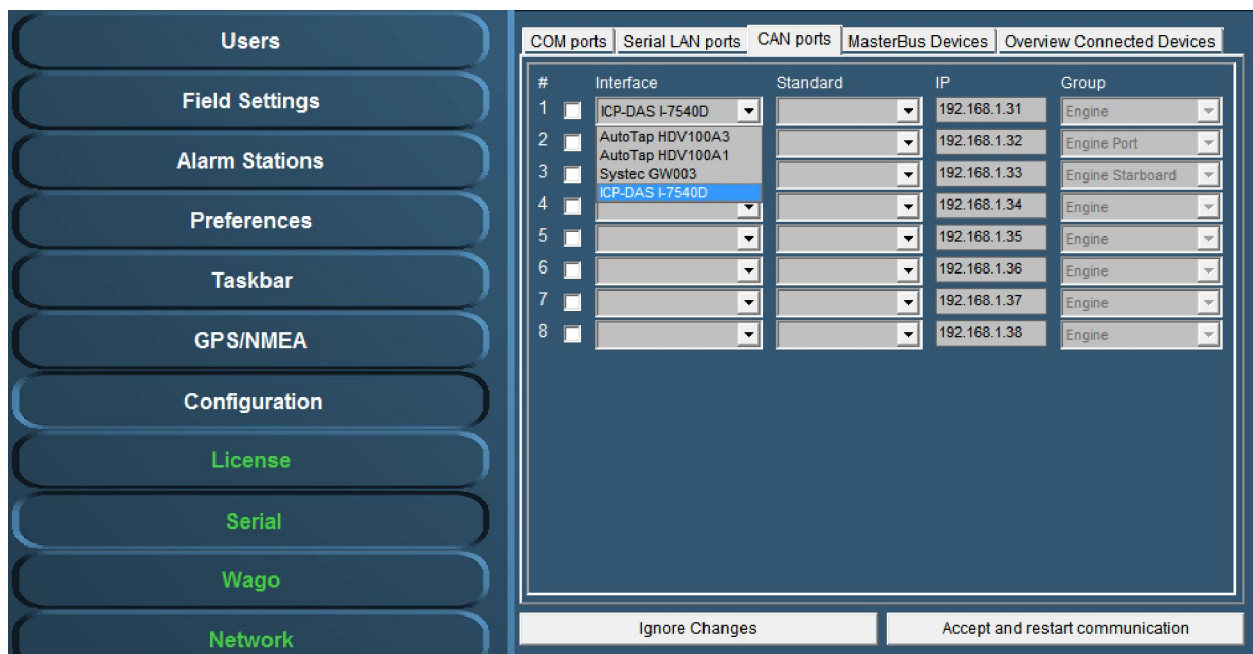
#### 14.3.4 Com Ports

In addition there are also two Com-ports available on the ICP DAS i-7540D (see Figure 14-6). COM1 is a RS232 connection and COM2 is a RS485H connection.

#### 14.3.5 CAN ports

Under “Serial > CAN ports” the following menus are available:

