Contents

1 Analysis of RS422/485 Bus systems  
1.1 Special serial driver software .................. 3  
1.2 Bus-tap 2-Wire Bus .............................. 3  
1.3 Double bus-tap 4-Wire bus ....................... 4  
1.4 Sampling ....................................... 4  

2 MSB-RS485 Analyzer  
2.1 Advantages of a hardware solution ............... 5  
2.2 Innovative software concept ..................... 6  
2.3 Application fields ................................ 7  

3 Features & Benefits ................................ 9  

4 Specifications ....................................... 11  

5 Program Installation  
5.1 Installation under Linux .......................... 16  
5.2 Manual installation under Linux .................. 16  
5.3 Installation for all users ........................ 16  
5.4 Program Updates ................................ 17  

6 Connection of the Analyzer  
6.1 Definition of the Signal lines .................... 19  
6.2 Internal Signal Processing ....................... 20  
6.3 Digital In/Outputs ................................. 21  
6.4 Bus Termination and Tapping ..................... 21  
6.5 Tapping 2-wire system ........................... 22  
6.6 Segment Analysis 2-wire system ................. 22  
6.7 Tapping 4-wire system ........................... 23  
6.8 Segment Analysis 4-wire system ................. 24  
6.9 Signal assignment ................................ 24  
6.10 Lightment elements LEDs ....................... 25  
  Green LEDs ........................................ 25  
  Red LEDs .......................................... 26  

7 Program start  
7.1 User Interface .................................... 28  
7.2 Select kind of connection ....................... 28  
7.3 The first start ................................... 28  
  Automatical protocol scan ....................... 28  
  Manual Protocol setup ............................ 29  
  Start/stop a recording ............................ 29  
7.4 Status display .................................... 29  
  Display I ......................................... 30  
  Display II ........................................ 30  
  Display III ....................................... 30  
7.5 Config a recording ................................ 31  
  Connection ....................................... 31
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus wiring</td>
<td>32</td>
</tr>
<tr>
<td>Signals</td>
<td>32</td>
</tr>
<tr>
<td>Record mode</td>
<td>33</td>
</tr>
<tr>
<td>Autosave</td>
<td>35</td>
</tr>
<tr>
<td>General</td>
<td>35</td>
</tr>
<tr>
<td>7.6 The analysis tools</td>
<td>35</td>
</tr>
<tr>
<td>7.7 Save a recording</td>
<td>36</td>
</tr>
<tr>
<td>7.8 Save a session as a project</td>
<td>36</td>
</tr>
<tr>
<td>7.9 Open an earlier recording</td>
<td>37</td>
</tr>
<tr>
<td>7.10 Open an earlier session (project)</td>
<td>37</td>
</tr>
<tr>
<td>7.11 Last opened recordings and projects</td>
<td>37</td>
</tr>
<tr>
<td>7.12 Drag and drop</td>
<td>37</td>
</tr>
<tr>
<td>7.13 Connecting multiple analysers</td>
<td>38</td>
</tr>
<tr>
<td>7.14 Automational start after computer boot</td>
<td>38</td>
</tr>
<tr>
<td>Activate the autostart feature</td>
<td>39</td>
</tr>
<tr>
<td>7.15 Short commands</td>
<td>39</td>
</tr>
<tr>
<td>7.16 Additional program arguments</td>
<td>40</td>
</tr>
<tr>
<td>7.17 Special program parameters</td>
<td>41</td>
</tr>
<tr>
<td>8 The MultiView design</td>
<td>43</td>
</tr>
<tr>
<td>8.1 Synchronization</td>
<td>43</td>
</tr>
<tr>
<td>Follow (autoscroll)</td>
<td>44</td>
</tr>
<tr>
<td>Locked (fixed)</td>
<td>44</td>
</tr>
<tr>
<td>Linked</td>
<td>44</td>
</tr>
<tr>
<td>8.2 Views (displays)</td>
<td>44</td>
</tr>
<tr>
<td>Virtual Ledtester</td>
<td>45</td>
</tr>
<tr>
<td>DataView - Data Monitor</td>
<td>45</td>
</tr>
<tr>
<td>EventView - Event Monitor</td>
<td>45</td>
</tr>
<tr>
<td>ProtocolView - Protocol Monitor</td>
<td>45</td>
</tr>
<tr>
<td>SignalView - Signal Monitor</td>
<td>45</td>
</tr>
<tr>
<td>Regions</td>
<td>46</td>
</tr>
<tr>
<td>8.3 Copy Views</td>
<td>46</td>
</tr>
<tr>
<td>8.4 Saving the state of the Views</td>
<td>46</td>
</tr>
<tr>
<td>9 Session management</td>
<td>47</td>
</tr>
<tr>
<td>9.1 Projects</td>
<td>47</td>
</tr>
<tr>
<td>9.2 Store and reload projects</td>
<td>48</td>
</tr>
<tr>
<td>9.3 Automatic storing of a session</td>
<td>48</td>
</tr>
<tr>
<td>10 The virtual Ledtester</td>
<td>49</td>
</tr>
<tr>
<td>10.1 The toolbar</td>
<td>50</td>
</tr>
<tr>
<td>11 The Data View</td>
<td>51</td>
</tr>
<tr>
<td>11.1 User Interface</td>
<td>51</td>
</tr>
<tr>
<td>Data channel selection</td>
<td>53</td>
</tr>
<tr>
<td>Synchronizing</td>
<td>53</td>
</tr>
<tr>
<td>Addressing the window content</td>
<td>53</td>
</tr>
<tr>
<td>11.2 Data selection</td>
<td>54</td>
</tr>
<tr>
<td>Copy and Paste</td>
<td>54</td>
</tr>
<tr>
<td>Save data selection</td>
<td>54</td>
</tr>
</tbody>
</table>
CONTENTS

Export a data selection ........................................... 55
11.3 Data displaying ............................................ 56
  Columns and data format .................................... 56
  Coloring data .................................................. 56
  Change the font .............................................. 57
11.4 The data inspector ......................................... 57
11.5 Searching the record ....................................... 57
  Pattern search ............................................... 57
  Search for time distances .................................. 59
  Search for transmission errors .............................. 60
11.6 The Watch window .......................................... 60
  The script editor ............................................ 60
  Example scripts ............................................. 61
  Limitations .................................................. 62
11.7 The toolbar ................................................ 62
11.8 Short commands ............................................ 63

12 The Event View ............................................. 65
  12.1 User Interface ........................................... 65
    Each line is one event .................................... 66
    All event types at a glance .............................. 66
  12.2 Navigation through the event list ....................... 67
  12.3 Event search with the LevelFinder ....................... 67
    Enter a search pattern .................................. 67
    Formulate a level condition .............................. 68
    Formulate a data error .................................. 68
    Formulate a data value .................................. 69
    Search input and search ................................ 69
    Search for signal changes ................................ 70
    Searching with time specification ....................... 71
  12.4 Mark a selection ......................................... 71
    Save a selection as a region ............................. 72
    Export a selection as CSV file ........................... 72
  12.5 Measure time distances .................................. 74
  12.6 The toolbar .............................................. 75
  12.7 Short commands .......................................... 75

13 The Protocol View .......................................... 77
  13.1 User Interface ........................................... 77
  13.2 Protocol Display ......................................... 78
    Synchronizing the display ................................ 79
    Choosing a range ......................................... 79
    Copy and Paste .......................................... 79
    Saving a protocol ....................................... 80
  13.3 Protocol Templates ....................................... 80
    Define your own templates ............................... 80
    Modify an available template ......................... 81
    Saving templates ........................................ 81
  13.4 Template language syntax ................................ 81
    Splitting into datagrams ................................ 81
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions for splitting into datagrams</td>
<td>82</td>
</tr>
<tr>
<td>BCHG</td>
<td>83</td>
</tr>
<tr>
<td>CAB</td>
<td>83</td>
</tr>
<tr>
<td>CNT</td>
<td>84</td>
</tr>
<tr>
<td>CNTA</td>
<td>84</td>
</tr>
<tr>
<td>CNTB</td>
<td>84</td>
</tr>
<tr>
<td>EOS</td>
<td>85</td>
</tr>
<tr>
<td>RTL</td>
<td>85</td>
</tr>
<tr>
<td>TO</td>
<td>85</td>
</tr>
<tr>
<td>Individual display of the datagrams</td>
<td>86</td>
</tr>
<tr>
<td>Output functions</td>
<td>86</td>
</tr>
<tr>
<td>ALIAS</td>
<td>87</td>
</tr>
<tr>
<td>ASCII</td>
<td>87</td>
</tr>
<tr>
<td>CTRL</td>
<td>88</td>
</tr>
<tr>
<td>HEX</td>
<td>89</td>
</tr>
<tr>
<td>MASK</td>
<td>89</td>
</tr>
<tr>
<td>REPEAT</td>
<td>90</td>
</tr>
<tr>
<td>Comments</td>
<td>91</td>
</tr>
<tr>
<td>13.5 Changing Date and Time</td>
<td>91</td>
</tr>
<tr>
<td>Change the font</td>
<td>92</td>
</tr>
<tr>
<td>13.6 The Toolbar</td>
<td>92</td>
</tr>
<tr>
<td>13.7 Short commands</td>
<td>93</td>
</tr>
<tr>
<td>13.8 Some examples...</td>
<td>93</td>
</tr>
<tr>
<td>Modem Communication with echo</td>
<td>94</td>
</tr>
<tr>
<td>S-Record data transmission</td>
<td>96</td>
</tr>
<tr>
<td>14 The Signal View</td>
<td>99</td>
</tr>
<tr>
<td>14.1 Signal representation</td>
<td>100</td>
</tr>
<tr>
<td>14.2 Navigation</td>
<td>101</td>
</tr>
<tr>
<td>Navigation by mouse wheel</td>
<td>102</td>
</tr>
<tr>
<td>Shift with the hand cursor</td>
<td>102</td>
</tr>
<tr>
<td>14.3 The time base</td>
<td>102</td>
</tr>
<tr>
<td>14.4 Undo and Redo</td>
<td>102</td>
</tr>
<tr>
<td>14.5 Settings dialog</td>
<td>102</td>
</tr>
<tr>
<td>The signal dialog</td>
<td>103</td>
</tr>
<tr>
<td>Signal inverting</td>
<td>103</td>
</tr>
<tr>
<td>Signal sequence</td>
<td>103</td>
</tr>
<tr>
<td>Fade in the transferred data</td>
<td>103</td>
</tr>
<tr>
<td>Grafical effects</td>
<td>104</td>
</tr>
<tr>
<td>14.6 Cursor operating</td>
<td>104</td>
</tr>
<tr>
<td>Signal selection</td>
<td>105</td>
</tr>
<tr>
<td>Regions</td>
<td>105</td>
</tr>
<tr>
<td>14.7 Synchronizing</td>
<td>105</td>
</tr>
<tr>
<td>14.8 The toolbar</td>
<td>106</td>
</tr>
<tr>
<td>14.9 Short keys</td>
<td>107</td>
</tr>
</tbody>
</table>
# CONTENTS

## 15 Regions

- 15.1 Switch regions on/off ........................................... 110
- 15.2 Remove a region ................................................ 110
- 15.3 Rename a region ................................................. 110
- 15.4 Move regions into view ........................................... 110

## 16 A quick start with Lua

- 16.1 Getting started .................................................. 113
- 16.2 Accessing the Data Monitor ..................................... 114
- 16.3 Mark sequences in the data grid ................................. 115

## 17 Lua beginners guide

- 17.1 Lua is case-sensitive ............................................. 117
- 17.2 Whitespaces and line ends ...................................... 117
- 17.3 Comments ....................................................... 118
- 17.4 Types and values ................................................ 118
  - Numbers ........................................................................ 119
  - Booleans ................................................................. 119
  - Strings ....................................................................... 119
  - nil ............................................................................. 120
  - Tables ....................................................................... 121
  - Functions ............................................................... 122
- 17.5 Identifiers ............................................................. 122
- 17.6 Keywords ............................................................. 122
- 17.7 Variables .............................................................. 123
  - Assignment ............................................................. 123
  - Global and local variables ......................................... 123
- 17.8 Operators ............................................................ 124
  - Arithmetic operators ............................................... 124
  - Conditional operators ............................................. 124
  - Logical operators .................................................... 124
  - String concatenation operator .................................. 125
  - The length operator ................................................. 125
  - Precedence ............................................................ 125
- 17.9 Control structures .................................................. 125
  - if then else ............................................................. 125
  - while ...................................................................... 126
  - repeat ................................................................. 126
  - Numeric for ............................................................ 126
  - break ..................................................................... 127
- 17.10 Functions ............................................................ 127
  - Function call .......................................................... 127
  - Function definition ................................................ 127
- 17.11 Modules ............................................................. 128
  - Standard Modules ................................................... 128
  - Analyzer Modules .................................................... 129
    - Bit Module .......................................................... 129
    - Data View Module ............................................... 130
    - Record Module .................................................... 131
- 17.12 Analyzer specific data types ..................................... 131
## CONTENTS

17.13 Limitations ................................................. 132
17.14 Further information ......................................... 133

18 Synchronize two analyzers ................................. 135
18.1 Technical requirements ................................. 135
18.2 Master Slave operation ................................. 136
18.3 Establish a synchronous record ......................... 137
18.4 Analyse a synchronous record ......................... 138
18.5 Conclusion ................................................. 139
Synchronous recording .......................................... 139
Synchronous analysis ........................................... 140

19 Commandline API .............................................. 141
19.1 Combine the programs as a tool chain .......... 142
Data source ....................................................... 142
Manipulators ....................................................... 142
Data sink ......................................................... 142
Some examples ..................................................... 142
19.2 Record data with msb_record ....................... 143
Connection settings and events ......................... 144
Usage in your own application ......................... 144
Remote control .................................................... 145
msb_record program parameters ....................... 146
19.3 Formatted output with msb_format ............... 148
Output of any character ......................................... 149
File output ......................................................... 149
Format parameters ............................................... 150
User defined date and time ................................... 152
msb_format program parameters ....................... 153
19.4 Filtering data output with msb_filter .......... 154
Filter data ......................................................... 154
Filter certain signal events ..................................... 155
Filter a given record part ..................................... 155
msb_filter program parameter ......................... 155
19.5 Split records with msb_split ....................... 156
Split existing record files ..................................... 156
Splitting the current recording from msb_record .... 158
Program Parameter ................................................ 158
19.6 One config file for all ...................................... 159

A ASCII character table ........................................ 161

B Baudrate measuring ........................................... 163

C Colors ......................................................... 165
C.1 RGB short form ............................................... 165
C.2 RGB long form ............................................... 165
C.3 Predefined color names ..................................... 165
Grey colors ......................................................... 166
Basic colors ......................................................... 166
CONTENTS

Extended colors ........................................... 166

D Windows Trouble-Shooting ............................... 169
  D.1 Windows doesn’t found the analyzer (Part I) .......... 169
  D.2 Windows doesn’t found the analyzer (Part II) ....... 170
  D.3 MSB-RS485 Device quit working unexpectedly ........ 170
  D.4 Other problem ...................................... 171

E Linux Trouble-Shooting .................................. 173
  E.1 Linux doesn’t found the analyzer (Part I) ............. 173
  E.2 Linux doesn’t found the analyzer (Part II) .......... 173
  E.3 Linux doesn’t found the analyzer (Part III) ........ 174
  E.4 Recording doesn’t work .............................. 175
  E.5 Segmentation fault during installation .............. 176
  E.6 The help menu Help→Content F1 doesn’t work ....... 176
  E.7 The software doesn’t run within a 64 bit Linux (Part I) .... 176
  E.8 The software doesn’t run within a 64 bit Linux (Part II) .... 176
  E.9 Other problem ...................................... 177
Analysis of RS422/485 Bus systems

In contrast to other busses RS422 or RS485 define only the electrical characteristics. All further protocol levels can be specified freely. So beside the physical features an analysis also has to regard the different protocols.

RS485 and RS422 (or. EIA-422 and EIA-485) are mostly used synonymously because of their great similarity, where the EIA-422 standard is seen as a subset of EIA-485. But this is correct only partially. Both standards use a pair of twisted wires to transmit the inverted and non-inverted levels of a one bit signal. The receiver reconstructs the original data signal from the difference of both signal levels. In this way common mode distortions do not have much effect on the transmission which leads to a significant higher noise immunity.

