Basic Training

FT NavVision©

Day 4

Version 1.1 ● December 30, 2013

Protocols, Modbus, Serial, interfacing and sensorlist.

**Last edited: 30 December 2013**

**Copyright © 2011 FT NavVision© . All rights reserved.**

**No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission FT NavVision© .**

**All copyright, confidential information, patents, design rights and all other intellectual property rights of whatsoever nature contained herein are and shall remain the sole and exclusive property of FT NavVision© . The information furnished herein is believed to be accurate and reliable.**

**However, no responsibility is assumed by FT NavVision© for its use, or for any infringements of patents or other rights of third parties resulting from its use.**

**The FT NavVision© name and FT NavVision© logo are trademarks or registered trademarks of FT NavVision©**

**All other trademarks are the property of their respective owners**

|  |  |
| --- | --- |
| Publication type: | Training |
| Publication number: | FTT1301004 |
| Title: | Basic training FT NavVision® day 4 |
| Subject: |  |
| Issue: | 1.1 |
| Publication date: | Februari 26, 2013 |
| Total number of pages: |  |
| Author: |  |

Inhoud

[2. Serial Communication 7](#_Toc349645642)

[2.1 Introduction 7](#_Toc349645643)

[2.2 DTE and DCE 8](#_Toc349645644)

[2.3 RS232 specifications 8](#_Toc349645645)

[2.3.1 RS232 Bit Streams 8](#_Toc349645646)

[2.3.2 Start Bit 9](#_Toc349645647)

[2.3.3 Data Bits 9](#_Toc349645648)

[2.3.4 Parity Bit 9](#_Toc349645649)

[2.3.4.1 Even Parity 9](#_Toc349645650)

[2.3.4.2 Odd Parity 9](#_Toc349645651)

[2.3.4.3 Disadvantages on the Parity System 9](#_Toc349645652)

[2.3.5 Stop Bit(s) 10](#_Toc349645653)

[2.3.6 Voltages 10](#_Toc349645654)

[2.3.7 Maximum Cable Lengths 10](#_Toc349645655)

[2.3.8 RS232 Pinout 11](#_Toc349645656)

[2.3.9 RS232 Flow Control and Handshaking 11](#_Toc349645657)

[2.3.9.1 Software flow control 12](#_Toc349645658)

[2.3.9.2 Hardware flow control 13](#_Toc349645659)

[2.3.10 Special connections 13](#_Toc349645660)

[2.3.10.1 RS232 DB25 to DB9 converter 13](#_Toc349645661)

[2.3.10.2 RS232 serial loopback test plugs 14](#_Toc349645662)

[2.3.10.3 RS232 Null Modem Cables 15](#_Toc349645663)

[2.4 RS485 Specifications 17](#_Toc349645664)

[2.4.1 Differential signals with RS485:Longer distances and higher bit rates 17](#_Toc349645665)

[2.4.2 Network topology with RS485 18](#_Toc349645666)

[2.4.3 RS485 functionality 19](#_Toc349645667)

[2.4.4 What Pins Are Needed for 2- and 4- Wire Transmission with RS-485 Serial Communication? 19](#_Toc349645668)

[3. TCP/IP 20](#_Toc349645669)

[3.1.1 Introduction 20](#_Toc349645670)

[3.1.2 Internet Protocol Suite 20](#_Toc349645671)

[3.1.3 Layers in the TCP/IP stack 21](#_Toc349645672)

[3.1.4 The Network Access layer 21](#_Toc349645673)

[3.1.4.1 The physical layer 21](#_Toc349645674)

[3.1.4.2 The data link layer 22](#_Toc349645675)

[3.1.4.3 The Internetwork layer 22](#_Toc349645676)

[3.1.4.4 The transport layer 22](#_Toc349645677)

[3.1.4.5 The application layer 23](#_Toc349645678)

[3.1.5 TCP/IP Relevance 23](#_Toc349645679)

[4. Modbus 24](#_Toc349645680)

[4.1 Introduction 24](#_Toc349645681)

[4.2 Modbus message structure 24](#_Toc349645682)

[4.3 Modbus serial transmission modes: Modbus/ASCII and Modbus/RTU 25](#_Toc349645683)

[4.4 Modbus addressing 26](#_Toc349645684)

[4.5 Modbus function codes 26](#_Toc349645685)

[4.5.1 Function 01: Read coil status 27](#_Toc349645686)

[4.5.2 Function 02: Read input status 27](#_Toc349645687)

[4.5.3 Function 03: Read holding registers 28](#_Toc349645688)

[4.6 Addressing (0- or 1-based) 29](#_Toc349645689)

[4.7 Understanding Raw Data 29](#_Toc349645690)

[4.7.1 How is data stored in Standard Modbus? 29](#_Toc349645691)

[4.7.2 Examples of questions and answers 30](#_Toc349645692)

[4.7.2.1 Example 1 30](#_Toc349645693)

[4.7.2.2 Example 2 31](#_Toc349645694)

[4.7.2.3 Example 3 31](#_Toc349645695)

[4.7.3 Data Types 32](#_Toc349645696)

[4.7.4 Byte and Word ordering 32](#_Toc349645697)

[4.7.5 Modbus Map 33](#_Toc349645698)

[5. Bus-Protocols 33](#_Toc349645699)

[5.1 Introduction 33](#_Toc349645700)

[5.2 Canbus 33](#_Toc349645701)

[5.3 Data transmission 34](#_Toc349645702)

[5.4 Speed, cable and termination 35](#_Toc349645703)

[6. Implementation in the FT NavVision© environment 36](#_Toc349645704)

[6.1 Introduction 36](#_Toc349645705)

[6.2 Moxa UC-7110 37](#_Toc349645706)

[6.2.1 Overview 37](#_Toc349645707)

[6.2.2 Wiring requirements 37](#_Toc349645708)

[6.2.3 LED indicators 37](#_Toc349645709)

[6.2.4 Connecting to the network 38](#_Toc349645710)

[6.2.5 Pinouts 38](#_Toc349645711)

[6.2.6 Connecting to a serial device 39](#_Toc349645712)

[6.2.7 Serial LAN server for Moxa 39](#_Toc349645713)

[6.2.7.1 Type (Moxa UC-711X) 40](#_Toc349645714)

[6.3 ICP DAS i-7540D 41](#_Toc349645715)

[6.3.1 Power supply connection 41](#_Toc349645716)

[6.3.2 Connection to CAN bus 41](#_Toc349645717)

[6.3.3 Resistance check 42](#_Toc349645718)

[6.3.4 Com Ports 43](#_Toc349645719)

[6.3.5 CAN ports 43](#_Toc349645720)

[6.3.6 Type (ICPdas i7540D) 44](#_Toc349645721)

[6.4 COM port assignment 45](#_Toc349645722)

[6.5 485LDRC9 49](#_Toc349645723)

[6.5.1 Operation 49](#_Toc349645724)

[8. What is the Sensorlist 52](#_Toc349645725)

[8.1 Introduction 52](#_Toc349645726)

[8.2 Excel 52](#_Toc349645727)

[8.3 Saving and naming 53](#_Toc349645728)

[8.4 Saving as sensorlist for import 54](#_Toc349645729)

[9. Devicelist 55](#_Toc349645730)

[9.1 Introduction 55](#_Toc349645731)

[9.2 Columns 56](#_Toc349645732)

[9.3 IP-addresses 60](#_Toc349645733)

[9.3.1 Introduction 60](#_Toc349645734)

[9.3.2 IPAddressUp- IPAddressDown 61](#_Toc349645735)

[9.3.3 IPAddressUp2- IPAddressDown2 61](#_Toc349645736)

[9.4 Mac addresses 62](#_Toc349645737)

[9.4.1 Introduction 62](#_Toc349645738)

[9.4.2 MACAddressUp(2)- MACAddressDown(2) 62](#_Toc349645739)

[9.5 How to implement this in the devicelist 62](#_Toc349645740)

[9.5.1 Introduction 62](#_Toc349645741)

[9.5.2 The devices 62](#_Toc349645742)

[9.5.3 Comment 65](#_Toc349645743)

[9.5.4 Location 66](#_Toc349645744)

[9.5.5 Protocol 66](#_Toc349645745)

[9.5.6 Interface 68](#_Toc349645746)

[9.5.7 Port and Source 69](#_Toc349645747)

[9.5.8 Type 70](#_Toc349645748)

[9.5.9 Speed, Datalink and Hardware 72](#_Toc349645749)

[9.5.10 Options 73](#_Toc349645750)

[9.5.11 IP addresses and MAC addresses 73](#_Toc349645751)

[9.5.11.1 Other devices 76](#_Toc349645752)

[10. Sensorlist 80](#_Toc349645753)

[10.1 Introduction 80](#_Toc349645754)

[10.2 Columns 80](#_Toc349645755)

[10.3 Implementation in the sensorlist 88](#_Toc349645756)

[10.3.1 Introduction 88](#_Toc349645757)

[10.3.2 Import Result 88](#_Toc349645758)

[10.3.3 ID, CableLabel, GroupLabel 89](#_Toc349645759)

[10.3.4 Item 89](#_Toc349645760)

[10.3.4.1 Conjunction with SensorType 90](#_Toc349645761)

[10.3.5 SensorType 90](#_Toc349645762)

[10.3.5.1 Fields 91](#_Toc349645763)

[10.3.5.2 Back to SensorType 91](#_Toc349645764)

[10.3.6 Connection 92](#_Toc349645765)

[10.3.7 Device 92](#_Toc349645766)

[10.3.8 Location 93](#_Toc349645767)

[10.3.9 Interface 93](#_Toc349645768)

[10.3.10 Module 93](#_Toc349645769)

[10.3.11 Pin 94](#_Toc349645770)

[10.3.11.1 Wago 94](#_Toc349645771)

[10.3.11.2 Serial Protocols 97](#_Toc349645772)

[10.3.12 Type 97](#_Toc349645773)

[10.3.12.1 Wago 97](#_Toc349645774)

[10.3.12.2 Serial Protocols 98](#_Toc349645775)

[10.3.13 Min-Max 98](#_Toc349645776)

[10.3.14 DefaultUnit 98](#_Toc349645777)

[10.3.15 Manufacturer 99](#_Toc349645778)

[10.3.16 Supplier 99](#_Toc349645779)

[10.3.17 Comment 99](#_Toc349645780)

[10.3.18 Revision 99](#_Toc349645781)

[10.3.19 Field 100](#_Toc349645782)

[10.3.19.1 How to work with fieldlist.txt 100](#_Toc349645783)

[10.3.19.2 Back to the Field column 102](#_Toc349645784)

[10.3.20 Label 103](#_Toc349645785)

[10.3.21 Rate 103](#_Toc349645786)

[10.3.22 Index 103](#_Toc349645787)

[10.3.23 Datatype 103](#_Toc349645788)

[10.3.24 Enum 103](#_Toc349645789)

[10.3.25 Count 104](#_Toc349645790)

[10.3.26 Multiplier 104](#_Toc349645791)

[10.3.27 Offset 104](#_Toc349645792)

[10.3.28 Unit 104](#_Toc349645793)

[10.3.29 Other columns 104](#_Toc349645794)

[10.4 Filter sensorlist 104](#_Toc349645795)

[10.5 Special issues 107](#_Toc349645796)

[11. Importing in FT NavVision 109](#_Toc349645797)

[11.1 Introduction 109](#_Toc349645798)

[11.2 How to import 110](#_Toc349645799)

[11.3 Check the import 111](#_Toc349645800)

[11.4 Devicelist generated 111](#_Toc349645801)

[11.5 Sensorlist generated 113](#_Toc349645802)

[11.6 Sensorlist generated diff 113](#_Toc349645803)

[11.6.1 Making an export 114](#_Toc349645804)

[12. Keeping up to date 115](#_Toc349645805)

[12.1 Introduction 115](#_Toc349645806)

[12.2 Direct changing 115](#_Toc349645807)

[12.2.1 insert 116](#_Toc349645808)

[12.3 Bigger changes 116](#_Toc349645809)

[12.4 Keep the sensorlist up to date afterwards 119](#_Toc349645810)

[12.4.1 Introduction 119](#_Toc349645811)

[12.4.2 What do you need 119](#_Toc349645812)

[12.4.3 Cleaning up after a day on board 120](#_Toc349645813)

[12.4.3.1 Copy devicelist.dat and sensorlist.dat 120](#_Toc349645814)

[12.4.3.2 The old sensorlist 120](#_Toc349645815)

[12.4.3.3 Startup your project folder 121](#_Toc349645816)

[12.4.3.4 Inspecting the generated files 122](#_Toc349645817)

[12.4.3.5 Comment 123](#_Toc349645818)

[12.4.3.6 Changed 123](#_Toc349645819)

[12.4.3.7 Failed 124](#_Toc349645820)

[12.4.3.8 Missing 125](#_Toc349645821)

[12.4.3.9 New 125](#_Toc349645822)

[12.4.3.10 Keep importing 125](#_Toc349645823)

[13. Special notes 126](#_Toc349645824)

[13.1 Introduction 126](#_Toc349645825)

[13.2 PLC 126](#_Toc349645826)

[13.3 Search 128](#_Toc349645827)

Figures

Figure 10‑1: RS232 DB9 Pinout 13

Figure 10‑2: RS232 DB25 to DB9 Convertor 16

Figure 10‑3: DB9 RS232 Loopback Plug 17

Figure 10‑4: Null modem cable without handshaking 17

Figure 10‑5: Null modem with loop back handshaking 18

Figure 10‑6: Null modem with full handshaking 19

Figure 10‑7: Noise in straight and twisted pair cables 20

Figure 10‑8: RS485 network topology 21

Figure 10‑9: 422/485 DB9 Pinout 22

Figure 13‑1: Canbus topology 38

Figure 14‑1: Overview (MOXA) 39

Figure 14‑2: 8-pin RJ connector and pinouts 40

Figure 14‑3: Pin assignments (DB9 male port) 41

Figure 14‑4: Type (Moxa) 41

Figure 14‑5: Comm Port Settings 42

Figure 14‑6: Power connection 43

Figure 14‑7: CAN bus connector 44

Figure 14‑8: Electronic circuit CAN bus connector 44

Figure 14‑9: Terminator resistor 45

Figure 14‑10: Interface 45

Figure 14‑11: Standard 46

Figure 14‑12: Type (ICPdas i7540D) 47

Figure 14‑13: Drop-down menu (device interfaces) 48

Figure 14‑14: COM port assignment 49

Figure 14‑15: additional configuration 49

Figure 14‑16: Comm Port Settings 50

Figure 14‑17: 2-wire RS485 52

Figure 14‑18: 4-wire RS485 52

Figure 14‑19: RS422 53

Figure 14‑20: Extend and isolate RS232 53

Figure 9‑1: devicelist 55

Figure 9‑2: Sensorlist 55

Figure 9‑3: Excel saving 56

Figure 9‑4: saving as sensorlist.xls 57

Figure 10‑1: single line drawing 58

Figure 10‑2: numbering the drawing 63

Figure 10‑3: Filling device column 1 65

Figure 10‑4: Filling device column 2 65

Figure 10‑5: Filling device column 3 66

Figure 10‑6: Filling device column 4 67

Figure 10‑7: Filling location column 68

Figure 10‑8: Filling protocol column 1 69

Figure 10‑9: Filling protocol column 2 69

Figure 10‑10: Filling protocol column 3 70

Figure 10‑11: Filling interface column 71

Figure 10‑12: Port and Source 1 71

Figure 10‑13 Port and Source 2 72

Figure 10‑14: Network Serial colors 73

Figure 10‑15: Filling Type column 1 73

Figure 10‑16: Filling type column 2 74

Figure 10‑17: speed, datalink and hardware 75

Figure 10‑18: Addresses and connection 1 76

Figure 10‑19: Addresses and connection 2 76

Figure 10‑20: Addresses and connection 3 76

Figure 10‑21: Addresses and connection 4 76

Figure 10‑22: Addresses and connection 5 77

Figure 10‑23: Addresses and connection 6 77

Figure 10‑24: Addresses and connection 7 77

Figure 10‑25: Addresses and connection 8 78

Figure 10‑26: Wago Addresses 1 78

Figure 10‑27: Wago Addresses 2 79

Figure 10‑28: Wago Addresses 3 79

Figure 10‑29: Network Serial addresses 1 80

Figure 10‑30: Network Serial addresses 2 80

Figure 10‑31: Network Serial addresses 3 80

Figure 10‑32: Client addresses 81

Figure 11‑1: ID, CableLabel, GroupLabel Example 91

Figure 11‑2: Item example 92

Figure 11‑3: Double fault 93

Figure 11‑4: Sensorlist device column 94

Figure 11‑5: Sensorlist Interface column 95

Figure 11‑6: Sensorlist Module column 1 96

Figure 11‑7: Sensorlist Module column 2 96

Figure 11‑8: Wago Numbering 1 97

Figure 11‑9: Wago Numbering 2 98

Figure 11‑10: Pin column 1 98

Figure 11‑11: pin column 2 99

Figure 11‑12: Pin column 3 99

Figure 11‑13: Type column 1 100

Figure 11‑14: Type column 2 100

Figure 11‑15: Default Unit column 101

Figure 11‑16: open with Excel 102

Figure 11‑17: Excel 1 102

Figure 11‑18: Excel 2 103

Figure 11‑19: Excel 3 103

Figure 11‑20 : Excel 4 104

Figure 11‑21: Excel 5 104

Figure 11‑22: Field column 1 104

Figure 11‑23: Field column 2 105

Figure 11‑24: Custom sort 107

Figure 11‑25: Custom sort window 107

Figure 11‑26: Sorting device and interface 107

Figure 11‑27: Sorted sensorlist 108

Figure 11‑28: Cut and paste 1 109

Figure 11‑29: Cut and paste 2 109

Figure 11‑30: Duty names 110

Figure 11‑31: Call names 110

Figure 11‑32: Special issues 1 111

Figure 11‑33: Special issues 2 111

Figure 12‑1: Root folder 112

Figure 12‑2: Import devicelist 112

Figure 12‑3: Import sensorlist 112

Figure 12‑4: Checking system layout 113

Figure 12‑5: devicelist\_generated good 114

Figure 12‑6: Fault codes 114

Figure 12‑7: sensorlist\_generated 115

Figure 12‑8: Diff example 1 115

Figure 12‑9: Diff example 2 115

Figure 12‑10: Diff example 3 116

Figure 12‑11: Export shortcut 116

Figure 13‑1: changing baudrate old 117

Figure 13‑2: Changing baudrate new 117

Figure 13‑3: Changing Wago original 118

Figure 13‑4: Changing Wago lines 118

Figure 13‑5: Changing Wago numbers 118

Figure 13‑6: Inserting a Wago slice 1 119

Figure 13‑7: Inserting a Wago slice 2 119

Figure 13‑8: Inserting a Wago slice 3 120

Figure 13‑9: Excel trick 1 120

Figure 13‑10: Excel trick 2 121

Figure 13‑11: clean FT NavVision© folder 122

Figure 13‑12: Devicelist.dat and sensorlist.dat in network folder 123

Figure 13‑13: sensorlist.xls in root of project folder 123

Figure 13‑14: root folder after import sensorlist 124

Figure 13‑15: Comment example 1 125

Figure 13‑16: Comment example 2 125

Figure 13‑17: Comment example 3 125

Figure 13‑18: Changed example 1 126

Figure 13‑19: Changed example 2 126

Figure 13‑20: Changed example 3 126

Figure 13‑21: Failed example 1 126

Figure 13‑22: Missing example 1 127

Figure 13‑23: New example 1 127

Figure 14‑1: PLC added 129

Figure 14‑2: Wago overview PLC 129

Tables

Table 10‑1: RS232 Voltages Values 13

Table 10‑2: RS232 Cable Length 14

Table 10‑3: RS232 DB9 Pinout 14

Table 10‑4: DB9 – DB25 conversion 17

Table 10‑5: DB9 RS232 Loopback Plug 18

Table 10‑6: : Null modem cable without handshaking 18

Table 10‑7: Null modem with loop back handshaking 19

Table 10‑8: Null modem with full handshaking 20

Table 11‑1: Internet Protocol Suite 24

Table 12‑1: Modbus Message Structure 28

Table 12‑2: Properties of Modbus/ASCII and Modbus/RTU 29

Table 12‑3: Device and Modbus address ranges 29

Table 12‑4: Common Modbus function codes 30

Table 12‑5: Function 01 query structure 30

Table 12‑6: Function 01 answer structure 30

Table 12‑7: Function 02 query structure 31

Table 12‑8: Function 02 answer structure 31

Table 12‑9: Function 03 query structure 31

Table 12‑10: Hexadecimal values 32

Table 12‑11: Modbus Storage 33

Table 14‑1: Moxa Led Indicators 41

Table 14‑2: Pin assignment (CAN bus) 45

Table 14‑3: Communications mode selection 52

Table 14‑4: Data rate selection 52

Table 10‑1: Devicelist Columns 60

Table 10‑2: Protocol Options 61

Table 10‑3: Interface Options 62

Table 10‑4: Type Options 62

Table 10‑5: Device options 63

Table 10‑6: IP Ranges 63

Table 11‑1: Sensorlist columns 86

Table 11‑2: Sensor Type mode Read 88

Table 11‑3: Sensor Type mode Write 88

Table 11‑4: Unit Type 91

Table 11‑5: (Default) Unit options 102

Table 13‑1: Import Result fields 126

1. 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.

# Serial Communication

## 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.

## 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.*

## 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.

### 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.

### 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.

### 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.

### 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.

#### 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.

#### 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.

#### 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.

### 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 date 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 incomming bits.

For resynchronizing, the receiver scans the incomming 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.

### 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 2‑1: 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.

### 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 2‑2: RS232 Cable Length

### 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.

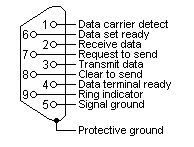


Figure 2‑1: RS232 DB9 Pinout

|  |  |  |  |
| --- | --- | --- | --- |
| **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 2‑3: RS232 DB9 Pinout

### 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.

#### 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 ment 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.

#### 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.

### Special connections

#### 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.

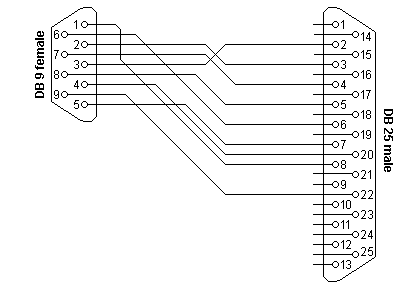


Figure 2‑2: RS232 DB25 to DB9 Convertor

|  |  |  |
| --- | --- | --- |
| **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 2‑4: DB9 – DB25 conversion

#### 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 2‑3: 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 2‑5: DB9 RS232 Loopback Plug

#### 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.

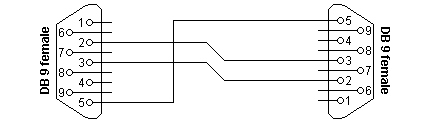


Figure 2‑4: Null modem cable without handshaking

|  |  |  |
| --- | --- | --- |
| **Connector 1** | **Connector 2** | **Function** |
| 2 | 3 | Rx <= Tx |
| 3 | 2 | Tx => Rx |
| 5 | 5 | Signal Ground |

Table 2‑6: : Null modem cable without handshaking

##### 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 normaly 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.

##### 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 2‑5: 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 2‑7: Null modem with loop back handshaking

##### 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. Incomming 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.

##### 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.

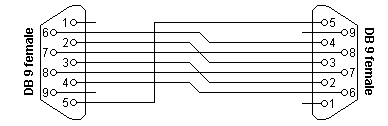


Figure 2‑6: Null modem with full handshaking

|  |  |  |
| --- | --- | --- |
| **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 2‑8: Null modem with full handshaking

##### 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.

## 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.

### 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 ±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.

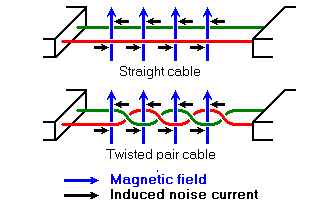


Figure 2‑7: Noise in straight and twisted pair cables

n 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 oposite 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.

### 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 kΩ 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.

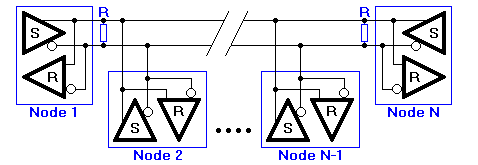


Figure 2‑8: RS485 network topology

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 Ω 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.

### 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 explicity 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.

### 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+ <=> RXD+

TXD- <=> RXD-

RXD+ <=> TXD+

RXD- <=> 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).

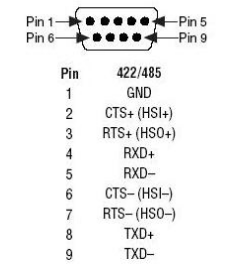


Figure 2‑9: 422/485 DB9 Pinout

*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.*

# TCP/IP

### 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.

### 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 3‑1: Internet Protocol Suite

### 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:

### The Network Access layer

#### 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 sychronization 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.

#### 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.

#### 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.

#### 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.

#### 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).

### 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.

# Modbus

## 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 hunderds 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 equiped 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.

## 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 4‑1: Modbus Message Structure

## 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 comming and the receive buffer is cleared. The main advantage of Modbus/ASCII is, that it allowes 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 4‑2: Properties of Modbus/ASCII and Modbus/RTU

## 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 substracted 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 4‑3: Device and Modbus address ranges

## 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 4‑4: Common Modbus function codes

### 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 4‑5: 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) mod 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 **N** |
| 4…**N**+3 | 0…255 | Bit Pattern of Coil Values |
| N+4 (…N+5) | LRC/CRC | Error Check Value |

Table 4‑6: Function 01 answer structure

### 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 4‑7: 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 **N** |
| 4…**N**+3 | 0…255 | Bit Pattern of Input Values |
| N+4 (…N+5) | LRC/CRC | Error Check Value |

Table 4‑8: Function 02 answer structure

### 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 4‑9: 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

## 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 view 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!

## 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 4‑10: Hexadecimal values

### 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 4‑11: 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.

### 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:

#### 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 to 218 = 22 = 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 Inputs / 8 bits per byte = 3 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.

#### 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).

#### 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).

### 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 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).

### 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

### 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.

# Bus-Protocols

## 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.

## 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).

## 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.

## 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.

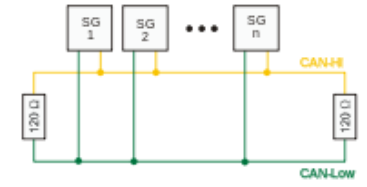


Figure 5‑1: Canbus topology

# Implementation in the FT NavVision© environment

## 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.

## Moxa UC-7110

### Overview

The Moxa UC-7110 series of RISC-based communication platforms (see Figure 6‑1) 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.

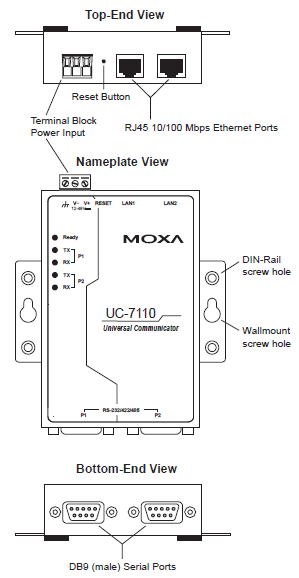


Figure 6‑1: Overview (MOXA)

### 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.

### 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 6‑1: Moxa Led Indicators

### Connecting to the network

Connect one end of the Ethernet cable to UC-7110’s 10/100M LAN1/LAN2 Ethernet port (see Figure 6‑1) 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 6‑2)
* 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.

### 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 6‑2).



Figure 6‑2: 8-pin RJ connector and pinouts

### 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:

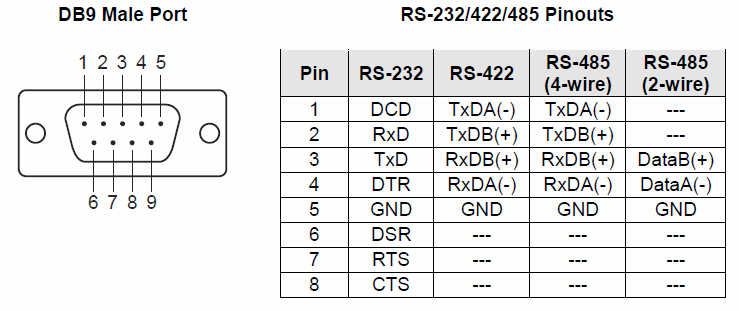


Figure 6‑3: Pin assignments (DB9 male port)

### Serial LAN server for Moxa

Under “Serial LAN ports > Serial LAN server” (see Figure 6‑4) the server to be assigned can be selected. In addition under “Type” the LAN server type can be selected.