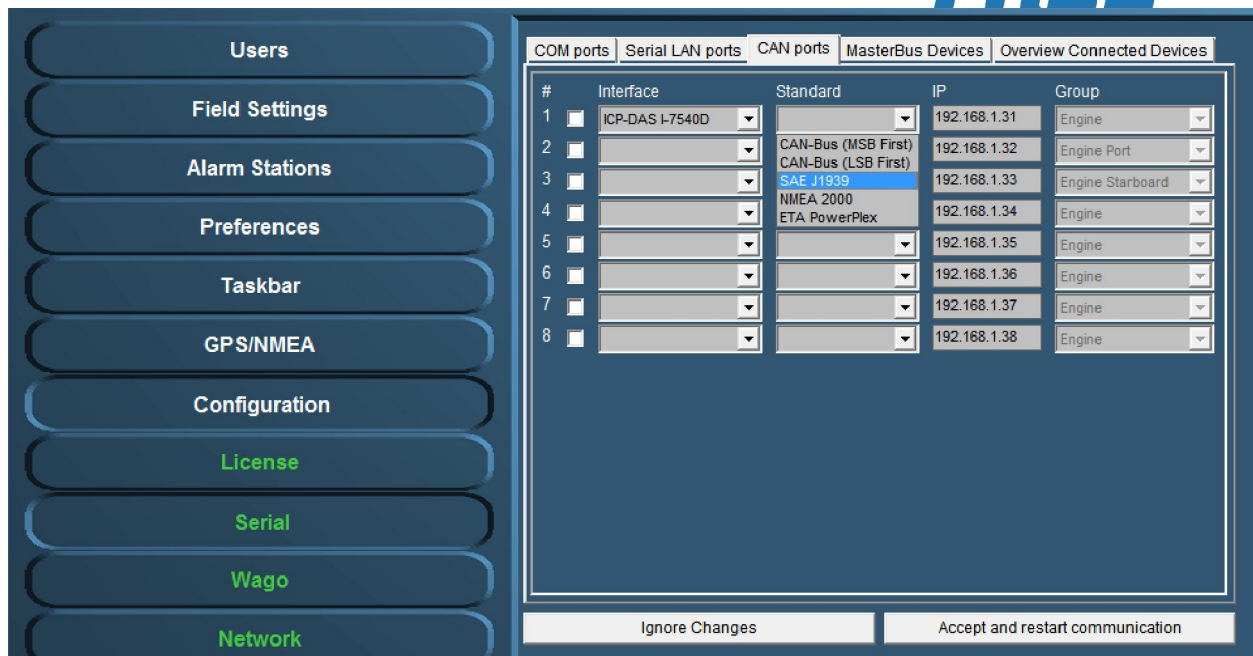
- Interface
- Standard
- IP
- Group.



**Figure 14-10: Interface**

Under interface you can choose different kinds of Can-interfaces. The most used one is the ICP. If you come across an older version, you can choose it here. (see Figure 14-10).





**Figure 14-11: Standard**

Under Standard you choose the protocol you want to use with the interface (see Figure 14-11). Most widely used are the NMEA 2000 and the SAE J1939. Which to use is depending on your attached protocol.

Under IP you can select the right IP address that reflects the connected ICP for example. You can best leave it as it is by default (which will become the 172.16.1.x range). For information on how to set the right IP-address in the ICP, please refer to the ICP installation manual.

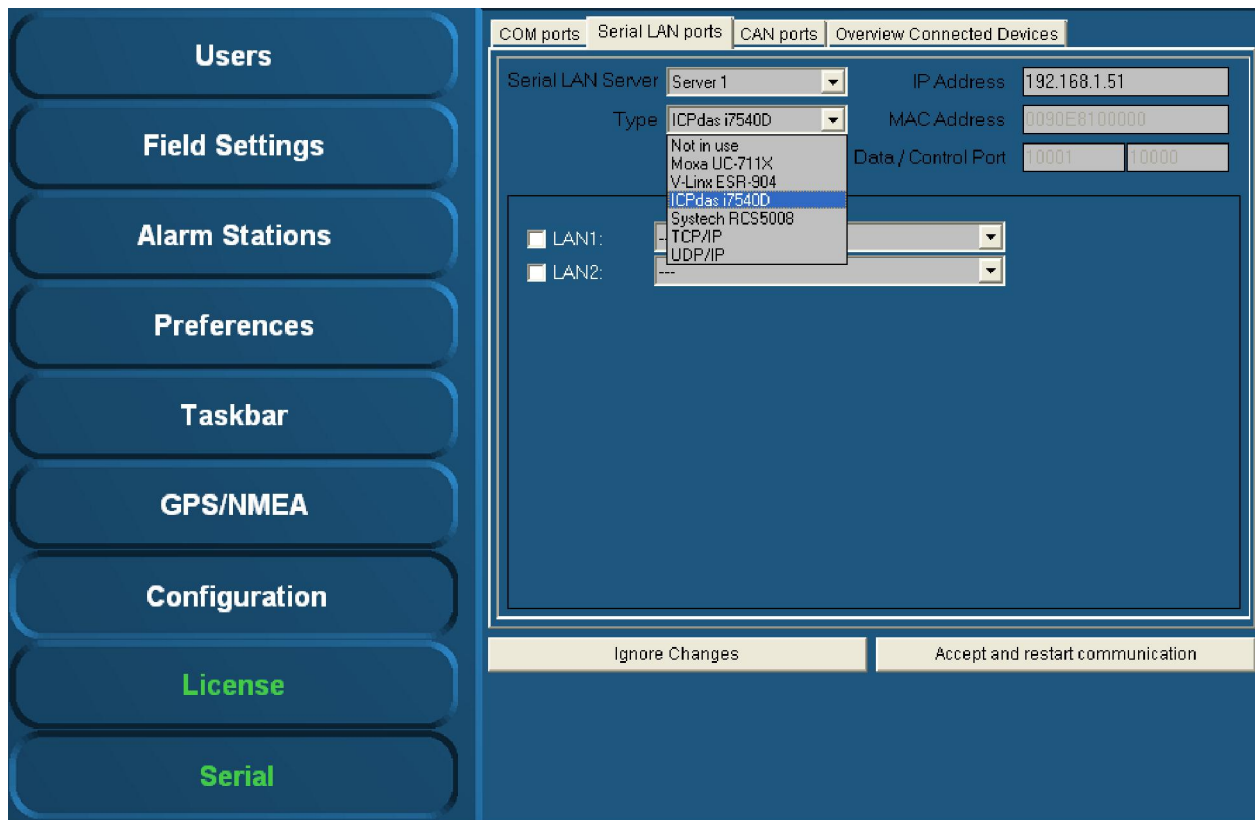
The group you choose reflects under which group the information will be stored in NavVision. If you, for example, want the information from the interface to show up under Engine Port, you select that under Group (see Figure 14-11).