As a consequence all data and handshake lines are designed as wire pairs. However no standardized terminal assignments for EIA-422 and EIA-485 exist.

A EIA-422 connection generally consists of two pairs of wires for send and receive and a common ground line which is conform to the classical EIA-232 connection. In case of RTS/CTS Handshake two additional wire pairs have to be used. EIA-422 primarily was developed to overcome the limitations of EIA-232 connections.

With EIA-422 Full-Duplex point-to point transmissions and Multidrop networks can be realized. The latter allows the unidirectional connection of up to 10 receivers, in which the transmission takes place in one direction only from the sender to the maximal 10 receiver.

EIA-485 was designed as a bidirectional bus system for up to 32 (and more\(^1\)) participants. Data can be transmitted optionally over a singe pair of wires Half-Duplex (the so-called 2-wire technology or short 2-wire) or in a Fullduplex capable way with two separate send and receive wire pairs (4-wire).

In a 2-wire system all sender and receiver are connected together through a single pair of wires. The main advantage of the 2-wire technology is its Multi-Master capability. Each bus device can exchange data with any other device. A well known application based on a 2-wire system is the PROFIBUS.

\(^1\) Depending on the so-called unit load it may be up to 256 participants
CHAPTER 1. ANALYSIS OF RS422/485 BUS SYSTEMS

4-wire busses (for instance the DIN-Messbus) are solely used as Master-Slave systems. That means that the data output of the master is connected to all data inputs of the slaves through a single wire pair. The data outputs of the slaves are all connected to the second wire pair which leads to the data input of the master.

In both variants only one device can drive (send), all other devices have to set their sender into tri state mode.

Some EIA-485 devices automatically care for a correct implementation of the tri state status (tri state when no data is sent), others have to be explicitly set into tri state by software control.

Physically both interfaces are almost identical so that EIA-485 devices can be used without problems in EIA-422 systems. But this is not possible the other way round, because EIA-422 drivers do not have the tri state mode which is necessary to operate multiple devices on one wire pair.

The analysis of a EIA-422/485 connection does not only have to care about the different connection varieties (and the bus signals). The EIA-422/485 specifications do not make a statement about the protocol levels. So a number of different transfer protocols are established, protocols with asynchronous (UART based) and synchronous serial data transmission.

Protocols with synchronous data transmissions use different kinds of bit coding, with or without synchronizing clock, and use special coding hardware.

In contrast the asynchronous transmission technologies are based on the UART which is the standard for serial interfaces and is installed in every PC and microcontroller. Because of its simply way of connecting (with EIA-232 or USB to EIA-422/485 converters) these protocols are widespread why in the following we focus on UART based asynchronous protocols. These are some of them:

1. Din-Messbus
2. Modbus ASCII
3. Modbus RTU
4. Profibus
5. Application specific protocols

The kind of the protocol plays a very important role not only for the logging and evaluation of the communication but also for the right choice of the analysis tools.

After this short side-note to the specification of EIA-422/485 the basis is laid for the question: Which possibilities for analysis of EIA-485\(^2\) communications are available and where are they appropriate for?

Because of the simple connection of EIA-485 busses, based on asynchronous data transmission, to a standard PC the following techniques for logging and evaluation are possible with appropriate software:

1. Data logging by the serial driver and additional software of the PC

\(^2\)EIA-485 Analysis tools are also appropriate for EIA-422 connections. In the following chapters we speak about EIA-485 only, meaning both standards.
1.1. SPECIAL SERIAL DRIVER SOFTWARE

2 Bus-tap by one EIA-485 converter (2-wire bus)

3 Bus-tap by two EIA-485 converters (4-wire bus)

4 Sampling of the bus lines by special additional hardware

1.1 Special serial driver software

If one of the participants of the communication is a PC (usually the master) an appropriate driver can be installed to protocol every sent and received data byte. This procedure only allows the logging and detecting of the data bytes which are processed by the serial driver of the operating system. The disadvantages are:

Data losses due to buffer overflows are not detected.

A precise time stamping is not possible. Indeed the received and sent data bytes are signalized by interrupt. But the interrupt is processed by the operating system after a not exact predictable time delay.

Therefore the time measurement of such Sniffer programs, most time in the millisecond range, have to be regarded with suspicion. The times are the times when the operating system handles the input and output of the characters. They are not the moment the character is really present on the data line. A statement about correct data and protocol timing can only be done with care. Some busses as Modbus RTU or Profibus define a send pause as the start or end of the data telegram. Data within a telegram have to be transmitted without gaps. Information about the tri-state condition get lost since the serial driver can process the two logical bus states only. Errors caused by data collision by reason of multiple sending bus participants can not be detected.

1.2 Bus-tap 2-Wire Bus

The PC is connected via an appropriate EIA-485 interface converter (EIA-232 to EIA-485 or USB to EIA-485) as an additional bus participant to the bus. This variant allows the logging of all transmitted data bytes with e.g. Hyperterm. The disadvantages:

Since send and receive data run over the same wires it is not possible to distinguish between sent and received data. A detailed examination of the telegrams is necessary to detect the data direction.

The behavior of the time measurement is the same as mentioned under 1.1.

The connection of a EIA-485 converter is normally done as a serial COM port (either as a virtual COM port in case of a USB to EIA-485 converter or direct in case of a EIA-232 to EIA-485 interface converter). In both cases the information about the tri-state condition gets lost with the consequence that bus errors, caused by multiple active senders, can not be recognized.
CHAPTER 1. ANALYSIS OF RS422/485 BUS SYSTEMS

1.3 Double bus-tap 4-Wire bus
The send and receive lines are separately recorded. However this way needs two EIA-485 converter and special drivers since the recorded data bytes have to be marked with a time stamp or unique number to synchronize both data streams.

The data direction is correctly recognized but the data sequence is not always clear because of the interrupt delays, explained above. This possibility was used under MS-DOS since this operating system allowed the direct real-time processing of the interrupts.
The statements about time measurement and the tri-state condition are the same as under 1.2.

1.4 Sampling
This method needs additional independent hardware to simultaneously sample all signal lines and produce correspondent results. The advantages:

Because the sampling and evaluation is independent of the connected devices and the PC invalid tri-state conditions, wrong baudrates or UART settings are clearly detected and logged.

Furthermore the parallel sampling of all signal lines allows precise time stamps for the data lines and optional handshake lines like RTS/CTS for point to point connections. This is mandatory for examination of protocols which have to follow exact timing rules.
More: Even jittering or slightly deviating baudrates of single bus devices can be detected.

Sampling analyzers combine the advantages of protocol analyzers with the features of a digital scope and offer beside the logging of the data transfer also the physical or logical display of the line levels.
The **MSB-RS485 Analyzer** is an essential tool for analysis and optimizing of RS485/422 connections. As an autonomous device it gathers exact information about every line change with micro second precision, independent from the PC and its operating system. Equipped with a multitude of visualization tools it allows a detailed view into every RS485/422 communication and detects conditions which can be recorded by a true ‘hardware solution’ only.

The Analyzer MSB-RS485 samples all eight signals simultaneously with a sampling rate of maximum 16 MHz. Thereby all events, that means level changes on the line, are stamped with the exact time in a micro second precise resolution. As events all changes of the line levels are regarded, including the tri-state. That means a change from 0 (space) to 1 (mark) and from an inactive (tri-state) to an active logical level (and of course vice versa). The recorded data are transferred via USB to the PC at which a 500kByte cache memory serves as a buffer to avoid data losses. The recording is not limited. Date types with 64 bytes for the \( \mu \)s precise time stamps guarantee any length of recording, even with this time resolution (if you define 500000 years as unlimited). The recording depends only on the maximum allowed file length (disc capacity).

### 2.1 Advantages of a hardware solution

The MSB-RS485 Analyzer offers the capabilities of a logic analyzer, combined with a very low price. With it the disadvantages of pure software solutions are avoided by the direct evaluation of the signal changes in an independent hardware.

Analyzing solutions, based on software, depend on the not constant reaction and computing times of interrupts in the operating system. The usage of EIA-422/485 to USB converters add unpredictable delays of the USB subsystem. Furthermore the hardware input Fifos of the PCs are normally limited to 16 characters and 115200 Baud. If the interrupt handling is too slow characters will get lost because of the input buffer overflow. The resulting time stamps are the times of the interrupt execution, not the real...
CHAPTER 2. MSB-RS485 ANALYZER

time of the occurred events.

Correspondingly the chronological relationship between the send and receive data is imprecise if two EIA-485 converter are used for logging of a full duplex 4-wire connection (T+, T-, R+, R-).

Especially if using protocols with a an exact to follow timing relationship (e.g. max. pause between the single data bytes of a telegram or pause between the telegrams themselves like requested by ModBus RTU or Profibus)) you will not be sure if the measured timing is the real bus timing.

In contrast to traditional converters the MSB-RS485 analyzer also supports protocols with 9 bit data length. 9-bit values are used for certain binary protocols to differ between data and address bytes or to indicate the frame or telegram start.

By connecting the EIA-485 bus via serial interface all information about the tri-state condition get lost. The consequence is that data collisions caused by multiple simultaneously active driving senders can not be detected. The MSB-RS485 detects the tri-state level correct even if the differential signal is drawn to a certain rest level (Idle, Pull up, pull down resistors).

The MSB-RS485 analyzer and pure software solutions in comparison:

<table>
<thead>
<tr>
<th>Feature</th>
<th>MSB-RS485</th>
<th>Software solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detects invalid levels (tristate)</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Any Baudrate 1 Baud to 1 MBaud</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Real time stamps</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Time resolution 1 micro second</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Display of the real level changes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Automatic detection of baudrate and protocol</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Supports protocols with 9 Bit data word length</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Correct time relationships between data and control lines</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Detects baudrate jitter and wrong bit times</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

2.2 Innovative software concept

Already while recording any section of the data transfer can be investigated. This includes the physical display of the signals in different time resolutions (scope display) as well as the display of the transferred data bytes.
2.3 APPLICATION FIELDS

The software of the analyzer is designed as a Multi-Process Architecture. While the control program controls the recording, the transferred data can already be checked, evaluated and searched in any number of analysis windows in different time resolutions.

In this way the logical signal of a 2-wire bus or of both data lines can be followed and at the same time an earlier section can be watched in a higher resolution. That is also possible in all other analysis windows and allows the comparison of transferred data at different moments.

Extensive search mechanisms allow the search for defined data sequences, where also complex search requests are possible. That is done via regular expressions and could be:

All data strings which start with an 'A' and end with 'Z'. Also a search for defined levels or level changes can be done. The MSB-RS485 analyzer is supplied via USB from the PC and is appropriate for mobile operation when using a Laptop or notebook.

Application specific protocols and telegrams can be displayed with the help of the integrated script language LUA.


2.3 Application fields

The analyzer MSB-RS485 finds its use for logging and evaluating of asynchronous data transmissions based on the EIA-422/485 specifications. This includes 2-wire, 4-wire and EIA-422 full duplex connections inclusive handshake lines.

The high chronological time resolution of one microsecond allows a precise timing analysis of the watched communication and detailed information about the reaction times in EIA-422/485 protocols.

By the active sampling of all wire pairs bus conflicts, evoked by faulty implementation of the tri-state condition, can be clearly detected. This is also possible if the data bus is drawn to the usual idle level of 200mV by pull up, pull down resistors.

Typical application are:

- Industrial interface applications
- Fabric automation
- Industrial networks
- Building services engineering
- Maschine controlling and automation technics.
- Embedded devices
The MSB-RS485 analyzer offers all necessary features for an effective examination of EIA-422/485 connections. In particular for debugging, recording, tests and 'reverse engineering'.

- **Simultaneous sampling of all lines by external hardware**: Exact measurement of all EIA-422/485 signals with a precision of 1 µsec and a maximum sampling rate of 16 MHz, independent from the PC operating system. No wrong time stamps or event sequences due to delayed or not answered system interrupts (software solutions).

- **Any baudrate with FLEXUART**: High-precise set and measurement of standard and non-standard baudrates in the range from 1 Baud up to 1 MBaud with a resolution of 0.1% of value. Recording and analysis with any, even unusual, baudrates. Detection of asynchronous or drifting baudrates between sender and receiver.

- **Automatic protocol detection**: Simple check and analysis of any communication with unknown connection parameters.

- **Supports 9 Bit Data words**: Recording and analysing also of protocols with 9 Bit data word length.

- **Scope-like display of the data lines**: Simultaneous display of the logical signals as well as the transferred data. That makes the error analysis and search easy for transmission errors, i.e. improper bit rates (jitter) or wrong data formats. Measuring of the real signals with the integrated bit ruler.

- **Segment-Analysing**: Direction specific analysis of single bus segments or bus participants and therewith isolating of erroneous send devices by transparent bus disconnection.

- **2 digital input/output channels**: Recording of two additional control lines (signals), output of the bus direction or bus state (active/inactive) for triggering of external measuring equipment.

- **Protocol template**: Define own rules how your data shall be displayed or visualize any application specific protocols.

- **Data analysis in real time**: Examination of the connection already while recording the data.

- **Detection of invalid line levels**: Detecting of open lines, invalid Tri-States and bus conflicts.
CHAPTER 3. FEATURES & BENEFITS

- **Framing, Parity, Break Detection**: Direct analysis of error conditions and the reactions of end devices thereon.
- **Pattern search with regular expressions**: Makes the search for any data sequences possible with wild card characters and time distances or pauses between data strings.
- **Integrated LevelFinder**: Finds any static level, level change or error condition. Combined with the search of defined data bytes it is a precious tool to analyze hardware protocols.
- **Integrated Lua script language**: To define, visualize, compute (check sum test) and convert the recorded data.
- **MultiView concept**: Simultaneous analysis of the recorded data at different positions with multiple tool windows. That’s a very powerful help to compare the transferred data with their logical signal level or to compare different sections of the data stream.
- **Copy And Paste**: Simple copying of recorded protocol or data sequences into other applications for further evaluation or documentation purposes.
- **Data export as CSV**: For further evaluation of the logged data in Microsoft Excel or other spread sheet programs. That makes the full toolset of these programs available for statistic examination, sorting and other computations.
- **Direct display of the data stream by green Leds**: Additional indication of the data flow, quick check for a correct data connection.
- **Future-proof by modern FPGA technology**: Integrated state of the art gate array technology allows permanent advancements and adaption to different applications. The updating is done simply at start of the software.
- **Synchronize of two analysers with mikrosecond precision**: The internal Link jack provides the user with a time synchronous recording of two different RS232/RS422/485 connections.
- **Internal memory of 512 kByte**: USB transfer buffer of 512 kB for measurement data to avoid data losses while recording at high baudrates.
- **Multi-Platform Support**: The MSB-RS485 software is delivered as 'native binary' for Microsoft Windows and Linux. No emulation, no additional libraries, no installation of .NET® or Java®.
- **Multi-Language Support**: German and English language support. The selection is done automatically according to the used operating system, but can be changed manually.
- **Multi-Process Architectur**: The splitting of different functions into different programs or processes guarantees high data security while recording and provides a better adaption to the system resources and CPU load time.
- **Compact housing with USB connector**: No additional power supply necessary. Mobile operation even with laptop.
Spezifications

General
Protocol analyzer for recording and analysis of asynchronous EIA-422/485 connections (2-wire, 4-wire, half- and fullduplex) by parallel sampling with a maximum of 16 MHz. Precise measurement of all data and bus signals with a resolution of 1µs. Decoding of the bus lines T+, T–, R+, R– including the Tri-State with any baudrate in the range from 1 Baud to 1 MBaud. Automatic detection of baudrate and protocol.

EIA-422/485 Measurement
- **Any baudrate with FlexUart**
  High-precise setting and measuring of standard and non-standard baudrates in the range from 1 Baud to 1 MBaud with a resolution of 0.1% of the set resp. measured value.
- **Data formats**
  Parameter for serial data transmission: 5 to 9 data bits, parity off, even, odd, constant 0 or constant 1.
- **Logical line state**
  Logical level (A-B): 1 (V+), 0 (V–), invalid (–0.7V < In < +0.7V)
- **Time resolution**
  All lines are exactly sampled and marked with 1µs time stamps, independent of the operating system of the PC.