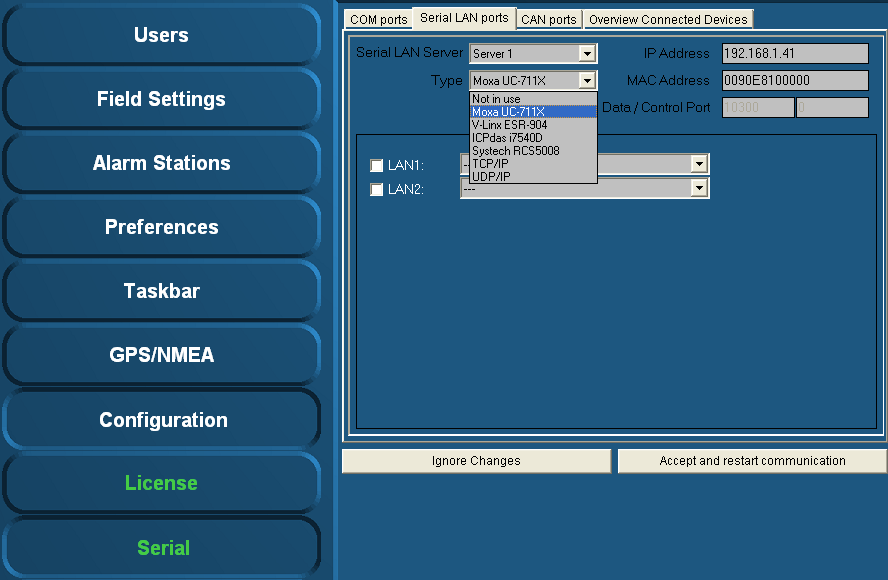


Figure 6‑4: Type (Moxa)

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

The Moxa is found under “Type” > “Moxa UC-711X” (see Figure 6‑4).   
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 6.4 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 6‑5: Comm Port Settings

In this additional configuration menu (see Figure 6‑5) 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 **resynchronize** 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

## ICP DAS i-7540D

### 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 6‑6).

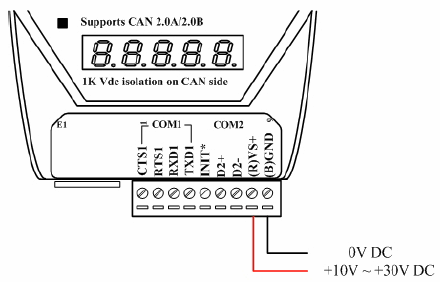


Figure 6‑6: Power connection

### 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 6‑7.

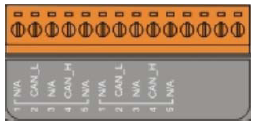


Figure 6‑7: CAN bus connector

|  |  |  |
| --- | --- | --- |
| **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 6‑2: Pin assignment (CAN bus)

*NOTE:   
The electronic circuit comprises of a 120 Ω 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 6‑8.

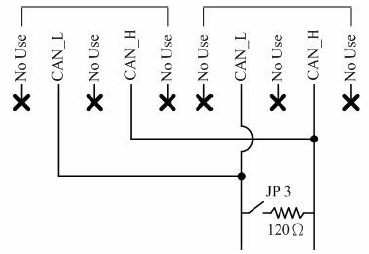


Figure 6‑8: Electronic circuit CAN bus connector

### Resistance check

Users should check the resistances of their CAN bus, before installing a new network as shown in Figure 6‑9.

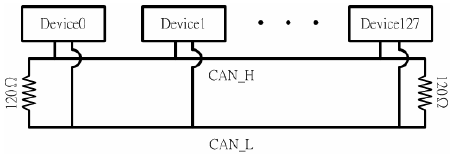


Figure 6‑9: Terminator resistor

### Com Ports

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

### CAN ports

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

* Interface
* Standard
* IP
* Group.

****

Figure 6‑10: 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 6‑10).

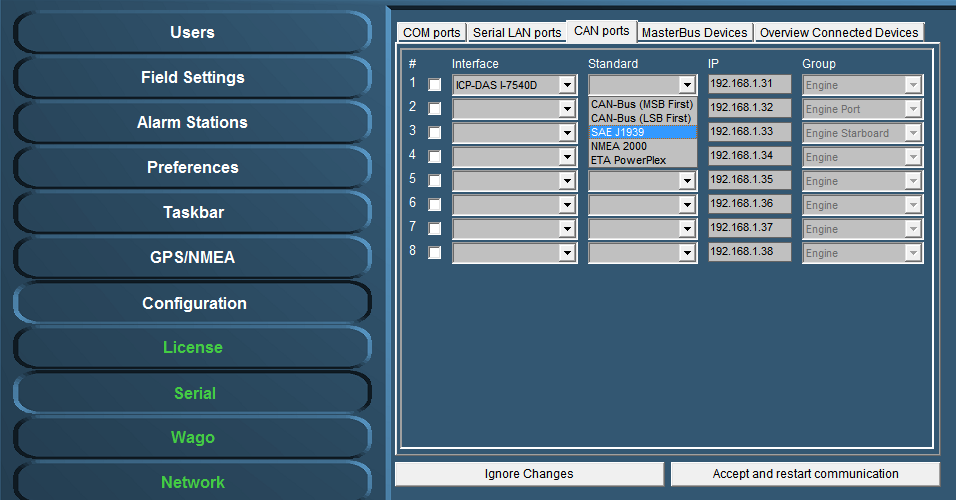


Figure 6‑11: Standard

Under Standard you choose the protocol you want to use with the interface (see Figure 6‑11). 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 6‑11).

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

### Type (ICPdas i7540D)

The ICPdas is found under “Type” “ICPdas i7540D” (see Figure 6‑12).   
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.

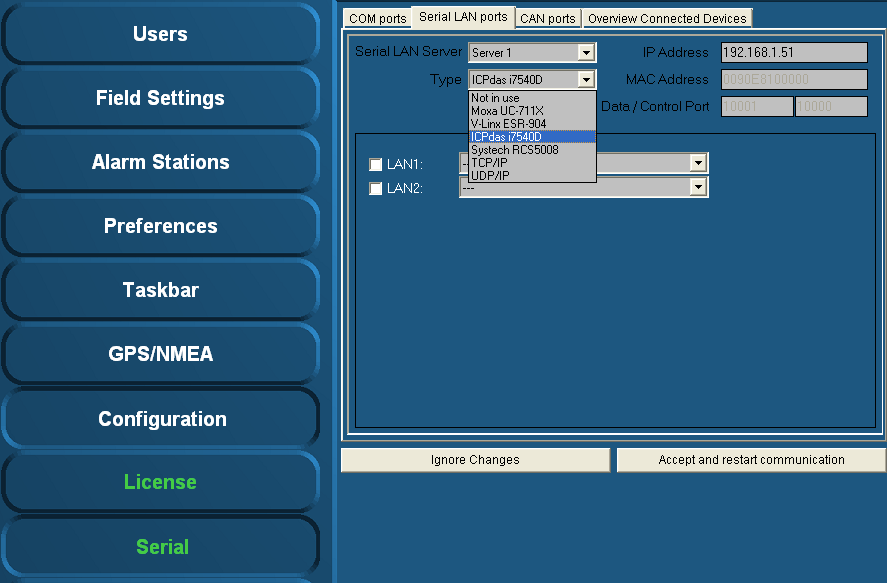


Figure 6‑12: Type (ICPdas i7540D)

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®.

## 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 6‑13).

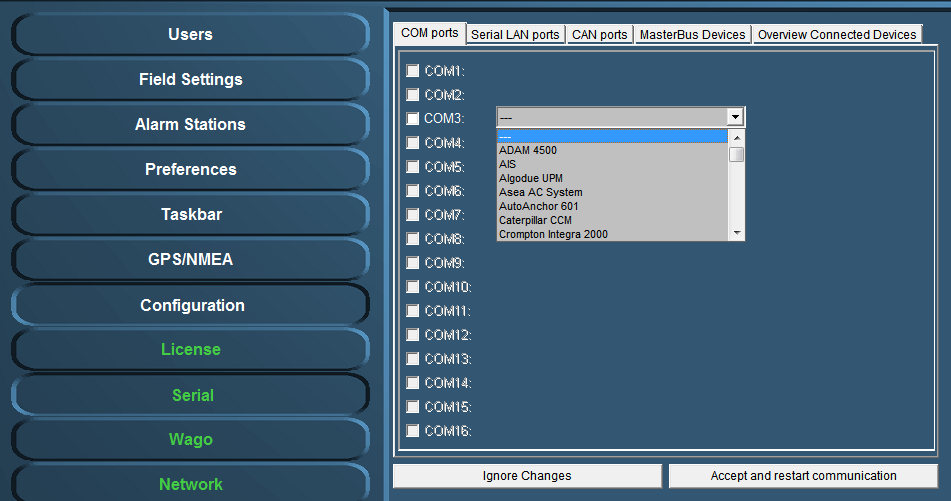


Figure 6‑13: Drop-down menu (device interfaces)

At completion, confirm the settings by clicking “Accept and restart communication” (see Figure 6‑13).

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 6‑14) to verify that the device interface (protocol) on “COM1” is correct. Repeat this procedure for all other listed COM ports.

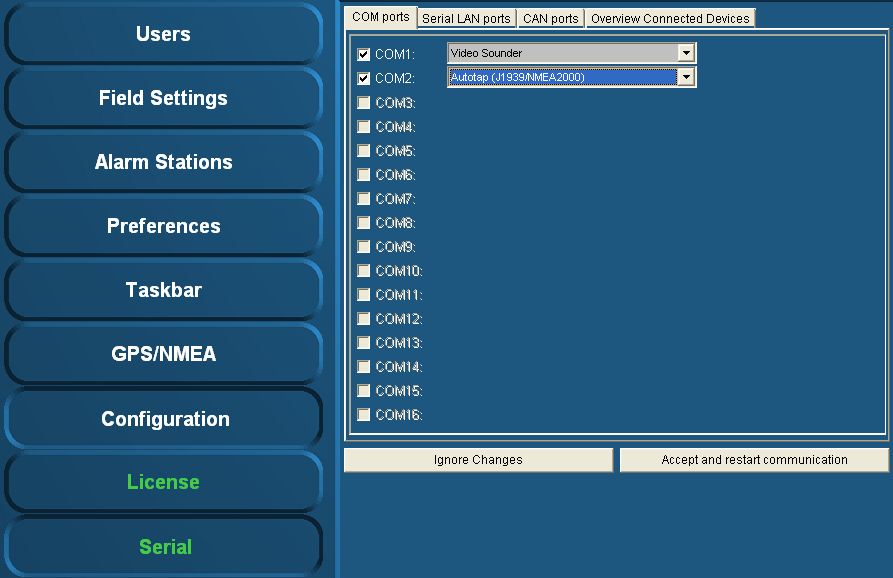


Figure 6‑14: COM port assignment

Additional information on the selected port can be configured by clicking on the sign behind the drop-down menu (see Figure 6‑15). A new box will open.

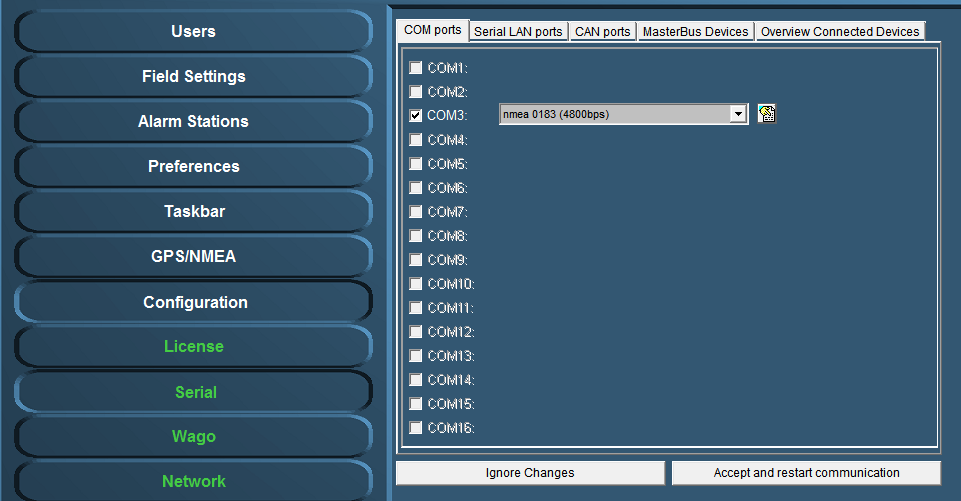


Figure 6‑15: additional configuration



Figure 6‑16: Comm Port Settings

In this additional configuration menu (see Figure 6‑5) 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

## 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.

### Operation

Select Data rate and mode by positioning the DIP Switches in accordance with Table 6‑3 and Table 6‑4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 6‑3: Communications mode selection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Baud rate** | **Switch 6** | **Switch 7** | **Switch 8** | **R11** |
| 1200 | OFF | OFF | OFF | 820 KΩ |
| 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 KΩ |
| 57600 | OFF | OFF | OFF | 16 KΩ |
| 115200 | OFF | OFF | OFF | 8.2 KΩ |

Table 6‑4: 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 6‑4 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 6‑17, Figure 6‑18 and Figure 6‑19 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.

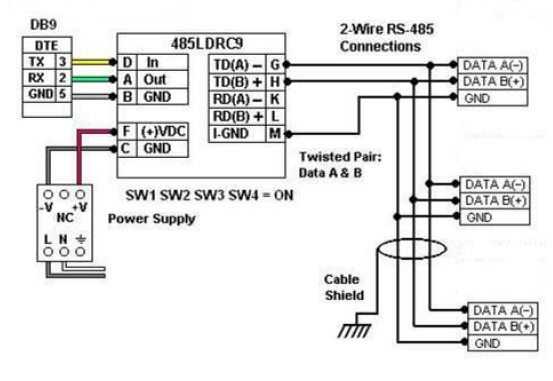


Figure 6‑17: 2-wire RS485

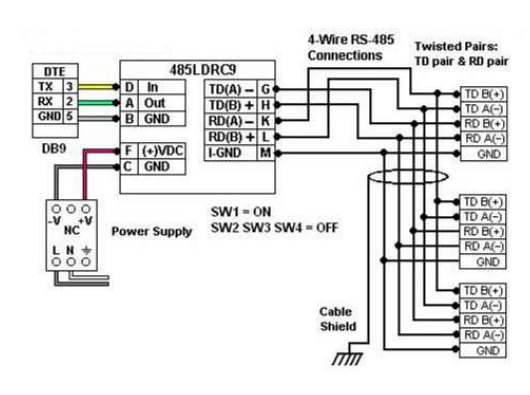


Figure 6‑18: 4-wire RS485

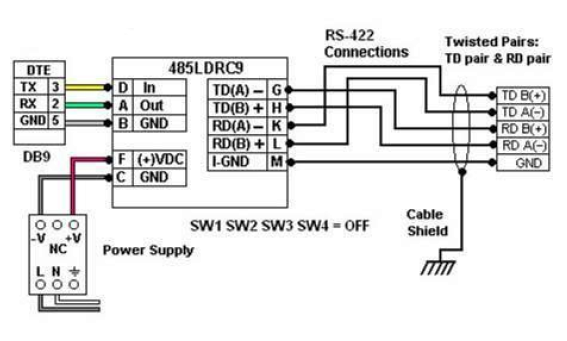


Figure 6‑19: RS422

Figure 6‑20 demonstrates how to use two converters to extend and isolate RS-232 signals.

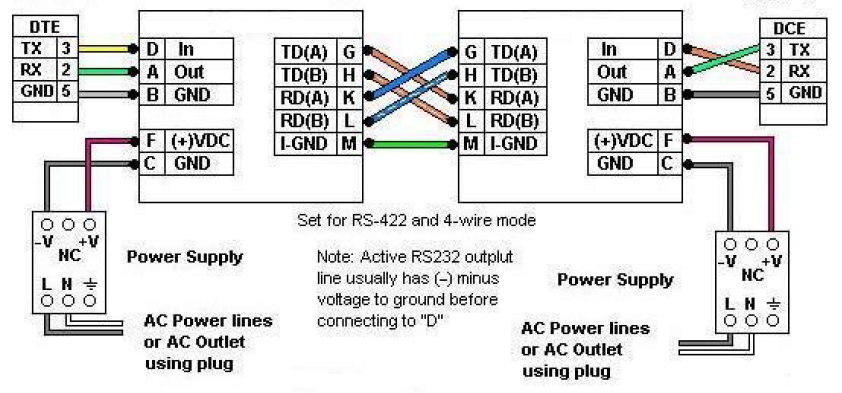


Figure 6‑20: Extend and isolate RS232

1. Introduction Sensorlist

In the course of time changing or altering settings in FT NavVision© has become quite time consuming. Due to the large infrastructure of the program, the vast amount of data and protocols we embed and the complexity of the large vessels that FT NavVision© is used on, just changing a simple connection type on a lot of sensors is a lot of work.

To automate this work we developed the “sensorlist” This list is a combination of a devicelist and a sensorlist in which all connections will be defined. The advantage is the fact that for changes, you can now work in a more simple “excel” document which allows you to change data more accurate and quicker and all in the same place.

This sensorlist has to be imported in FT NavVision© and after a new start-up the program will be up to date again.

This sounds easy. But you will need clear knowledge on how the sensorlist works to enjoy the full benefits of it. This manual will try to teach you everything you need to know about the sensorlist.

# What is the Sensorlist

## Introduction

The sensorlist is a validated description of the total FT system. Everything that is connected, whether it is a sensor, a serial connection, an engine etc. is represented in the sensorlist. The sensorlist is the start of where we build the topology, the network and all connected devices. Once imported, the sensorlist will be updated into a sensorlist\_generated file that is pretty much automated. Missing files, wrong connections, etc. will be highlighted or even changed. New connections will be highlighted and fails will be highlighted as well.

If the sensorlist is kept well up to date all changes to the system can be made from within the sensorlist. It will be your ultimate tool to easily change your setup, change names or even add new sensors or complete devices. Also the sensorlist is a nice tool to troubleshoot the system. It makes it easy to find double connections, wrong terminations, strange values etc.

Learn how to work with the sensorlist and you’ve got half the job done.

## Excel

The sensorlist is composed in Microsoft Excel. Some knowledge on working with Excel is absolutely necessary. That part of training lies beyond the scope of this manual. We refer to books and courses for Microsoft excel to learn the basics.

It is enough to have basic knowledge because the sensorlist itself is merely a form you have to fill in with the appropriate data. The sensorlist exists of two parts (tabs) which are the “devicelist” and the “sensorlist”. The devicelist (see Figure 8‑1) contains all the devices where he system consists of and is namely an enumeration of the topology of the system. The sensorlist (see Figure 8‑2) on the other hand is a list of all the sensors attached to the system together with the necessary information for the connection.

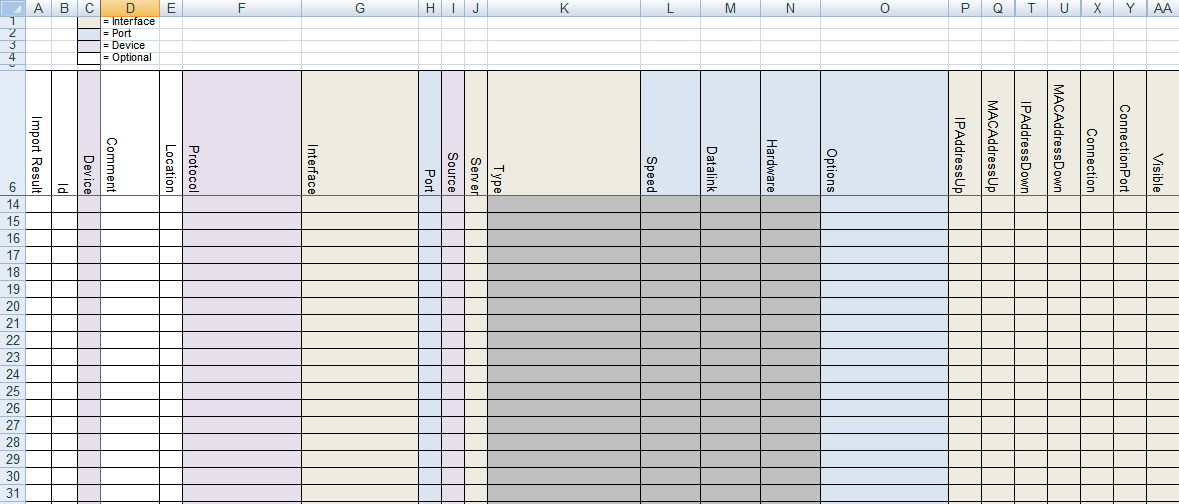


Figure 8‑1: devicelist

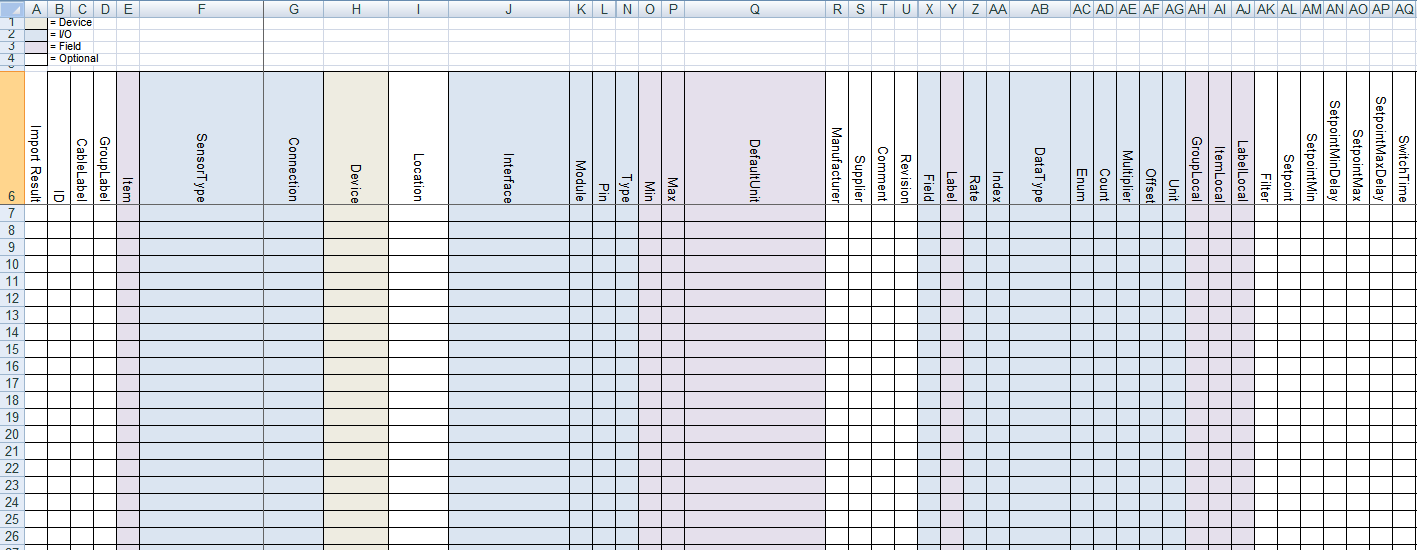


Figure 8‑2: Sensorlist

It goes unsaid that for filling in the sensorlist properly, you need to make sure that you have all the appropriate data form the shipyard available.

*:*

*Always ask Free Technics for the latest sensorlist. The sensorlist changes with upgrades in FT NavVision©.*

## Saving and naming

For working with the sensorlist always make sure that you use the latest version of Microsoft Excel. At this moment this is Microsoft Excel 2010. Although it is also possible to work with an earlier version, we will use this version as an example in this manual.

While working on the sensorlist, make sure that you save your work regularly to prevent loss of data. Goto startbutton>save as>Excel workfolder (See Figure 8‑3).

Make sure you choose the right folder to save to and the right format (in this case .xlsx) and save the sensorlist with a distinctive name. When working on ship A you can use for example “sensorlist\_shipA\_v1.1.xlsx”. When renewing or changing the sensorlist you can add a new version number to distinguish the different versions (i.e. “sensorlist\_shipA\_v1.2”).

When working on the ship or on the original configuration, make sure that, together with the newest sensorlist, you take a backup of the complete NavVision folder.

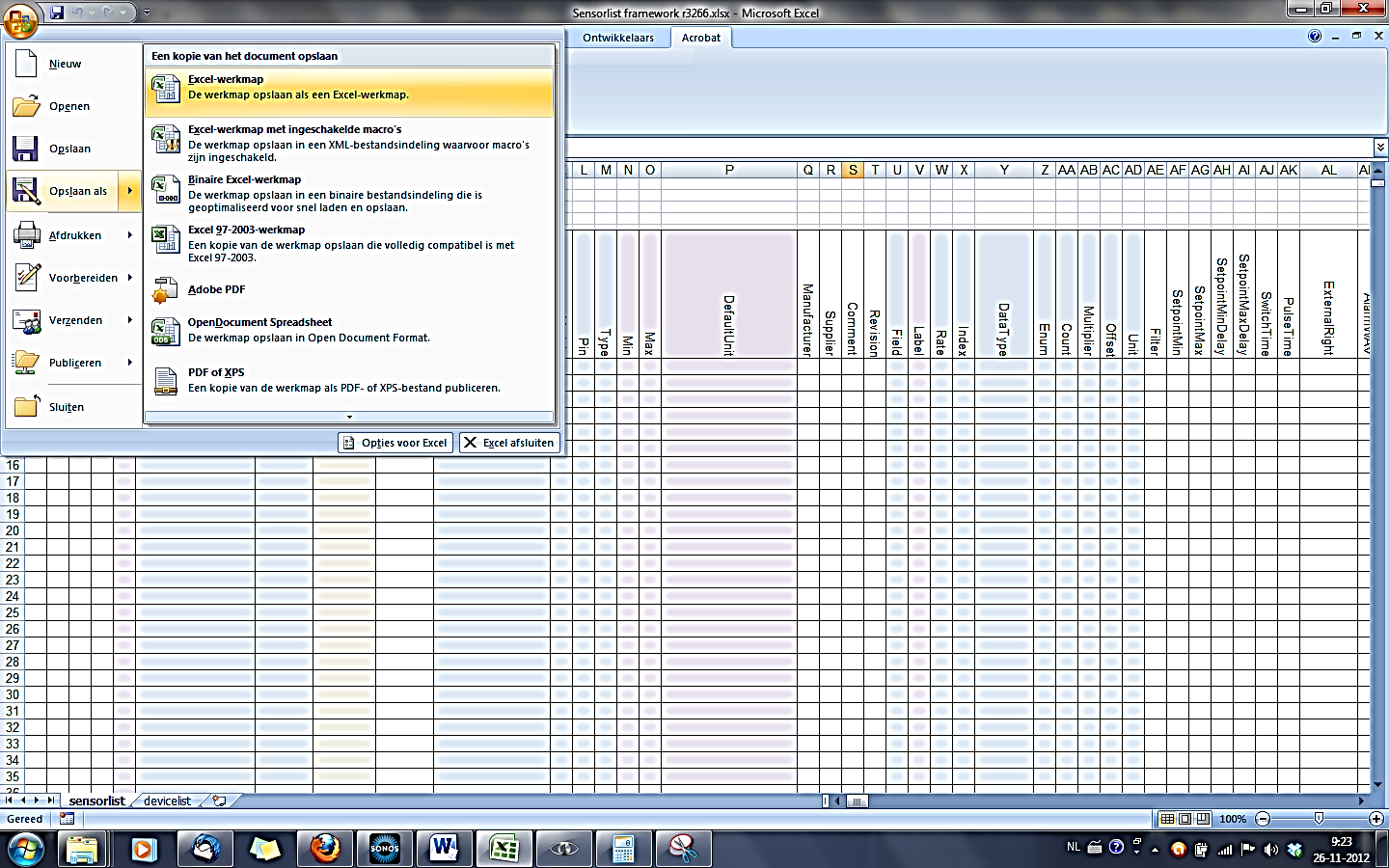


Figure 8‑3: Excel saving

## Saving as sensorlist for import

When saving the sensorlist to be used as import-file for FT NavVision© you need to do two things:

After choosing “save as” you go to the drop-down menu for the file type and choose

“Excel 97-2003-workfolder (\*.xls)” (see Figure 8‑4) while this is the supported format

for importing a sensorlist.