After each change you need to hit “Accept and restart communication” to save it to the system.

#### **14.3.6 Type (ICPdas i7540D)**

The ICPdas is found under “Type” “ICPdas i7540D” (see Figure 14-12).

Fill in the IP address of the ICPdas server under “IP Address” (same range as the PC i.e. 172.168.x.x, for ICP the last digits are in the 30 range). The very first connected ICP is set to IP address 172.16.1.31 and the next available to 172.16.1.32 etc.



**Figure 14-12: Type (ICPdas i7540D)**

Verify the LAN1 and LAN2 settings (if available) and select the appropriate protocol (see chapter 4.9.2.1 Hardware installation and commissioning manual).

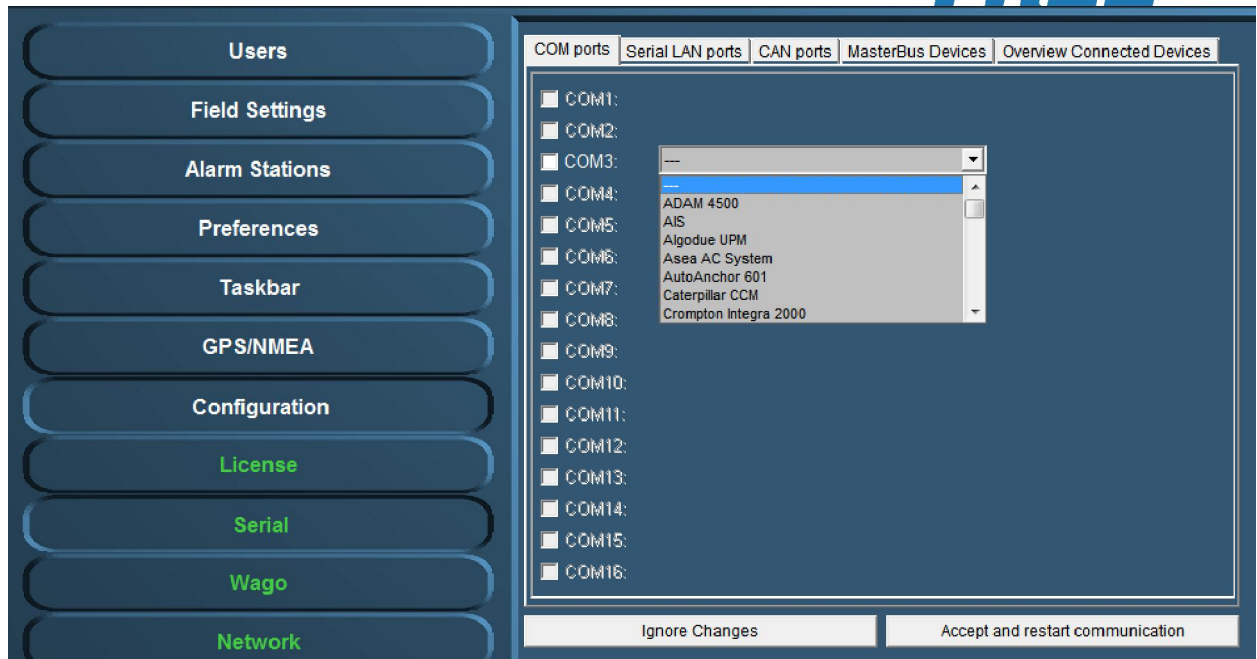
To confirm the settings, click “Accept and restart communication” and verify if the serial data is working within FT NavVision®.