EIA-422/485 connectors
- **Signal levels**
  Standard EIA-422/485 level ±0.2V to ±12 V, ESD protected inputs 12kOhm, Common Mode ±7V. Detection of the tri-state level of differential signals below ±0.7V
- **Bus Connectors**
  Connector: 2* Phoenix MC 1,5/ 6ST-3,5 with 2mm screw connectors, 6 pins each.
- **Intern connections**
  All connections from Port 1 and 2 are connected through high speed transceivers and are automatically switched in correspondence to the selected connection mode and data direction.
CHAPTER 4. SPEZIFICATIONS

Additional features

- **Auxiliary In-Outputs**
  Two additional terminals, each individually switchable for recording of external signals or for outputting of bus status signals. Input: 0-5V, Trigger level 1,65V, 25KOhm Pull down Output: 0/5V ca. 10mA

- **Cache**
  Internal cache memory of 512 kB for buffering of measuring data when recording data with high transfer rates.

- **Status LEDs**
  Leds for displaying of: red: recording status and buffer load, green: bus data flow.

Power supply

The analyzer is directly supplied from the USB cable. The consumption is about 200mA. USB Ground is the same as EIA-422/485 Ground. No external power supply necessary.

Supported Operating Systems

- **Windows**
  Windows 2000, Windows XP, Vista (32 and 64 Bit)

- **Linux**
  All Linux with kernel from 2.4.18 and installed Gtk2 libraries (are standard). In case of doubt you can test the Linux version from our download page. 32 and 64 Bit Systems.

Dimensions

- **Abmessungen**
  100mm x 50mm x 25mm (Length, width, height)

- **Weight**
  ca. 100g

Scope of delivery

- **Analyser**
  MSB-RS485 analyzer device.

- **Connection Set**
  Connection set consists of:
  2* 6-pin Phoenix screw connector
  4* termination resistors 120 Ohm if analyzer is end devcie
  4* short circuit wires for various connection variants.
  1* Screwdriver for Phoenix connectors
  1* USB Cable for connection to PC

- **Software**
  CD for Windows and Linux, Manual as online help and PDF document in German and English.
Requirements

- **Graphical display**
  Graphics board and monitor with at least 1024x768 pixel resolution and 16 bit color depth or more.

- **Disk space**
  100 MByte empty space for the software installation plus additional space for the recording files.

- **Memory**
  256 Mbyte or more

- **USB connector**
  One empty USB 1.1 or 2.0 connector (full speed).
CHAPTER 4. SPEZIFICATIONS
The MSB-Analyzer software is available for Microsoft Windows as well as for Linux. Both versions are contained on the program CD and are both offered to your system for installation. What you have to regard is mentioned in the following chapter.

The MSB-Analyzer is connected via USB to the PC and communicates through a virtual COM port. Under Microsoft Windows the respective VCOM driver is installed automatically. Linux distributions from kernel 2.4.18 already contain the right module ftdi sio, functional for the analyzer.

The software is bilingual (German and English) and can be installed even without analyzer, e.g. to evaluate earlier captured data. Or if you want to check out the capabilities of the Analyzer by examining the enclosed sample files.

Installation under Windows

Close all running applications before inserting the CD-ROM. Do not connect the MSB-Analyzer before you insert the CD-ROM. Connect the analyzer after the program installation is finished.

1 Insert the installation CD-ROM
   The IFTOOLS product installer is invoked. When it does not automatically start double click onto the My computer symbol on your desktop or open it from the start menu. Double click onto the IFTOOLS-Setup-CD icon to start the IFTOOLS product installer.

2 Selecting the product
   In the product list at the left side click on the relating analyzer (MSB-RS232 or MSB-RS485). Start the software installation inclusive the necessary driver with a single click onto installation (Version 3.2.5). Probably it takes a moment until the installer is displayed.

3 Install the software
   Proceed according to the hints on the screen. The necessary driver is automatically installed together with the operating program.
CHAPTER 5. PROGRAM INSTALLATION

5.1 Installation under Linux
Modern Linux distributions offer the same comfort for program installations as Windows. Just as under Windows this behavior has to be activated before. Your Linux system has to mount the CD as executable. If this is the case the installation runs as under Windows.

1 Insert the installation CD-ROM
The IFTOOLS product installer is invoked. Depending on the distribution you are asked if you want to activate the autorun feature. Answer with 'yes'. If the installer does not automatically start read the following chapter 'Manual installation under Linux'.

2 Selecting the product
Click on the relating analyzer (MSB-RS232 or MSB-RS485) in the selection list at the left side. Start the software installation with a click onto Installation (Version 3.2.5).

3 Install the software
Proceed according to the hints on the screen. The necessary kernel module is part of all kernel since kernel version 2.4.18 and does not have to be installed.

5.2 Manual installation under Linux
If the IFTOOLS product installer does not start after inserting of the CD please follow these steps:

1 Open a console
2 Copy the installation file onto your desktop
   Enter the following command:

   cp PATH_TO_CDROM/programs/msb/msb-3.2.5-linux-installer.run ~/Desktop

3 Make the installation file executable
   by setting the executable flag with:

   chmod +x ~/Desktop/msb-3.2.5-linux-installer.run

4 Start the installation file
   via mouse click (double click). Alternatively you can also run the installer in the text mode in case of the graphical installer did not start:

   sudo ~/Desktop/msb-3.2.5-linux-installer.run --mode text

5.3 Installation for all users
The installation needs only root rights if you like to install the software for all users (system wide). For instance: if the relating software files should be wrote to /opt or /usr/local. In this case start the installation file under KDE with:

   kdesu ~/Desktop/msb-3.2.5-linux-installer.run
5.4 PROGRAM UPDATES

Under Gnome either with:

gnomesu ~/Desktop/msb-3.2.5-linux-installer.run

or

gksu -k ~/Desktop/msb-3.2.5-linux-installer.run

Alternatively you can also start the installer in the text mode. This makes sense if the graphical installer did not start or you want to execute it simply by root or via sudo command:

sudo ~/Desktop/msb-3.2.5-linux-installer.run --mode text

5.4 Program Updates

IFTOOLS issues software updates in irregular intervals with new features and improvements. These updates are free of charge can be loaded from the following address https://iftools.com/msb-rs485/download.en.php

The update are complete program versions which also contain in the Windows version the current driver. Updates can be installed in parallel to your current MSB-Analyzer program version. Windows user only have to execute the update file.

Under Linux it is necessary to make the file executable and then to start it like described above.
How do I connect the analyzer to my PC? How do I insert it into the connection I want to monitor? What is the meaning of the LEDs? These and other questions will be answered in the following chapter.

The EIA-422/485 specification do not specify a certain association to the connector pins. Therefore the type of connection is depending on the application. To allow an easy adaption of the analyzer to different bus systems the MSB-RS485 is equipped with two 6-pin sockets for Phoenix connectors with screw terminals. An appropriate connection kit including plug connectors, termination resistors, wires and screw driver is enclosed.

The connector marked with PC USB is used to connect the analyzer with an unassigned USB port of your PC where your analyzer software is or shall be installed. The energy support is made through the USB cable so that you do not have to use an extra power supply. In this way you can easily use the analyzer together with a laptop in mobile operation, the current consumption is about 250 mA.

6.1 Definition of the Signal lines

Unfortunately the naming of the both twisted lines of an EIA-422/485 connection is not consistent.

The EIA-485 specification defines lineA as the inverted signal or ‘–’ terminal and line B as the not inverted or ‘+’ terminal.

This is in conflict with the A/B naming of a number of transceiver manufacturers. Details can be found at: http://en.wikipedia.org/wiki/Rs485 Even if their naming of the signals (A+,B–) is in contrast to the standard it is wide-spread.
CHAPTER 6. CONNECTION OF THE ANALYZER

To avoid more confusion the terminals of the analyzer MSB-RS485 are simply marked with ‘+’ and ‘−’, which correspondents to the not inverted and the inverted signal of a EIA-RS485 wire pair.

So connect the not inverted bus line with the ’+’ input and the inverted line with the ‘−’ input of the analyzer.

The following table lists some of the most used line names:

<table>
<thead>
<tr>
<th>EIA-485</th>
<th>MSB-RS485</th>
<th>Customize naming</th>
</tr>
</thead>
<tbody>
<tr>
<td>A−</td>
<td>−</td>
<td>TX−, TX−/RX−, D−, Data−, (G)</td>
</tr>
<tr>
<td>B+</td>
<td>+</td>
<td>TX+, TX+/RX+, D+, Data+, (Y)</td>
</tr>
<tr>
<td>A−</td>
<td>−</td>
<td>RX−(^1)</td>
</tr>
<tr>
<td>B+</td>
<td>+</td>
<td>RX+(^2)</td>
</tr>
</tbody>
</table>

\(^{1,2}\) only for full duplex 4-wire systems.

6.2 Internal Signal Processing

The MSB-RS485 Analyzer has four differential inputs CH1 to CH4 which are simultaneously sampled and recorded. For every channel the physical signal is available as display of the logical states.

Two integrated UARTs handle the decoding of the serial data stream into single data bytes. These UARTs are automatically connected to two of the four differential inputs CH1 to CH4. By this variable connecting a number of connection types and analysis functions can be implemented.

In this way the MSB-RS485 Analyzer allows besides a plain tapping of the bus lines also to feed the bus through the analyzer. This is done by splitting the bus into two parts whose ends are connected to CH1 and CH2. Both UARTS care for the independent decoding of both bus segments while the bus data flows bidirectionally through the analyzer.

A combined mode where one bus is split and a second one is tapped allows the filtering of single bus devices even in full duplex 4-wire connections.

Two additionally generated tri-state signals support this way of analysis by delivering information about the validity and direction of the bus data and the common (fed through) data signal. Both tri-state signals are listed in the following table:

<table>
<thead>
<tr>
<th>Notation</th>
<th>Mark (1)</th>
<th>Space (0)</th>
<th>invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus-Dir (Data direction)</td>
<td>CH2 → CH1</td>
<td>CH1 → CH2</td>
<td>Bus undriven</td>
</tr>
<tr>
<td>Bus-Signal fed through</td>
<td>Bus-(Data)-Signal</td>
<td>Bank through</td>
<td>no data</td>
</tr>
</tbody>
</table>

20
6.3 Digital In/Outputs

The MSB-RS485 Analyzer offers two additional digital IO-channels which can be optionally used as auxiliary inputs for recording of logic signals or as outputs for indication of status information. The latter allows the output of the bus data direction and the validity of the bus either of single segments or of the fed through bus lines. The following settings are possible, separately for both IO-channels:

<table>
<thead>
<tr>
<th>IO-Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Input with pull down resistor</td>
</tr>
<tr>
<td>Output</td>
<td>Output static 0</td>
</tr>
<tr>
<td>Output</td>
<td>Output static 1</td>
</tr>
<tr>
<td>Output</td>
<td>Bus data direction between CH1 and CH2 (segment analysis): 0 : CH1 → CH2, 1 : CH2 → CH1</td>
</tr>
<tr>
<td>Output</td>
<td>Activity of the bus through CH1 and CH2 (segment analysis): 0: inactive (Tri-state), 1: active</td>
</tr>
<tr>
<td>Output</td>
<td>Bus activity of CH1: 1: active, 0: inactive (tri-state)</td>
</tr>
<tr>
<td>Output</td>
<td>Bus activity of CH2: 1: active, 0: inactive (tri-state)</td>
</tr>
<tr>
<td>Output</td>
<td>Bus activity of CH3: 1: active, 0: inactive (tri-state)</td>
</tr>
<tr>
<td>Output</td>
<td>Bus activity of CH4: 1: active, 0: inactive (tri-state)</td>
</tr>
</tbody>
</table>

6.4 Bus Termination and Tapping

EIA-422 are usually designed as full duplex point-to-point connections, sometimes together with additional line pairs for hardware handshake. Whereas a EIA-485 connection is generally implemented as a multi-master capable 2-wire system (half duplex) or as a full duplex 4-wire system based on a master-slave configuration.

All serial bus systems have in common that the signals allow no inferences on direction and source of the data. The MSB-RS485 analyzer offers the possibility to split the inspected connection into two segments and/or bus participants. With this so called ‘segment-analysis’ the data of a single (or several) bus devices can purposefully be monitored independently of the rest of the bus and can be directly assigned without regarding the protocol.

The MSB-RS485 Analyzer has 4 differential Inputs and 2 integrated FLEXUART’s available. These Uarts are connected to the inputs according to the selected connection mode and process the decoding of the serial data stream into single data bytes.

To make the connection and assignment as simple as possible the analyzer offers a selection of various combinations for bus system and taping (the connection mode or short wiring), which are in accordance with the following variants.
CHAPTER 6. CONNECTION OF THE ANALYZER

6.5 Tapping 2-wire system

For connecting a 2-wire system the bus resp. the wire pair is simply connected to the terminals at Port1, CH1, CH2 and the pins of Port2 remain unconnected. Connect the inverted line of the bus to the ‘–’ input of CH1 and the not inverted line to the terminal ‘+’ of CH1.

The name 2-wire system implies the use of the line pair only but the correct treatment of ground is mandatory. If the inspected bus has a signal ground line connect it to the correspondingly marked terminal at port 1.

This simple method of tapping does not need additional terminating resistors.

In this connection mode the analyzer records all transmitted data independently of source and direction. To get more information about the sender of the data you have to know the corresponding used protocol.

6.6 Segment Analysis 2-wire system

In contrast to the plain tapping of the data signal the MSB-RS485 is inserted into the bus. In doing so the bus is split into one segment on the left side and one segment on the right side of the analyzer.

This kind of wiring is more complex but it has some advantages over the plain tapping.

The analyzer becomes the interface between any two bus segments. The data, flowing through this interface, are collected together with their direction so that they can be clearly assigned to the corresponding segment. If the segment consists of only one bus device the data sent from this device can be easily assigned to this device independently from the remaining bus communication - even without having to know the used protocol.

Split the bus at the required point and connect the wire pair of the first segment to the terminals of the first analyzer channel Port1, CH1 (CH1+, CH1–) The second bus segment is connected to the second analyzer channel Port1, CH2 (CH2+, CH2–).

The bus is now split into two segments. Please note that you possibly have to terminate the new bus ends. In this case you can directly connect the resistors to the terminals of CH1, CH2.

The same applies for pull up/down resistors. By the splitting the bus one segment is now without these resistors for setting the idle level. They are normally attached at one bus end only and their values are system dependent. Please
check if they have to be added.

The analyzer records the data of both segments independently while the data of both directions are passed fully transparent. Data of the first segment are marked as coming from channel 1 with the internal name ‘A’, data of the second segment are marked as coming from channel 2 with the internal name ‘B’. A and B are initially two different data sources within the analyzer and are assigned to the physical input channels according to the selected wiring.

6.7 Tapping 4-wire system

In point-to-point connections like used for EIA-422 transmissions over long distances (EIA-232 replacements) only two bus devices are available communicating over two different send and receive channels. A double tapping is sufficient and guarantees the correct recording of the data direction.

This kind of wiring is also used for analysis of full duplex EIA-485 connections (as DIN-Messbus, Master-Slave) if you do not need to watch a special bus device singularly and if you can assign the data to the bus participants by evaluating the protocol.

Connect the send line pair to the terminals CH1+, CH1- of Port 1 and the receive line pair to CH2+, CH2- of Port 1.

All data from channel 1 are named as A and all data from channel 2 are named B.
CHAPTER 6. CONNECTION OF THE ANALYZER

6.8 Segment Analyse 4-wire system

Bus systems with full duplex 4-wire connection (Master-Slave bus, Din-Messbus) also use separate send and receive channels. While the master is connected as sender (Masterbus) to the receivers (Slaves) these return their answers on the second channel (Slavebus) to the master. To monitor the send data from the master a single tapping of the master bus is sufficient.

In contrast the slaves share one channel to send their data back to the master. With the help of the segment analysis a singular device can intentionally be separated and its communication with the master monitored without regarding the other devices.

Like for the 2-wire segment analysis you have to split the slave bus at the appropriate point.

Both segments of the slave bus have to be connected to CH1 and CH2 of Port 1. Please note that possibly the ends have to be terminated as explained in ??.

Additional you have to consider about existing pullup resistance. The tapping of the master bus is connected to CH3 at Port 2. A termination is not necessary.

In this configuration all data sent by the slaves on both segments are gathered at CH1 and CH2 and internally named as A. The data from the master, received at CH3 are named B.

The assignment of the data A to a segment or a certain bus device (CH1 or CH2) is done via an additional internally generated bus direction signal. This signal can be evaluated to visualize and differing the sender, resp. the active sending segment.