Save the sensorlist as “sensorlist.xls” no capitals.

Save this “sensorlist.xls” in the same folder as the sensorlist.xlsx you derived it from, so you can always check what has been changed and/or you can get back to previous versions.

For importing the sensorlist into FT NavVision© we refer you to Chapter 11.

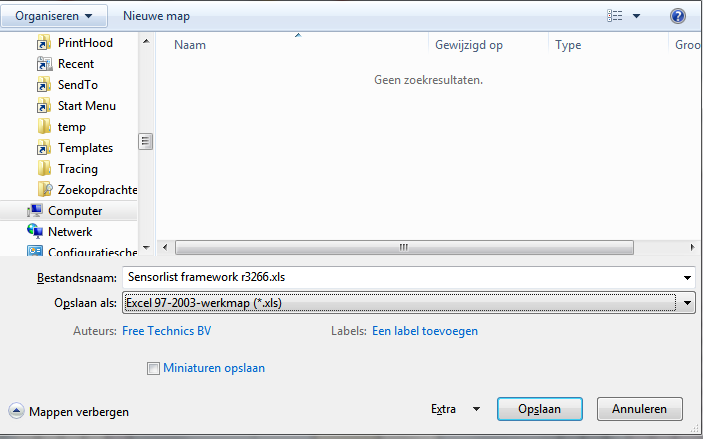


Figure 8‑4: saving as sensorlist.xls

# Devicelist

The devicelist is the part of the sensorlist that contains all the devices that are connected to FT NavVision© together with all the specific data concerning that connection. When opening a sensorlist framework you will see 2 tabs from which you will have to choose the tab named “devicelist”. (see Figure 8‑1).

## Introduction

The devicelist is separated in different columns which need to be filled with the right data. A few of the columns are optional and merely there for you to put your own comment. These columns are white. The other columns are almost all necessary for the proper working of the system and are colored differently. These colors belong to the different groups which can be divided into interface, port and device. Columns with the same color belong to the same group.

By defining all the devices the right way in the devicelist you will get a properly closed network once you import the sensorlist into the system. To do so you need to make a plan on how you need the network to be applied, a list of all the devices and a list of how everything will be connected. To make it visual it is best to make a single-line drawing of the topology for reference (see Figure 9‑1).

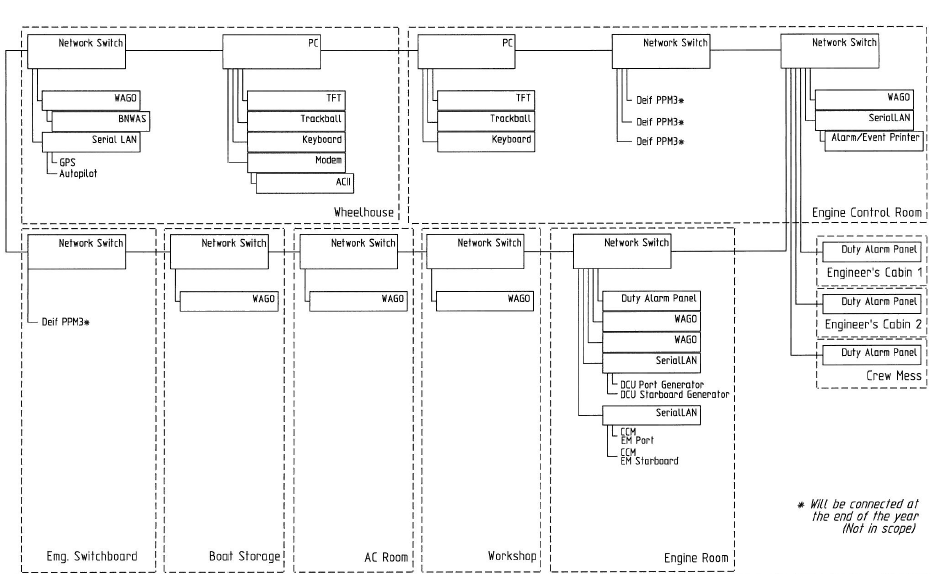


Figure 9‑1: single line drawing

## Columns

The columns in the devicelist are labeled in the first row. The fields underneath can be filled with free text or have a drop-down menu where you can choose a tag. These tags are mandatory and the devicelist won’t except tags that are not in the list for these columns.

The following columns are in the devicelist:

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| Import Result | Text | Checking value by FT NavVision© |
| ID | Text | Any given ID you want or need. |
| Device | Text | Identification of the device where the sensor/control or serial device is connected to. This text should be unique for each FT NavVision® device. The text is case sensitive |
| Comment | Text | Freely to add comment |
| Location | Text | Identification of the substation where the sensor/control is connected to in the FT NavVision® system. (i.e. ER or WH) |
| Protocol | Select | The protocol used for serial connections. (for options see Table 10‑2) |
| Interface | Text  (Index) | Choose the appropriate interface to distinguish the different interfaces in the system (for options see Table 10‑3) |
| Port | Value  (Index) | Port number on the FT NavVision® interface. For MOXA serial servers it’s 1 or 2. On a WAGO it’s always 1. |
| Source | Value  (Index) | Identification of multiple devices on a bus protocol. Used for example for Mod bus (ID byte) and CAN bus (SA byte). Default address is 1. |
| Server | Text | In some cases (like with OPC and WatchIO), you need to specify a server name. |
| Type | Text  (Index) | defines the type of module used to read/control the I/O. (for options see Table 10‑4) |
| Speed | Value  (Index) | The Baudrate the device is communicating with. See devices manual for the appropriate speed. |
| Datalink | Value  (Index) | Defines the parity, databits and stopbit. See devices manual for appropriate settings |
| Hardware | Value  (Index) | Serial communication protocol |
| Options | Text  (comma separated) | Divers special settings for various devices. See devices manual for need of these special demands. (for options see Table 10‑5). |
| IPAddressUp | IP-address | IP address of the FT NavVision® interface that’s connected to the device or sensor/control. Up-side (for explanation see Chapter 10.3). |
| MACAddressUp | MAC-address | MAC address of the FT NavVision® interface that’s connected to the device or sensor/control. Up-side (for explanation see Chapter 10.4). |
| IPAddressDown | IP-address | IP address of the FT NavVision® interface that’s connected to the device or sensor/control. Down-side (for explanation see Chapter 10.3). |
| MACAddressDown | MAC-address | MAC address of the FT NavVision® interface that’s connected to the device or sensor/control. Down-side (for explanation see Chapter 10.4). |
| Connection | Text | Specify the device (see first column) to which this device is connected |
| Connection Port | Value | Specify the port on the device where this device is connected to |
| Visible | Yes/No | Non mandatory field to tell FT NavVision© if the node needs to be visible in the network topology. |

Table 9‑1: Devicelist Columns

|  |  |  |
| --- | --- | --- |
| Option | Devicetype | Description |
| Adam | Serial | Advantech 4500/5000 series |
| AIS | Serial | AIS Data over Nmea |
| Algodue | Serial | Algodue AC monitoring module |
| Asea | Serial | Asea Shore converters |
| AutoAnchor601 | Serial | Chaincounter |
| BMV501 | Serial | Victron battery monitoring modules |
| BMV602 | Serial | Victron battery monitoring modules |
| BTM1 | Serial | Mastervolt battery monitoring modules |
| Can | I7540D | CAN bus |
| Cat | Serial | Caterpillar CAT-Link protocol. Link via CCM |
| CF Smartview | Serial | Broadband |
| Crompton | Serial | Crompton AC monitoring module |
| DssKeypad | Serial | CAN-based keypad |
| EM4000 | Serial | ELEQ AC monitoring module |
| EmpirBus | Serial | power supply systems |
| Frigomar\_626C | Serial | Airconditioning |
| FSI\_2DACM | Serial | Current measurement sensors from Falmouth Scientific Instruments |
| Generic | Serial | Gen-set |
| Gensys | Serial | GenSYS power management system (PMS) monitoring |
| J1708 | I7540D | SAE J1708 |
| J1939 | I7540D | SAE J1939 |
| KiloPakIguard | Serial | Kilopak I-Guard Generators |
| Littau Anchor | Serial | Anchoring |
| Lutron | Serial | Lutron Light system |
| MalinDraught | Serial | Draft System |
| Masterbus Modbus | Serial | Mastervolt charger/inverter modules through Modbus |
| Mastervolt | Serial | Mastervolt charger/inverter modules |
| Mitsubishi\_DMS\_II | Serial |  |
| ModBus | Serial/IP | Modbus ASCII/RTU Serial or TCP/IP |
| ModBus Slave | Serial/IP | Modbus ASCII/RTU Serial or TCP/IP |
| MPC30 | Serial | Inkjet printer |
| MTU | Serial | MTU MCS-5 system. Connections to be made through LOP, PIM or PCS. |
| MVECP | Serial | PaxMAN Engine Control Unit |
| Nke | Serial | NKE Navigation Instruments and Autopilots |
| Nmea | Serial | NMEA 183 |
| Nmea2000 | I7540D | NMEA 2000 over CAN. |
| PC |  | Server or Client PC |
| PPM3 | Serial | Deif power management system (PMS) monitoring |
| Printer | Serial | Printer |
| Sae | I7540D | SAE |
| SD41 | Serial |  |
| SMS | Serial | SMS Module (Tango blackbox modem) |
| Sounder | Serial | Black box video sounder |
| SygoDraft | Serial | Sygo Draft systems |
| TMA4S | Serial | Tank Gauging System |
| Vaisala\_CL31 | Serial | Vaisala cloud detection sensor |
| Vaisala\_LT31 | Serial | Vaisala LT series visibility sensor |
| Vaisala\_PTB330 | Serial | Vaisala Digital Barometer |
| Vaisala\_PW | Serial | Vaisala PW series visibility sensor |
| VDR | Serial | VDR output connection (NMEA 183 based) |
| Victron | Serial | Victron charger/inverter modules |
| VictronVEBus | Serial | Victron BUS |
| VisiplexPaging | Serial | Alarm paging system |

Table 9‑2: Protocol Options

|  |  |
| --- | --- |
| Interface | Description |
| Camera 01, Camera 02, etc. | Define the different IP cameras on the network. Do not use the same Camera twice. |
| CAN 01, CAN 02, etc. | Use a separate interface-ID for each Canbus device. If you, for example, have two I7540D devices, you choose CAN 01 for the first and CAN 02 for the second |
| Client 01, Client 02, etc. | Clients, can be DAP’s, Client PC;s and al workstations that aren’t servers. Each one needs to be provided with a separate Client-ID |
| Local Serial | Choose this interface setting for a serial connection that is directly connected to the server. |
| Network Serial 01, Network Serial 02, etc. | Network Serial devices are devices like the MOXA that are used as an interface between serial to LAN. Each interface needs a distinctive interface. More ports on the same device will get the same interface |
| Printer | When a printer is connected |
| Server 01, Server 02, etc. | The main workstations will act as server. Each server gets its own interface |
| Settings | Use if the line contains a setting for FT NavVision© |
| Switch 01, Switch 02, etc. | Interface for network switches. Although the switches have multiple ports, you only use one interface for each switch. |
| Wago 01, Wago 02, etc. | When a Wago is connected, choose Wago as interface. Each Wago gets its own interface |
| WatchIO | Special connection type for WatchIO |

Table 9‑3: Interface Options

|  |  |
| --- | --- |
| Type | Description |
| Axis 241Q | Axis IP camera interface |
| Carlisle Finch | Searchlight interface |
| GW003 |  |
| ICPdas i7540D | CANbus to serial interface |
| ModBus TCP/IP | Modbus over TCP/IP |
| ModBus TCP/IP Slave | Modbus over TCP/IP slave |
| Moxa UC-711X | Serial to Ethernet interface |
| PC | Server, DAP, panel PC, etc. |
| Printer | Printer |
| Serial TCP/IP Client | TCP/IP client over serial connection |
| Serial TCP/IP Server | TCP/IP server over serial connection |
| Serial UDP/IP Client | UDP/IP client over serial connection |
| Serial UDP/IP Broadcast | Typical broadcast over UDP/IP |
| Switch | Switch to connect different devices |
| Telnet | Telnet |
| V-Linx ESR-904 | Serial to Ethernet interface |
| Wago | PLC |
| Wago 750-881 | PLC type specific |
| Wago 750-882 | PLC type specific |

Table 9‑4: Type Options

|  |  |
| --- | --- |
| Device | Description |
| AlarmDataLoss | Gives an alarm on loss of data on the specific port. Works only when the interface have had a connection before. |
| DTR | When Data Terminal Ready needs to be set High |
| dtr | When Data Terminal Ready needs to be set Low |
| RTS | When Request to send needs to be set High |
| rts | When Request to send needs to be set Low |
| RTU | Sets the port to RTU |
| ASCII | Sets the port to ASCII |
| MSBFirst | Set reading of Most Significant Bit First |
| LSBFirst | Set reading of Least Significant Bit First |
| MSWFirst | Set reading of Most Significant Word First |
| LSWFirst | Set reading of Least Significant Word First |
| MaxWordCount= | Some Modbus protocols can read only an x-amount of registers at one time. While FT works with the Modbus standard of 123 registers, you need to limit the max value of words that FT is questioning. For Heinen Hopman for example it is “MaxWordCount=10” |
| NoHoles | Some Modbus protocols can’t handle it when there are a lot of unused registers between the different calls. With the option “NoHoles” all the registers that are not used will be ignored. |
| KeepAlive | Especially for H&H interfaces, but can be used in other Modbus protocols. When a Modbus call doesn’t get an answer in the predefined time, it will keep the question alive until answered. |
| OutputFirst | Especially for H&H interfaces, but can be used in other Modbus protocols. If a request is send (Modbus function 6) it will be handled before other questions |

Table 9‑5: Device options

## IP-addresses

### Introduction

At Free Technics© we use a specific set of IP-addresses for our connections. We use the 172.16.x.x range for the i/o side of our system and the 172.17.x.x range for the next ring. If there are more rings connected than these two we go on with 172.18.x.x etc. As you can find in the “installation and commissioning manual” we use also specific ranges for the different devices and interfaces (see Table 9‑6).

|  |  |
| --- | --- |
| **Detail** | **IP-Address** |
| PC I/O | 172.16.x.x (172.16.24.35 for key number 2435) |
| PC I/O next ring | 172.17.x.x (172.17.24.35 for key number 2435) |
| Duty Alarm Panels  (DAP) | Using range x.x.1.8y  Depending on the network connected, this will result in:  DAP 1: 172.16.1.81  DAP 2: 172.16.1.82  DAP 3: 172.16.1.83 |
| Serial LAN servers | Using range 172.16.1.4x (attached to I/O subnet 172.16) INT 1: 172.16.1.41 INT 2: 172.16.1.42 INT 3: 172.16.1.43 |
| Wago | Using range 172.16.1.9x (attached to I/O subnet 172.16) Wago substation 1: 172.16.1.91 Wago substation 2: 172.16.1.92 Wago substation 3: 172.16.1.93 |
| CAN-Interface | Using range 172.16.1.3x (attached to I/O subnet 172.16) CAN interface 1: 172.16.1.31 CAN interface 2: 172.16.1.32 CAN interface 3: 172.16.1.33 |
| Axis | Using range 172.16.1.24x (attached to I/O subnet 172.16) Axis cam server 1: 172.16.1.241 Axis cam server 2: 172.16.1.242 Axis cam server 3: 172.16.1.243 |

Table 9‑6: IP Ranges

We work from the single line drawing to make it possible to get all the IP-addresses to the right line in the devicelist. Also it is wise to start with building the topology of the single line drawing in to the devicelist. This way you will get closed rings.

### IPAddressUp- IPAddressDown

In the single line drawing you best number all the connections upfront so you minimise the mistakes. Say that the IPAddressUp is number 1 and the IPAddressDown is number 2 (try to make the Up-address to go to the i/o side of the system). The drawing will look like the following:

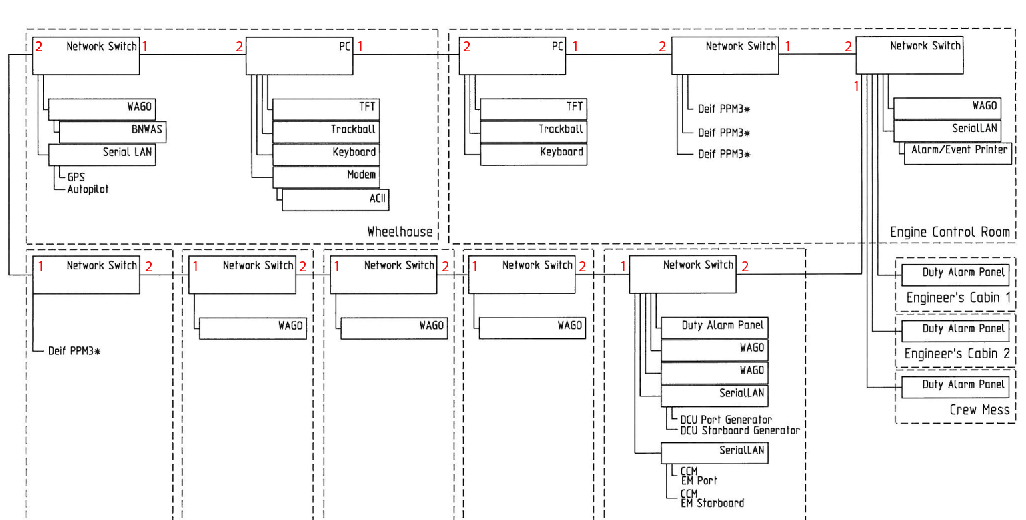


Figure 9‑2: numbering the drawing

As you can see we have numbered all the devices with the numbers 1 and 2. Now number 1 is the “IPAddressUp” and number 2 is the “IPAddressDown”. So, for example. for the PC (let’s assume it has key number 3035) the number 1 side , in the sensorlist IPAddressUp, will be 172.16.30.35. the number 2 side, in the devicelist the IPAddressDown, will be 172.17.30.35. You’ll notice that de down-side is considered as another ring and will get another IP-range.

While the Switches do not have an IP-address they need not have one of the above mentioned IP addresses assigned. More on how to build that in to the devicelist in chapter

9.5.11

The interfaces such as the Wago, the SerialLan etc. will get their own IP address as well as a port connection (also see Chapter 9.5.11)

## Mac addresses

### Introduction

To be identified in a network it is sometimes needed that the MAC address is available to distinguish two or more of the same devices. The MAC address is a unique number that is only conjugated to one device. So if there are two or more Moxa’s on a network, FT needs to separate them with their distinct MAC number. For these devices you need to fill in the MAC address in the devicelist.

## How to implement this in the devicelist

### Introduction

To implement all the devices in the devicelist we will now give an example on the basis of the single-line drawing we presented earlier. We will try to give you a method that is easy to use, yet with the least possibilities to make any mistakes. Once you get familiar with the devicelist, you can derive your own method of working. The ultimate goal of the devicelist is to make the topology connected flawlessly which can be checked in FT NavVision©.

### The devices

As seen earlier a device name is “Identification of the device where the sensor/control or serial device is connected to. This text should be unique for each FT NavVision® device. The text is case sensitive”

So to use it later on in the sensorlist it is mandatory that you give each device a name that is unique. So use names as “GPS” or “Port Engine” or any other name as long as it is descriptive for your device.

As the example single line drawing shows we have two servers that are in the topology. For now let’s call them Server 1 and Server 2. So the first devices that we put into the devicelist are these two. Please remember wich one you call Server 1 and which Server 2 (easiest way to do is to write it down in the drawing). Now let’s put them in the devicelist (see Figure 9‑3).

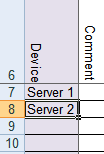


Figure 9‑3: Filling device column 1

Next thing we find in the drawing are a lot of switches. We can number the switches or give them the location as a tag, or even both. That way we know later on about which switch we are talking. In this case we work with the descriptive name and we get the following:

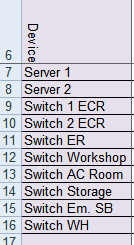


Figure 9‑4: Filling device column 2

Next we take the Wago’s. Lets also take the descriptive name for that and we get the following:

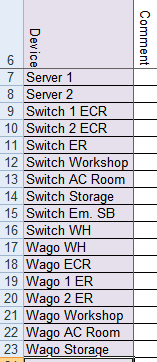


Figure 9‑5: Filling device column 3

Last items are some serial Lan interfaces and some DAP’s (Duty Alarm Panels). Adding these makes the device column complete and gives the following result:

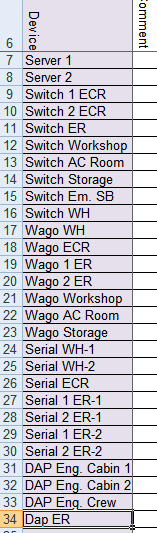


Figure 9‑6: Filling device column 4

*:Serial Lan’s have multiple ports as well as some other interfaces. Make sure you put both ports (if in use) in the device column. You can give it a name like “serial 1 ER-1”wich is the first port of the first serial Lan interface in the engine room, or you can name it after the sensor or device that is connected to that port (in this case “Port Generator”). We prefer the first option because sometimes you have multiple GPS’s on the system or whatever and you will get confused.*

This completes the filling of the device column. If you compare it to your drawing, you can see that all the devices in the topology are now in the devicelist. Time to move to the next column.

### Comment

We leave this up to you. If you have something you need to remember with a certain device you can keep it here. FT NavVision© doesn’t use this information.

### Location

The location is the Identification of the substation where the sensor/control is connected to in the FT NavVision® system. You can easily get that from the drawing and put it in the Location column. In our case we will get the following:

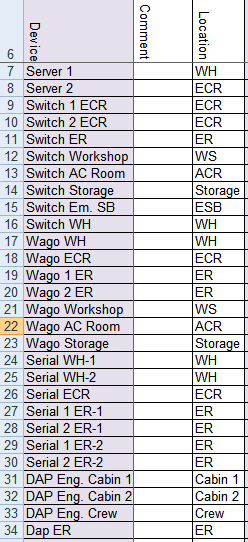
`

Figure 9‑7: Filling location column

### Protocol

To choose the protocol you have a lot of options. To make it easy there is a drop-down box. Just click the appropriate field and look in de drop-down menu if you can find the right protocol. (for explanation on the options see Table 9‑2).

The first ones are easy. We have two server and a couple of switches and Wago’s. The protocols for these are quite clear. Fill them in and you get the following:

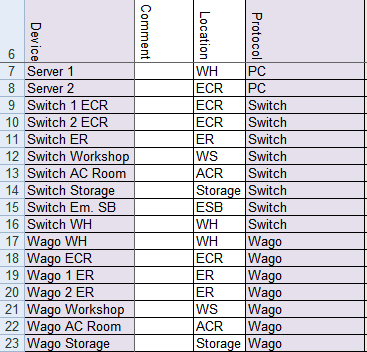


Figure 9‑8: Filling protocol column 1

For the serial Lan’s we need to know what is connected. Look at the drawing and find the right protocol for the connections you see there. For example the GPS and Autopilot are NMEA, the generators and engines are Caterpillar and the printer is a printer. Find these protocols in the drop-down menu and you’ll get the following:

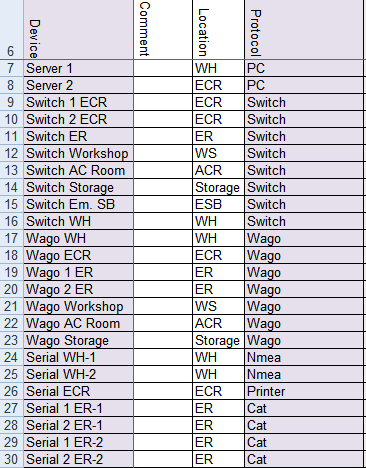


Figure 9‑9: Filling protocol column 2

The DAP’s are PC’s so that ends the filling of the protocol column as follows:

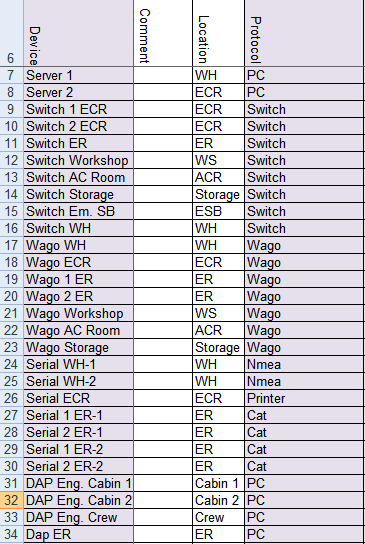


Figure 9‑10: Filling protocol column 3

### Interface

The interface is the name of the sort of interface that is used to get the data into the system. This is used to distinguish the same sort of interfaces by a separate number.

It speaks for itself for most of the interfaces. Only notice that the PC’s will be divided into Servers and Clients and for an interface that has multiple ports you need to add the same interface for each port. (for options see Table 9‑3).

The result will be as follows:

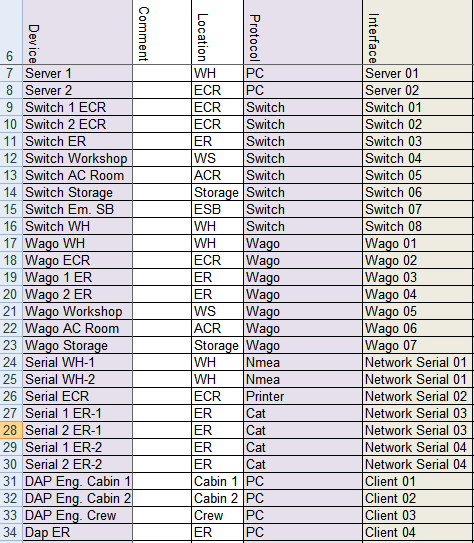


Figure 9‑11: Filling interface column

### Port and Source

The port defines the port on the device that the sensor or whatever is connected. So in our case for example we have a port and a stbd engine that are both connected to the same Serial Lan. While they are separately connected one will be on port 1 and one will be on port 2. In this “Port” column you can specify this as follows:



Figure 9‑12: Port and Source 1

As you can see, the first port on the Serial Lan gets number 1 and the second port gets number 2. In this case the “source” will stay at number 1.

Normally the Port and Source will be “1”

This will result in the following list:

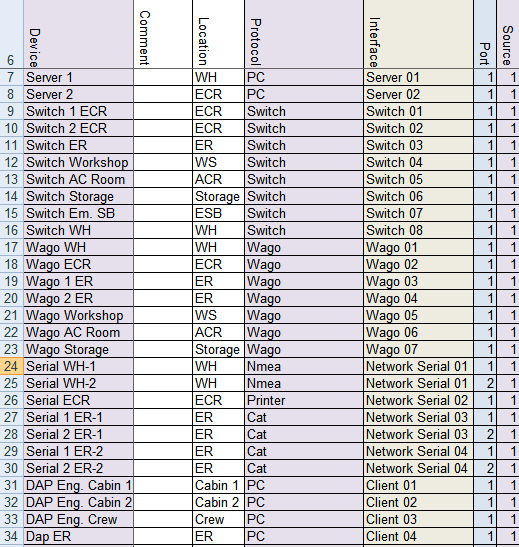


Figure 9‑13 Port and Source 2

*: The source can be as high as 256. When, for instance, you have Modbus/TCP connected through a serial Lan it is possible that there are multiple devices connected through the same bus. The ID’s of these devices can also be put in the “source” column.*

### Type

defines the type of module used to read/control the I/O. (for options see Table 9‑4). As you can tell from the options table it is mostly used when the i/o source is connected to the FT system through some type of interface. This can be Serial Lan, TCP/IP (Modbus or Serial) and a few more options.

Keep in mind that this is part of the interface-side in the sensorlist/devicelist. If the interface needs some extra specification, you will put it here. Most of the fields will be head on what it says but as you may have noticed earlier the Network Serial interface will need some additional information. This is directly shown when you choose Network Serial, the fields “type, speed, datalink and hardware” will change color (see Figure 9‑14).