## 14.4 COM port assignment

### *NOTE:*

*Use the right device interface (protocol) and verify the baudrate etc.*

Check the respective wiring schematics to determine the COM port arrangement and assignment. Tick off the relevant COM port (1, 2, 3, etc.) and select the required device interface (protocol) by means of the drop-down menu (see Figure 14-13).



**Figure 14-13: Drop-down menu (device interfaces)**

At completion, confirm the settings by clicking “Accept and restart communication” (see Figure 14-13).

Check the appropriate FT NavVision® viewer to verify if the COM-port is correct and if there is any data communication. For example: select the “Video Sounder” viewer (see Figure 14-14) to verify that the device interface (protocol) on “COM1” is correct. Repeat this procedure for all other listed COM ports.

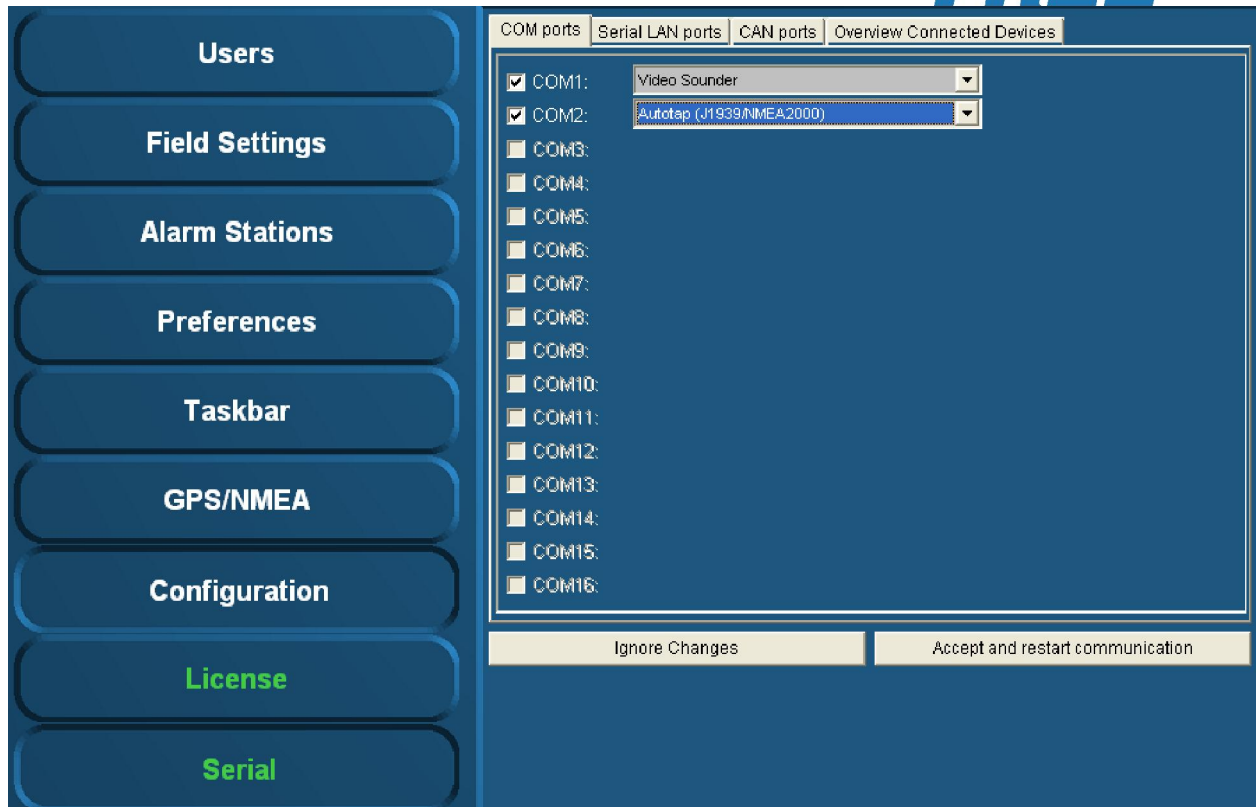


Figure 14-14: COM port assignment

Additional information on the selected port can be configured by clicking on the sign behind the drop-down menu (see Figure 14-15). A new box will open.

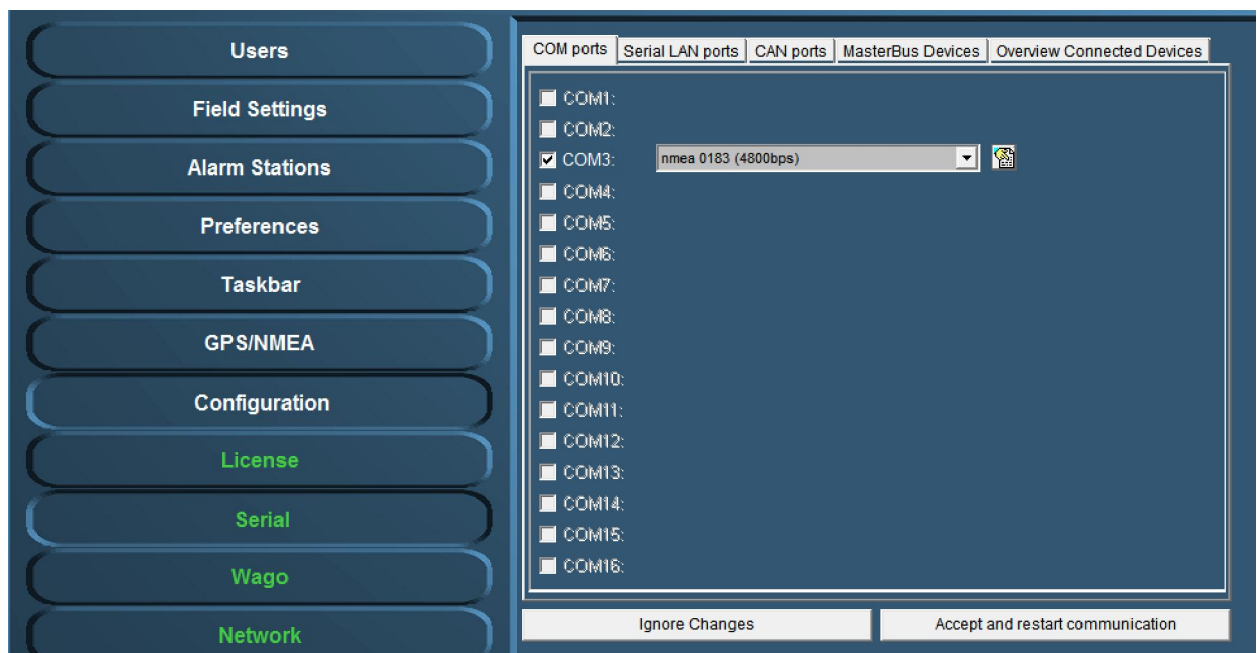
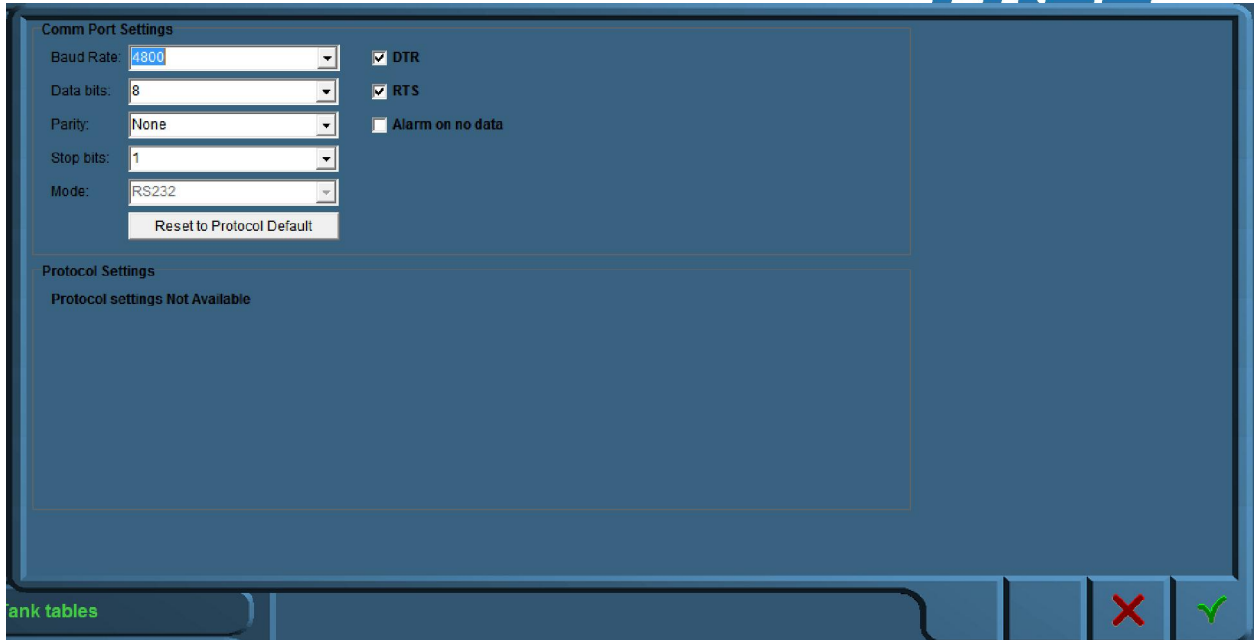