6.9 Signal assignment

The MSB-RS485 Analyzer has 10 data display channels for the visualization of the recorded information.

Two data channels are used for the display of both bus data A and B which are generated by the UARTs.

Further 8 logical channels are used to display the tri-state levels of the differential signal inputs CH1 to CH4. They also display the bus activity and data direction and the two digital auxiliary inputs. The assignment of the display channels varies according to the selected connection mode (wiring).
6.10. LIGHTMENT ELEMENTS LEDs

Gray entries indicate signals which are recorded by the analyzer but are not used in the selected connection mode. However they can be used for recording of additional signals like handshake lines. The following table shows the available signal information. You do not have to use this table for your data evaluation, the analyzer software automatically names the display channels depending on the selected wiring.

<table>
<thead>
<tr>
<th>Display channel</th>
<th>2-wire Tap</th>
<th>2-wire Seg</th>
<th>4-wire Tap</th>
<th>4-wire Seg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data A (Data A)</td>
<td>Data from Bus at CH1</td>
<td>Data from Bus segment at CH1</td>
<td>Data from Bus at CH1</td>
<td>Data from Bus segments at CH1 + CH2</td>
</tr>
<tr>
<td>Data B (Data B)</td>
<td>Data from Bus at CH2</td>
<td>Data from Bus segment at CH2</td>
<td>Data from Bus at CH2</td>
<td>Data from Masterbus at CH3</td>
</tr>
<tr>
<td>Signal 1 (CH1)</td>
<td>Logic signal at CH1</td>
<td>Logic signal at CH1</td>
<td>Logic signal at CH1</td>
<td>Logic signal at CH1</td>
</tr>
<tr>
<td>Signal 2 (CH2)</td>
<td>Logic signal at CH2</td>
<td>Logic signal at CH2</td>
<td>Logic signal at CH2</td>
<td>Logic signal at CH2</td>
</tr>
<tr>
<td>Signal 3 (CH3)</td>
<td>Logic signal at CH3</td>
<td>Logic signal at CH3</td>
<td>Logic signal at CH3</td>
<td>Logic signal at CH3</td>
</tr>
<tr>
<td>Signal 4 (CH4)</td>
<td>Logic signal at CH4</td>
<td>Logic signal at CH4</td>
<td>Logic signal at CH4</td>
<td>Logic signal at CH4</td>
</tr>
<tr>
<td>Signal 5 (BDIR)</td>
<td>unused</td>
<td>Data direction CH1 =&gt; CH2</td>
<td>unused</td>
<td>Data direction CH1 =&gt; CH2</td>
</tr>
<tr>
<td>Signal 6 (BSIG)</td>
<td>unused</td>
<td>Logic signal CH1 + CH2</td>
<td>unused</td>
<td>Logic signal CH1 + CH2</td>
</tr>
<tr>
<td>Signal 7 (IO1)</td>
<td>IO1</td>
<td>IO1</td>
<td>IO1</td>
<td>IO1</td>
</tr>
<tr>
<td>Signal 8 (IO2)</td>
<td>IO2</td>
<td>IO2</td>
<td>IO2</td>
<td>IO2</td>
</tr>
</tbody>
</table>

*The short notation as displayed in the control program in parenthesis

6.10 Lightment elements LEDs

The MSB-RS485 analyzer has four LEDs to display its operating status and the status of the data recording. They are located between both 6-pin Phoenix jacks (Port 1 and 2) and are marked with Red1, Red2, Green1 and Green2. The red LEDs serve for the signaling of the recording status and the internal buffer load while the green LEDs show the state of the connection and data flow.

Green LEDs

The green LEDs show the bus states. LED1 is assigned to data channel A, LED2 is assigned to data channel B.
CHAPTER 6. CONNECTION OF THE ANALYZER

1. **LED is off:**
   Undefined bus state, drivers in tri state or lines are twisted (interchanged polarity).

2. **LED is on:**
   Active bus, correct connection.

3. **LED flickers, mostly on:**
   Active bus with data transfer, corresponds to the normal EIA-422 operation.

4. **LED flickers, only short pulses:**
   Active bus with data transfer, but wrong polarity or EIA-485 bus with rest (tri-state) conditions. The latter is the normal condition for EIA-485.

**Red LEDs**
The red LEDs serve as a display for the operating condition of the MSB-RS485.

1. **Both LEDs permanently on:**
   The MSB-RS485 was not yet initialized by the PC. Data is not fed through (segment analysis).

2. **Both LEDs blink alternatively:**
   The MSB-RS485 was initialized but is not yet active, that means no recording was started.

3. **Red 1 is on, Red 2 is off:**
   Recording is active, the PC logs all interface events.

4. **Red 1 is on, Red 2 is blinking:**
   The loading of the internal data buffer is displayed. The filling degrees $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$ are indicated by different length of pauses between the blinks. The more full the memory the shorter the pauses.

5. **Both LEDs blink at the same time:**
   The buffer memory is full and recording data gets lost. The duration of the data loss is recorded.
Program start

Selection of the interesting events and the recording mode (continuous or Fifo-mode). Store or load recorded data. All these functions are controlled through the main program.

The Firmware of the MSB-RS485 is not installed in the device but has to be loaded after powering up. This takes place only once. As long as the device is powered from the USB connection the firmware is kept active. Therefore the loader appears as soon as you double click onto the MSB-RS485 desktop icon.

The firmware loader automatically detects if an analyzer is connected and if the firmware is already loaded or has to be transferred. After the MSB-RS485 was identified and the firmware was successfully transferred (observable at the progress bar in the lower part of the dialog window) the MSB-RS485 control program starts automatically.

If there are more than one analyzer connected with your PC the Loader will show you a selection list of all detected devices.

If no MSB-RS485 was found, even though it is connected to your PC, read the hints for Trouble shooting (Windows D.1, Linux E.1) in the appendix.

If you simply forgot to connect the MSB-RS485 to your PC, just connect it now and click on the `Search` button to update the list of the detected analyzers.

You also can work without the MSB-RS485, e.g. to evaluate recorded data or to work through the Tutorial. Because the program cannot detect the kind of analyzer (MSB-RS232 or MSB-RS485) in offline mode, you have to choose the wanted type from the selection list.

The MSB-RS485 software uses a multi process architecture. That means, that the program does not run in a single window but starts special tools according to the different tasks.

In view of this feature the start of a small control panel may seem poor. But the software shall not confuse you with not necessary windows, multiline toolbars and nested menus. Instead it shall offer you a set of easy to operate tools which are appropriate for your application.
CHAPTER 7. PROGRAM START

The MSB-RS485 control program is your cockpit to supervise the analyzer. With it you start a recording, save or load records or projects and open the different analysis tools to examine the recorded data.

7.1 User Interface

A Recording control: Easy start, pause or stop a recording.
B Protocol Scanner: Automatically detecting of (unknown) baudrate and protocol.
C Settings: All necessary settings for the recording available with one click. No long winded navigation through complex menu items.
D Recording control: Clear display of the recorded data/events, settings, record time and record state. More information with a right mouse click.
E Analysis tools: Start the proper view with it’s last settings.

7.2 Select kind of connection

The MSB-RS485 Analyzer has four differential inputs which are used as data or signal loads, according to the selected wiring. Before starting the first recording you have to inform the program about how the analyzer is connected to the EIA-422/485 bus. The only thing you have to do is to select the connection mode in the setup menu. Also see section ??.

7.3 The first start

Imagine the control program as a kind of recorder. To record a data connection you need at first not more than the communication parameters (baudrate, protocol). All further tools to display the transferred data or their physical level can be optionally opened or closed without influencing the recording. The MSB-RS485 analyzer contains a so called FLEXUART core, an specially developed decoder hardware, for the serial data transmission which allows not only the measuring of the baudrate but also the detection of the used protocol. The only thing you have to do is to click onto the button to open the protocol scanner dialog.

Automatical protocol scan

A correct detection of the connection parameters implies an appropriate transmission. It is sufficient to receive data at Port 1 or Port 2. A recording does not
have to be started.
Start the automatic detection with a click onto the 'Start' button of the scan dialog.
After starting the protocol detection the MSB-RS485 analyzer at first measures the baud rate of the data at Port 1 or Port 2. In the further process the data stream is analyzed and the correct number of data bits and parity is evaluated. The complete process lasts only a few seconds and can be repeated at any time by clicking the start button. As soon as the parameters are correctly detected you can adopt the found settings with the button 'Use scan'.

**Manual Protocol setup**
If you already know the baudrate and protocol you can directly set them in the settings dialog. The MSB-RS485 software memorizes these settings so that you do not have to enter them any more.
All relevant parameters for the recording can be reached with one click onto the button. The most important settings are immediately shown. Settings for advanced users or more specific applications can be reached in the page selection of the dialog and are discussed in chapter 7.5.

**Start/stop a recording**
As soon as you determined the communication parameters one click onto the record key starts the recording. The record key starts to glow red and the active symbol in the display starts to turn.
You can halt the recording at any time by clicking on the Pause button. The recording of occurring events is discontinued until you continue the recording by another click on the Pause button.
If you want to end the recording press the Stop button. The recording is not deleted but finally stopped. If you want to start a new recording the program reminds you to first save the recorded data of the last session.
The MSB-RS485 analyzer software allows to examine the data even while recording. In this respect you will stop the recording not before you want to start a different one or to save it for a later examination.

**Test the software without analyser**
You can test the program even without a connected analyzer. Simply load a sample recording with Ctrl+O from the example folder in the installation directory.

---

**7.4 Status display**
The central part of the control program is the display of the current recording. To indicate all information clearly arranged the display can be operated in three different modes. The selection is simply made by a right mouse click onto the display.
In addition to the connection data also the available recording capacity is displayed. The necessary disc capacity is depending on the data traffic and the events, selected for recording. An estimation of the space consumption shows the marker (dot) on the horizontal dividing rule. It indicates the empty space (right hand of the marker) in relation to the total available space.
CHAPTER 7. PROGRAM START

The default directory is the usual Windows or Linux temporary directory. You can change it by calling the program call with an addition parameter (see also section 7.16 Additional program arguments). The speed of the moving marker is depending on the quantity of the occurring (and selected) events as well as on the size of the empty space in the used temporary directory.

Display I
It contains in the lower part the set connection parameters (Set: baud rate, data length, parity, stop bits), the measured baudrate (Mete:) and the for logging activated input channels resp. signals. Additionally the quantity of the transmitted data bytes are displayed depending on the direction, bus or bus segment. The time display corresponds to the time point of the last occurred event, relative to the start of the recording.

Display II
The second display informs you about the total sum of transmitted data bytes and about the quantity of the remaining events as level changes and alternations of the bus direction a.s.o. In this display mode the time indication is not related to the last occurred event. The time is the running recording time, independent of any events.

Display III
The third display alternative serves as a control display for the connection of the Analyser to the controlling PC. The analyzer MSB-RS485 is controlled and supplied directly through the USB connection. The USB connection is done via a virtual COM port and is shown as a normal COM connection under Windows. For Linux it is typically /dev/ttyUSBx. The fields Gaps and Fifo indicated if the analyser has recorded or sent more data than the PC could handle (as a result of a too slow connection). Normally both values should be zero. Other values indicate that either the internal buffer of the analyser (Gaps) resp. the Fifo of the serial ports (UART) could not handle the data rate (overflow, lost data). In this case you should reduce the number
of recorded events.
All signal and data lines can be individually enabled and disabled for logging. Inactive are shown as lines.
In the examples above the level changes of the input channels CH3 and CH4 are not logged.

7.5 Config a recording
In the program defaults all signals are activated and prenamed, the connection parameters are set to 8N1, 115200Baud.
These defaults probably do not correspond to the connection which shall be monitored. Before you start a recording these parameters have to be changed according to the connection settings.
You get access to these parameters through Settings → Configure Control Program... or by click onto the set symbol on the left side of the display.
Because some settings directly influence the logging they are deactivated while a logging session is running. The respective setup menus are displayed in gray.
The setup dialog is divided into the following sections:
- Connection
- Bus Wiring
- Signals
- Record
- Auto Save
- General

The recording of the bus line(s) takes place as data bytes and additional as the recording of the logical signal levels.

Connection
These settings concern the analyzed connection, not the connection between the analyser and the PC! They are mainly important if you want to record the transmitted data bytes.
This includes the baudrate and also the numbers of the databits and the parity setting. The count of stopbits doesn’t matter. The analyser takes care about the stopbits automatically.
If you are unsure about the connection parameters just let the integrated protocol scanner detecting the right settings for you.

Additional to the standard baudrates, the MSB-RS485 analyser also supports any rate in a wide range of 1 Baud to 1 Mbaud.
To use an more special rate like 123456, just input it in the baudrate field. Or select a standard baudrate by click on the button. Valid entries are 1 Baud to 1 Mbaud.

Besides the common data lengths the MSB-Analyser also allows you the record 9 bit data transmissions, which are often used for address decoding or to mark a telegram start.
Please note that a 9 bit data length excludes any other parity except for none.
CHAPTER 7. PROGRAM START

By default, the display of the control program shows the measured baudrate were alternatively the detected data at CH1 or CH2 are used for the evaluation.

**Bus wiring**

Bus wiring It is decisive for the signal assignment how the MSB-RS485 analyzer is connected to the examined bus. Since the connection can not be automatically determined you have to inform the analyzer about the chosen wiring.

Depending on the connection setting only a part of the differential inputs are used. The unused inputs and the digital auxiliary inputs can be used freely in your application. Not available signals are marked with 'disabled'. The setup menu shows a respective graphic and a table for the signal assignments. The signal names are automatically assigned. In the setup menu of the signal names (32) you can change this behavior and assign own names. The signal assignment is separated into:

- **2 Data channels**
  
  These channels contain the decoded data of both UARTs. 9 bit data are supported and displayed as well as occurred transmission errors like parity and framing.
  
  Depending on the selected wiring a data channel can also contain the data of several differential inputs. In this case the inputs are explicitly listed with a '+' For example CH1+CH2.

- **8 Logical signal channels**
  
  All differential inputs CH1 to CH4 are additionally displayed as a 'plain' logic signal independent of their data decoding. The four inputs are directly shown in the four signal channels 1 to 4.
  
  The signal channels 5 and 6 have a special status. For the selected segment analysis they display additional internally generated logic signals. These are the bus-direction between CH1 ↔ CH2 (signal channel 5) and the combination of the logic signals from both inputs CH1 and CH2 (Signal channel 6).
  
  The signal channels 7 and 8 are assigned to the two auxiliary channe

- **Digital auxiliary inputs/outputs**
  
  As the name implies both channels can be individually operated as input or output. That allows the recording of additional signals or the output of the current bus state (bus validity, activity or direction). Reasonable if you need a signal to trigger external measuring equipment.
  
  By default both terminals are set to open inputs with pull down resistor. A detailed description of the possible settings can be found in chapter ??.

**Signals**

You can select and rename each of the sampled input channels or signals separately. The latter one can be done even with a running recording.

The chosen bus connection mode specifies reasonable signal names (default bus connection).

It is up to you which one you choose or if you define own names as 'User defined'. In this case set the signal naming to 'User defined' and enter the new names for each signal.

The names Sig1 to Sig8 are used as place holder. Every name can consist of
7.5. CONFIG A RECORDING

a maximum of 7 characters, allowed are: all digits and letters, the underscore, colon, and full stop.
The modified signal names are automatically adopted by the control program and all analysis windows.

You can individually enable or disable the line events (signal alternations) to be monitored by the Analyser by setting or removing the checkmark beside the relating signal. As default all level changes detected on the input channels CH1 to CH4, both auxiliary digital inputs and the decoded results of the two UARTs are switched on.
Please note, that the selected signal names are shown. If you have chosen 'User defined' but haven’t entered names then here appear - no names (blank)!

The decoding of data by the UARTs is done independently of the level change recording. If you need the data only but not the logical signal you can deactivate the recording of the channels CH1 to CH4 because each level change is stored as an additional event and strikingly increases the quantity of the recorded data.

Please keep in mind that the more unnecessary events you admit the more (needless?) data is stored onto your hard disk.