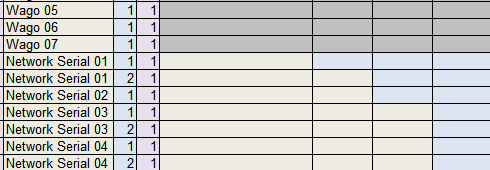


Figure 9‑14: Network Serial colors

In our example, while we do not have any special interfaces, it is quite obvious what to choose. Only some extra information will be needed for the Network Serial. The rest will look like the following:

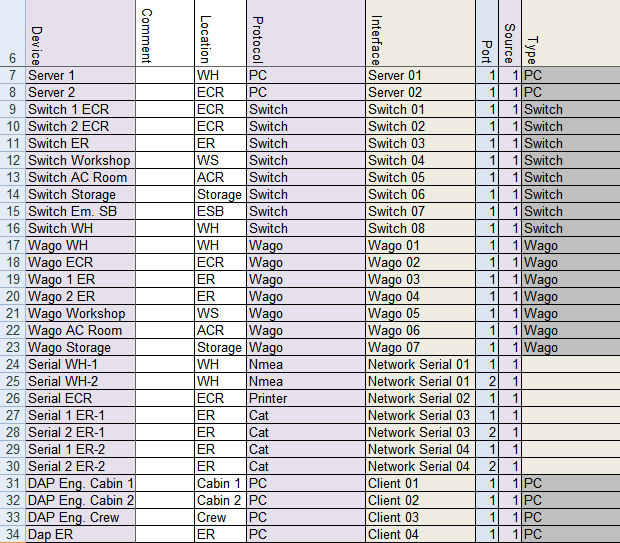


Figure 9‑15: Filling Type column 1

For the Network Serial you will have to look at the Interface in the same way you do as in the rest of the “Type” column. Only in this case it will probably be a serial to Ethernet interface that will go here. You can’t get that directly from the single line drawing, so you need to find out up front. In this example all the network serial interfaces are Moxa’s so we will fill that in. This results the Type column as follows:

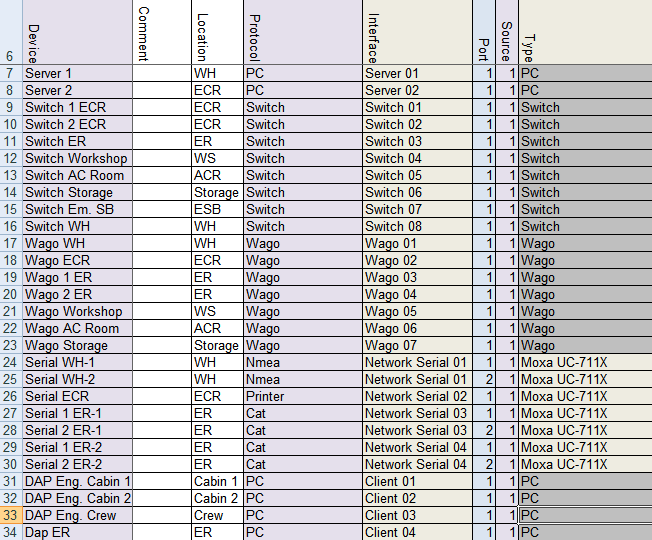


Figure 9‑16: Filling type column 2

### Speed, Datalink and Hardware

The speed, datalink and hardware are figures that you will find in the manuals of the attached sensors, engines, i/o or whatever. If a GPS is connected to the Serial interface, you probably Will find a paragraph describing that it is NMEA, at a speed (baudrate) of 4800. None parity, eight data bits and 1 stop bit and that the serial connection is RS232. (see FT Port Connections and Protocols manual for more information)

This is the data that you need for these columns. FT NavVision© will set the ports on the devices, accordingly to what you put here. Make sure that you have the data ready before starting to build the devicelist.

On the basis of our example drawing we know the following:

* GPS 9600 None 8 1 RS232
* Autopilot 115200 None 8 1 RS232
* Printer 9600 None 8 1 RS232
* Generator 115200 None 8 1 RS485
* Engines 115200 None 8 1 RS485

So with this we can finish these columns and we get the following:

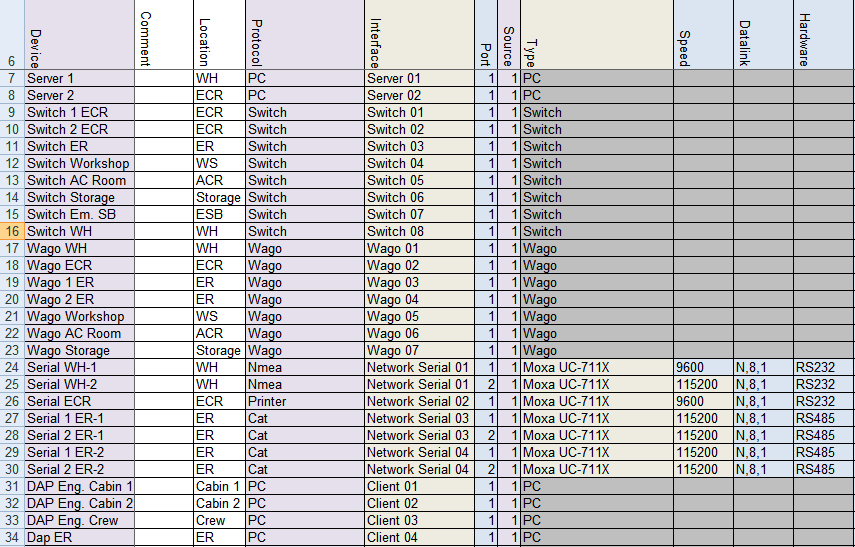


Figure 9‑17: speed, datalink and hardware

### Options

Some devices need some special attention. Mostly because they have another interpretation on protocols, or just that their interpretation deviates from the one that Free Technics uses. To make it easier we have made a separate column where we can put those differences. (for options see Table 9‑5). You can use more options on one device. Just put them in the same cell “comma separated”.

### IP addresses and MAC addresses

As described earlier in Chapters 9.3 and 9.4, we now need to put in the diverse IP addresses and MAC addresses to let the system know how everything is connected. This is essential because the system needs to know where to transfer requests and to make sure that the system is connected the right way. Also the alarms on lost connections depends on these figures.

Let’s look back at the single line drawing that we made (see Figure 9‑1). We already gave the Up-link the number 1 and the Down-link the number 2. These are two separate rings an so they will get a separate IP-range. For the Up-link we start with 172.16.x.x.

Based on our drawing we state that the WH-pc has the key 2637 and the ECR-pc has the key 2636. In this case we can fill in the x.x with the key number. While we always start in the direction of the i/o we will start at the ECR-pc. The ECR-pc port 1 (Up-link) will get the IP address from the first IP range ending with the key number. In this case that will be 172.16.26.36. This address we put in the IPAddressUp behind the Server 2 (see Figure 9‑18).

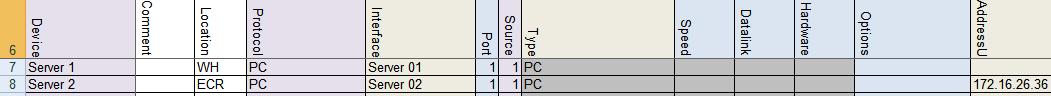


Figure 9‑18: Addresses and connection 1

From port 1 at the ECR pc we come at a switch in the ECR. As we use the single line drawing as our reference, it is easier that we fill in the devicelist as we follow the main lead of this drawing instead of filling the IP addresses one by one following the order in the devicelist. This will also give you a good indication on any mistakes that you might have made in the devicelist.

So the next column we now will look at is the “connection” column. Here you need to put the device that the server 2 is connected to. In this case that will be the “Switch 1 ECR” as we named that switch in the “device” column (see Figure 9‑19).

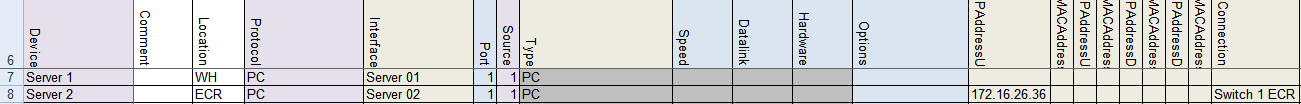


Figure 9‑19: Addresses and connection 2

As we connect from the Up-link from one device to the Down-link of the other device, we now that we connect Server 2 Port 1 to the Switch 1 ECR Port 2. So in column “ConnectionPort” We type “2” (see Figure 9‑20).

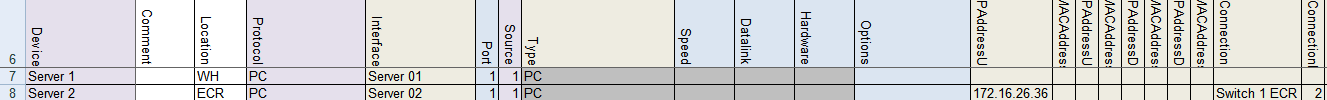


Figure 9‑20: Addresses and connection 3

As we mentioned, we will follow the single line drawing. So the next row that we will process is the row of the “Switch 1 ECR”. A switch doesn’t have an IP address nor a MAC address is needed. This only leaves us to fill in where a switch is connected at. Following the drawing we see that the “Switch 1 ECR” is connected to the “Switch 2 ECR”. So in this case we need to fill in that it is connected to “Switch 2 ECR” at Port 2 (the Down-link of that switch). This way we come to the following:

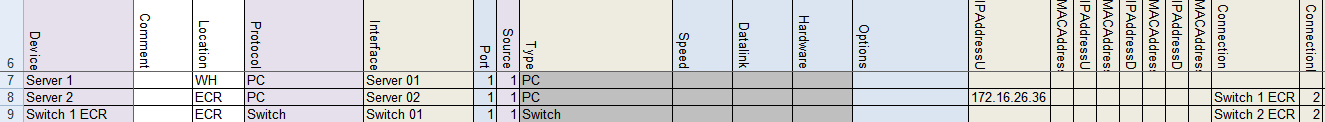


Figure 9‑21: Addresses and connection 4

As you can see in the single line drawing, the main ring is connected through a bunch of switches until you come to the WH server. So the rest of the rows are quite the same. After filling in all the switches you will get to the following:

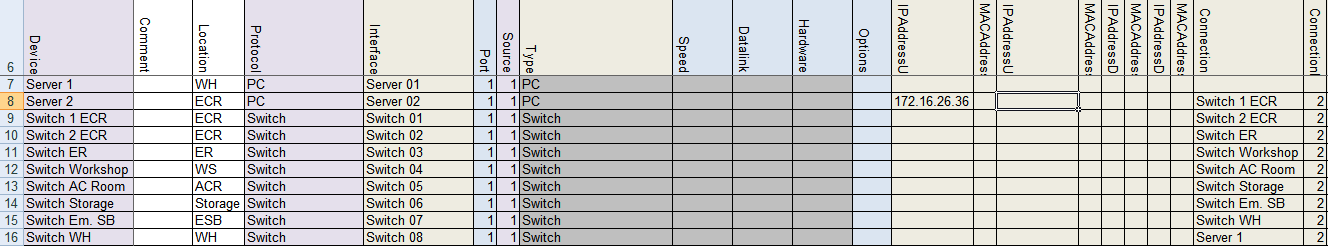


Figure 9‑22: Addresses and connection 5

From the last switch we come to the WH Server or as in the devicelist “Server 1”. This is the tricky part. As you find in the single line drawing, that switch is connected to Port 2 of the Server. As we mentioned earlier, the Up- and Down-link are two separate rings. These rings need to go round all the way. So the Port 2 of the WH server has to be in the same IP-range. With the key number of the WH sever being 2637 the IP address of that port will have to be 172.16.26.37. Now while this is Port 2 on the WH server (the Down-link) you will have to put that IP address in the “IPAddressDown” column. See the following figure:

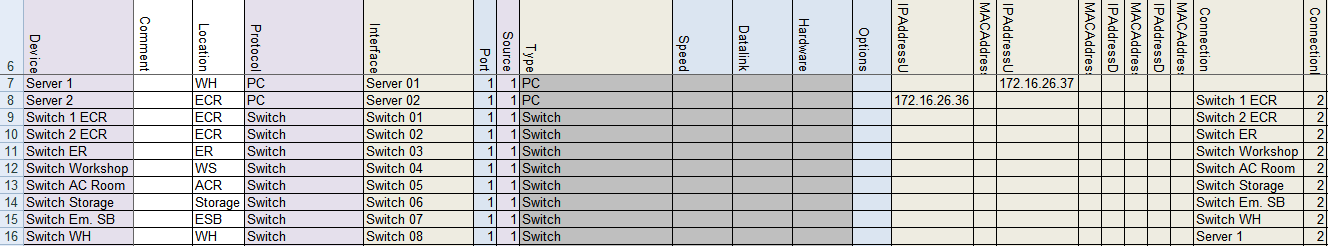


Figure 9‑23: Addresses and connection 6

So now the ring is completed. You can use the devicelist to check if the lines are correct.

Next thing we do is close the ring in the opposite direction. This will be the next IP-range, so 172.17.x.x.

Starting again with the Server 2 the ECR server We are going to address the Down-link port or Port 2 of that server. While it has the key 2636 the IP address for that will be 172.17.26.36 and has to be filled in at the “IPAddressDown” column at the Server 2 row. See following:

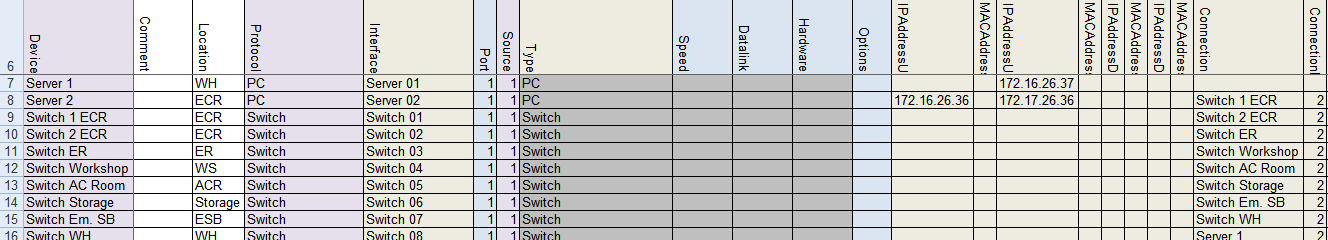


Figure 9‑24: Addresses and connection 7

Concluding that it is connected to Port 1 on the WH server (Server 1) we can now say that the IP address in the “IPAddressUp” column at the Server 1 row must be 172.17.26.37. See following:

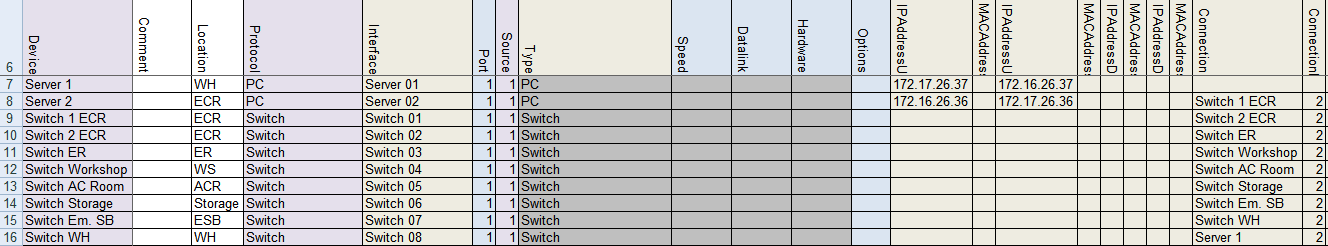


Figure 9‑25: Addresses and connection 8

Now the circle is really connected properly and FT NavVision© can calculate all the connections etc.

#### Other devices

The other devices such as Wago, Network Serial and Clients will not have a Down-link (unless they are in a double-wired systems which goes beyond the scope of this manual), but they do need an IP address, a connection port and sometimes a MAC address.

Let’s start at the Wago. As we saw in Table 9‑6 the Wago use the IP range x.x.1.9y. While the connection lies in the 172.16.x.x range the first Wago will get the address 172.16.1.91, the second Wago will get the address 172.16.1.92 etc. Resulting for our example in the following:

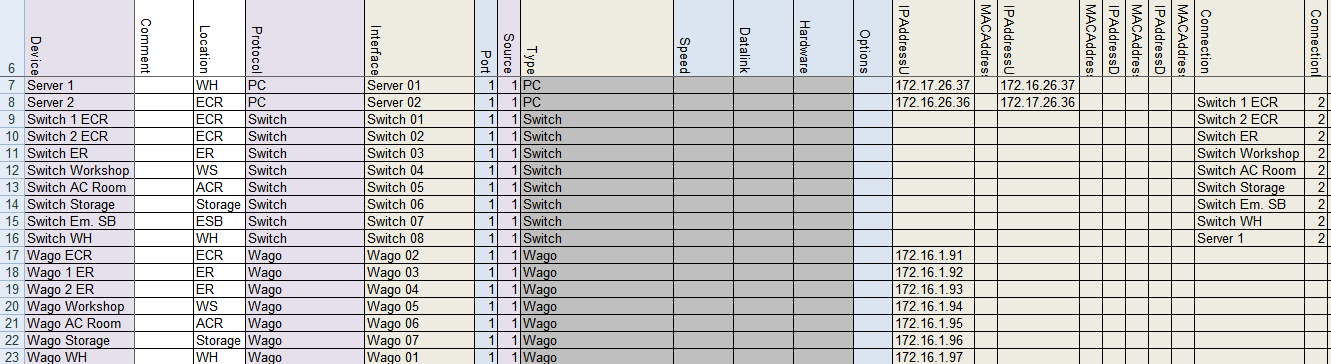


Figure 9‑26: Wago Addresses 1

Wago does need a MAC address but is doesn’t have a Down-link. The MAC address can be found on the Wago PLC itself and will probably lie in the range of 0030DE. Fill in the MAC address in the appropriate row. Also we do need to fill in where they are connected at. For that we again use the single line drawing. As we started earlier at the ECR server we now start again in the ECR and go clockwise to find all the Wago’s. There is one Wago in the ECR (that is why it gets the address 172.16.1.91) and it is connected at the “Switch 2 ECR”. The first free port at the switch is port 3. This results in the following:

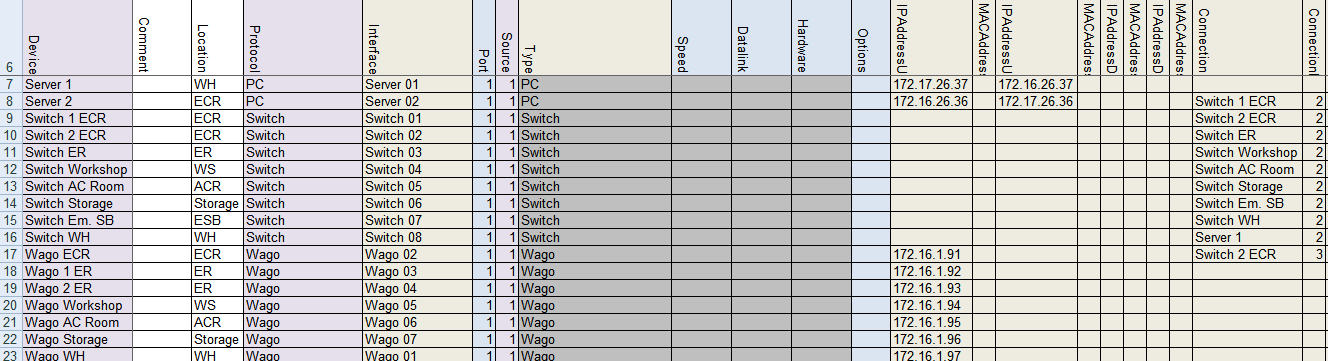


Figure 9‑27: Wago Addresses 2

*: FT NavVision© calculates all the port 1 connections itself. So it is not possible that you find a “1” in the “ConnectionPort” column.*

You can follow this for all the Wago’s. If, like in this example, there are two Wago’s on one switch, than you need to give them separate connection ports. In this case the firs adjacent free ports will be port 3 and port 4.

*: The ports you assign in the devicelist, must be connected exactly the same in the installation. Because FT NavVision© works with multicast, it would be impossible to troubleshoot the system if you mix up the ports.*

The devicelist will be like the next figure after filling in the information (including the MAC addresses) for the Wago’s:

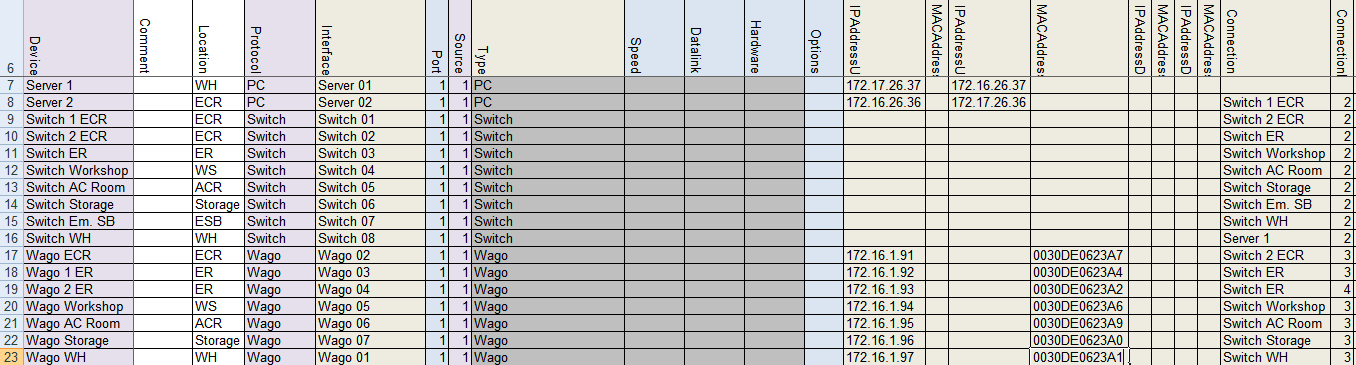


Figure 9‑28: Wago Addresses 3

The Network Serial connections need some special attention. Network Serial Connections can be a variety of interfaces with different approaches in the devicelist. In our example we have Moxa’s as interface. These Moxa’s have an Up-link and you need to specify the MAC address. Also you must specify the connection and the connection port.

According to Table 9‑6 the Moxa falls in the range of x.x.1.4y. so in this case, starting again in the ECR the first Moxa (Serial Network 01) will get the address 172.16.1.41 (as it exists in the 172.16.x.x. range.

*: if you use multiple ports on a Serial Network interface, make sure that you give the same IP address and MAC address to these ports as they are on the same interface.*

The MAC address range of a Moxa will probably be within the 0090E8 range. You’ll find it on the backside of the interface. Put it in the appropriate row.

The first Moxa we find in the ECR with the printer connected to it. This will get the address 172.16.1.41. While only one port is in use, we only need to fill in one row. See the following:



Figure 9‑29: Network Serial addresses 1

Now we can do that for the rest of the Network Serial connections. Be sure that you fill in the same addresses at multiple port connections.

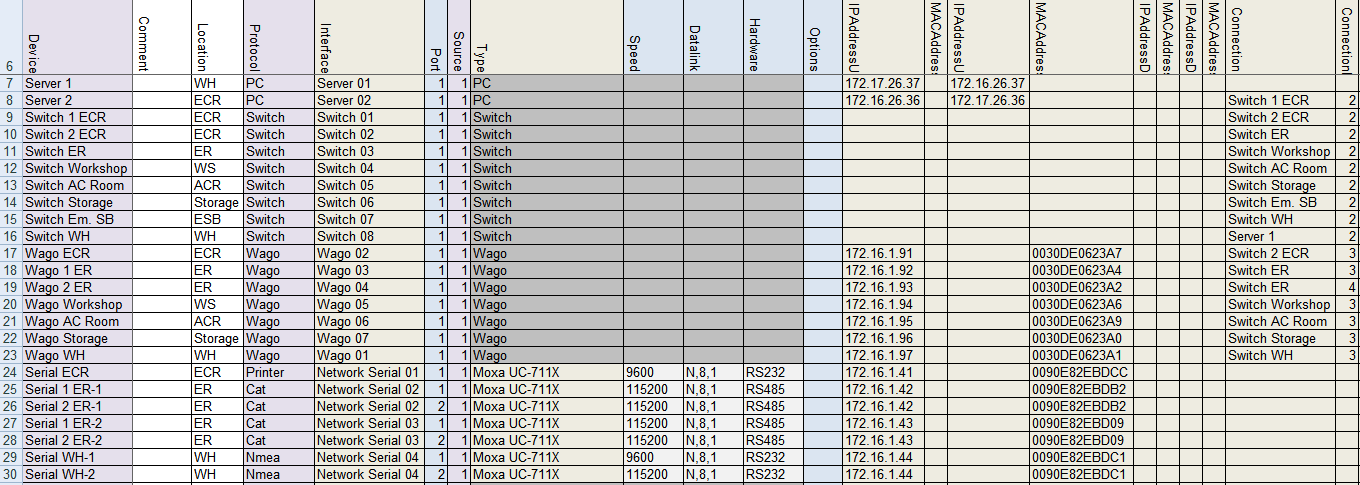


Figure 9‑30: Network Serial addresses 2

Finally assign the Connection and ConnectionPort Wher the ConnectionPort will be the first free port on the switch and you will get the following:

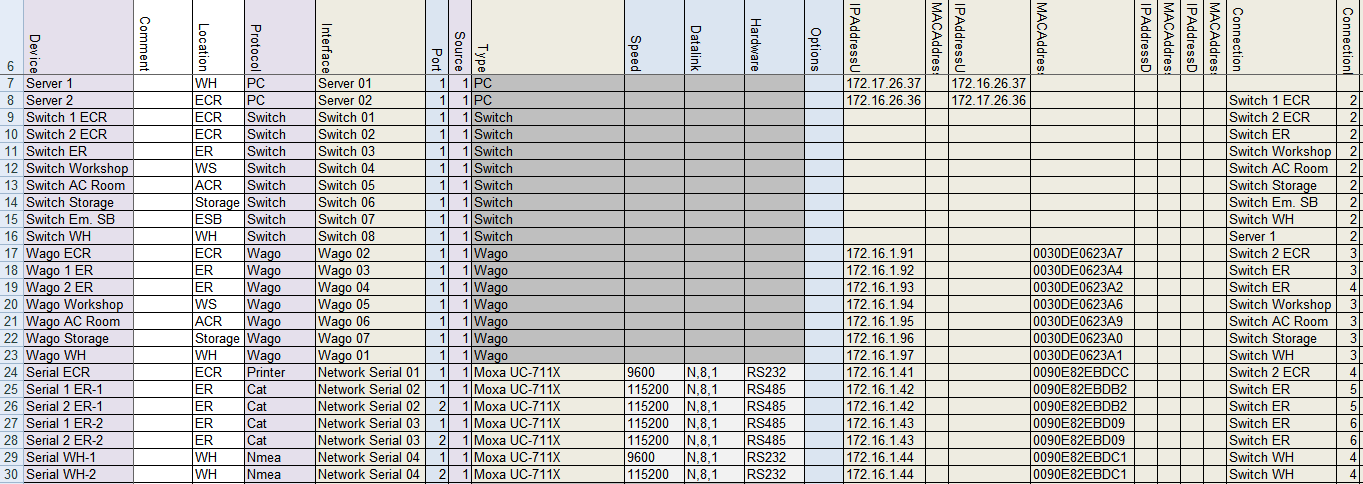


Figure 9‑31: Network Serial addresses 3

*: other Network Serial interfaces can be: ICPdas i7540D, Modbus TCP/IP, Serial TCP/IP and a few others. They mainly work the same way in the devicelist, with the exception that you don’t need a MAC address for TCP/IP.*

Finally we have a few clients in the single line drawing. These are the so called DAP’s (Duty Alarm Panels). As we know from Table 9‑6 the IP range for DAP’s lies within the x.x.1.8y range Where the first one will be x.x.1.81 and so on. While these DAP’s are also in the 172.16.x.x. range, the first DAP will get the address 172.16.1.81.