Figure 14-15: additional configuration



**Figure 14-16: Comm Port Settings**

In this additional configuration menu (see Figure 14-5) you can force all the settings for the regarding Comm port. The following fields apply:

- **Baud Rate:** Set the appropriate baudrate (see manual attached device)
- **Data Bits:** The number of data bits in each character can be 5 (for Baudot code), 6 (rarely used), 7 (for true ASCII), 8 (for any kind of data, as this matches the size of a byte), or 9 (rarely used). 8 data bits are almost universally used in newer applications. 5 or 7 bits generally only make sense with older equipment such as teleprinters.
- **Parity:** The parity bit in each character can be set to none (N), odd (O), even (E), mark (M), or space (S). None means that no parity bit is sent at all. Mark parity means that the parity bit is always set to the mark signal condition (logical 1) and likewise space parity always sends the parity bit in the space signal condition. Aside from uncommon applications that use the 9th (parity) bit for some form of addressing or special signalling, mark or space parity is uncommon, as it adds no error detection information. Odd parity is more common than even, since it ensures that at least one state transition occurs in each character, which makes it more reliable. The most common parity setting, however, is "none", with error detection handled by a communication protocol.
- **Stop Bits:** Stop bits sent at the end of every character allow the receiving signal hardware to detect the end of a character and to resynchronise with the character stream. Electronic devices usually use one stop bit.
- **Mode:** In mode you can set the protocol that the serial port is using to communicate. Refer to your device for the proper protocol. You can choose between RS232, RS422

and RS485. In some occasions you can't choose Mode cause the interface protocol can only work in a predefined Mode (i.e NMEA is always RS232).