Record mode
For troubleshooting of serial connections you often get the problem that you do not know when the error occurs but you need a sufficient big quantity of data to get a statement about potential reasons for the fault.
Of course you can run the logging up to the occurrence of the fault. But this can cause a rapidly increasing amount of data. With 115200 Baud and recording of all level changes this can mean 2MBytes data per second!
Therefore the analyzer supports 2 modes of logging:

1 Continuous recording:
In the continuous mode all occurring events are stored until the recording is stopped. This mode is appropriate if you want to watch and analyze the data stream already while recording.

2 Time loop with Fifo mode:
In the Fifo Mode a certain amount of data of the last (before stop of recording) occurred events is stored. The amount can be defined by setting the maximum size of the recorded events (1000...1000000 events) or by setting a time limit (10...600 seconds).
This behaviour quasi corresponds to an analogue endless tape (used with observation cameras). With this tape always the last time, defined by the tape length, is recorded. In this case you can define the 'tape length' in a given range.
Please note that in the Fifo mode no analysis tool can be used while recording. The reason is that the tools need a random access to the recorded data which is not possible in the Fifo mode. In this mode data is always overwritten from the beginning of the buffers. As the Fifo mode is normally used for recording with later analysis this behaviour is not necessarily a disadvantage. As soon as you stop the recording all recorded data are normalized. That means that they are sorted according to their time stamps and can then be analyzed as usual.

**Time synchron recording**

In the program defaults each MSB-Analyzer works autonomously unless it received so called synchron impulses on its MSB-Link jack from a connected 'Master'. If you like to record two independent connections at the same time, for instance a RS232 and RS485 port of a level converter, you have to choose one of the analysers as the record 'Master'.

You can see the current analyser status in the display. A 'Master' above the running record time indicates that the device works as the master, a 'Slave' means that the analyser is linked as the slave. In the latter case all settings are disabled since there is only one Master allowed.

You can use the 'Flash connected analyser' if you are in doubt which one you are currently setting.

**Adapt the record date and time**

Sometimes you want to adjust the date and start time of an analysing record perhaps if you examine the record in another time zone or need to adapt it to the date and time of a second comparing record.

To do so, just click the 'Adapt now' button and enter the new date and start time.

Close the dialog with 'Ok'. The Views automatically refresh their display.

You can always switch back to the original record date in the same dialog. The new date/time doesn’t change the original record. The software only uses the new value as an offset added to the initial record time and date.

**Save new record date**

The modification of the record date and time doesn’t alter the record file. The new date/time is only temporary for the current session. If you want to store the modifications permanently you have to save the record.
7.6. THE ANALYSIS TOOLS

Autosave
The amount of data can rapidly increase during a record. Therefore the pro-
gram only stores the data if the user explicitly want to save it in a file.
But there are conditions which require the storage of the record. For instance:
If you like to make sequent records for a later analysis or if an analyser in slave
mode isn’t accessible.
For both cases you can preset an automatical storage of the record. The stor-
age always takes place:

1. After stop of a synchronous record
2. After each stop of a record

The place and folder of the saved files are freely selectable. The program cre-
ates a unique file name according to the serial number and the record start
date-time which prevents to overwrite existing files. But you can add an addi-
tional prefix for a better identification or classification.

General
This page is intended for general settings. Among other things you can switch
of the security question for not yet stored data and/or suppress the display of
single views in the task bar or the task changer. The latter is possible for Win-
dows only. Linux desktops usually group the program views within one taskbar
entry.

As a special feature the software offers you to synchronize the Views of two
running MSB-Analyzer programs with each other. This comes in handy if you like
to compare and analyse two synchronous records. All views of boths records
interact as it used to be in a single recording.
For this case you have to allow the external synchronisation first.

7.6 The analysis tools
The control program solely makes the collected data available. The actual
Display and analysis of the data is done by analysis tools - separate program
modules to visualize the data at different time points and in different display
modes.
You can open any number of analysis windows by either using the below men-
tioned short commands or by clicking on one of the quick start buttons at the
right side of the display.

A (P.49) Virtual Ledtester: Das virtual counterpart of a real ledtester.
B (P.51) Data View: Data dump of the transmitted data with special
search features.
CHAPTER 7. PROGRAM START

C (P.65) Event View: All line changes in a clear look, search for line modifications.

D (P.77) Protocol View: Display any protocol with your own definition.

E (P.99) Signal View: Digital Scope like view of all lines.

7.7 Save a recording

Independend of the status of the recording (aktiv, paused or stopped) you can save the data, collected so far, in a file. Press the keys Ctrl+S or select in the file menu the entry Save→Save recording. In the opening dialog you can enter a new file name (The extention .msblog is automatically added) or you can overwrite an already existing recording. This file contains all information about the selected and recorded events and data bytes. Settings of the control program and opened analysis windows are separately stored as a project file.

Every time you save a recording the choosen file is stored in the list of last opened recording files and can be loaded at any time. More information can be found in chapter ‘Last opened Recordings and Projects’.

Save a special section

To save any section of the recorded data use the event monitor and mark the interesting range. To save the transmitted data bytes only or a part of it use the data monitor and mark the interesting range.

7.8 Save a session as a project

A session contains the current state of the opened analyser program. This includes all current settings and views which represent the program on the screen. That means that beside the connection parameters also position, size and content of all opened analysis tools and all the marked regions are combined into the session.

You can save the session at any time by pressing: Ctrl+Shift+S or by selecting Save→Save project in the file menu.

When a session is saved also the data, recorded until this time, are saved in a separate file with the same project name, but with a different extention.

Separate files for project and record

Project files always have the extention *.msbprj, the recorded data files the extention *.msblog.

Accordingly a record data file is also loaded (if available) when the project file is opened. With it you have all informations you need to resume an analysis of recorded data at exactly that point where you paused or finished the examination before.
7.9. OPEN AN EARLIER RECORDING

Saved projects are also managed in the list of last opened project files, see 'Last opened Recordings and Projects'. Each session can saved as a independent project template. For this purpose clear all recorded data by New → New record in the file menu or press Ctrl+N. Afterwards save the session under a name of your choice.

7.9 Open an earlier recording

A devision between project files and record files was intentionally made. The reason is that you can load an earlier data recording into your current project without loosing your current settings. Press Ctrl+O or click on Open → Open Record in the file menu to load the data into your current session. Please note that this can be done only if no recording is running and that your recorded data are overwritten.

7.10 Open an earlier session (project)

Press Ctrl+Shift+O or click on Open → Open project in the file menu to open a saved session. The control program loads the associated recording, places the analysis tools and makes the corresponding settings. In short it restores the program state as it was at the moment of saving the session.

7.11 Last opened recordings and projects

As mentioned earlier all saved recordings and sessions (Projects) are listed in two separate lists. You get quick access to the files you used last. The lists contain the names in sequential order so that the newest files are on top. All files are listed with full path to clearly identify them.

Click on 'Last opened Recordings' in the file menu and select the appropriate one. This is the same like open a recording with the open dialog but is much faster because you do not have to move through the different menus and directory trees. If the chosen file is no more available, i.e. because you deleted it, you are asked by the program if the file shall be removed from the list. If the file really does no longer exist you can answer with 'yes'. But if the file is on a data medium that is only temporarily unavailable you can answer with 'no' and the entry is kept.

7.12 Drag and drop

You can load any record or project file by simply drag and drop it into the application. Just drag the wanted file from your file browser or desktop into the control program. This will replace the current session with the data of the new dragged file. In case of a project file also all stored session settings are restored including all Views. Please note that drag and drop isn’t possible during an active recording.
CHAPTER 7. PROGRAM START

7.13 Connecting multiple analysers

You can use multiple analyser at one PC at the same. Furthermore it is possible to compare the data and events of one analyzer to the data of another one or to data recorded earlier. Every control program acts independent to the others.

To explicitly connect the control program to a certain MSB-RS485 you have to start the program with declaration of the serial number of the MSB-RS485. The serial number is attached to the bottom of the instrument and is displayed in each window frame of the running program. It has the following format: MSB####.

If you do not add the serial number the program connects to the first instrument it finds on the USB. This is the default behaviour.

Depending on the PC and the sequence of search the analyzer found in each case may vary.

To select a certain analyzer by double click on the start icon proceed as follows:

1. Right click onto the MSB-RS485 Icon and select the entry Copy.
2. Right click onto an empty space of your desktop and select Insert to add a copy of the start icon.
3. Rename the Copy to e.g. MSB##### (##### is the serial number).
4. Right click on the renamed icon and select the entry Properties.
5. Add in the field Target: to the call of the control program the parameter -nMSB#####. I.e. something like this:
   C:\Programme\msb-3.2.5\msb_serv.exe -nMSB#####
   resp. for Linux: /opt/msb-3.2.5msb_serv -nMSB#####
6. Click on OK to apply the addition.

Take care that the added serial number of your analyzer is the same as the one on the instrument. Otherwise the analyzer will not be found and an error message will be issued.

Following this procedure you can define an own start icon for each analyzer.

7.14 Automatical start after computer boot

The analyzer can be started automatically and set into the logging mode after booting of the computer. That means in detail:

1. As soon as the windows boot process is finished an analyzer is searched and load with the firmware.
2. Subsequently the analyzer is set to the recording state and starts logging the connection. For this application the last connection settings are used.
3. Every new recording is stored in an own logging file. Its name is combined from the serial number of the analyzer and the start date and time of the recording. For example: MSB00237-20120702093107.msblog means a record taken on 2th July 2012 at 09:31:07.
4. The analyzer software closes the record file as soon as the computer is shut down.
Activate the autostart feature
Windows automatically starts all programs, which are located in the autostart folder. An additional parameter for the analyzer program is necessary to start logging after searching and loading the analyzer. At the same time this parameter takes care, that the logging and its file is correctly closed before the system is shut down.
Open the autostart folder of the active user with the file explorer. Commonly it is the folder:

C:\Documents and Settings\Username\Start Menu\Programs\Startup

Copy the MSB-RS485 start icon from your desktop into the autostart folder (copy, not move). Then right click the new icon in the autostart folder and select Properties.
Add to the target entry the parameter -a, i.e.:

C:\Program Files\msb-3.2.5\msb_serv.exe -a

Click on Apply and OK to save the change. When the computer is rebooted the analyzer program is executed and a new recording is performed.
Please note, that the last events of the recording are possibly not stored when the computer is incorrectly switched off (power off without shut down command).

Autostart with high data increase
Please not! The time to store the data depends on the amount of data and can take several minutes with very large volumens of data.
An alternative would be to use the command line tools from chapter 19 and put an according script or batch file in the autostart folder.

7.15 Short commands

<table>
<thead>
<tr>
<th>Action</th>
<th>Short command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online help for the control program</td>
<td>F1</td>
</tr>
<tr>
<td>New recording</td>
<td>Ctrl + N</td>
</tr>
<tr>
<td>New project</td>
<td>Ctrl + Shift + N</td>
</tr>
<tr>
<td>Open recording</td>
<td>Ctrl + O</td>
</tr>
<tr>
<td>Open project</td>
<td>Ctrl + Shift + O</td>
</tr>
<tr>
<td>Save recording as...</td>
<td>Ctrl + S</td>
</tr>
<tr>
<td>Save project as...</td>
<td>Ctrl + Shift + S</td>
</tr>
<tr>
<td>Start recording</td>
<td>R</td>
</tr>
<tr>
<td>Pause recording</td>
<td>P</td>
</tr>
</tbody>
</table>
CHAPTER 7. PROGRAM START

<table>
<thead>
<tr>
<th>Stop recording</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open a virtual Ledtester</td>
<td>Ctrl + Alt + L</td>
</tr>
<tr>
<td>Open a Data View</td>
<td>Ctrl + Alt + D</td>
</tr>
<tr>
<td>Open a Event View</td>
<td>Ctrl + Alt + E</td>
</tr>
<tr>
<td>Open a Protocol View</td>
<td>Ctrl + Alt + P</td>
</tr>
<tr>
<td>Open a Signal View</td>
<td>Ctrl + Alt + S</td>
</tr>
<tr>
<td>Save settings and close program</td>
<td>Alt + F4</td>
</tr>
</tbody>
</table>

7.16 Additional program arguments

The MSB control program can be called with a series of additional parameters to set explicit defaults like language, offline mode or the type of the connected analyser.

In most cases the default setting (automatical search and initializing of the analyser) is sufficient. If the analyser is not found (this can happen if Bluetooth converters are used because they reserve some COM ports) or if you want to set another directory for storing your temporary logging data you can change this with the following program parameter.

You can add additional program arguments to your desktop start icon like described in chapter 7.13.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>Starts the analyzer in autostart mode. That means that after loading the firmware into the connected analyzer the device is immediately switched into logging mode and all recording files have serial numbered names.</td>
</tr>
<tr>
<td>-D directory</td>
<td>Set the working directory.</td>
</tr>
<tr>
<td>-e</td>
<td>Starts the control program with the default settings. All stored program and session settings will be ignored.</td>
</tr>
<tr>
<td>--force</td>
<td>Forces the program to use the given port name and serial number in cases the automatical detection of the analyzer fails. Please note! This parameter only works with a given port name and serial number, i.e. msb_serv -pCOM12 -nMSB01234 --force</td>
</tr>
<tr>
<td>-i</td>
<td>Forces the loading of the firmware even when the MSB-RS485 is already loaded.</td>
</tr>
<tr>
<td>-j</td>
<td>Forces the program windows to appear on the current screen. Use this parameter, if you want to open a project file, which was saved on a workstation with more than one monitor. (And therefore the windows doesn’t appear, because they are saved on a non-visible screen).</td>
</tr>
</tbody>
</table>
7.17. SPECIAL PROGRAM PARAMETERS

Parameter | Description
--- | ---
-`language` | Select the language. Values for language are:
0: System default, depending on your operating system, 1: english, 2: german
Syntax: `msb_serv -l 1`

-`serno` | Select a analyzer by it's serial number `serno`. Important, if you are connecting more than one analyzer at the same time.
Syntax: `msb_serv -n MSB12345`

-`o` | Starts the control program offline. A connected Analyser is not searched for. Recordings are not possible but saved data can be examined.

-`p port` | Virtual com port to be used. For example COM1 (Windows) or /dev/ttyUSB0 (Linux).
Please note! You have to pass the serial number of the connected analyzer with parameter -n (see above) otherwise the program will use the default MSB00000 for setting storage.
Syntax: `msb_serv -p COM1 -n MSB12345` (Windows) respectively `msb_serv -p /dev/ttyUSB0 -n MSB12345` (Linux)

-`r number` | Reduces the firmware transfer speed by the given number. Default is 0 (full speed), maximum value 50.

-`T directory` | Presetting of the directory where the temporary logging data is stored. By default this is C:\Documents and Settings\Username\Local Settings\Temp (Windows) respectively /tmp (Linux).

7.17 Special program parameters
Beside the 'normal' program arguments the control program also offers a few parameters to affect the program in some special cases.
The relating parameters are listed below and are not stored after the program end. That is you have to give it to the program each time you start it again.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--ignore-unsaved-data</td>
<td>Disables the warning about recorded but not saved data. This may useful if you running some tests without a need to store the data always afterwards.</td>
</tr>
<tr>
<td>--socket=portnumber</td>
<td>Specifies another socket port for the communication with the SwitchEditor. The default ports are in the range 50000...50100, but sometimes other applications have already reserved these. A validate port number starts with 1024, the max. number is 65535. A zero port number disables the socket completely, the use of the SwitchOption isn't possible then.</td>
</tr>
</tbody>
</table>
Already while recording the data can be displayed at different points in time in different formats with different time resolution. We call this concept MultiView, the actors Views or Analysis Tools.

The MSB-RS485 analyzer software uses a multi-process architecture to guarantee a high maximum in stability and scalability. The Recording of data from the via USB connected analyzer and their display and evaluation are done by separated and independent programs and processes which communicate with each other. That has a lot of advantages:

- A recording can be examined at the same time at different segments of the data stream and in different representations with different analysis tools.
- Visualization in real time already while recording.
- The number of views only depends on the computing and system power (scalability).
- Application errors in the displaying programs do not have effect on the recording.