Also the MAC address is necessary so we put that in the devicelist (DAP’s are in the 00506C range) and also the Connection and ConnectionPort has to be put in. We will finish the devicelist like this:

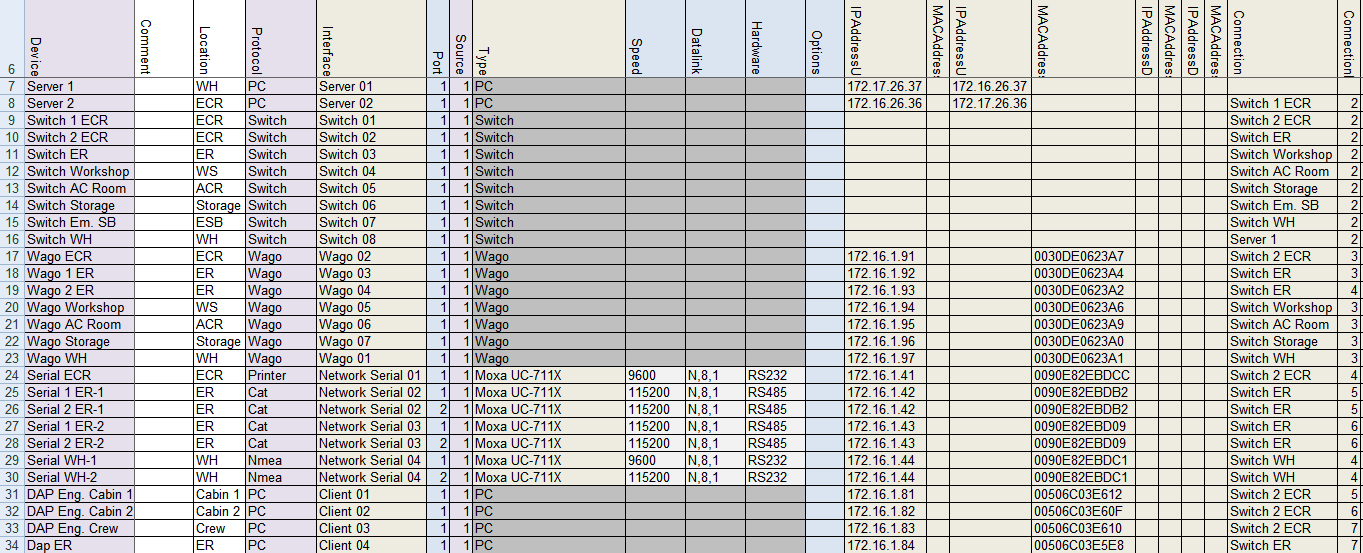


Figure 9‑32: Client addresses

Now the Devicelist is ready you can import it into FT NavVision© to check if it works. We refer to Chapter 11 for further information.

*: We didn’t describe all the possibilities that you can change in the Devicelist, merely the basic ones. Other interfaces or devices can roughly be treated as we described above. If you do find something not working or don’t know how to implement that, please contact Free Technics.*

# Sensorlist

## Introduction

In the devicelist we started to list all the devices with their respective interfaces and ports. The sensorlist (tab sensorlist) will break this up even further. It will go from device through I/O to the field that is attached.

Every single I/O that comes in to the system will have its own line here in the sensorlist. This is done to control all the incoming data as accurate as possible. Every I/O gets its own Field-ID which will be kept in the database of FT NavVision©. Once the program knows that a certain I/O belongs to a specific field, you can add possibilities to that field to control the I/O. Just as example, you can add min/max values, alarm values, unit types, offsets, inhibits and much more. You can even use the specific I/O in PLC programs, Whether internal or external.

With the devices already assigned in the devicelist, you can start out filling the columns in the sensorlist. Be aware that you need all the information on the I/O’s upfront. So for I/O’s on Wago you need to know the sort and type of I/O, but also for protocols such as Modbus, Canbus and other types you will need the right details. Without these details it is almost impossible to make a good sensorlist

## Columns

The sensorlist is also divided in to columns. Some columns are free for your own information, but the colored ones are mainly mandatory. Same as in the devicelist you can find the columns in the sensorlist are labeled in the first row. The fields underneath can be filled with free text or have a drop-down menu where you can choose a tag. These tags are mandatory and the sensorlist won’t except tags that are not in the list for these columns.

The following columns are in the sensorlist:

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| Import Result | Text | For troubleshooting purposes. See Chapter 12.3 |
| ID | Text | A unique ID for the I-O provided by the shipyard or installation company |
| CableLabel | Text | The cable label as labeled in the real installation. Mostly provided by the installation company |
| GroupLabel | Text | Group labels are for dividing I/O into dedicated groups, like Bilge, Fire, Engines etc. |
| Item | Text | The description of the Data Field. Default item text belonging to the Data Field is preferred. The name of the I/O as you want it to appear in the Alarm List. |
| SensorType | Select  (Text) | SensorType defines which subfield or action of the Data Field is set by this value. By default it’s [Standard]. Standard means it’s not defining a subfield or action, but the value of the Data Field itself. (For more options see Table 11‑2 and Table 11‑3). |
| Connection | Select  (NO,NC) | Connection defines the type of connection for digital in- and outputs. Connection is NO by default. If an in- or output is normally closed it’s NC. |
| Device | Select | Identification of the device where the sensor/control or serial device is connected to. This text should be unique for each FT NavVision® device. The text is case sensitive This comes from the devicelist |
| Location | Text | Identification of the substation where the sensor/control is connected to in the FT NavVision® system. Every substation should have a unique text. The text is case sensitive |
| Interface | Select  (text) | Select the type of interface that the data is coming in. For Wago this is divided in the slice’s type-number. For Modbus, Canbus and other protocols it is Serial (Digital/Analog) in or out. |
| Module | Value  (Index) | Module index where the sensor/control data can be found. For CAN bus it is the parameter group number (PGN), for Modbus it is the Modbus mapping and for WAGO it is the slice number. Module 1 for WAGO is the first slice after the 750-626 module. |
| Pin | Value  (Index) | The I/O index on the module for WAGO and the bit offset in the message for serial protocols. (NOTE: The pin index is 1 based) |
| Type | Select | defines the type of module used to read/control the I/O. This is mainly used for WAGO. It can be between 750-400 and 750-612. For Modbus here goes the function code. |
| Min | Value | Minimum instrument value |
| Max | Value | Maximum instrument value |
| DefaultUnit | Select | The default unit used to present this Data Field. (For options see Table 11‑4) |
| Manufacturer | Optional | Manufacturer |
| Supplier | Optional | Supplier |
| Comment | Optional | Comment |
| Revision | Optional | Revision |
| Field | Select  (FTSelect) | The ID of the Field. References to this ID can be found in the file “fieldlist.txt” that is in the root folder of the FT NavVision® software installation after the first time FT NavVision® has been started. |
| Label | Text | The short description of the Data Field when shown in an instrument. Default label text belonging to the Data Field is preferred. The name of the I/O as you want it to appear in an instrument, a value, a button, etc. |
| Rate | Value  (Hz) | Rate describes the number of samples per second of a sensor/control. This is defined by the protocol. Leave empty. |
| Index | Value  (Index) | Index defines when this Data Field Definition [DFD] is valid. The Index column can only be used in combination with a Data Field Definition [DFD] that has the SensorType set to Index and is in the same message as this DFD. Default is empty. |
| DataType | Select  (Unsigned, Signed, Bool, Enum, Float) | DataType is used to define the type of value on serial protocols. For analogue values it’s Float, Signed or Unsigned. For digital values it’s Bool. For enumerations this is Enum. See Enum column. |
| Enum | Value  (Index) | Enum is the index value where the received value should compare to, to switch the Data Field on. If the value is not equal to the Enum index the Data Field is switched off. |
| Count | Value  (Count) | Count is the number of bits starting from the pin index. For a digital value it’s typically 1 with a pin index between 1 and 16 and for analog values it’s for example for Mod bus typically 16 with pin index 1. |
| Multiplier | Value | Multiplier defines the factor between the sensor/control value and the real value.  For inputs/read:  *value = sensor value \* multiplier + offset*  For outputs/write:  *sensor value = (value – offset) / multiplier* |
| Offset | Value | Offset defines the offset between the sensor/control value and the real value. See Multiplier column. |
| Unit | Select | The Unit in which the sensor/control value is received or send. (See Table 11‑4) |
| GroupLocal | Text | Local language text (see 11.3.29) and: chapter 11.1.14 Software installation and commissioning manual 1.9 |
| ItemLocal | Text | Local language text (see 11.3.29) and: chapter 11.1.14 Software installation and commissioning manual 1.9 |
| LabelLocal | Text | Local language text (see 11.3.29) and: chapter 11.1.14 Software installation and commissioning manual 1.9 |
| Filter | Value  (Seconds) | The filter used in the instruments for this Data Field to eliminate short spikes in measurements. Default is 1 second. Maximum is 10 seconds |
| SetpointMin | Optional | SetpointMin |
| SetpointMax | Optional | SetpointMin |
| SetpointMinDelay | Optional | SetpointMinDelay |
| SetpointMaxDelay | Optional | SetpointMaxDelay |
| SwitchTime | Optional | SwitchTime |
| PulseTime | Optional | PulseTime |
| ExternalRight | Optional | Read, Write or Read/Write rights |
| Decimals | Optional | Set number of decimals in values. (See also chapter 11.2.2.3 Software installation and commissioning manual 1.9) |
| Log | “Y” or “N” | Defines whether a field will be logged for remote monitoring (see Remote monitoring manual v1.0.2) |
| AlarmSMS | Obsolete | Set if an SMS will be sent at alarm |
| AlarmWAV | Filename | The filename of the sound that will be played over the sound card when this Data Field is in alarm. Default is “alarm.wav”. Files can be found in the “sound” sub folder of the FT NavVision® software installation |
| WarningLow | Value  (in “Unit”) | The threshold for the low alarm. Empty is off |
| WarningHigh | Value  (in “Unit”) | The threshold for the high alarm. Empty is off |
| WarningDelay | Value  (Seconds) | The delay for the low and high alarms |
| WarningGroup | Select | The ID of the alarm group that the low and high alarms are assigned to. References to this ID can be found in the file “fieldlist.txt” |
| WarningAction | Text | The action an operator should take when a low or high alarm occurs. |
| CriticalLow | Value  (in “Unit”) | The threshold for the too low alarm. Empty is off |
| CriticalHigh | Value  (in “Unit”) | The threshold for the too high alarm. Empty is off |
| CriticalDelay | Value  (Seconds) | The delay for the too low and too high alarms |
| CriticalGroup | Select | The ID of the alarm group that the too low and too high alarms are assigned to. References to this ID can be found in the file “fieldlist.txt” |
| CriticalAction | Text | The action an operator should take when a too low or too high alarm occurs. |
| InhibitAll | Value  (“” or “Y”) | Inhibit all alarms for a specific field. This will show in the alarmlist. Empty is off. |
| InhibitLevels | Value  (“” or “Y”) | Inhibit all Level alarms for a specific field. This will show in the alarmlist. Empty is off. |
| InhibitTimeout | Value  (“” or “Y”) | Inhibit all Timeout alarms for a specific field. This will show in the alarmlist. Empty is off. |
| InhibitNotReady | Value  (“” or “Y”) | Inhibit all NotReady alarms for a specific field. This will show in the alarmlist. Empty is off. |
| InhibitDefect | Value  (“” or “Y”) | Inhibit all Defect alarms for a specific field. This will show in the alarmlist. Empty is off. |
| InhibitField1 | Select  (FTSelect) | Field That this I/O should be inhibited or not inhibited to. Se definition “Field”. |
| InhibitType1 | Value  (Higher, Lower) | Inhibits the field depending on if the type is Higher or Lower. |
| InhibitValue1 | Value | Value when to inhibit. ( i.e. Inhibit when RPM is Lower than 500). So choose 500 here. |
| InhibitLogic | Value  (AND, OR) | Logic for second inhibit field. Choose between different possibilities. |
| InhibitField2 | Select  (FTSelect) | Field That this I/O should be inhibited or not inhibited to. Se definition “Field”. |
| InhibitType2 | Value  (Higher, Lower) | Inhibits the field depending on if the type is Higher or Lower. |
| InhibitValue2 | Value | Value when to inhibit. ( i.e. Inhibit when RPM is Lower than 500). So choose 500 here. |
| InhibitBeforeDelay | Value  (seconds) | Delay before inhibit kicks in |
| InhibitAfterDelay | Value  (seconds) | Delay after inhibit stops |
| Weight | Optional | Weight |
| CableLength | Optional | CableLength |
| Connector | Optional | Connector |
| Supply | Optional | Supply |
| Consumption | Optional | Consumption |

Table 10‑1: Sensorlist columns

Sensor types can be used for in- and outputs (read/write). The interpretation of the read values and written values differs a bit, so they are described separately

|  |  |  |
| --- | --- | --- |
| SensorType (Mode: Read) | | |
| Option | Sensor | Description |
| Standard | Value | Sensor value represents the state of the Data Field itself (Default) |
| Set | On | Request to turn on |
| Off | No action |
| Reset | On | Request to turn off |
| Off | No action |
| Pending | On | Processing a request. |
| Off | No action |
| Auto | On | Switched by an automatic control sequence |
| Off | Controlled by an operator |
| Manual | On | Controlled by an operator |
| Off | Switched by an automatic control sequence |
| Low Speed | On | Running at low speed |
| Off | Off, when not in “High Speed”. Otherwise no action |
| High Speed | On | Running at high speed |
| Off | Off, when not in “Low Speed”. Otherwise no action |
| Closed | On | Switched off |
| Off | Processing a request, when not “Open” |
| Open | On | Switched on |
| Off | Processing a request, when not “Closed” |
| Ready | On | Ready for use |
| Off | Not ready for use |
| Remote | On | Remote control. Controlled by AMCS |
| Off | Local control. Not controlled by AMCS |
| Ack | On | Acknowledgement of alarm on the assigned field |
| Off | No action |
| Request | On | Request to turn on |
| Off | Request to turn off |
| Push | On | Request to turn on, when off.  Request to turn off, when on. |
| Off | No action |
| Too Low | On | Value is too low |
| Off | Value is not too low |
| Low | On | Value is low |
| Off | Value is not low |
| High | On | Value is high |
| Off | Value is not high |
| Too High | On | Value is too high |
| Off | Value is not too high |
| Failure | On | Defect |
| Off | Not defect |
| Precision | On | High precision frequency counter in 0.01 Hz accuracy up to 10kHz |
| Off | Low precision frequency counter in 1 Hz accuracy up to 100kHz |
| Counter | Value | The changes in this counter value will be added to the field |
| Sign | On | The value read by “Standard” is negative |
| Off | The value read by “Standard” is positive |
| Index | Value | Value is the index of a serial message. See “Index” description |
| Pulse | On | Field’s value is counted 1 up |
| Off | No action |
| Pulse 1/2 | Value | Used in combination with “Pulse 2/2” to detect movement with two proximity switches. |
| Pulse 2/2 | Value |
| Pulse 1/3 | Value | Used in combination with “Pulse 2/3” and “Pulse 3/3” to detect movement with three proximity switches. |
| Pulse 2/3 | Value |
| Pulse 3/3 | Value |

Table 10‑2: Sensor Type mode Read

|  |  |  |
| --- | --- | --- |
| SensorType (Mode: Write) | | |
| Option | Sensor | Description |
| Standard | Value | Requested state of the Data Field itself (Default) |
| Set | On | Request to turn on |
| Off | No action |
| Reset | On | Request to turn off |
| Off | No action |
| Pending | On | Processing a request. |
| Off | No action |
| Auto | On | Request to turn automatic control sequence on |
| Off | Request to turn automatic control sequence off |
| Low Speed | On | Request to run at low speed |
| Off | Request to turn off, when not in “High Speed”. Otherwise no action |
| High Speed | On | Request to run at high speed |
| Off | Request to turn off, when not in “Low Speed”. Otherwise no action |
| Impulse | On | Request to turn on, when off.  Request to turn off, when on. |
| Off | No action |
| Status | Value | Output value represents the state of the field/ device itself  (No control) |
| Ready | On | Ready for use |
| Off | Not ready for use |
| Remote | On | Remote control. Controlled by AMCS |
| Off | Local control. Not controlled by AMCS |
| Too Low | On | Value is too low |
| Off | Value is not too low |
| Low | On | Value is low |
| Off | Value is not low |
| High | On | Value is high |
| Off | Value is not high |
| Too High | On | Value is too high |
| Off | Value is not too high |
| Failure | On | Defect |
| Off | Not defect |

Table 10‑3: Sensor Type mode Write

|  |  |  |
| --- | --- | --- |
| Unit Type | Select | Description |
| Alarm | Alm | Alarm |
| Ampere Hour | Ah | Ampere hour |
| Angle | ° | Angle |
| Angular Acceleration | °/s^2 | Degrees per square second |
| Angular Speed | °/sec | Degrees per second |
| °/min | Degrees per minute |
| Content | % | Percentage |
| G | Gallon [US] |
| M3 | Cubic meter |
| L | Liter |
| Guk | Gallon [UK] |
| Consumption per Distance | l/nm | Liter per nautical mile |
| l/km | Liter per kilometer |
| G/nm | Gallon [US] per nautical mile |
| l/m | Liter per meter |
| Consumption per Time | G/H | G/H |
| G/M | G/M |
| Guk/M | Guk/M |
| Guk/H | Guk/H |
| l/m | l/m |
| G/S | G/S |
| Guk/S | Guk/S |
| l/h | l/h |
| L/S | L/S |
| Counter | x | Count |
| Course | ° | Course |
| Current | mA | MilliAmpere |
| kA | Kilo Ampere |
| A | Ampere |
| Dampening | D | Dampening |
| DistanceContent | nm/G | nm/G |
| nm/l | nm/l |
| km/l | km/l |
| m/l | m/l |
| Force | Pdl | Poundal |
| Lbf | Lbf |
| N | Newton |
| Kgf | kg |
| ForceLength | Kgm | Kgm |
| Lbf-ft | Lbf-ft |
| kips | Kips |
| Nm | Nm |
| Frequency | Hz | Hertz |
| FuelEconomyGaseous | nm/kg | nm/kg |
| m/g | m/g |
| km/kg | Km/Kg |
| FuelEconomyPower | kWh/l | kWh/L |
| kWh/Guk | kWh/Guk |
| kWh/G | kWh/G |
| Length | km | Km |
| mi | mi |
| cm | cm |
| nm | NM |
| ft | Feet |
| fm | Fathom |
| mm | mm |
| m | M |
| in | Inch |
| Luminance | cd m-2 | cd m-2 |
| Magnetic | ° | Magnetic |
| MassSpeed | g/s | g/s |
| t/s | t/s |
| kg/h | Kg/H |
| Name |  |  |
| Number |  |  |
| okta | Okta |
| Percentage | % | Percentage |
| Position | ° | Degrees |
| Pressure | psi | Psi |
| Pa | Pascal |
| kPa | kPa |
| mbar | mBar |
| hPa | hPa |
| Hg | Hg |
| bar | Bar |
| Resistance | ohm | Ohm |
| mOhm | MilliOhm |
| kOhm | KiloOhm |
| RPM | rpm | RPM |
| RPMAccelaration | rpm/s | RPM/s |
| Speed | km/h | Km/H |
| m/min | M/Min |
| m/s | M/S |
| ft/min | Feet/Min |
| kn | Knots |
| B | Beaufort |
| mph | Miles per hour |
| SpeedAcceleration | g | g-force |
| m/s2 | M/S2 |
| Status | Open | Open |
| On | OnOff |
| Switch | Take Over | Take Over |
| S | Switch Off |
| Alarm Group | Alarm Group |
| General Alarm | General Alarm |
| Alarm Deadman Group | Alarm Deadman Group |
| P | Push |
| S | Switch |
| PS | Popup Switch |
| Temperature | K | Kelvin |
| °C | Celsius |
| °F | °F |
| Time | Mn | Month |
| H | Hour |
| D | Day |
| DTL | Date & Time Left |
| D | Date |
| ms | mSec |
| us | uSec |
| Wk | Week |
| M | Min |
| T | Time |
| DT | Date & Time |
| S | Sec |
| Yr | Year |
| True | ° | True |
| Voltage | mV | MilliVolt |
| kV | KiloVolt |
| V | Volt |
| VoltAmpere | VA | VA |
| kVA | kVA |
| VoltAmpereHour | kVAh | kVAh |
| VAh | VAh |
| MVAh | MVAh |
| Watt | MW | MegaWatt |
| W | Watt |
| kW | kW |
| WattHour | Wh | WattHour |
| MWh | MegaWattHour |
| kWh | kWh |
| Weight | lbs | Lbs |
| kg | kg |
| g | Gram |
| t | Ton |

Table 10‑4: Unit Type

## Implementation in the sensorlist

### Introduction

As the sensorlist is way bigger and more complex than the devicelist, we will not fill in all the columns and fields. We will give some excerpts from what you can expect at the different devices and the different columns. On the basis of the single line drawing we used for the explanation of the devicelist, we will give as many examples as possible. After this explanation, you should be capable to work out the rest of the sensorlist.

*: Once you have imported the sensorlist into FT NavVision©, most of the fields will be automatically added. This will be done by FT NavVision© on standard basis. This will not always be right, so you need to check that. We will come back at that in a separate Chapter*

### Import Result

The import result is a checklist. When you have imported the sensorlist, FT NavVision© will generate a few files at which we will come back later. One of these files is the sensorlist\_generated. In this file you will see in the first column the import result. For more information we refer you to Chapter 11.5.

### ID, CableLabel, GroupLabel

These columns are optional. They are not needed for the proper functioning of the program. However it could come in handy when you fill up some of those fields.

The ID column you can use for your own reference. Maybe you use some kind of numbering that is different from the one you get from the shipyard.

Many installation companies use cable labels (numbers) for the connections of the wires at the terminals and/or at the sensor/control. If you fill in these Cable labels in this column, you will have a reference in the sensorlist which is searchable. You also get a reference in FT NavVision© where the Cablelabel is shown in the Wago screen.

In the GroupLabel column you can separate different (alarm)groups and their I/O. This makes it quite easy to search specific I/O or just select a whole group that you need to adjust.

Next figure will show a small example:

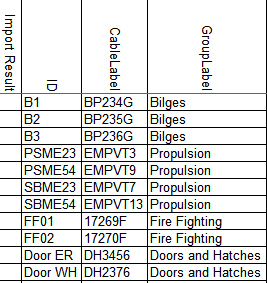


Figure 10‑1: ID, CableLabel, GroupLabel Example

### Item

The Item is somewhat different and needs some attention. In consultation with the installer or even with the shipyard, you need to come up with a descriptive name for each field (I/O, sensor, control). As this is the name that comes up in the logbook and the alarmlist, you need to be clear about what it is.

Sometimes people come up with texts like “Preferential Trip & Em. Stop System Power Failure”. As you can see it is quite long and also very confusing. It can mean a lot of things. Maybe this one would be easier to understand if you called it “PMS Power Failure”. It is certainly more descriptive and short and concise.

In other cases, the crew can be very familiar with certain names. The example “N.16 Fr 20-21 Bilge Level High Alarm” may seem confusing, but the crew knows exactly what it means cause they have been working with this name for years.

Remember however that the text is free to choose, but it will appear in alarm lists and the logbook. So keep it as simple as possible.

#### Conjunction with SensorType

You also need to understand the conjunction with the “Item” column and the “SensorType” column. As explained in Chapter 10.3.5 SensorType defines which subfield or action of the Data Field is set by the value in that column. So if it is not “standard” you better check the “Item” text again.

For example: A sensortype can be “High Alarm” or “Running” or even just “Alarm”. This means that you trigger an extra action with the sensortype field. Now let’s say that you have the sensortype defined as Alarm. When you put “Bilge ER Alarm” as text in the “Item” field you get it double. With an alarm you now will get “Ext: Bilge ER Alarm Alarm” in your alarm screen. Easy to understand that if you use the sensortype “Alarm” you leave the word Alarm out of the Item-text. This is valid for all the conjunctions between these two columns.

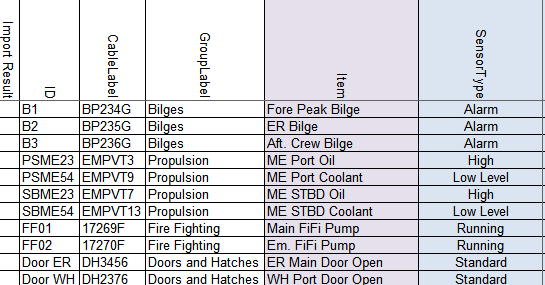


Figure 10‑2: Item example

### SensorType

SensorType defines which subfield or action of the Data Field is set by this value. By default it’s “Standard”. Standard means it’s not defining a subfield or action, but the value of the Data Field itself. (For more options see Table 10‑2 and Table 10‑3).

With “standard” as option in the sensortype column FT NavVision© will only act upon the field itself. So if the field is an alarmfield FT NavVision© will give an alarm when that field gets triggered. This goes for all the different type of fields. So if for example it is a Pressure field (analog value) FT NavVision© will show the pressure value. If you don’t fill in anything in the sensortype column, it will automatically be “Standard”.

If no extra action is necessary on a field you probably won’t use the sensortype column. This comes in play when you want something extra. An analog field that needs a “high” alarm. An output that needs a “Set” request and so on. Before we elaborate on this we need to explain something about the “Fields” within FT NavVision©.

#### Fields

FT NavVision© works with a database with all kind of ID’s in it. These ID’s are represented by fields that are divided into sup-parts. Every action in FT NavVision© revolves around this database of field-id’s. You can use one field over and over again cause the main value is set in the database.

Once you connect a sensor or control to a field you can do almost everything you like. For example if you want to control a pump with a hardwired button, you can connect that pump in FT NavVision© to let’s say the field “Pump1”. Through a Wago PLC you now get to control that pump. On a Wago Digital Input, you connect the field “Pump1” and you hardwire a button to the same Slice. Now if you push the button the Wago input will get high. If you put the same field “Pump1” to an output on the Wago. This output will get active as soon as the input gets active. While this is an output, you can hardwire it to the actual Pump1. So than when you push the button the pump will start to run.

These fields you can find in the “fieldlist.txt”. Once that FT NavVision© is started for the first time, you will find it in the root folder. You can open and control this .txt-file best with Excel. For people not familiar with Excel there is a small explanation in Chapter 8.2.

As there is a lot of intelligence in the fields already it is good to understand the interaction between the field and the sensortype. You can mess things up when you use this wrong.

#### Back to SensorType

So, as mentioned earlier, there is a conjunction between the “Item” and the “SensorType” and now also between the “Field” and the “SensorType”. We use the same example as in Chapter 10.3.4.1 to show how it all fits together.

As we mentioned in that chapter, you need to pay attention to the name you use in the Item-column so you do not get confusing or double values. Same goes for the fields and the sensortype. If you choose a field that is already an alarm-field this means that, when the value gets high, the field will give an alarm. So it is not necessary to put an extra alarm in the sensortype column. This is not only double but also can confuse the system or the user. On the other hand, if you use a field that holds Level information, you might want to trigger an alarm when you get to a certain level. This is possible by putting “High Level” in the SensorType column. You see there is quite some interaction between those different columns.

You need to practice a lot with the sensorlist to learn how to work with it. For now we will give an example on how it is not supposed to be concerning “Item” “SensorType” and “Field”.

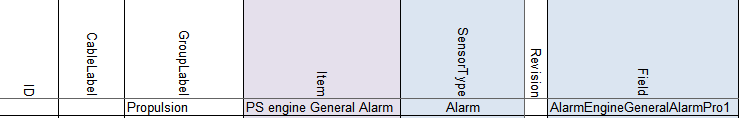


Figure 10‑3: Double fault

As you can see we have an alarmfield in the field column, a sensortype that triggers an alarm and the name in the Item column that will make it double. Easiest in this case is: keep the alarmfield in the Field column, put Sensortype to “Standard” and take “Alarm” out of the Item column name.

### Connection

Connection defines the type of connection for digital in- and outputs. Connection is NO by default. If an in- or output is normally closed it’s NC. If you have problems with switches that go the wrong way around or there is an alarm where the sensor itself is not in alarm, this is the first place to look.

### Device

Identification of the device where the sensor/control or serial device is connected to. This text should be unique for each FT NavVision® device. The text is case sensitive.

This device is already been set in the devicelist. See chapter 9.5.2 to see how you’ve done that. Now all the I/O that you put into the sensorlist must be connected to the right device, so FT NavVision© knows where to look for it and how to process it.

When you click on a field you can see there is a drop0down menu. In the menu you will find all the previous assigned devices. All you have to do now is choose the right device.

As we look at the single line drawing and we take the example we had earlier we can tell that the Fore Peak Bilge is connected to the Wago AC Room and the ER Bilge is connected to the Wago 2 ER.