- **DTR:** Data Terminal Ready, indicates presence of DTE to DCE (set high or low)
- **RTS:** Request to send, DTE requests the DCE prepare to receive data (set high or low)
- **Alarm on no data:** Gives an alarm when there is no data on the Comm port
- **Reset to protocol default:** Resets standard configuration for chosen protocol

## 14.5 485LDRC9

The 485LDRC9 is an industrial RS-232 to RS-422/485 converter. RS-232 signals interface via a terminal block or a convenient DB9 (DCE) female connector. RS-422/485 signals are connect to a terminal block. B&B's Automatic Send Data Control circuitry eliminates the requirement for software control of the RS-422/RS-485 handshake signals. Position the DIP Switches in accordance with tables one and two to change the communications mode and data rate. You can also use a pair of these converters to extend and isolate RS-232 signals. An external 10 – 30 VDC power supply (not included), is required.

### 14.5.1 Operation

Select Data rate and mode by positioning the DIP Switches in accordance with Table 14-3 and Table 14-4.

RS	Switch 1 Tx Enable	Switch 2 RX Enable	Switch 3 2/4 Wire	Switch 4 2/4 Wire
RS485 2-wire (Half-Duplex)	ON	ON	ON	ON
RS485 4-wire (Full-Duplex)	ON	OFF	OFF	OFF
RS422 (Full-Duplex)	OFF	OFF	OFF	OFF

Table 14-3: Communications mode selection

Baud rate	Switch 6	Switch 7	Switch 8	R11
1200	OFF	OFF	OFF	820 K
2400	OFF	OFF	ON	Not Used
4800	OFF	ON	OFF	Not Used
9600	ON	OFF	OFF	Not Used
19200	ON	ON	ON	Not Used
38400	OFF	OFF	OFF	27 K
57600	OFF	OFF	OFF	16 K
115200	OFF	OFF	OFF	8.2 K

Table 14-4: Data rate selection

Automatic Send Data Control: The first bit of data from the RS-232 side enables the transmitter and disables the receiver. After receiving the last RS-232 data bit, the timeout circuit waits one character length, then disables the transmitter and enables the receiver. Select the timeout by positioning the DIP Switches or changing the value of R-11. Refer to Table 14-4 for R-11 values and DIP Switch positions.

If necessary, use termination resistance for high data rates or long cable runs by positioning Switch 5 to "on."

Figure 14-17, Figure 14-18 and Figure 14-19 are examples of a DTE to DCE connection. The DB9 female connector on this converter will make the same connections using a straight through DB9F to DB9M cable. If the RS-232 device is wired for DCE, then cross pins 2 and 3.