By the capability of the single programs (Views) to communicate with each other a number of new possibilities to make the analysis of EIA-422/485 connections easy are opened. So different views of the recorded data can be linked. What does that mean? Every display program can be selected as master. All other data views automatically follow this master view and synchronize their displays to it. For instance: The graph of the physical data signal (scope view) follows the cursor of the data monitor and vice versa. The search for a defined level change or a specific data delay fades in the respective data sequence. A click onto the recorded parity error shows the respective signal, a.s.o

8.1 Synchronization

This way of communicating is called synchronization, the handling is identical for all Views. Each display program may alternately follow the current recording and display the last occurred events (data byte or level change). Or it can lock the current view to compare it with another sector or recording.
CHAPTER 8. THE MULTIVIEW DESIGN

If the display program is switched to interlocked operation it reacts on all synchronizing requests which are triggered from other Views and fades in the respective section of the recorded data in its own display mode. Thereby the program, which is just operated by the user, is automatically seen as the master.

With this simple concept any views can be synchronized, completely independent of the running recording.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓</td>
<td>Follow (autoscroll)</td>
<td>The display follows the recording and always fades in the last recorded data.</td>
</tr>
<tr>
<td></td>
<td>Locked</td>
<td>If locked the content of the View is frozen, e.g. to compare it with other views from other parts of the recording.</td>
</tr>
<tr>
<td>←</td>
<td>Linked</td>
<td>If linked the View is synchronized with the content of the master View.</td>
</tr>
</tbody>
</table>

**Follow (autoscroll)**

If your interest is in the last events of the examined data connection, for instance if you like to see the current data flow or you want to control the current bus direction and/or bus validation you have to activate the Follow button in the tool bar.

The analysis window is switched to the autoscroll mode and shifted its window content always so that the last event is visible.

Please note that in this autoscroll mode no synchronization with other analysis windows is performed. An active autoscroll is limited to the respective window and has no effect on other analysis windows.

**Locked (fixed)**

In case the opened windows shall represent different data sections a synchronizing or following of the display is not wanted. You just want to intendedly watch the different data sections. An update by synchronization would delete the window content. Therefore set the display mode to locked.

**Linked**

As soon as you activate this button the content of the window follows the cursor movements of the active input window. That means it synchronizes with the analysis window which currently has the input focus and is operated by you (master window).

If more than one analysis window is opened at the same time automatically the window which has the input focus is the master. All cursor movements or shifts are also transferred to all those windows, which are set to the linked mode.

### 8.2 Views (displays)

Views are autonomous programs which link into a current running recording and visualize data in a certain format. The MSB-RS485 analyzer software follows the concept to offer a specially optimized display tool for each kind of
examination. Each view provides functions which represents its kind of data interpretation. Thereby the handling stays easy and clear, multiline toolbars and overload menus are avoided.

You are searching in a data View for data sequences, while you watch out for level changes in the event monitor? Each View provides just the search dialog you would assume to find there.

Simply close data views which you do not need or do not open them. Since they all are independent programs you can place them on your desktop as you like and vary their size and position. The session management saves all settings. Views are automatically shown with their last adjustments and can be copied with a single click.

The following Views are available:

**Virtual Ledtester**
The current line level displaying LED tester is a standard tool for checking RS232 communications. We modified its virtual EIA-232 counterpart for the operation on EIA-422/485 connections. In this way a fast check of the bus states (inactive /active), data direction, handshake conditions and digital auxiliary inputs is possible.

**DataView - Data Monitor**
The data monitor represents the transferred data as a series of data bytes in different formats (ASCII, decimal or hexadecimal). As a special feature the data monitor allows the search for defined pattern by the use of regular expressions, which exceeds the normal search for words or sequences by far. In addition you can search for pauses between sent and received data and in general between any data.

With the help of the integrated script language the displayed data can be computed and colored in any way. Protocols can be visualized, checksums tested in real time and data transformed into other forms.

**EventView - Event Monitor**
Every line change is an event and is logged. Be it the change of a control line or the change of a single bit of a transferred data byte. The event monitor lists them all and allows a simple navigating between all or certain event types, the measuring of times between events and the search for defined conditions or condition changes. E.g. Changing of the bus state (tri-state) or of a handshake signal during a data sequence.

**ProtocolView - Protocol Monitor**
The protocol view enables you to display the recorded data according to special rules.

Define your own protocol so that every data sequence is displayed in an own line. Also color any section of the sequence to make them more readable.

**SignalView - Signal Monitor**
The MSB-RS485 analyzer samples the logical state of all signals with a maximum of 16 MHz. The result can be watched in the signal monitor. Analogous to a digital scope you can move to any section and examine in different resolutions.
CHAPTER 8. THE MULTIVIEW DESIGN

By synchronizing to other views you immediately see the basic signal behavior of every data byte and therewith the real world of your EIA-422/485 connection.

Regions
Regions are definable sections of the recording. They can be compared with bookmarks and define time ranges in the recording file. Regions can be named, they also can send out synchronization requests to other views.
A click onto the start or end of the region is sufficient to let them be faded in into other windows. In this way it is easy to compare recorded sectors in different representations.

8.3 Copy Views
The Clone symbol in the toolbar starts an exact copy of the current analysis window with all its features, settings and position within the recorded data.
By this you can fix a current view while you go on working with the copy (or the original). This makes sense when you want to compare various data regions.

8.4 Saving the state of the Views
To make the working with views easy the current settings of a view like size and position are saved as default when the window is closed by click onto the symbol. The view is restored when you reopen the view. That avoids that you have to enter all settings when you start a new analysis tool.

In case you do not want to save the settings, e.g. because you experimented with the settings and want to close it without ‘consequences’, simply click onto the close symbol in the window frame instead of the quit symbol in the toolbar.

The saved defaults stay active even after the end of a session. They are part of the session settings and are also regarded in project files.
Project files contain a complete description of the current session. They are the subject of the following chapter.
Session management

A program session contains a variety of opened windows in different views. The session management cares that at program start you will find everything again like you left when closing the program.

The session management takes care for the correct storing of all relevant settings for a session. All recording parameters, window properties (position, size) and content (colors, text size, formats) of the open views are saved as configuration when closing the window and restored at next start.

The storage of the current program settings are completely transparent. You do not have to trigger this process, but you also can save the complete session including the recorded data as a project. In this case you can proceed with the examination of the data at a later time simply by opening the project file.

9.1 Projects

Projects are used for saving of your current work (analysis) with the MSB-RS485 software, that means the recorded data is also stored. Therefore a project always consists of two files:

1. **Project file**: Project file: Describes the condition and properties of all open views. Project files do have the extension `.msbprj`.

2. **Record file**: Record file: Contains the actual data and all information, relevant for the data recording which are: data rate, protocol, defined regions, which event types are recorded and time of their recording. Record files do have the extension `.msblog`.

**Why this splitting into two files?**

Stored sessions (projects) corresponds to the user request to configure the program individually for his own needs. These are mostly independent of the recorded data. Perhaps he wants to adapt the placement and display of the views to the screen resolution or use other fonts than the default ones.

On the other hand record files contain information which are independent of the session settings. These information about the protocol, time stamps, regions, used signal names and (de)activated event types. Furthermore recordings should be analyzed by different persons with different ideas about the configuration.

By this splitting some advantages are added:
CHAPTER 9. SESSION MANAGEMENT

- The storage of the recording is done independent of the current session.
- A recording can be loaded into an existing session without disturbing it.
- Other users can examine the data with their individual configuration.
- Project files can be purposefully defined for certain analysis and forwarded.

By the clear separation between project and record file you always can examine a recording with your own program settings or you can use another predefined program configuration for the analysis. Project and record files have their own icon to make the distinction easier. They are linked to the MSB-RS485 software while installation and can be opened by a double click.

9.2 Store and reload projects

Storing and reloading of projects are executed from the control program. The seperation into a session and recording file is done automatically like described before. Likewise a recording file (if existing) is loaded when you open a project. The same applies when you start the MSB-RS485 software with double-click onto a project file *.msbprj from the Windows file explorer. Opening of a project file automatically loads the associated recording file msblog too.

Please note that certain settings like the baudrate are stored as default in the session file as well as in the recording file as mandatory part.

A record file contains the data and also all transmission parameters. It has the extension *.msblog. As soon as a recording is loaded by the software this information is fetched from the recording file and (over)written into the session configuration. This applies for the protocol settings (baudrate, parity, stopbit) and for definition of the signal names and activated events. These settings are inseparably linked to the recorded data.

Create a pure project file without data recording

To save a current session as configuration for later examinations you have to save it as project without data or you can delete the record file to get the pure project file.

9.3 Automatic storing of a session

This process is done transparently in the background as soon as you close the current session by closing of the control program. The MSB-RS485 software stores all necessary settings in a configuration file with the name MSB#####.db in your home directory whereat ##### is the serial number of your MSB device. Under Microsoft Windows this is:

C:\Documents and settings\User name\MSB#####.db

Under Linux:

/home/User name/MSB#####.db

If you did not connect an analyzer the settings are stored in the file MSB00000.db.
The virtual LED tester is modeled on a customary serial line state tester and shows the state of all differential inputs CH1 to CH4 and of both auxiliary inputs. Additionally, the bus signal and the bus direction between CH1 and CH2 (segment analysis) is visualized.

For better clarity, the LED tester (or line monitor) has two separate diode columns for every active line state, consisting of red and green LEDs in each case. The red LEDs on the right side signal a positive line level, the green LEDs on the left side a negative level.

In the range from $\pm 0.7V$ both LEDs are off. This corresponds to the inactive Bus/line condition. The trigger level of the EIA-485 receivers is about $\pm 200mV$. By the higher level of the MSB-RS485 analyzer inactive bus levels are still recognized, even if the bus rest level is drawn by pull up resistors to more than $\pm 200mV$.

By default, the current state is displayed independent of a running recording. This corresponds to the synchronization to the last recorded event. Therefore, the scroll button in the toolbar is activated. So the monitor is comparable to a real tester.

You also can use the monitor for watching the status of earlier data in the
recorded data stream.

Click the ‘Sync’ symbol in the toolbar. With this the line state monitor is synchronized with the active display window, e.g. a data monitor. Or you freeze the current state by clicking the ‘Lock’ Button.

The active levels of a EIA-422/485 connection are alternatively described with logical 0/1 as space/mark or as a physical positive or negative voltage. Most time this is more confusing than helpful. To make it a bit easier the Ledtester fades in additional information about the line conditions. Simply move the cursor over the Tester to make this information visible.

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
<td>A logical 1, Mark refers to a negative voltage level (–0.7V...–7V) on the difference signal input (green LED with a minus Symbol)</td>
</tr>
<tr>
<td>0S</td>
<td>A logical 0, space refers to a positive voltage level (+0.7V...+12V) on the difference signal input (red LED with a plus Symbol)</td>
</tr>
</tbody>
</table>

As all other Views the virtual Ledtester also updates the signal names as soon as these are changed in the control program. This is also true when you change the bus wiring.

10.1 The toolbar

The toolbar offers a quick access to the most needed functions.

A End: Saves all settings and closes the window.

B Display mode: According to the mode the ledtester either shows always the current (last recorded) line states or locked or actualizes its content synchronous to the other windows.
The Data View

You are searching for certain data sequences? For communication breaks of a certain length? The data monitor shows the data in their real time sequence, alternatively in decimal, hexadecimal or ASCII and additionally contains parity, framing or break information. Regular expressions allow the search for any data pattern and much more...

The data view displays all transmitted and by the MSB-RS485 recorded data bytes in their sequence. Changes in the control lines are fade out, so that only the pure user data is shown.

The data can be displayed separated for each data channel A/B (the assignment input signal / Data channel depends on the connection mode) or together (see signal selection). The latter makes sense if the reaction on sent data shall be inspected.

If you want to examine the data separate for data channel A and B without mixing them simply start two data monitors.

You also can watch different sections of the same data stream. You can open as many windows as you need. The PC-Resources are the only limitation.

11.1 User Interface

The data monitor shows the transmitted data bytes like a hex editor. Default are 8 characters or bytes\(^1\) per line, displayed in hexadecimal notation and in the ASCII representation. Every line starts with the current address or position as the offset from the beginning of the data stream. Non printable bytes e.g. the carriage return sign are displayed as a dot.

Use the arrow keys to move the Cursor while additional information is displayed in the Statusline like the exact time, position and quantity in relation to the complete data stream.

In case of a communication error (framing or parity) the data monitor fades in the error into the associated data byte. This is also done for the break condition, which could be misinterpreted in the data stream as a null byte.

\(^1\)Strictly spoken 9 bit values because the MSB-RS485 analyzer supports transmissions with 9 bit data length.
CHAPTER 11. THE DATA VIEW

With the integrated Lua script interpreter you can calculate the displayed data in any form, convert a sequence of data in another format and output the result in the watch window (see section 11.6).

Even more - you can colorize and mark the displayed data in real time controlled by a Lua script. For instance if you like to emphasize a curtain protocol or some special sequences of interest. And if you wish to validate an additional checksum too - no problem. A little Lua script will do the job and marks the according bytes in the display as correct or wrong.

The information in the status line is always related to the current cursor position. In the default settings the left field contains the exact position inside the data stream. Only data bytes are counted, other events like level changes are ignored. The right field contains the exact time when the data byte has occurred.
11.1. USER INTERFACE

You can modify the output of each statusbar field anytime by an own Lua script. For instance, if you like to show the time in another format, or if you are interest in the distance to the previous and/or next byte. The chapter 16.1 will give you an introduction how you can realize this.

**Data channel selection**
The data monitor optionally displays both data channels (sources) A and B or a single one. According to the bus connection (tapping 2/4-wire system or segemnt analysis) the display of both channels makes more or less sense. You can switch between the several directions anytime just by click the channel selection in the toolbar.

The data channel selection also defines which data are stored. If you choose both data channels A+B both are stored, otherwise only the data of the selected one. In this way it is possible to save the recorded data or parts of it depending on the data source in one file.

**Synchronizing**
Each analysis window can synchronize its current view with other windows (see Synchronizing the data view). Is the data monitor the active window, that means the one which gets your inputs, than every move of the cursor sends a sync signal to other opened windows.

This includes the cursor movement as a result of a search or positioning. In this way you can watch the signals in the signal monitor, remotely controlled by the search for certain data sequences.

Leftclick the designated data byte to fade in its representation in other views. Likewise the data monitor reacts on a synchronization from other views and fades in the respective data section, where the cursor is positioned onto the data byte nearest to the original event.

How the data monitor acts when it receives the sync signal from another active window determine the sync-buttons in the toolbar. By default the data view is locked, the window does not react on changes. Please note that the data monitor always generates a sync signal, independent of the choosen display lock/unlock function. Windows which shall not react on sync signals have to be locked.

**Addressing the window content**
Besides the navigation by cursor or the left scroll bar the data monitor offers an absolute positioning and shift relative to the current position.

Click the symbol in the toolbar or open the dialog via View→Goto. Or simply just press Ctrl+G.

Simply enter the absolute address or the wanted offset and click one of the following keys.

- **Absolut**: Moves the data sector to the entered address, shortkey Alt+A.
- **Plus**: Adds the entered value to the current position and moves the data view towards data end, shortkey Alt+P.
- **Minus**: Subtracts the entered value from the current position and moves the data view towards data start, shortkey Alt+N.
The input can be made in decimal, hexadecimal or binary format. Simply click on the used number format. Like most other dialogs you can leave this dialog open as long you need it.

11.2 Data selection

If you press the left mouse key while the cursor is on one of the displayed data byte a context menu opens. In this menu any section of the recorded data can be selected. You can mark the beginning or the end of the selection.

You can also mark the beginning of a region with Ctrl + left mouse key and the end with Shift + left Mouse key. This corresponds to the file selection in Microsoft Windows Explorer. The selection is marked with a light blue color. If you want to select all data, just press Ctrl+A.

The data selection can be stored separately or assigned to a Region with F4. By storing special data sequences you can examine these data for transmission errors or compare to other data sequences.

With the export or copy and paste mechanism you are allowed to evaluate any desired section of data in other applications.

Copy and Paste

Copy and paste copies the selected range as text into the clipboard and paste it into another program. If the target application supports RTF like the most word processing software (for example WordPad®, Microsoft Word® or OpenOffice Writer®), the copied data will be inserted with the origin color information, i.e. the data of port A are shown as red, data of port B as blue².

Pure text editors (like Notepad) doesn’t provide any text formatting. Therefore the information of the data direction has to be visualized in a different way. Data bytes received via the first Data channel (A) precedes a dot, data from the second data channel (B) a colon.