The Port Engine is connected to the second port at the second SerialLan in the ER so you choose Serial 2 ER-2 as device. It will look a bit like the following figure:

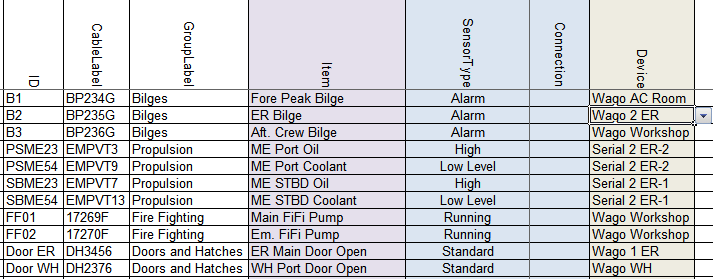


Figure 10‑4: Sensorlist device column

Of course, while you probably will start filling all the I/O’s from one device at the time, you will get a long row with only Wago WH and then for example Wago Workshop. You will see that once you start working with it.

In the example we only have serial and Wago connections, but it can be anything that you filled in as a device. It is probably best to start to fill the list with the Wago devices as these are mostly already assigned. Later on you take the serial connections with for example Modbus or CANbus on it.

### Location

For location you can use the same field as in the devicelist. It is optional, but also usable for sorting the list and/or localizing sensors or I/O’s.

### Interface

Here you define what kind of interface is used to connect the sensor/control to FT NavVision©. For Wago this is divided in the slice’s type-number. For Modbus, Canbus and other protocols it is Serial (Digital/Analog) in or out.

If you have the Wago drawings available, it is easy to choose the right module for that. If you have trouble finding it, you can always fall back to the documentation of Wago. For the protocols you just need to look if it is a digital or analog value and if it is an input or an output. More on these serial interfaces we discuss later.

To give you an idea, we go back to our example. The bilges in the example will be most likely digital inputs. As Wago works standard with 24V it will be a Dig in (24V) you have to choose there. This goes also for the fire pumps and the doors. Probably normal switches so an input of 24V for High or Low (On or Off).

The engine however is somewhat different. As we can see in the SensorType field it is just a digital input where On is High or On is Low Level. However, this is the SensorType Field. This field will give FT NavVision© a reason to calculate an alarm on an analog value. So don’t be misled. This will be an analog field coming in (Oil is a pressure field and Low Level is a level field). So you will have to connect it to an analog interface module on the Wago. This can be 4-20mA, 0-10V or a lot of other sorts. Let’s say the oil pressure field is a 4-20mA signal and the level field is a 0-10V signal. We will come to the following:

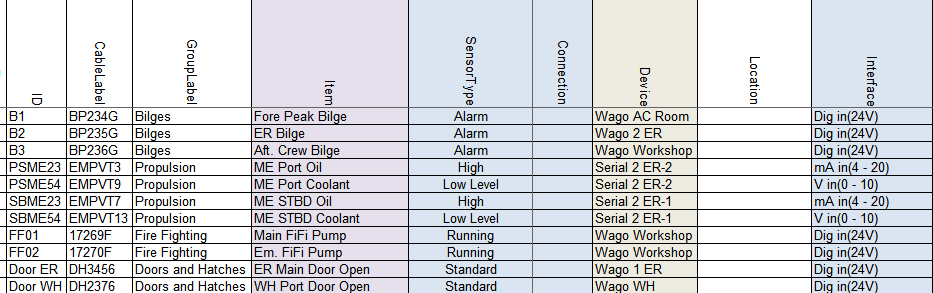


Figure 10‑5: Sensorlist Interface column

### Module

For Wago you start counting the slices after the 750-626 module. Starting with 1 and so on. If you do not filter the sensorlist, than it will be hard to look if the numbers are alright. But as we will explain that in a later stadium we now just have to watch carefully. As example we show you the next figure:

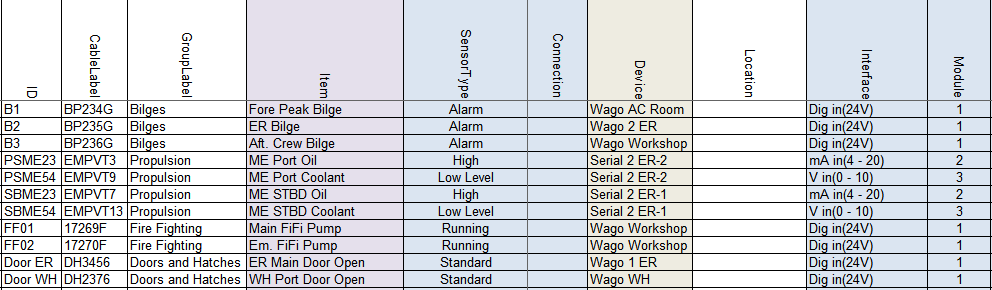


Figure 10‑6: Sensorlist Module column 1

This may look a bit odd, but realize that we put the Dig in (24V) on a module with 8 contacts (Pin). So the first 8 DI you find are on the first module. Same goes for the mA in (4-20). These modules have 4 contacts. It will become more clear in the next paragraph.

For CANbus in this column you put the PGN or Parameter Group Number as index for the I/O. With Modbus you take the Modbus mapping as starting-point. The register of the Modbus mapping you put here. See following example:



Figure 10‑7: Sensorlist Module column 2

### Pin

The I/O index on the module for WAGO and the bit offset in the message for serial protocols. (NOTE: The pin index is 1 based).

#### Wago

So if you look at a Wago slice you will see openings for the wires to be attached. It needs some attention because Wago has a different numbering than FT NavVision© and this can be confusing. First let’s look at the numbering Wago uses:

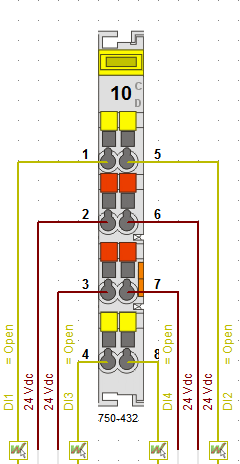


Figure 10‑8: Wago Numbering 1

As you can see Wago numbers the pins vertically so left side 1-4 and right side 5-8.

FT NavVision© has to number different because of program issues. We number the Wago horizontally. So 1=1, 5=2 and so on. You have to keep that in mind to work properly with the sensorlist. The FT NavVision© numbering will look as follows:

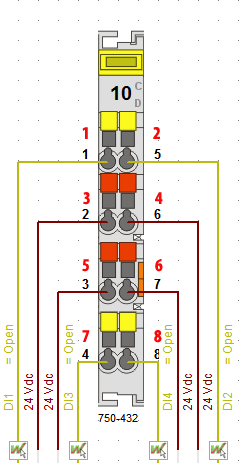


Figure 10‑9: Wago Numbering 2

So when you number it this way in the sensorlist, it will mainly look like the following figure:

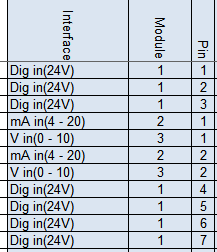


Figure 10‑10: Pin column 1

Or, when you allready sorted the sensorlist, it will make it even clearer. See the following figure:

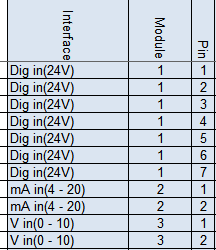


Figure 10‑11: pin column 2

Of course I can’t show you the example from where we started off. While all the connections where on different Wago’s there, we should than have divided all the modules over the different Wago stations. Therefore, before you begin numbering the modules and pins, you need to have all the Wago connections in the sensorlist. Then you can filter the sensorlist first (as explained in Chapter 10.4) and then do the modules and pins.

#### Serial Protocols

For Serial protocols the pin number defines the bit-offset. So if you need to connect to a serial protocol at bit level, this column is where you assign this. Note that the “pin index” is 1 based. So if you need bit 3 for PGN 65280 you have to put 4 in the column. See next figure:



Figure 10‑12: Pin column 3

### Type

#### Wago

For Wago you fill in here the module number. You can find the module number on the Wago slice itself, on the drawings or look it up in the Wago documentation. See following figure:

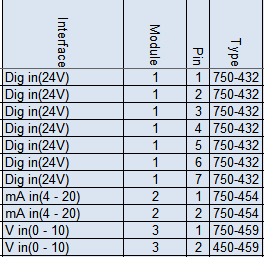


Figure 10‑13: Type column 1

#### Serial Protocols

For CANbus we do not use this column. For Modbus we use this column to define the function code of the Modbus register. So for example if you read actual values in Modbus, this will be Modbus function 04. Type a 4 in the “Type” column. See following figure:



Figure 10‑14: Type column 2

### Min-Max

The columns Min and Max show the range of the data field. This will come back in instruments and value-bars. If you choose them wrong then you get values that go beyond the range of an instrument. Once you see this, you know that you need to change the values. If you get the right data from the shipyard you can fill it in in these fields. For digital data it is Min=0 and Max=1. It is not necessary to fill in the Min- Max-values for digital values. FT NavVision© will do this for you. You can also change these values at a later time.

### DefaultUnit

The defaultUnit is used to set the unit to present this Data Field in. This can also be changed in the instrument or mimic itself, but for big amounts of data it is easier to use the sensorlist. If you do not choose anything FT NavVision© will fill it in for you. For options see the next figure:

|  |  |  |  |
| --- | --- | --- | --- |
| Alarm | High Alarm | Low Alarm | Ampere-Hour |
| Radians | Degrees | Grads | °/sec² |
| rad/sec | °/sec | °/min | Normal |
| Normal | Mirror | Liter | Gallon |
| GallonUK | Cubic Meter | Percentage | L/km |
| G/Nm | L/min | L/Nm | G/S |
| l/h | G/H | Guk/H | G/min |
| L/S | Guk/min | l/m | Guk/S |
| Count | Degrees | Grads | Radians |
| Kilo Ampere | MilliAmpere | Ampere | Dampening |
| kg/m³ | kg/L | lb/gal | nm/G |
| nm/l | m/l | km/l | Poundal |
| Newton | Lbf | Kgf | Kips |
| Newton Meter | Kgm | Lbf-ft | Hertz |
| m/g | nm/kg | Km/Kg | kWh/L |
| kWh/Guk | kWh/G | Fathom | Nautical Mile |
| Feet | mi | cm | Km |
| mm | Inch | M | cd/m² |
| Kg/H | g/s | t/s | Name |
| Okta | Mask | Number | Percentage |
| Degrees | Bar | mBar | kPa |
| Hg | hPa | MPa | Psi |
| Pascal | MilliOhm | Ohm | KiloOhm |
| RPM | Hertz | RPM/s | Km/H |
| Knots | M/Min | M/S | Beaufort |
| Miles per hour | Feet/Min | g-force | m/s² |
| OnOff | Open | Alarm Group | General Alarm |
| Switch Off | Alarm Deadman Group | Switch | Take Over |
| Push | Popup Switch | Kelvin | Fahrenheit |
| Celsius | Date | Day | Date & Time |
| Month | Date & Time Left | Time | Sec |
| Week | Hour | Year | Min |
| mSec | uSec | MilliVolt | KiloVolt |
| Volt | Volt Ampere | kVA | Volt Ampere Hour |
| kVAh | MVAh | Watt | MegaWatt |
| KiloWatt | MegaWattHour | WattHour | kWh |
| Ton | kg | Lbs | Gram |

Table 10‑5: (Default) Unit options

For our example it will be the following:

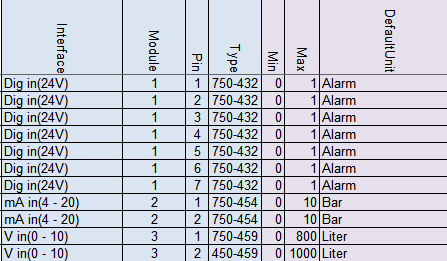


Figure 10‑15: Default Unit column

### Manufacturer

This is an optional field for your own convenience

### Supplier

This is an optional field for your own convenience

### Comment

This is an optional field for your own convenience

### Revision

This is an optional field where you can give a revision number. Easy if you need to see when something has been changed or what has been changed after a certain revision.

### Field

This is one of the most important columns within the sensorlist. This is the place where you assign a dedicated field from the database of FT NavVision©. This field will be inextricably linked to that I/O, sensor or control. All the in- and outputs and all the calculations, as well as connection to instruments and mimics, will be represented with that field. Also the alarmgroup and behavior will be defined by hat you choose here.

You can understand that it is utmost important that this field is chosen properly and a field is only used for one particular sensor/control. These field-column is also the one that will consume most of the time in building the sensorlist.

As mentioned before these fields can be found in the file “fieldlist.txt” in the root of NavVision after the first start of FT NavVision©.

#### How to work with fieldlist.txt

To find all the right fields you first have to open “fieldlist.txt” the right way. You need to know that, to work with the .txt-file you need to open it in Excel. To do so, right-click on the .txt file and choose “open with” and go for Excel (see Figure 10‑16)

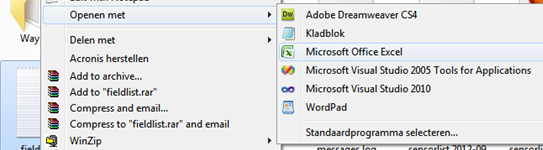


Figure 10‑16: open with Excel

Now the program will open as an Excel sheet, with all the opportunities. There are two things you must do first (this is basic Excel knowledge).

Click in the upper left corner of the sheet (see Figure 10‑17) to select all fields. Put your mouse between row “A” and row “B” (see Figure 10‑18)and doubleclick. The fields now will be all on the right width.

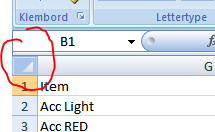


Figure 10‑17: Excel 1

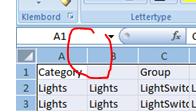


Figure 10‑18: Excel 2

Now select the first row by clicking with you mouse on the number “1” in front of the row. Goto Start>sort and filter and then filter (see Figure 10‑19). Click it

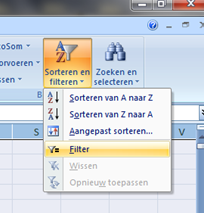


Figure 10‑19: Excel 3

The first row with the index names has now drop down menus and you can choose what to filter. For our example we need Bilges. Goto the index name “Category” click on the dropdown menu, deselect the “select all” checkmark and then select the “bilges” checkmark (see Figure 10‑20). You now have only all the bilges-fields available.

You can narrow it down by going to the index name “Group” and make another selection (see Figure 10‑21). In our case it is AlarmBilge

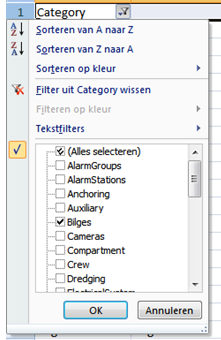


Figure 10‑20 : Excel 4

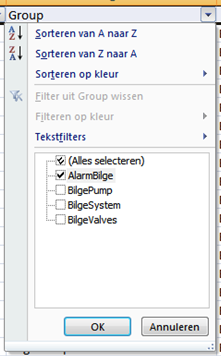


Figure 10‑21: Excel 5

Now we’ve done this we have only the Bilge alarmfields available. You can figure out yourself how you can further narrow it down, or use it for other fields.

#### Back to the Field column

So now we have narrowed it down to the right fields, it is time to give all our I/O a separate field tag. In the adjusted fieldlist.txt we now see all the alarms for bilges available. We need three bilge alarms, so we need three distinctive bilge alarm fields.

In the previous mentioned excel list, goto the column “Field”. As we are just starting, all the fields are still available. So we can choose the first three Bilge Alarm Fields. Select these three fields and copy them (CTRL-C). Go back to your sensorlist and past them into the field column behind the three bilge items. See the following figure.

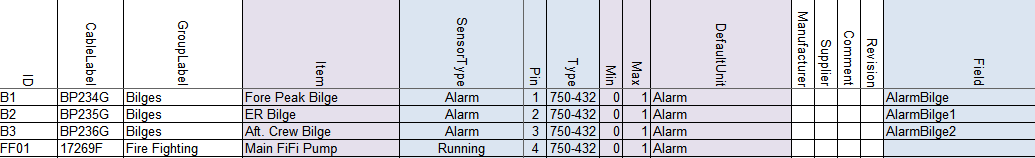


Figure 10‑22: Field column 1

You can follow this for all the other fields and you will get the following:

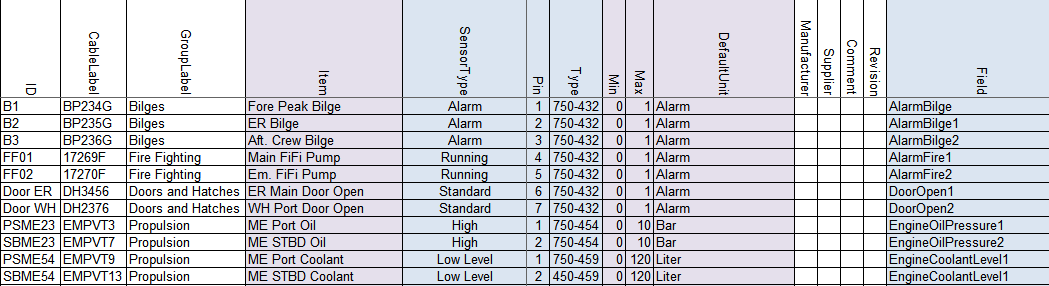


Figure 10‑23: Field column 2

*: with bigger projects it is easy to get mistaken. Easiest way to prevent this is that you color the fields u have used in the fieldlist.txt yellow. That way you will know which ones are used and which are free. Later on we show you that FT NavVision© has a way to trace the faults. See chapter 11.5*

### Label

The Label column exists of the short description of the Data Field when shown in an instrument. Default label text belonging to the Data Field is preferred.

So the easiest way is to copy the “Item” column and just past it into the “Label” column. This way you have a one-on-one connection. Off course this is the text that you find as a label in instruments etc. When the text is too big, it won’t fit in the instrument or just looks sloppy. So if this is the case, just alter the name here to a short description. “Emergency Generator Power Failure” can be changed into “Em. Gen. Power Fail.” And if the default unit is available in an instrument, you can even leave types as “Pressure”, “Voltage”, etc. out of it, cause they will see that it is a “Bar” value or a “V” value. So “Main Engine Lub Oil Pressure” can be set as “ME Oil”

### Rate

Rate describes the number of samples per second of a sensor/control. This is defined by the protocol. Leave empty.

### Index

Index defines when this Data Field Definition [DFD] is valid. The Index column can only be used in combination with a Data Field Definition [DFD] that has the SensorType set to Index and is in the same message as this DFD. Default is empty.

### Datatype

DataType is used to define the type of value on serial protocols. For analogue values it’s Float, Signed or Unsigned. For digital values it’s Bool. For enumerations this is Enum. See Enum column.

### Enum

Enum is the index value where the received value should compare to, to switch the Data Field on. If the value is not equal to the Enum index the Data Field is switched off.

### Count

Count is the number of bits starting from the pin index. For a digital value it’s typically 1 with a pin index between 1 and 16 and for analog values it’s for example for Mod bus typically 16 with pin index 1.

### Multiplier

Multiplier defines the factor between the sensor/control value and the real value.

For inputs/read:

*value = sensor value \* multiplier + offset*

For outputs/write:

*sensor value = (value – offset) / multiplier*

For example: if the temperature is send in from a sensor in whole numbers (210 for 21 degrees) you can put in a multiplier of 0.1. So when the sensor sends 210, it goes through the multiplier and FT NavVision© makes it 210\*0.1=21

### Offset

Offset defines the offset between the sensor/control value and the real value. See Multiplier column.

### Unit

The Unit in which the sensor/control value is received or send. Directly from the sensor control. This field differs from the DefaultUnit by the fact that FT NavVision© has no influence on this one. For options see Table 10‑5.

### Other columns

The rest of the columns in the sensorlist are optional, because FT NavVision© will fill them in for you. These fields will only be used for specific needs. If you want to know what you can do with these columns, it is enough to look in the Sensorlist Table (see Table 10‑1).

## Filter sensorlist

Once you start filling the sensorlist it is good habit that before you fill in the columns module and pin, you filter the sensorlist. This is also common Excel knowledge, but for your convenience we will give a short explanation here.

Let’s say you have filled in a few I/O that you got from a list and you just start to fill in in no particular order. Than it is impossible to address the right module and pin as the list will be extremely long and changes on mistakes will be huge. So before you start with the module and pin columns you will have to filter the sheet.

The columns that you did fill in contain the device-column and the interface-column. With these two you can filter the sheet for a first result.

What you need to filter first is that all the devices are grouped and the interfaces are grouped together. To do this you go to Start>Sort and Filter>Custom sort. You will get a menu like in the folloing figure:

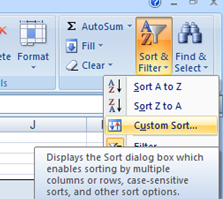


Figure 10‑24: Custom sort

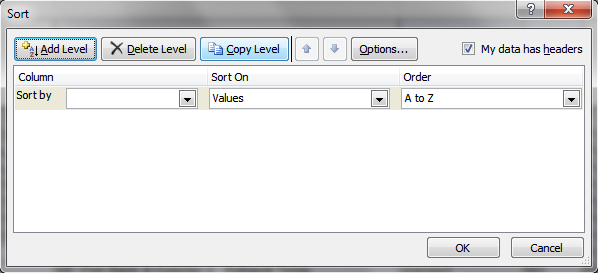


Figure 10‑25: Custom sort window

In this window you can add as many levels as you want to filter out the sheet. We need only two for now, “Device” and “Interface” as you see in the next figure:

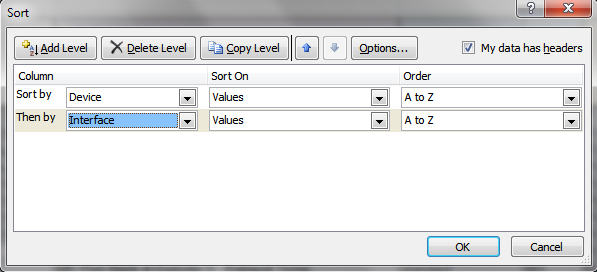


Figure 10‑26: Sorting device and interface

Sorting it this way gives you the devices ordered at the right Wago PLC and you get all the same slices together. This is the first step of filtering that is pretty easy and it gives the following example:

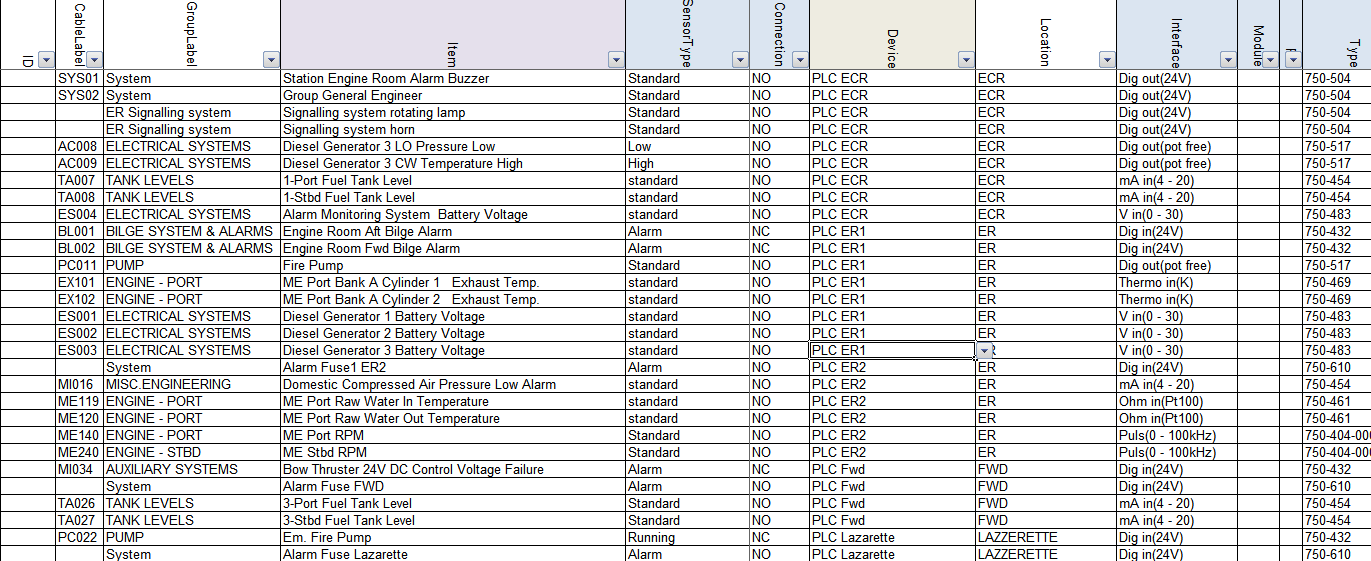


Figure 10‑27: Sorted sensorlist

As you can see we have all the devices put together and within these devices we have all the interfaces put together. Due to the running sequence Wago follows, we need to make some final adjustments by hand. There is a certain sequence that we have to build up the Wago PLC’s in. For more information we refer to Wago. For now we can say that we start the construction of Wago in the following (global) order:

First DI-modules

Than DO-modules

Than AI-modules

Than AO-modules

This is a global distribution, cause it sometimes needs some additional action. For now this is enough to understand.

As you look at Figure 10‑27 you can see in the column “Interface” that it worked out pretty well. The only thing in this example that is not right are the modules at line 28 and 29. This is need to know knowledge. These modules don’t work in that position and has to be places before the 750-454 module at line 25.

To do so select the two lines (28 and 29) and cut them.

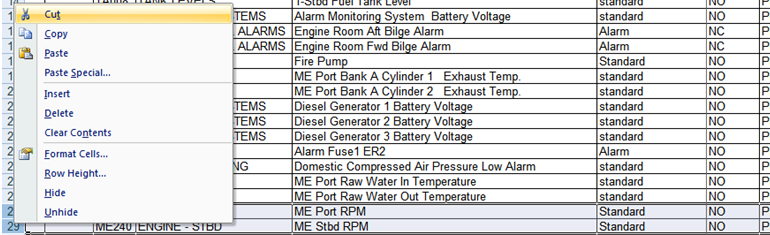


Figure 10‑28: Cut and paste 1

Once you’ve done that you go to the line that you need to insert them and right-click on the number of the row underneath that line. Choose Insert Cut Cells. See following figure”

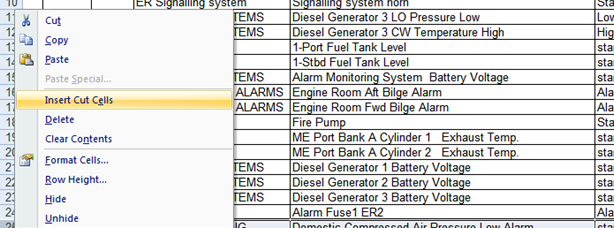


Figure 10‑29: Cut and paste 2

Now you have everything in the right order and you can start numbering the Modules and Pins.

*:You need to have good to excellent knowledge about Wago and Microsoft Excel. We recommend that you get some additional training on this as well.*

## Special issues

There are several special issues that you can put in the sensorlist. Changes you make in FT NavVision© itself will get lost as soon as you import a new sensorlist. To prevent this loss it necessary that you put all the changes you make in FT NavVision© are directly put into the sensorlist. In the hectics of a commissioning it will not always be possible to do that directly, for adjusting the sensorlist at a later time we refer you to Chapter 11.

However we do like to give an example of things you need to change by hand in the sensorlist. For this we assume that you have more than basic knowledge of working with FT NavVision©.

So let’s say that you have a ship with a lot of duty-stations. At some point the crew will ask you to change the names in the alarm/duty mimic, so they can see who is on duty or who they are calling through the FT NavVision© call function.

Given the next example (see Figure 10‑30 and Figure 10‑31) we have changed the names of a few files to match the names as the crew would like to see it. As you will know these names are changed in Fieldsettings>Comment>Crew>CrewAlarms within FT NavVision©. If you do not put this in the sensorlist, each time you import a new sensorlist these names will be changed. This is not desirable, so you need to put these changes into the sensorlist.

If you put this in to the sensorlist, the easiest way to do this is on top of the list. Add some extra rows and start filling the information there. You have to understand that it is FT NavVision© based so the device is FT NavVision© NavVision. SensorType is Standard, Connection is NO and in the “Item” column you fill in the name that you want to show in the alarm mimic of FT NavVision© (see Figure 10‑32).