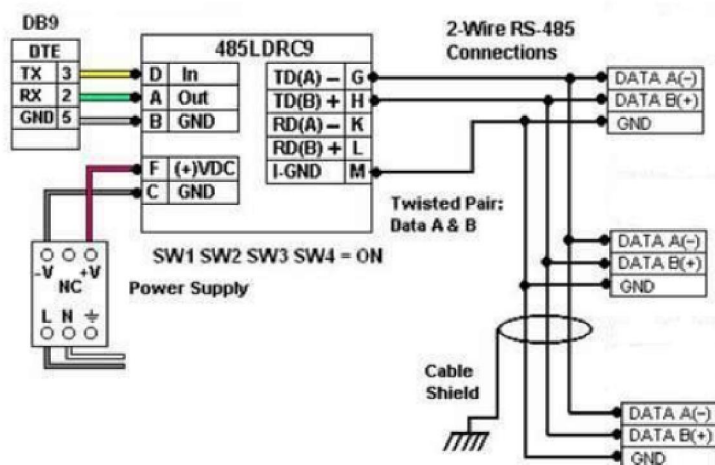


Figure 14-17: 2-wire RS485

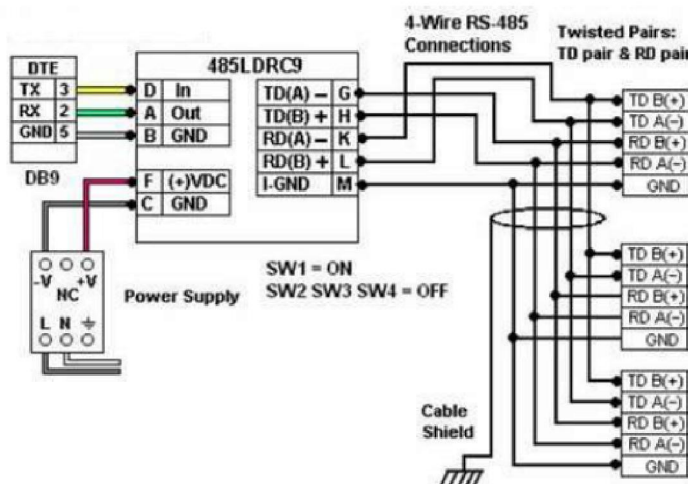


Figure 14-18: 4-wire RS485



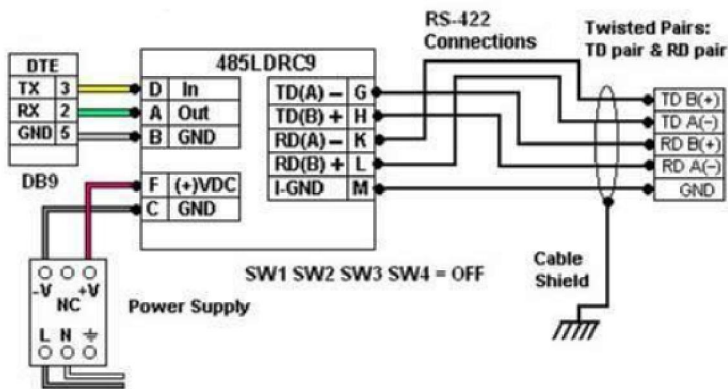


Figure 14-19: RS422

Figure 14-20 demonstrates how to use two converters to extend and isolate RS-232 signals.

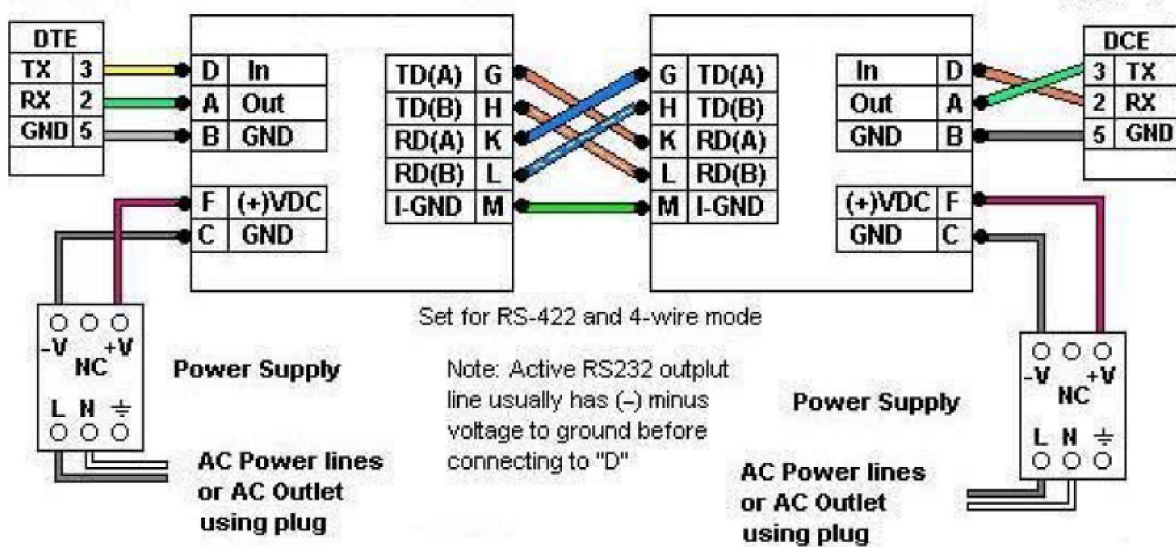


Figure 14-20: Extend and isolate RS232



Technical & customer support  
**The Netherlands**

Free Technics B.V.  
Eikenlaan 259J  
2404 BP, Alphen aan den Rijn  
The Netherlands

Telephone: +31 172418 890  
Fax: +31 172418 899  
[www.freetechnics.eu](http://www.freetechnics.eu)