The text display is generally in the hexadecimal format to avoid problems with different character fonts.

```
00000020 | :72 :6f :72 0a .73 .6f .6d .20 | ror.some r
00000030 | .65 .73 .70 :6f :6e .73 .65 .0a | esponse.fr
00000040 | :61 :6d :65 :21 :0a .66 .72 .61 | ame!.frame
00000050 | .20 .72 .65 :73 .70 :6f :6e .73 | response.
00000060 | :00 .00 :70 :61 :72 :69 :74 :79 | ..parity.p
00000070 | .61 .72 .69 .74 .79 :20 .61 .6e | arity answ
00000080 | .65 .72 .0a .00 .6e .6f :20 .6f | er..oop
00000090 | .73 .00 :31 :32 :33 :00 | s.123.
```

Save data selection

The data monitor allows to save any selected range (see Data selection) as binary data into a file. This file herewith contains an accurate series of the marked data. You will appreciate this when you want to compare the recorded data sequences to others, available as data files.

²Coloured Copy and Paste is supported only in the analyser software for Microsoft Windows.
11.2. DATA SELECTION

For example if you know the result of the sent data from a bus participant or the original data and you simply want to check if these data have been correctly transmitted. Simply select the wanted range or all data with Ctrl+A and click the menu item File → Save as.... If you selected both data channels for display (A+B) both are stored as well. For a comparison it makes sense to choose just one channel.

**Export a data selection**

To analyze a section or all recorded data with spreadsheet analysis you can export these as a CSV (Comma Separated Values) File. Spreadsheet programs offer extensive statistical tools to evaluate the data. For example the frequency distribution of single data or minimum and maximum times between the bytes. The export capabilities concern the data only. If you are interested in an analysis of other events read chapter Export selection in the event monitor. Select the wanted range and click on the entry Export as CSV in the File menu. In the opening export dialog you can select from the list of the available values any value by clicking on it and moving it with the right arrow to the list of the export values. Repeat this for all interesting values. To change the sequence of the export values click on the value to shift and move it up or down with the up or down arrow. Likewise you can remove a value from the export list with the left arrow. Then enter a name for the export file and click on 'OK' to start the export. For exporting the current view of the data monitor is regarded. The data is exported as hexadecimal values with prefix 0x, or as decimal value or as ASCII character included in apostrophes. The same applies for the addresses. (The address is the position of the data byte in the data stream). An example for the hexadecimal address and data format:

```
"Timestamp(us)"","Address","Input","Data"
3547,0x000050,A,0x20
3547,0x000051,B,0x20
3634,0x000052,A,0x21
3634,0x000053,B,0x21
3720,0x000054,A,0x22
3720,0x000055,B,0x22
...```

The same selection showing a decimal displaying address and the data as ASCII.

```
"Timestamp(us)"","Address","Input","Data"
3547,00000080,A,' '
3547,00000081,B,' '
3634,00000082,A,'!''
3634,00000083,B,'!''
3720,00000084,A,'"'
3720,00000085,B,'"'
...```

Note! The timestamp resolution is in micro seconds (us). Because we have record this samples with a loop back jack, always two data events on data channel A and B have the same timestamp.
CHAPTER 11. THE DATA VIEW

11.3 Data displaying

The data display can be adapted to your own requirements. Just open the setup dialog in the menu Settings→Configure Data Monitor...

Every view offers only those setup possibilities which are relevant for this view. In case of the data monitor it is:

- **Display**: Number and form of columns and data.
- **Colours**: Coloring rules for representation and marking of certain data.
- **Font**: Font type and size.

All settings can be tested with the apply button before they are finally accepted with the OK button.

Columns and data format

In this part of the settings dialog Settings→Configure Data Monitor... you can individually select the number of columns as well as the kind of display (hexa, decimal, ASCII). The number of lines are changed by extending or reducing the display window.

In addition you can fade in the generally defined names for the first 32 characters of the ASCII character set (control characters), e.g. to display 'LF' for linefeed instead of Hex 0A.

Not printable characters can be displayed alternatively as a dot or as the original character according to the chosen font type.

Coloring data

The data monitor allows to color any data depending on the data source. That is an important feature if you want to highlight certain data bytes or sequences. For example the EOS character like carriage Return and/or Line Feed. Or characters wit a set 8th or 9th bit as often used in bus protocols to separate data from address commands.

To activate this feature you can define any number of *coloring rules* which are applied on the display of the transmitted data.

Each rule contains the data source or data channel (A or B), a range for the data value and the color to dye these data bytes. You can switch every rule on or off individually by enabling or disabling it.

The input of the data values from/to is done in decimal where the range is 0 to 511. Values above 255 makes sense only if you analyze transmissions 511. with 9 data bits.

The rules are processed in the sequence from 1 to n (or from top to bottom). Rules can overlap. In this case the last rule is regarded. With this it is possible to overwrite rules in parts to recolor single bytes of a rule defined before.

The presetting has four rules. By clicking onto the button you can add new rules at any time or remove them by using the button.

New rules are attached either at the end or are directly inserted over the selected rule. To remove a rule you first have to select it by clicking onto it.

All entered rules are automatically stored and are at the same time available for all later opened data monitors.
Color schemes are intended for simple applications. If you want to mark the data with complex rules use the integrated Lua script editor in the watch window, see chapter 11.6

**Change the font**
Besides the number of columns and the representation of the data bytes you also can alter the font type, e.g. to use a font with letters of equal width instead of the default proportional font or to adapt the letter type and height.

Click on Settings→configure data monitor to open the settings dialog.
The chosen font type is automatically stored.

## 11.4 THE DATA INSPECTOR

Watching the transferred data is one thing. To find out the reasons for communication problems sometimes it is necessary to analyse the exact timing for the transferred data. What is the time difference between two bytes? Or how long does it take to receive the answer for a sent data string? Click on the 'Inspect' symbol in the toolbar or press Ctrl+I to open the data inspector. The data inspector offers some informations related to the current byte:

- **Position**: The absolute position of the byte and it’s source (Port A or Port B).
- **State**: Error state of the byte, i.e. Parity, Framing or Break.
- **Absolute Date/Time**: Shows the absolute date and time of the received byte in your locale time format.
- **Time difference**: Displays the distance between the previous and next byte.
- **View as...**: Converts the data byte value in different formats.

### Display the line states

To watch the current line state in parallel to the data byte simply open the 'virtual LedTester' in the control program and switch it to synchronizing operation.

## 11.5 SEARCHING THE RECORD

The data monitor contains some functions which are optimal adapted to the search for data and data sequences. So different searches are possible. The search for a certain series of data bytes, for too short or too long times between request and answer, or simply for transmission errors like parity, framing or break. Since every search starts from the beginning or the current cursor position all search functions can be combined in any way.

### Pattern search

One of the outstanding attributes of the data monitor is the search function for special data sequences. The search input is not limited to simple comparisons of data strings. In fact the Search dialog allows the input of so called regular expressions.

Regular expressions are extended by the wild card characters ‘*’ and ‘?’, known from the MSDOS DIR Command. So the command DIR *.HTM lists all files which have the extention HTM.
CHAPTER 11. THE DATA VIEW

DIR FILE?.TXT lists the files (when available) FILE1.TXT, FILE2.TXT etc. Similar mechanisms for searching special data sequences are offered by the Search dialog of the data monitor. It is opened by the search symbol in the toolbar or simply with Ctrl + F.

Generally search starts at cursor position! By default the search is starts with the begin of the data recording. You also can start the pattern search beginning from any other time stamp within the data stream. For this purpose position the cursor of the data monitor at the desired start position and activate the button 'Search from cursor position'.

To find a special sequence you first have to describe this sequence in the input box. It can be a simple string, e.g. LOGIN in a modem connection. Click on the Search button to start the search in the recorded data stream.

The search is always restricted to the displayed data channel. If you have selected channel A only the bytes assigned to this channels are searched. The same applies for channel B. If both channels are displayed all recorded data are regarded. Often the searched data can not be described by a simple data series. For example the recorded data stream can contain the word 'LOGIN' in the following combinations: LOGIN, Login or login. The latter ones can be described like in MSDOS with ?ogin for search. To find all three variants you have to describe the search pattern as a series of the characters L,O,G,I,N, where each character can be a capital letter or not.

For the conversion a regular expression is used. The expressions are listed in the Table below.

```
[ L l ][ Oo ][ Gg ][ I i ][ Nn ]
```

Every character is described by a Set which exactly corresponds to the searched letters. Imagine you inspect a data connection where from time to time the CR of a CRLF(carriage return line feed) sequence is missing. That means you search for a single LF WITHOUT a CR directly before. The appropriate regular expression is:

```
[ ! ] x0D ] x0A
```

and means: all characters except Hex 0D(means CR), followed by a Hex 0A (LF).

A regular expression is a series of any charcter, where certain characters can have a special function. They are listed in the following Table. If you want to use one of the special characters as a normal one, e.g. if you search for Password? and '?' is NOT any charcter but really the question mark, you have to quote it. This is done by a preceding \ char. For instance

Password$ \backslash$?

With the "*" char in a search pattern any data sequence is marked. That makes sense only if this character is framed by other search patterns, otherwise everything will be found.

The following expression finds all names found between 'LOGIN' and 'PASSWORT' but not single 'LOGIN' Sequences without following 'PASSWORT' Sequence:

```
LOGIN$PASSWORT
```
11.5. SEARCHING THE RECORD

The following table lists the available expressions.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>any character</td>
</tr>
<tr>
<td>*</td>
<td>any character string</td>
</tr>
<tr>
<td>[abc]</td>
<td>a char out of the set abc</td>
</tr>
<tr>
<td>[!abc]</td>
<td>a char not member of the set abc angehört</td>
</tr>
<tr>
<td>\xHL</td>
<td>a char in hexadecimal notation, H is the upper half byte, L the lower half byte</td>
</tr>
<tr>
<td>?</td>
<td>the character ?</td>
</tr>
<tr>
<td>*</td>
<td>the character *</td>
</tr>
<tr>
<td>[</td>
<td>the character [</td>
</tr>
<tr>
<td>]</td>
<td>the character ]</td>
</tr>
<tr>
<td>\d</td>
<td>any decimal character 0...9</td>
</tr>
<tr>
<td>\n</td>
<td>the linefeed control character (Hex. 0x0A)</td>
</tr>
<tr>
<td>\s</td>
<td>any whitespace character (blank, linefeed, carriage return, horizontal tab)</td>
</tr>
<tr>
<td>\</td>
<td>the character \</td>
</tr>
</tbody>
</table>

A misentry will be showed as a selected marked input, so you just can retype a corrected version of your matching rule.

Search for time distances
Beside the pattern search facilities the data view also supports the search for defined time distances between two data events. Click onto the search symbol in the toolbar or press Ctrl+F and select the slider Delay. The time specification is always done in seconds, e.g. 0.0015 for 15 milliseconds. The smallest time unit corresponds to the resolution of the analyzer and is 0.000001 or 1μs. Time distances can be defined as limits for over or under stepping or as a range. The button with the link symbol decides if both times have to be valid for the search result (AND-relation) or only one of them, which is the default. Please take a look to the following table:

<table>
<thead>
<tr>
<th>Time(s)</th>
<th>Logic</th>
<th>Time(s)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=</td>
<td>OR</td>
<td>1.000000s</td>
<td>Finds all distances which are longer than 1s OR shorter than 0s. Negative times are not valid, so that the search is for times longer than the entered 1s.</td>
</tr>
</tbody>
</table>
CHAPTER 11. THE DATA VIEW

1.000000s AND 2.000000s Finds all times which are longer than 1s and shorter than 2s, i.e. Times between 1 and 2 seconds.

10000000s OR 0.001s Finds all times which are greater than 100000s or smaller than 1ms. Since such long times will never happen only times smaller than 1 ms will be found.

Besides the time specifications also the sequence plays a role. That means whether the timely distance between two data bytes of one source, e.g. data channel A, is measured. Or a data byte from channel A, followed by a data byte from channel B (a possible answer). Depending on the wiring you can intentionally search for answering times of a certain bus device and check the answering behavior.

The sequence which shall be obeyed for the search process can be set explicitly. Default is Any, i.e. the sequence is irrelevant.

Please note, that the sequence can be set only if both data sources A and B are activated in the data monitor, otherwise it is disabled.

Search for transmission errors
The search for error conditions refers to errors in the data transmission. These are framing and parity errors. Breaks are usually no errors, but because they must not be mixed with the nul byte and are sometimes used for initializing or resetting of communication partners they are also integrated into the search mechanism.

The search for errors is easy. Mark one or more error conditions and start the search with a click onto the start button.

11.6 The Watch window
The Watch Window serves as the input window for Lua scripts and as the display of their program outputs. If you open up the Watch window for the first time you see an empty list of eight lines or entries. Each entry a Lua script is assigned. As long as you do not enter a script or this script does not produce any output the entry stays empty.

You can display any information in these lines. Numbers, text or a combination of both but no line breaks. Every output is limited to one single but unlimited line.

Double click on any entry you want to write a script for or whose script you want to change. Or click on the Lua tab. Both opens the script editor of the selected entry.

The script editor
The integrated script editor offers all you await from a user-friendly editor! Syntax highlighting unlimited Undo/Redo, Copy and Paste and import and storing of scripts for exchange with other users.

The entries for storing or loading of script are activated in the file menu as soon
as you open the script editor. The file operations always refer to the active editor and therefore to the selected entry in the watch window. The MSB-RS485 software comes with a number of examples which you can find in the examples/Lua directory of the installation directory. You can load one of these examples into the editor (see section ??) or simply enter the following two lines to get you in the mood (presumed some recorded data):

```
1 dv.watch( "Cursor at ", dv.cursor() )
2 dv.mark( dv.cursor(), 1, #ff8080 )
```

This small script displays the current cursor position in relationship to the current segment and colors the corresponding data cell in a bright red.

To run the script click on the ‘Run’ button. Alternatively you can use the key combination Alt+R. The script output is shown as an entry in the Watch List and also in the text line of the script editor because you do not see the watch list entry when the editor is opened.

In case of a wrong input or syntax error an error message is displayed at this place. In our example the current cursor position- if you input the text correctly. With the start of the script it is transferred into the internal byte code of the Lua interpreter. This code is run each time the cursor is moved or the window content is changed. The same happens when you close the editor and return to the watch list. Move the cursor or click onto another data cell to see effects of the script.

A storage of the editor content is not necessary. The data monitor automatically saves its current status and restores it with the next start. Additionally you can save the editor content as a text file to reload it into the editor at any time.

**Example scripts**

In the folder examples/DataView of the installation directory you will find some examples to show the possibilities of Lua. Start the analyzer software by double clicking (Windows) or open the wanted project file *.msbprj* from the control program *msb_serv*. Alternatively you can directly load the project from the control program.

Every example project contains a recording and loads the data monitor with the corresponding Lua script and settings so that you can immediately start.

- **9bit.msbprj**
  Analysis of a 9 Bit data protocol including check sum test. Lua coded LRC (Logitudinal Redundancy Check) function tests for the correctness of the check sum when the cursor is positioned on the start of the protocol sequence. At the same time the sequence is visualized with different colors for Start (Address) byte, length, data and checksum byte.

- **errors.msbprj**
  Coloring of the data bytes for frame-, parity errors or breaks. Shows the information output in the status line, the iteration over the data shown in the data monitor and the indication depending on the error status.

- **modbus-rtu.msbprj**
  Shows the 2-wire segment analysis of a RS485 Modbus connection with one
CHAPTER 11. THE DATA VIEW

Master and two slaves. The single sequences are colored differently for address, function code, data and check sum. By clicking onto an address byte automatically the CRC16 check sum of the sequence is computed and displayed in the status line for comparison with the recorded check sum.

Coloring of all protocol sequences in the data area. Using the recorded data of a Motorola S-Record transfers the data display is shown in terms of color where two following bytes in the BCD representation are converted into the underlying 8 bit value.

Limitations
You can run any operation in Lua, write complex functions and perform extensive evaluations. But the data monitor allows each Lua script only a certain number of computing operations (recursions) or time period for the execution. As soon as your entered script exceeds this limit you get an error message. And that is of a good reason.
If you have programmed an endless loop for whatever reason the data monitor will kindly notify you instead of wordlessly stop further co-operation.

11.7 The toolbar
The tool bar is used for a quick access to the most needed functions. Some are identical to other views, some are specific for the data view.