Figure 10‑30: Duty names



Figure 10‑31: Call names

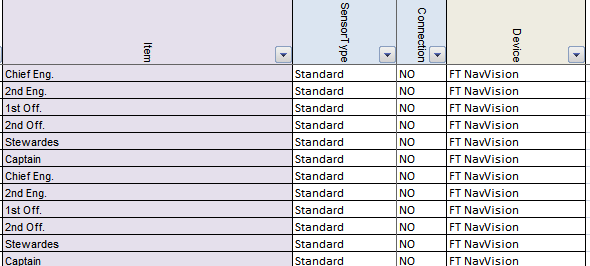


Figure 10‑32: Special issues 1

At the “field” column you assign the right fields (which you will find in the fieldlist.txt see Chapter 10.3.19.10). In the “Label” column you once again fill in the names as you described them in the “Item” column (see ).

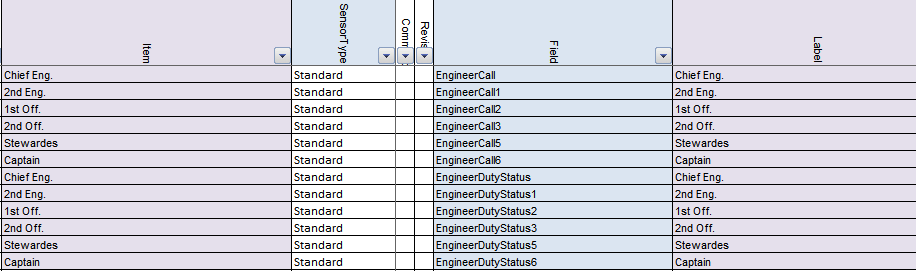


Figure 10‑33: Special issues 2

That is all. FT NavVision© will take care of the rest. Now if you import the sensorlist again, you will keep the names you gave to the Crew Alarms.

# Importing in FT NavVision

## Introduction

Once you are finished with (a part) of the sensorlist, you will at some point need to implement it in FT NavVision©. This is done by importing the sensorlist into FT NavVision©.

In Chapter 8.4 you can see how that is done. Once you have the sensorlist.xls file ready you will put it in the root folder of the NavVision installation. We will go over these steps in the next chapters.

## How to import

Make sure that FT NavVision© is closed and you are in the file explorer. You will have to be in the root folder. Here you will paste the sensorlist.xls file that you just created (see Figure 11‑1).

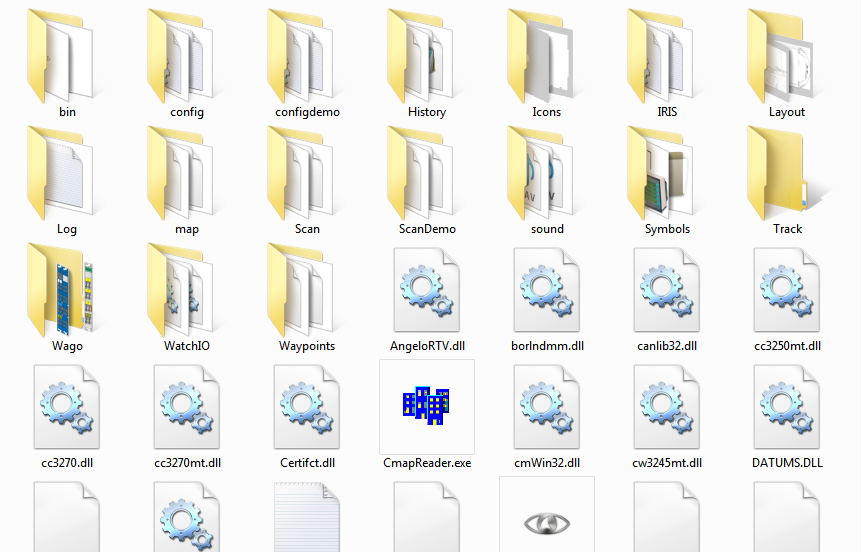


Figure 11‑1: Root folder

Once you have done this, you can start FT NavVision© again. During the startup you will be asked if you want to import the devicelist and/or the sensorlist (this is referring to the 2 tabs in the sensorlist.xls. You answer yes to both the questions (see Figure 11‑2 and Figure 11‑3) and FT NavVision© will continue the startup. At this time the sensorlist will overwrite the existing configuration.

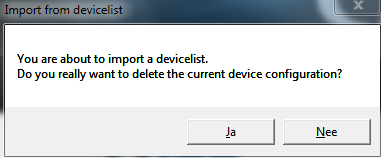


Figure 11‑2: Import devicelist

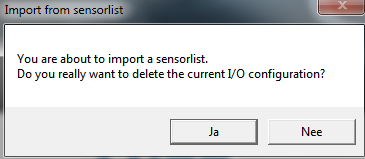


Figure 11‑3: Import sensorlist

Everything you have put into the sensorlist will now be in the configuration of FT NavVision©. This cannot easily be undone, so be very careful if you import. There is a possibility to preserve the old system. Therefor it is necessary that you back up the complete “config” folder. If than anything goes wrong, you can paste the old config folder back.

*: it is always wise to keep a backup of the last working system on for back up sake. Always make a backup of, at least, the “config” folder.*

*Make sure when importing a sensorlist (or even just working on the system) you work on one workstation only (close down all other stations). This way you prevent other workstations from interfering with your setup through the sync-function in FT NavVision©.*

## Check the import

There is not a simple way to check if the import has been successful. The import function has been tested thoroughly by FT NavVision© so the basic import function will work. It is wise to check the import anyway.

As you are probably the one that changed the sensorlist you will now which items has been changed, so you can check these items in FT NavVision©. Also check if the connections are still allright in network>system layout (see ) and if the Wago’s are still in place and connected right, etc. For more information on how to check these items we refer to the

“Installation and commissioning manual”.

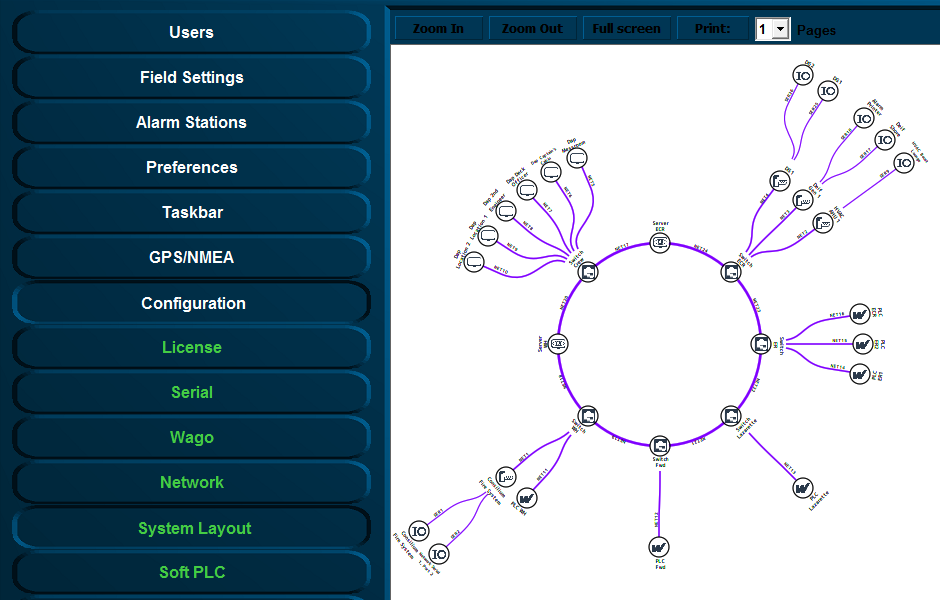


Figure 11‑4: Checking system layout

## Devicelist generated

Once you have made an import the system will make a “devicelist\_generated”. In this file you will find all the changes, faults, etc. that the system found. These are changes that are the differences between your devicelist import and the existing configuration. Also if you have made a mistake in the devicelist, it will be noted here so you can check whether you have to change something. The devicelist\_generated will look like the following figure:

*: Open the devicelist\_generated (or the other generated files which are all HTML-files) with right-click>open with> excel program.*

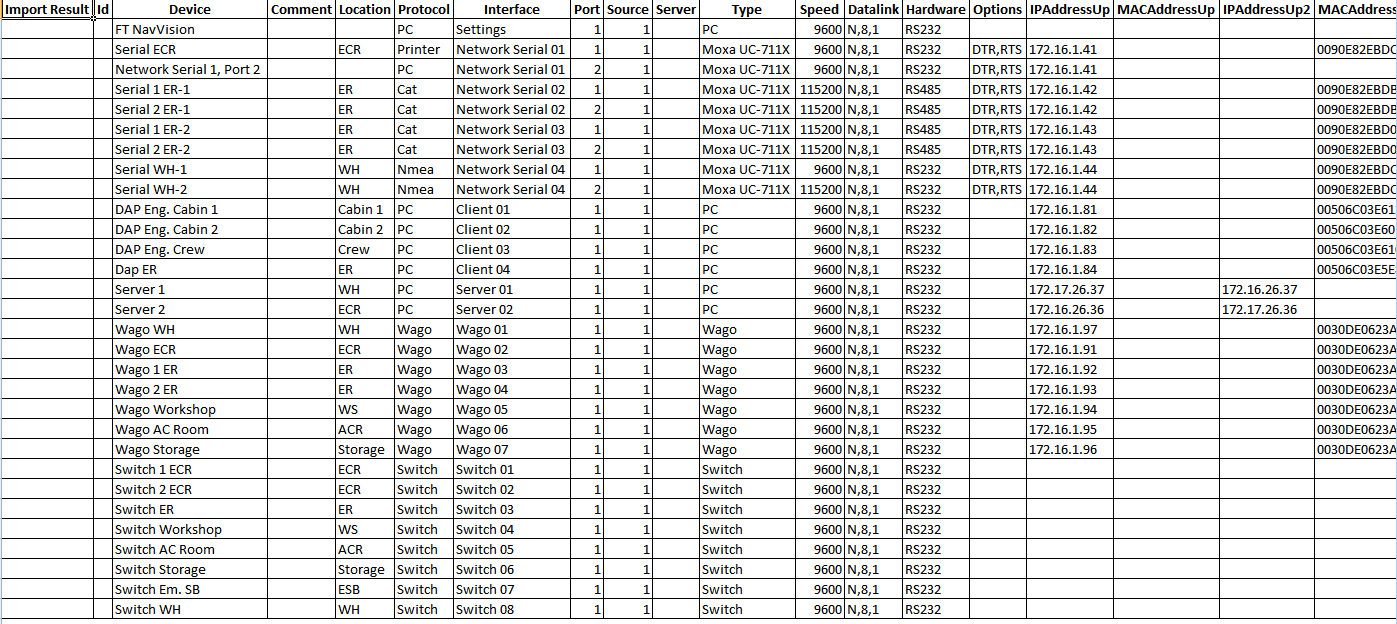


Figure 11‑5: devicelist\_generated good

This is of course when the devicelist was good in the first place. This is the kind of devicelist\_generated that you want to get back, because then you know you did well. When you have made a mistake you will find a comment (with a color) in the first column import result”. You can have something like the following figure:

|  |  |
| --- | --- |
| Field | Description |
| Comment | Comment that something is different in the field |
| Changed | Notice that something has changed |
| Failed | Critical failure somewhere in the field |
| Missing | Field tag is missing |
| New | Field is added since last import |

Figure 11‑6: Fault codes

The “comments” are merely there to make you aware that there is a small problem. Just check the line if there is an inconsistency in words or something. Sometimes it doesn’t even matter that there is a comment while you can deliberately made a difference in something. It doesn’t really affect he program.

The “changed” is there to warn you that there is something altered between the original configuration and the import. It can be two ways. Maybe FT NavVision© changed something because the program noticed that you made a mistake. Maybe you mixed up a protocol or whatever. The fault in the row behind the import result will also be colored yellow. Sometimes there is something changed that differs between the original configuration and what you imported with the sensorlist.

## Sensorlist generated

The sensorlist has the same import result column. It also has the same fault codes as the devicelist. The only extra field that the sensorlist\_generated has is the import result “New” in a green cell. This means that with the import of the sensorlist, you introduced a new i/o or control or that you changed something in the FT NavVision© program itself which is much more likely. In Chapter 12 we will explain that these fields are of much importance to keep the sensorlist up to date. For now you must know what you are looking at when you open up the sensorlist\_generated or the devicelist\_generated. The sensorlist\_generated is mostly much bigger than the devicelist\_generated, so you can imagine that it will be a lot of work to keep the sensorlist up to date. See the next figure for a small excerpt of a devicelist\_generated:

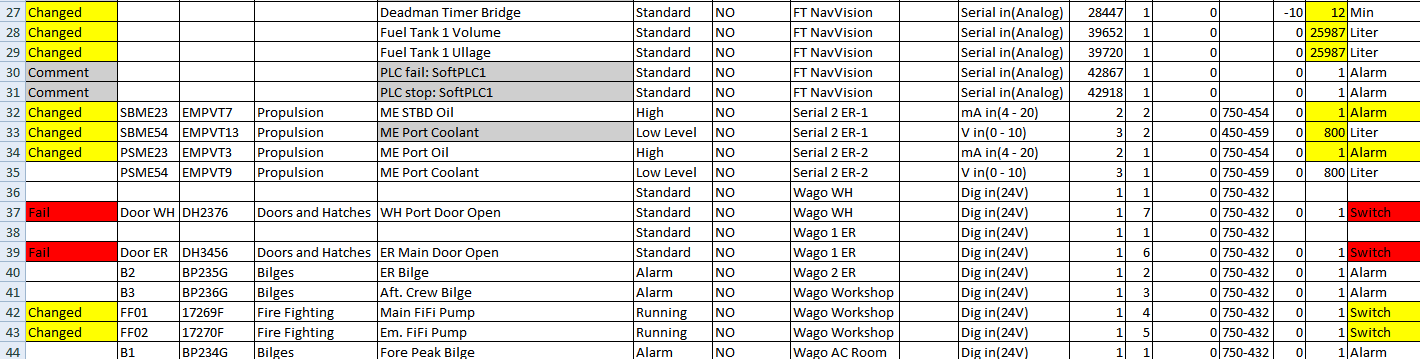


Figure 11‑7: sensorlist\_generated

## Sensorlist generated diff

The sensorlist\_generated\_diff is a help file that shows all the comments, fail and changes together with the corresponding original line (see ). This is ideal if you are troubleshooting the sensorlist. There you can see what is changed and the line underneath will tell you how it was original. In the next Chapter we will show you how you can make use of this file to keep the sensorlist up to date. There is no need to use it, but some people find it easier to work with. Others just use the sensorlist\_generated. It is up to you what you will use.



Figure 11‑8: Diff example 1

This is a typical example of a “comment”. You can see that FT NavVision© noticed that the name is changed. In the reference line you can see what it used to be. While this is probably the way you want it to be changed, you can ignore this comment.



Figure 11‑9: Diff example 2

Here you see a row that shows a changed state. In this case it is about the deadman timer bridge. Somehow in the program, somebody filled in 30 as the max amount of minutes. Later, probably after a new import, somebody changed it to 12 minutes. FT NavVision© notices this change and point it out for you here. If you feel it is alright you can leave it. You will, however, have to change it in the original sensorlist, or it will come back at the next import.

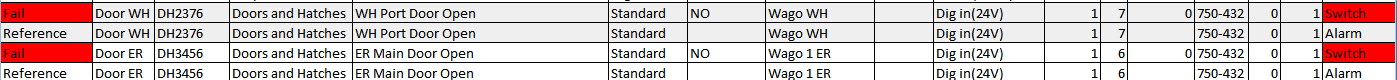


Figure 11‑10: Diff example 3

This concerns a real fault. FT NavVision© will look at the “field” column and see that the Field is not an alarm field, but a switch field. It will notice you that there is a fault and you have to change something in the sensorlist. Either you change the “DefaultField” into switch instead of alarm, or you need to change the “field” into an alarmfield. Either way you will have to adjust the sensorlist.

### Making an export

When you import a sensorlist FT NavVision© automatically generates the “\_generated”fields. There can be a time that you need to have one of these generated files without an import upfront. This can be done by stating FT NavVision© with the extension “EXPORT”.

Find the file “NavVision.exe” in the folder NavVision/bin/ and right click on it. Choose create a shortcut. Right click on that shortcut and choose “properties”. In the target window type EXPORT in capitals at the end of the line (see ).

Choose OK and start FT NavVision© up by doubleclick on the shortcut. When FT NavVision© has started you can close it directly. FT NavVision© will have generated the files. Now you can go further as planned.

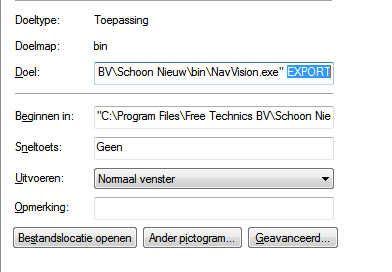


Figure 11‑11: Export shortcut

# Keeping up to date

## Introduction

Now you have seen what the sensorlist is capable of, you might have guessed that the sensorlist is the spill of the system. From the first build, up to changing large amount of data, the sensorlist is the tool for working with FT NavVision©.

It is very important that you keep the sensorlist up to date during commissioning. The best way to do this is probably have the sensorlist open at your laptop and change immediately everything that you change in FT NavVision© on board. We know that it is sometimes very hectic and you don’t have the time to do this directly. In that case it’s best that you change it right after you finished your days’ work. This way you can use the sensorlist the next day again.

We will explain here the different methods of keeping the sensorlist up to date.

## Direct changing

So this is the one that you keep the latest sensorlist open at your laptop, next to the workstation that you are working on. When you alter something directly on the workstation, you can immediately change that in the sensorlist.

I already gave an example in Chapter 10.5 with the crew names. But now lets say that you are working on the workstation and you find out that the serialnetwork on moxa 1 port 1 has a different baudrate. The seriallan is the 1st one in the ER en you have to change port 1 to a baudrate of 38400 instead of 115200. In FT NavVision© you change this on the workstation and the connection seems to be good.

Next time you import the sensorlist, you might wonder why the port isn’t working anymore. This is why you need to change it in the sensorlist in the tab “devicelist” to make sure next time the import will be in order. So go to your laptop, click on the devicelist tab and change the baudrate accordingly (see Figure 12‑1 and Figure 12‑2).

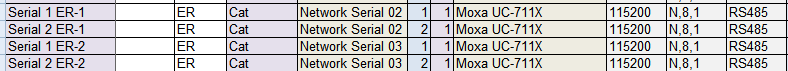


Figure 12‑1: changing baudrate old

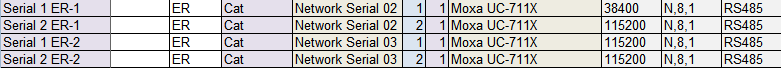


Figure 12‑2: Changing baudrate new

Same goes for the changes in the sensorlist. Again you’re working on the workstation and you notice that you have to change a connection at the Wago. It seems that the connections on the Wago Workshop are switched. The sensor on pin 3 is on pin 5 and the sensor on pin 5 is on pin 3. Of course you can change the wires on the Wago itself but for argument sake we say that you change the fieldnames in the Wago-section of the workstation.

Again you need to change this in the sensorlist or it will get back to the old state as you import the sensorlist again. The original lines you will find in the next figure:



Figure 12‑3: Changing Wago original

Now you can switch the whole line with names, fields and everything (see Figure 12‑4). Realize that you still need to change the pin-number, or nothing will change. For readability this will be the best option and also if you have to change a lot of pin numbers this is more synoptic. There will be an example later.



Figure 12‑4: Changing Wago lines

If it is about small amounts of changes it is easier to just change the pin-number. FT NavVision© doesn’t mind and will put it in the right order into the system. See next figure:



Figure 12‑5: Changing Wago numbers

### insert

When you need to insert a new connection into the Wago (an extra sensor for example), it could be very easy to do as you can read in the “Installation and commissioning manual”. Just choose a free pin in FT NavVision© Tools>Configuration>Wago. However, don’t forget to put that also in the sensorlist or you will lose that connection again after importing.

Same goes for extra devices in the “devicelist” tab. Just remember: importing a sensorlist will overwrite every change you have made on the system.

## Bigger changes

One of the bigger changes that can take place is that you have to change the order of the Wago slices or you will have to add a Wago slice somewhere. This will mess up the whole configuration. Without using the sensorlist this is almost impossible to do.

Let’s pretend you have the following configuration:

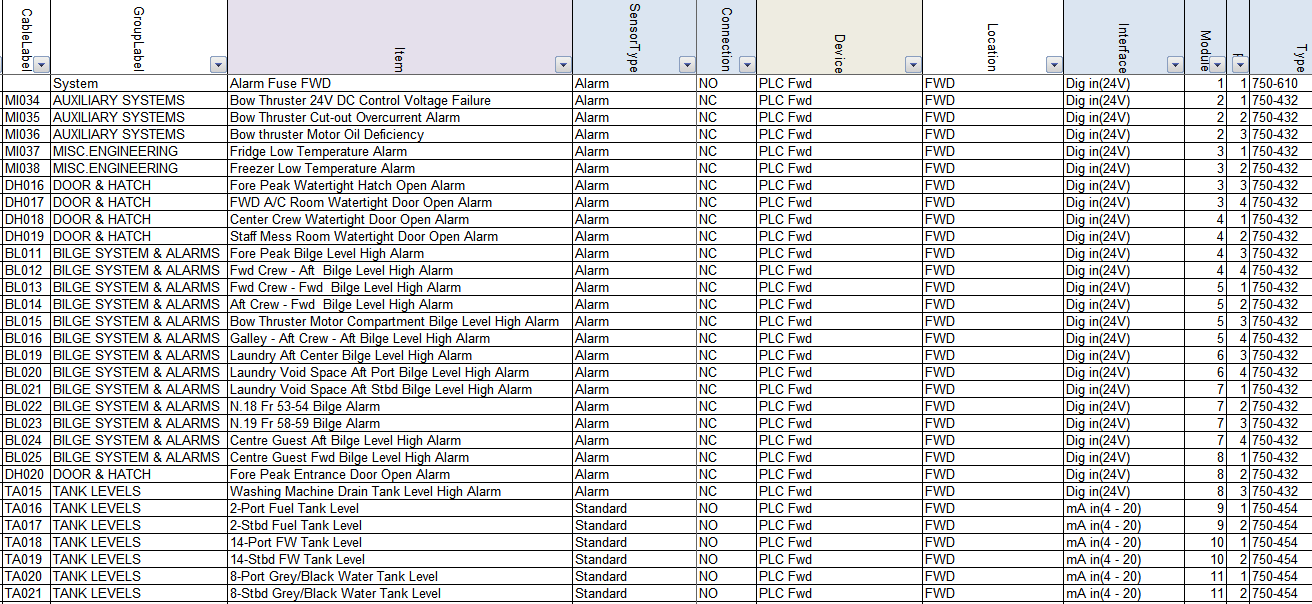


Figure 12‑6: Inserting a Wago slice 1

Now you need to put an extra slice (DI) 750-432 after the 3rd slice in the Wago. If you do that FT NavVision© will see that as a slice without a number and all the fields after slice 3 will go back one slice. You can imagine that is not what we want.

Now let’s do this with the sensorlist. You insert an empty row after the 3rd slice (see Figure 12‑7). Now this will be the 4th slice so at the module column you say it is number 4 and you fill in all the other appropriate fields (see Figure 12‑8).

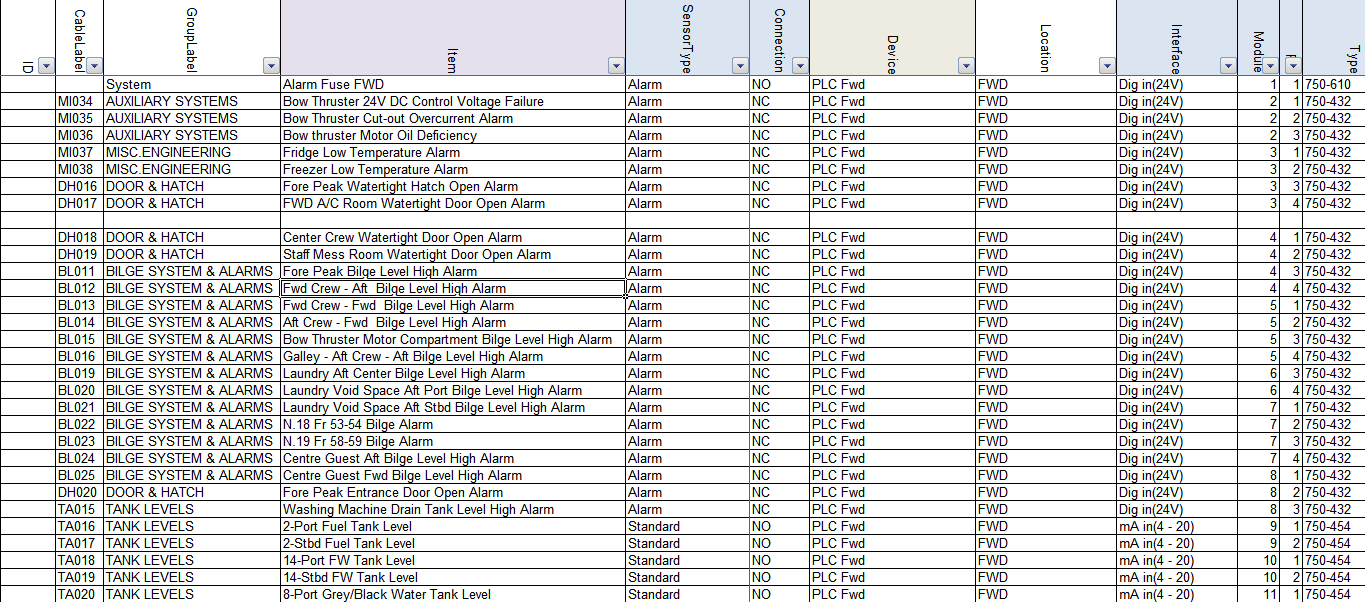


Figure 12‑7: Inserting a Wago slice 2

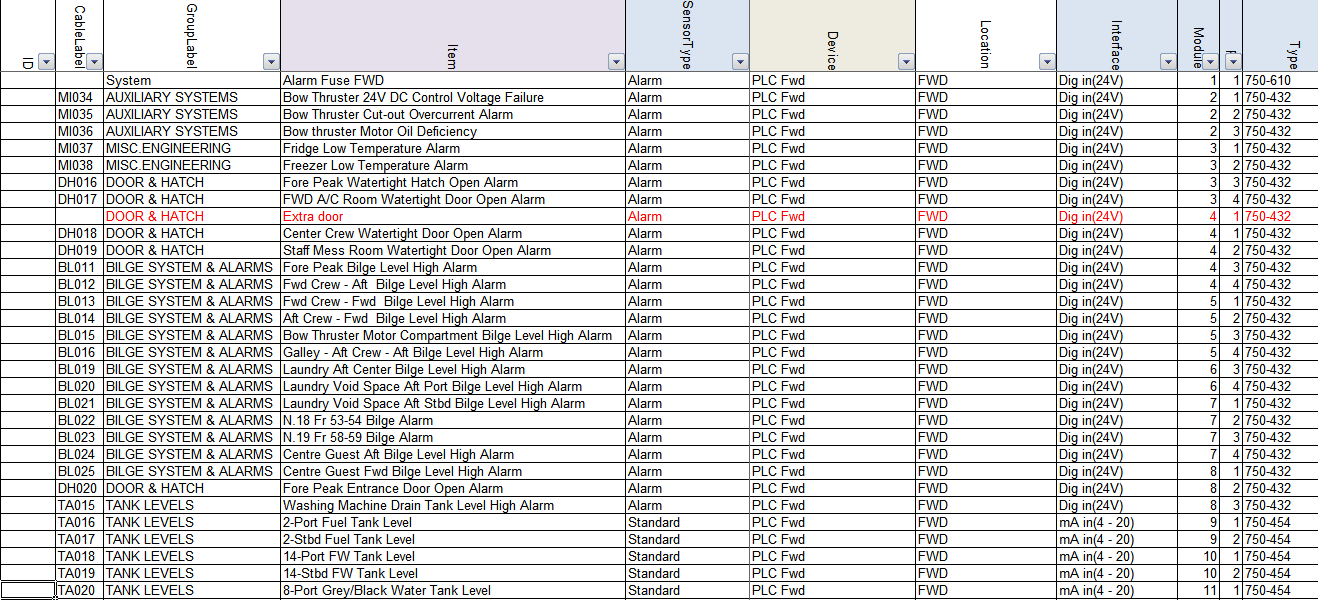


Figure 12‑8: Inserting a Wago slice 3

Now you will have two Wago slices with number 4 so you will need to increase the rest of the module numbers on that Wago. Of course you can do this by hand, but Excel is very helpful in this. Just find a cell with number 1 in it (cause we need to increase the modules by 1) an click CTRL-C to copy the number. Now select all the select all the cells in the module-column that need to be adjusted and right-click. Select “Paste Special” (see Figure 12‑9).

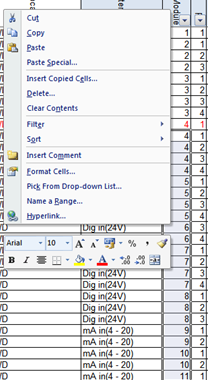


Figure 12‑9: Excel trick 1

In the next window choose “Add” and then click OK (see Figure 12‑10). You will see that all the module numbers has increased by 1.



Figure 12‑10: Excel trick 2

Now you can easily import the sensorlist (after you inserted the new Wago slice) and it will set everything in its right place.

## Keep the sensorlist up to date afterwards

### Introduction

Most likely you will find yourself occupied with work or you will get on board and the crew has made a lot of changes. In both cases it is impossible to use the sensorlist because it probably makes more problems than that it serves you. In that case you need to clean up the sensorlist first. After the clean-up you can use the sensorlist again.

To clean up the sensorlist you need to follow the instructions below. This is, for now, the best way to do this. The bigger the sensorlist is and the more changes, the more time-consuming it will be. But in the end you will only benefit.

### What do you need

You need a complete clean installation of the latest FT NavVision© on your pc/laptop. Keep this one clean and copy your key file (the \*.key.ini) into the folder NavVision/config/network.

If you start at a new project, or wish to make a new beginning, make a new folder and name it after your project. Copy al the files from the clean FT NavVision© folder into your new folder. You will get the following folder:

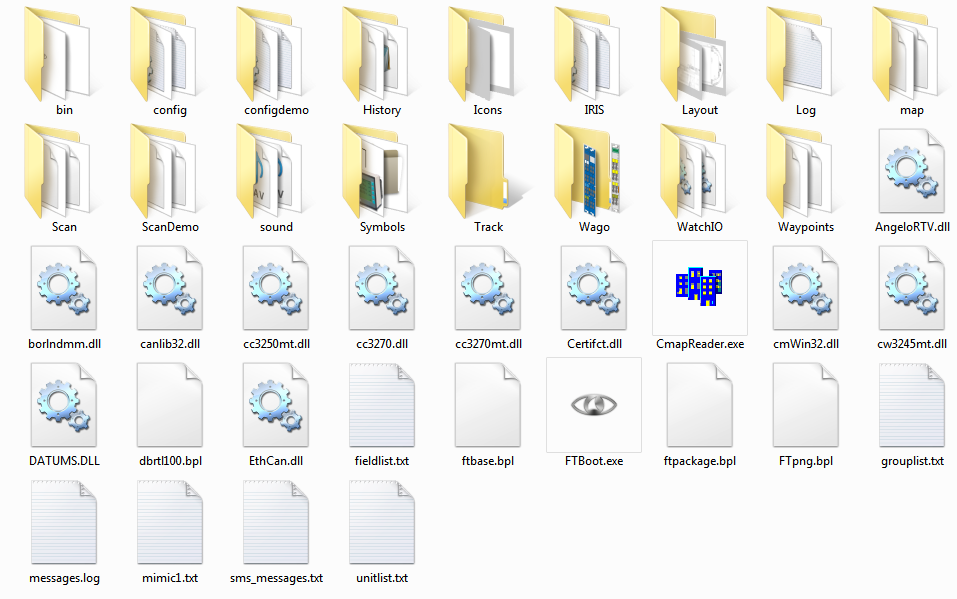


Figure 12‑11: clean FT NavVision© folder

Also you need the config-folder from the installation on board (better back-up the whole FT NavVision© folder). You can do this at the end of the day, when you have finished working on the system, or at a ship that you arrive at for commissioning.

### Cleaning up after a day on board

After you have been on board all day we assume that you have made a backup of the system. Now you do have an existing sensorlist, but we need to find out the changes. Here are the steps you need to take.

#### Copy devicelist.dat and sensorlist.dat

In the backup you took with you from aboard you find two files in the folder NavVision/config/network. These files are:

* Devicelist.dat
* Sensorlist.dat

Now copy these files and paste then in the folder NavVision/config/network of the folder you made on your pc/laptop as in Figure 12‑11. This folder now contains the configuration on board as it was when you left. Don’t start up yet.

#### The old sensorlist

You also have the old sensorlist.xls that you had before you went on board. If you do not already have the file as described, but only the raw sensorlist, we refer you to Chapter 8.4 to see how to save a sensorlist for import.

Copy this sensorlist.xls in to the root of your project folder. It will now look as follows:

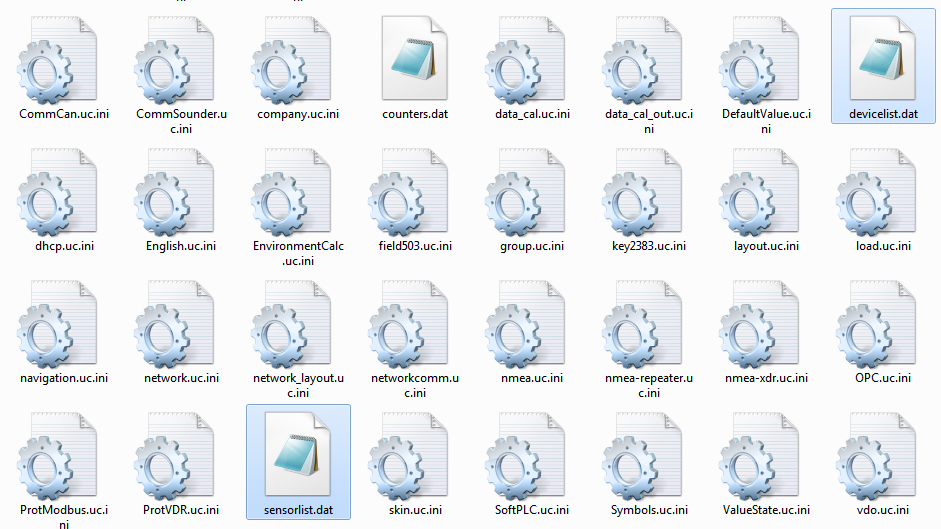


Figure 12‑12: Devicelist.dat and sensorlist.dat in network folder

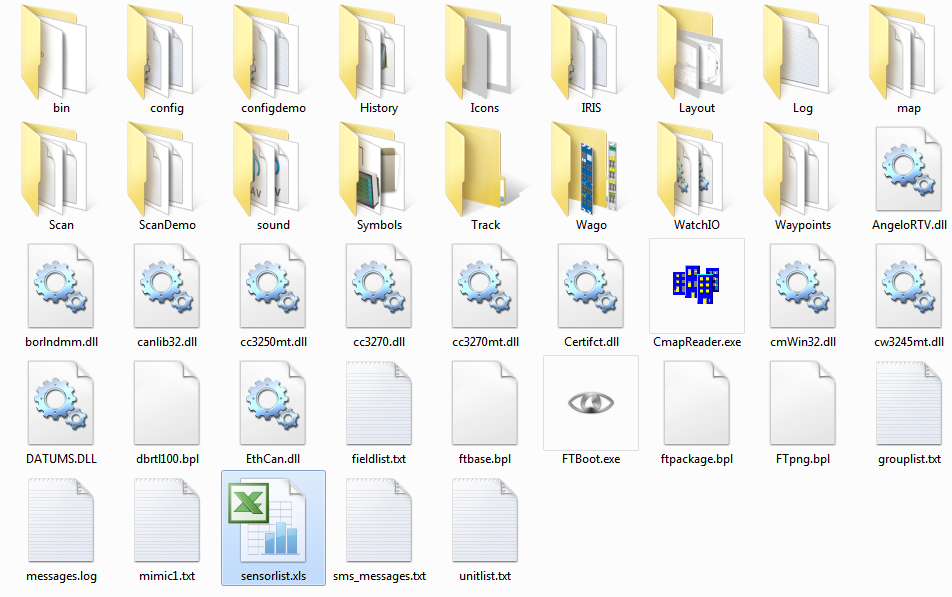


Figure 12‑13: sensorlist.xls in root of project folder

#### Startup your project folder

Now you must start up the FT NavVision© that is in your project folder. To do so, go to the folder NavVision/bin and double-click the NavVision.exe. This way you know that you start the right version.

During startup FT NavVision© will ask you if you want to import the devicelist and after that the sensorlist. Answer both questions with “Yes”. FT NavVision© will start up completely.

After it started up you can shut it down immediately. FT NavVision© will now generate de devices you need. These are:

* devicelist\_generated.html
* sensorlist\_generated.html
* sensorlift\_generated\_diff.html

These files can be found in the root of your projectfolder which now looks like the following:

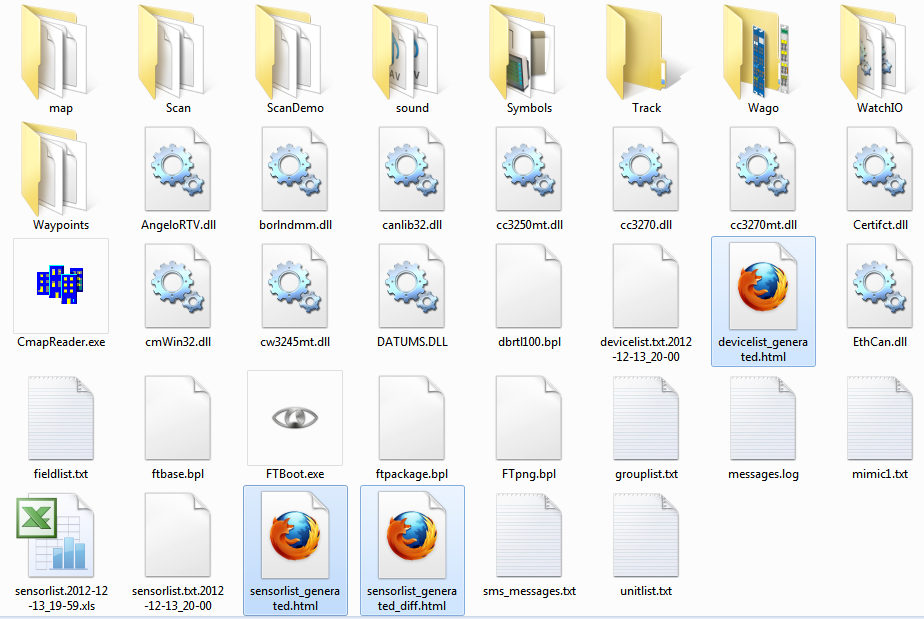


Figure 12‑14: root folder after import sensorlist

#### Inspecting the generated files

What goes for the sensorlist\_generated will also count for the other generated files, so we will only discuss this file here.

Open up the sensorlist\_generated.html (right-click, open with, Microsoft Office Excel). You will now have the sensorlist but also the column ImportResult filled in. If the field is blank than nothing has changed. Just pay attention to the fields that are colored and have a result in it.

This results can be:

|  |  |
| --- | --- |
| Field | Description |
| Comment | Comment that something is different in the field |
| Changed | Notice that something has changed |
| Failed | Critical failure somewhere in the field |
| Missing | Field tag is missing |
| New | Field is added since last import |

Table 12‑1: Import Result fields

This results will almost always be explained by the same color in the row that triggered the code.

Also you can open the sensorlist\_generated\_diff.html to see a reference to the same row (the old value that was there before you imported the sensorlist).

#### Comment

Comment usually indicates a minor problem or no problem at all, but you will need to check them. A simple example is that you see the following line:



Figure 12‑15: Comment example 1

If you look further down the row you will see that the problem is the text “bulb nav light SB 1” as you see in the next figure:



Figure 12‑16: Comment example 2

The fact is that “comment” usually indicates that the text is already in use somewhere in the sensorlist. Also it is possible that the field, in this case “AftNavLightSB” is already in use. Use the search function of Excel to find the text throughout the sensorlist.

In this case we will find that the text and the field is also used in line 71 as showed in the next figure:



Figure 12‑17: Comment example 3

You always have to check closely, but in this case it is fairly easy. Line 44 is the status connection as you will find in the SensorType column and it is connected to a DI-module. Line 71 is Standard connection and is connected to a DO-module. As you know how FT NavVision© works this is no problem. With line 71 you can switch the line on and if the light is on it will give a status back on line 44.

Now you now it is no problem and you can leave the row as is.

*: although it is only a comment, do check all fields for abnormalities. If you are sure it is ok, mark it in the sensorlist.*

#### Changed

Changed indicates that there is a bigger problem. It is a warning. It can be that a value has changed in the min/max settings, or an Item-name is changed or even the interface is changed. Eventually something can be changed in either column.

For your convenience FT NavVision© will show the changed cell in yellow as well. So it is easy to look up. It can even be in multiple cells, so have a good look. See the next figures as example:



Figure 12‑18: Changed example 1



Figure 12‑19: Changed example 2

As you can see there is a yellow colored field that will give you the changed value. In these examples it changed the interface. If you are not sure why it is changed or what was there before, you open up the sensorlist\_generated\_diff.html to see the reference. If we take the second figure as example and we look that up in the sensorlist\_generated\_diff.html, we’ll see the following:



Figure 12‑20: Changed example 3

Now you can check that in FT NavVision© it was defined as mV in(-125-125). As FT NavVision© knows that a Wago 750-469 slice is a Thermo in (K) slice it changed that interface to the right one.

Now that you know that it was changed because of the right reason, you also will have to change it in your sensorlist to keep that up to date.

*: make sure that you check all the changed fields and adjust them accordingly in your sensorlist. It is not possible with a changed field that you leave one unchanged. They all need to be altered in your basic sensorlist.*

#### Failed

Failed is a critical warning. There is something really wrong in that specific line. It can be anything, from missing information to double sensors. You will have to check the line very carefully. Sometimes it will show a red colored cell to show you what is wrong, but other times you will have to dig deeper to find the problem.

Failed always needs to be rectified in your original sensorlist. Here a simple example:



Figure 12‑21: Failed example 1

This is a sensor on a bus-protocol. As you can tell it was put twice in the sensorlist. Bus-protocols can hang on such information, so it is wise, in this case that you remove the Failed line from your original sensorlist.

#### Missing

Missing is an easy one. In this row the field tag is missing. You can go straight to the Field-column and you will find it is empty. See next figure:



Figure 12‑22: Missing example 1

Find the right field as described in Chapter 10.3.19.1 and put that in the original sensorlist.

#### New

Everything that was changed on board and that wasn’t already in the sensorlist will become visible as new. This could be a new sensor on a Wago, but also a complete new device or interface with, for example a bus-protocol.

The next example is when a new device or interface is connected. You will see the following:

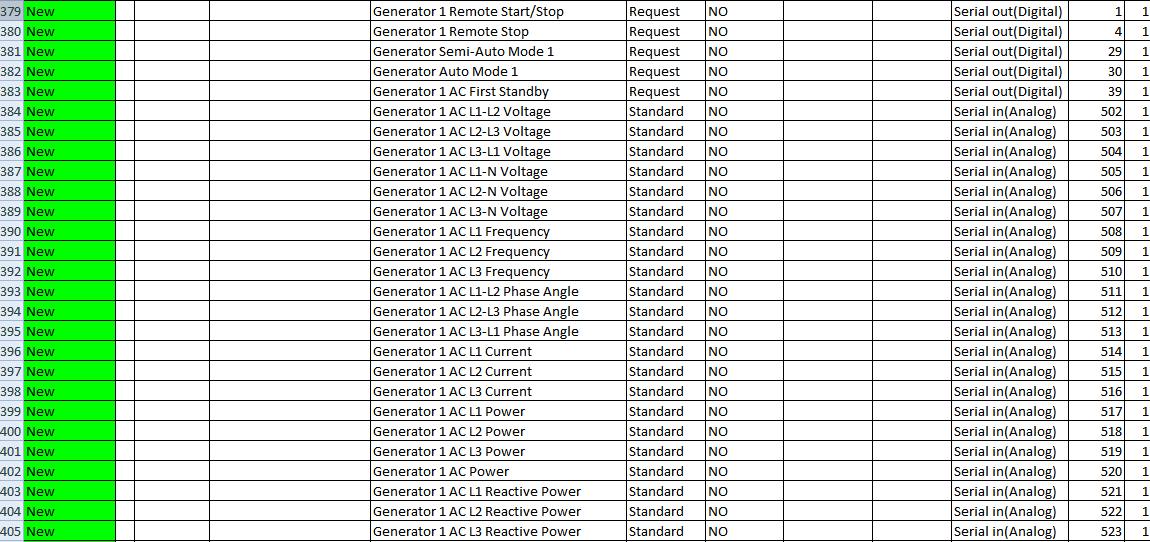


Figure 12‑23: New example 1

You can understand that you have to copy all these lines and paste them into the original sensorlist or they will get lost with a new import.

#### Keep importing

After you checked and replaced all the import results into the original sensorlist, you once again convert it to a sensorlist for import as described in Chapter 8.4 and put it in the root folder of your project folder.

Start FT NavVision© again and import the devicelist and sensorlist. Close FT NavVision© and open the new sensorlist\_generated.html.

If you did well there are no more import results except maybe for a few comments that you left there. If not you will have to repeat this process over and over again until there are no more import results and the sensorlist\_generated\_diff.html is empty.

Once you have reached that point you are finished and your original sensorlist is up to date again.

*: if you arrive on a ship after a long time and the crew has changed a lot, you can follow the same procedures. Just make a backup (or let them send one upfront) and go through all these steps. That way you can start directly with a good and working sensorlist.*

# Special notes

## Introduction

In this chapter we will discuss some special issues or things that are easy to know. It will just be a collection of extra knowledge randomly addressed and will be changed over time.

## PLC

When a PLC program is written and put into the Wago PLC itself it is necessary, especially for the outputs, that FT NavVision© doesn’t have field tags attached. To prevent the PLC program as well as FT NavVision© to address the output on the Wago, you do the following:

The rows in the sensorlist that hold the outputs that already are in use by the Wago PLC program, will need to be adjusted. First you add a “,PLC” after the module number in the column “Type” See the next figure:

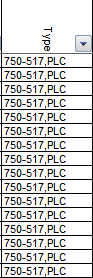


Figure 13‑1: PLC added

This way FT NavVision© knows that the slice is in use by the Wago itself, but will show up in the Wago overview in FT NavVision© see the next figure:

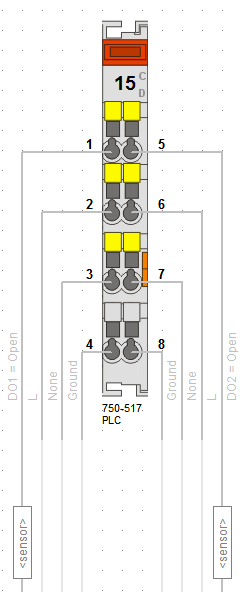


Figure 13‑2: Wago overview PLC

*: Leave the field column empty or it will interfere with the Wago PLC program.*

## Search

When you check the “sensorlist\_generated” you will often find comments. Most of the time it will be that you used the same field-ID in different rows. When you find a comment, go to the column “Type” and copy the field-ID. Press CTRL-F and you get a window where you can search. Paste the copied field-ID and select “Find all”. Now you can scroll through the fields to see if you have used the same Field-ID on multiple rows. If you find it, repair the problem and it will be fixed.

## Setting NMEA in the sensorlist

Since revision 3616 it is also possible to set the NMEA interfaces directly in the sensorlist. This needs an extra explanation cause it works a slightly bit different.

We will focus on the columns that are important. The other columns will all practically work the same as described earlier.

As example we will take a Voith NMEA interface. As you can see in the following figure, the standard columns will be the same as you already learned.

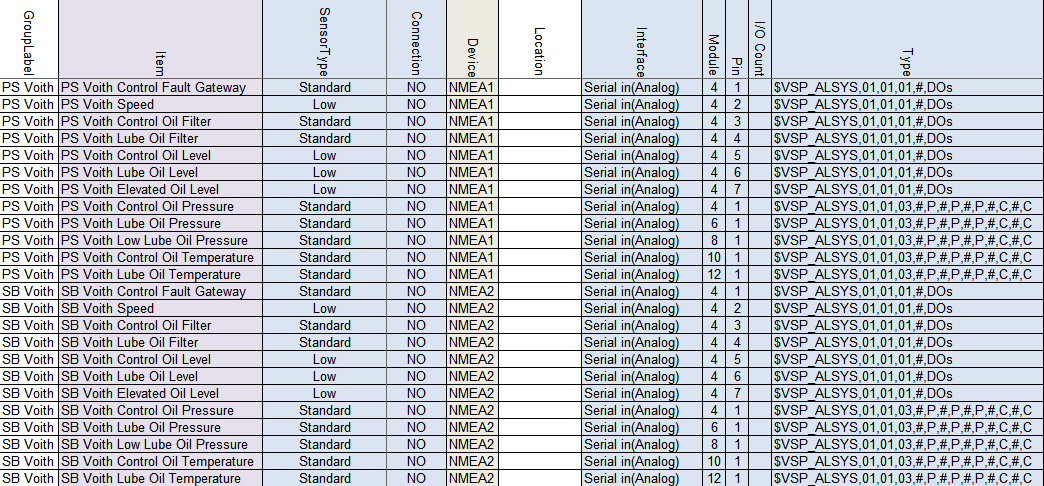


Figure 14‑3: NMEA sensorlist example

Grouplabel, Item, Sensor Type, Connection and Device are the same as described earlier. The alternative columns we’ll describe here.

### Interface

With NMEA you can choose between Serial in(Analog) and Serial out(Analog), depending if you want to receive or send.

### Module

As you will see in the column “type” you set the standard NMEA sentence there. All values are defined between comma’s in that sentence. To let FT NavVision© know which value you are looking at, you will set the comma after which the value is available in the NMEA sentence. So if you need the value after the 4th comma in the NMEA sentence, you will put a 4 here.

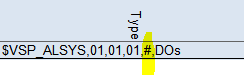


Figure 14‑4: NMEA example 1

*:the “#” sign is just to make it more visible and is not mandatory. You can leave the string without these.*

### Pin

To see which character behind the specific comma you need, under Pin you define the character number. In our example we have on that spot the digital values for the VOITH. So there are 7 zero’s or ones there, each representing one digital input. In our example we define all these values in the first seven rows.

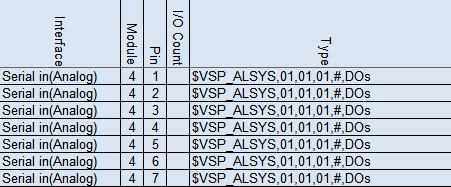


Figure 14‑5: NMEA example 2

*:Make sure that the count column is set to “1” cause you only want to read one character at the time.*

### Type

The Type column is the specific NMEA sentence that you are expecting. Lets analyse a sentence.

$VSP\_ALSYS = talker ID and Sentence Identifier

,01 = digital value

,03 = analog value

,# = wildcard

Or another example:

$GPRMC,220516,A,5133.82,N,00042.24,W,173.8,231.8,130694,004.2,W\*70

$GPRMC = talker ID and Sentence Identifier

,220516 = time stamp

,A = valid or ”V” invalid

Etc.

If you know the characters that are needed, you can fill it in.



The “P” and “C” represent Pressure and Celcius.

*:make sure that if you have an analog value, you set the Count column to the right amount of characters to read. Default is 16, which should be enough in most cases.*

### Count

At the count column you specify how many characters you will read at maximum on that specific location. So for digital values that will be 1. For analog values you will have to look at the original NMEA sentence. It can be that you need to read 4 characters max or 6. Whatever max number of characters you find for that field, you will define here at “count”.

*:the column “Data Type” is necessary if you send NMEA data. You will set the right parameter (see chapter 11.3.23). If NavVision is reading NMEA data it will ignore “Data Type” so you can leave it blank.*

### Count

At the count column you specify how many characters you will read at maximum on that specific location. So for digital values that will be 1. For analog values you will have to look at the original NMEA sentence. It can be that you need to read 4 characters max or 6. Whatever max number of characters you find for that field, you will define here at “count”.

## WatchIO in the sensorlist

Since revision 3904 The new setup of implementing WatchIO is in effect. WatchIO is the main protocol of the Unimacs bridges.

We will focus on the columns that are important. The other columns will all practically work the same as described earlier.

First make sure that the devices are set in the Devicelist. NavVision has to know where to look for the device. The following figure shows the mandatory fields for the devicelist:

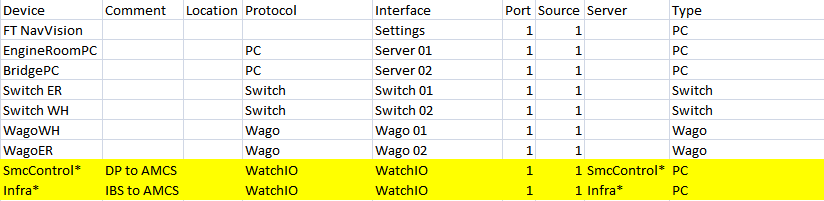


Figure 14‑6: Devicelist WatchIO

|  |  |
| --- | --- |
| Field | Description |
| Device | The name of the WatchIO view the asterix (\*) at the end is needed for redundancy |
| Comment | Free text |
| Location | Free location name |
| Protocol | Protocol must be WatchIO |
| Interface | Interface must be WatchIO |
| Port | Every protocol has to have a unique port number |
| Source | Mostly one |
| Server | Same as Device |
| Type | Type must be PC |

Table 14‑1: Devicelist WatchIO

In the sensorlist we also have a slightly different approach. While the main fields will be the same we will focus on the differences by the following example”

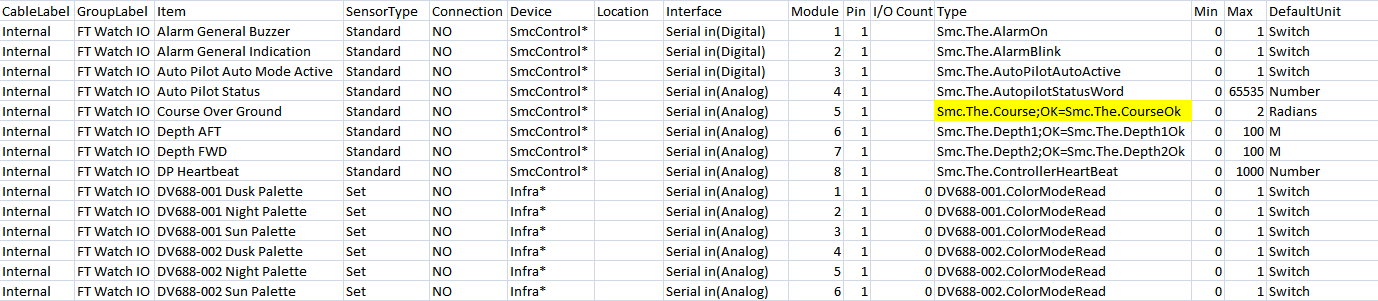


Figure 14‑7: Sensorlist WatchIO

|  |  |
| --- | --- |
| Field | Description |
| Cable label | Free (preferably “Internal”) |
| Group Label | Free |
| Sensortype | Preferably “Standard” only when you want to set something or you just need a part of the variable. For the last one use “Enum” |
| Connection | NO |
| Device | Choose the one from the devicelist |
| Location | Free |
| Interface | Serial in Analog or Digital |
| Module | A unique number. Start with 1,2,3…..etc. |
| Pin | 1 |
| I/O count | Free |
| Type | Here you need to put the type Variable. You can get that from the Unimacs program. For status you also put the OK-Variable behind a semi-colon. |
| Min | Normal Min settings |
| Max | Normal Max settings |
| DefaultUnit | Normal DefaultUnit |

Table 14‑2: Sensorlist WatchIO

The rest of the fields is to be treated the same as mentioned earlier in this document.

Free Technics

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