A End: Saves all settings and closes the window.
B Display mode: According to the mode the window either shows always the current (last recorded) event or locked or actualizes its content synchronous to the other windows.
C Data direction: The protocol monitor can display both data directions (port A and B) combined or separately to display them in different windows.
D New View: Opens a new window with the same sector and settings.
E Search dialog: Opens the dialog for pattern search and transmission interceptions.
F Goto...: Opens the Goto dialog to select the visible section by a absolut address or offset.
G Data inspector: Starts the data inspector.
11.8 Short commands

<table>
<thead>
<tr>
<th>Aktion</th>
<th>Kurzbefehl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runs the current script in the script editor</td>
<td>Alt+R</td>
</tr>
<tr>
<td>Online Help for the data monitor</td>
<td>F1</td>
</tr>
<tr>
<td>Save selection as region</td>
<td>F4</td>
</tr>
<tr>
<td>Start selection</td>
<td>Ctrl+Left mouse key</td>
</tr>
<tr>
<td>End of selection</td>
<td>Shift+Left mouse key</td>
</tr>
<tr>
<td>Select all</td>
<td>Ctrl+A</td>
</tr>
<tr>
<td>Clear selection</td>
<td>Shift+Ctrl+A</td>
</tr>
<tr>
<td>Copy selection into clipboard</td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Export selection</td>
<td>Ctrl+E</td>
</tr>
<tr>
<td>Open search dialog</td>
<td>Ctrl+F</td>
</tr>
<tr>
<td>Open goto dialog</td>
<td>Ctrl+G</td>
</tr>
<tr>
<td>Show data inspector</td>
<td>Ctrl+I</td>
</tr>
<tr>
<td>Open View in a new window</td>
<td>Shift+Ctrl+N</td>
</tr>
<tr>
<td>Save settings and close data view</td>
<td>Ctrl+Q</td>
</tr>
<tr>
<td>Save selection as binary file</td>
<td>Ctrl+S</td>
</tr>
</tbody>
</table>

Key commands of the most important functions

63
When did an event occur? Did a certain level change happen while data was transferred? Or was an error condition (break, parity, framing) recognized? How the status of the bus lines was at a specific time. The event monitor lists all occurred events, searches for event sequences or level conditions and exports events as CSV file.

Contrary to the data monitor the event monitor displays all occurred events (data and level changes) with their time relationship. While the data monitor offers a lot of mechanisms to investigate data streams and to represent the data view of the recording, the event monitor is optimized for the display and search of level changes. That concerns to changes in the level of the control signals as well as asynchronous events like framing or parity errors or breaks. Each event gets equipped with a time stamp which represents the exact time of its occurrence with a resolution of 1 µs. The time distance between has no influence on the display. So signal conditions and changes before and after the recording are easy to identify.

The search for certain data sequences is the task of the data monitor. As soon as a search contains a level condition, a change or an error condition the even monitor is the right tool.

With the LevelFinder not only any static level can be found but also sequence of events and level changes. The single search parameters can be combined optionally with AND or OR and can additionally be combined with a time duration to search for events within a defined time frame or to exclude them. In the chapter event search the search options are described in detail.

12.1 User Interface

The window of the event monitor at any time offers a quick overview over the level conditions. From here you start the search for event sequences, the export of any section or compare different sector of the recorded data stream.
CHAPTER 12. THE EVENT VIEW

Each line is one event
The event monitor displays the recorder events in list form where every line represents the current event and its changes compared to the preceding line status. The list display can be freely configured. Except for the first entry (the type of event) you can fade each column out by drawing its width to zero with the mouse. (In the view menu you can reactivate the faded out columns). The description of the columns automatically adapt to the defined names. These names can be set globally in the control program.

All event types at a glance
The event monitor distinguishes between the following event types:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Event type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Data byte</td>
<td>Data byte received at data channel A.</td>
</tr>
<tr>
<td>B</td>
<td>Data byte</td>
<td>Data byte received at data channel B.</td>
</tr>
<tr>
<td></td>
<td>Level change</td>
<td>Any change of the level of any signal, including level changes of the data lines.</td>
</tr>
<tr>
<td></td>
<td>Framing</td>
<td>Data byte received with a framing error.</td>
</tr>
<tr>
<td></td>
<td>Parity</td>
<td>Data byte received with a parity error.</td>
</tr>
<tr>
<td></td>
<td>Break</td>
<td>Break detected.</td>
</tr>
</tbody>
</table>

The indented error symbols do never occur singular, but always together with (followed by) a data byte event because it signals an error in the data transfer.
12.2. NAVIGATION THROUGH THE EVENT LIST

Changes in the levels of a data line (TxD or RxD) likewise trigger level events, followed by a data event as soon as the data bits are completely received.

**Level changes of the data lines**

If you do not see level changes of the difference signal inputs CH1...CH4 you have to activate them in the control program. If you are not interested in these changes deactivate them to save computing power and disk space since these events come very often.

12.2 Navigation through the event list

The event monitor offers beside the usual scrolling possibilities by mouse, mouse wheel, scroll bar or arrow and page up/down keys also the directed jump to the next or last event of the same type.

Click onto the desired event and than click the ctrl key together with the down arrow resp. up arrow. In this way you easily navigate from break to break or from data byte to data byte.

For longer records you can purposefully fade in ranges from a determined event or time stamp. The latter one awaits the input of a time offset from the start of the recording.

The specification of an event number allows to jump to a determined event or to move in determined steps from event to event. For example all 1000 events forth and back.

12.3 Event search with the LevelFinder

The detection of definite event sequences is one of the unique features of the event monitor. In contrast to the data monitor the search is not restricted to certain data sequences (for which the data monitor is the best solution) but intentionally adapted to changes in the physical level of the single lines. What does that mean?

You can set up the event monitor to search for the level change and/or the condition of any line. You can combine it with the occurrence of a certain data byte or a number of set bits within a data byte or an asynchronous event like break, framing or parity error. Click onto the magnifying glass symbol in the toolbar or press Ctrl+F to open the search dialog.

**Enter a search pattern**

The search pattern may be complicated. The integrated level finder accepts the search input in form of logical expressions where every expression can exist of one or more conditions which can be combined with AND or OR.

Expression: condition1 condition2 ... conditionN

For example:

AND: CH1=high BDIR=high

The formulation of the single conditions corresponds to the intuitive question for searching of defined conditions. In the preceding example: Search for the position in the recording where at the same time CH1 is high AND the bus
CHAPTER 12. THE EVENT VIEW

direction signal BDIR is high too. Each condition consists of a target (which the condition shall be applied to) and a description of this condition. So:

Target=Condition

*Target* is either a single signal line, described by its (also user defined) name or a data (channel) source A or B. *Condition* defines the status which the target has to have for the search. In case of one signal line this could be one of the three possible level conditions on, off or invalid (alternative names are mark, space, high, low). If the target is a data source the possible conditions are an exact data value, a bit pattern or an error.

**Correct definition of a condition**
Conditions must not contain blanks. Please note, that the names ‘A’ and ‘B’ are deserved and must not be used for signal names.

**Formulate a level condition**
Level conditions can be defined for each of the four difference signal inputs CH1...CH4, both of the additional auxiliary inputs IO1, IO2 and the internal by the analyzer generated bus signals BDIR and BSIG. Not defined lines are simply ignored and not regarded for the search.

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, on, high, mark, -V</td>
<td>Signal level is logical one which corresponds to a physical level of –0.7V...–7V (measured at a difference signal input). All listed descriptions are equal. CH1=on is the same as CH1=1 or CH1=high.</td>
</tr>
<tr>
<td>0, off, low, space, +V</td>
<td>Signal level is a logical Zero which corresponds to a physical level of +0.7V...+12V (measured at a difference signal input). All listed descriptions are equal. CH1=off is the same as CH1=0 or CH1=low.</td>
</tr>
<tr>
<td>none, invalid, 0V</td>
<td>Signal level is invalid. That corresponds to a physical level of -0.7V...+0.7V (according to the EIA-422/485 definition the trigger level of the receivers is about ±200mV. By the higher level of the MSB-RS485 analyzer rest levels which are set by pull-up and pull-down resistors are clearly detected as rest levels. The definition of an invalid level is as follows: CH1=none, CH1=invalid, CH1=inactive or CH1=0V.</td>
</tr>
</tbody>
</table>

**Formulate a data error**
Data errors occur only in connection with a data (channel) source. Therefore the target has to be A or B.

1This sample describes a bus conflict. A [high] level of the BDIR signal means an active transmission at CH2 and a bus participant at CH1 must not send at the same time.
12.3. EVENT SEARCH WITH THE LEVELFINDER

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>break, frame, parity</td>
<td>The data received by the data channel A or B shows a break or a framing or parity error. For example a parity error can be found by A=parity or B=parity.</td>
</tr>
</tbody>
</table>

**Formulate a data value**

Each of the data channel A and B (the data bytes received at the according sources) can be checked for equality or for set bits. The latter one is meaningful when certain bit combinations may are not allowed in the bytes or have a special meaning, defined by the used protocol.

Any data are described by with the * symbol.

The level finder offers four types of entry possibilities:

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=<em>, B=</em></td>
<td>Every data event (A or B) delivers a hit, the data value is not regarded. Makes sense if you search for any data event.</td>
</tr>
<tr>
<td>A='x', B='x'</td>
<td>Checks the target (A or B) for equality with the character, enclosed by the apostrophes. The search for a question mark, received at port A is written as: A='?'.</td>
</tr>
<tr>
<td>A=$xx, B=$xx</td>
<td>Checks the target (A or B) for equality with the hexadecimal value. A search for a question mark, received at port A is written as: A=$3F or A=$3f.</td>
</tr>
<tr>
<td>A=~xxxxxxxx, B=~xxxxxxxx</td>
<td>Checks the target (A or B) for the set bits in xxxxxxxx. For this check the bit pattern is logical AND combined with the data and then checked for equality. To find a data byte at port B with a set 7th bit enter: B=~10000000.</td>
</tr>
</tbody>
</table>

**Search input and search**

Before you start open in the control program the sample project levelfinder495.msbprj in the examples\EventView folder. It is about a 2-wire segment analysis with the additional recording of a digital output of a Modbus device with the help of the second digital IO terminal of the analyzer.

The recording contains a number of data errors and some combinations of level changes which we will search for in the following.

**Search for a break in data channel A**

- Open LevelFinder dialog with Ctrl+F
- Click onto the text field and enter \textgreater \textgreater \textless AND: A=break \textless \textless
- Click the start button
- Click the button ‘More’ to search for the next break
- Go back to the last hit by click on ‘Back’

The visible segment of the monitor changes its position with every hit and displays the found event as a black line.
Search for a break in data channel A or B

- Click onto the text field and enter \( \gg OR: A=\text{break} \ B=\text{break} \ll \)

Search Break at Port B with DTR high

- Click text field and enter \( \gg AND: B=\text{break} \ DTR=\text{high} \ll \) eingeben

That was easy. We make it a bit more complicated and search for:

Search for an inactive CH2 and BSIG as well as BDIR, IO1, IO2 all high

- Now enter into the text field:
  
  \( >\text{AND: CH2=none BSIG=none BDIR=high IO1=high IO2=high}< \)

- Click the button 'more' to find the next hits.

You can combine all level conditions, data and data errors combine in any way. It is not possible to combine different logical operations within one search expression, that means mixing of AND and OR expressions. But we will see, that AND and OR are allowed in succeeding expressions. Sequences with different search expressions are used for searching of signal changes. For instance the search for a change in the bus direction BDIR with an active IO2 line at the same time.

Search for signal changes

Changes are defined by two or more sequenced search expressions which describe the line state before and after the signal change. In this respect we expand the search inputs for the possibility to enter more than one expression in different lines. It is possible to vary the logical operators.

Watch the picture at the side. The interest is on the lines CH1, BDIR and IO2. We want to find the signal change displayed in the ideal zoomed display. All together there are three changes that have to happen one after another:

1. CH1=high BDIR=low IO2=low
2. CH1=none BDIR=none IO2=low
3. CH1=none BDIR=none IO2=high

In each state all three signal levels have to be fulfilled, that means that for every single search expression the AND condition has to be true. Enter each expression in an own line in the text field of the level finder. Since the AND operation is the default you can omit it and write the lines exactly as noted above. (Instead of high or low you also can write 1 or 0).

Wrong input, what happens

If you made a syntactical error, e.g. a wrong name or an invalid character, the level finder answers with a yellow text field as soon as you start the search.
12.4. MARK A SELECTION

You can write any search combinations, for instance the search for a certain data byte together with an active IO2 line, or any data byte from data channel A, followed by a change of the bus direction signal BDIR.

The number of search expressions is unlimited but you have to keep in mind that every expression costs additional computing time and slows down the search process.

The search process runs in parallel to the application and can be aborted at any time by clicking the abort button. Even during a longer search you can operate the event monitor in a normal way and speed.

The LevelFinder saves the current search expression automatically. That also is done when you close the monitor or the complete session.

Start a search beginning with a defined position
Click onto the line where the search shall start and activate Start search from the cursor position.

Searching with time specification
As a special feature the LevelFinder offers an integrated stop watch which can be started in every search expression and can be read out in the following expressions. Thus it is possible to search for level changes which exist a certain time only. For instance an active bus direction signal (BDIR) with a duration in the range of 0.1 to 0.3 seconds.

1 BDIR=none
2 BDIR=high watch.start
3 BDIR=none watch.time>0.1 watch.time<0.3

Please note that all conditions within an expression are combined per default with the AND operator. line 2 defines the change of the bus direction from an inactive level none (line 1) to high and at the same time the watch is started. Strictly spoken the watch is reset and loaded with the time of the occurred event, the change of the BDIR signal to high.

In the third line the change of the BDIR level back to an inactive state is AND-combined with a duration greater than 0.1 sec and smaller than 0.3 sec. Positive signal edges which occur later than after 0.3 sec. are ignored.

Indication of the hits in the signal monitor
The sample contains exactly two positions which fulfill the conditions above, nicely to be seen in the signal display. Open the signal monitor and set it to 'synchronization' with other views. With every search result the marker jumps to the corresponding signal position.

12.4 Mark a selection
Certain functions of the event monitor like the saving of events as an independent record file or the export in CSV format for later evaluation or as a region always refer to a before defined selection.

The selection of any event sequences is done like in other programs. Left click
CHAPTER 12. THE EVENT VIEW

in the list representation onto the line, which shall be the first event of the se-
lection. Then scroll by mouse wheel or scroll bar through the recording until 
the desired last event of the selection. This one click with the left mouse key 
while pressing the Shift key.

To jump to the last event of your selection you also can use the level finder or 
the Go-To dialog. It is only important that you click the last event together with 
the Shift key.

Please note that you can select only ranges and not single arbitrary events. 
With File→save as... you can save the selection as an own record file 
*.msblog for later evaluation with the analyzer tools. In this way the interest-
ing parts of a recording can be extracted and the necessary storage capacity 
can be reduced.

Save a selection as a region

A region serves for marking of certain sections of the recorded data which are 
of special interest. Contrary to a selection regions are valid for all analysis 
tools. As soon as a region is added it is visible for all analysis windows. 
Regions are displayed in different colored ranges and exist independently from 
the event monitor until they are explicitly deleted.

To save a selection as region press the F4 key or click onto Edit→copy 
to region. A maximum of eight regions are available. Under View→show 
region dialog the available regions can be fade in or out or removed. 
Regions are part of the record and are stored together with the data in the 
record file *.msblog.

Export a selection as CSV file

The even monitor allows to save any selection of recorded events as a Comma 
Separated List (Values) (CSV). You will use this when you want to import these 
values into a spread sheet program like Microsoft Excel, Gnumeric or Open 
Office.

At this time you will probably ask why you should want to import the recorded 
events into a spread sheet program. 
Assume you want to sort the recorded events for the longest pauses between 
two sent data bytes. Or you want to create a statistic of the events or data. Re-
quirements of this type are the domain of spread sheet calculation programs. 
The event monitor offers you the possibility to benefit from them.
At first mark the selection of the events to be exported. With Ctrl+A you can 
select all recorded events at one time. 
Click in the menu File→export as CVS.

An export dialog opens where you can select a value from the list of available 
values by clicking and moving with the right arrow button into the list of exported 
values. Repeat this for all desired values.
To change the sequence of values click onto the value to be moved and shift it 
in the desired direction with the up or down arrow.

Likewise you can remove a marked entry back to the selection list with the left 
arrow. Finally you have to enter a name for the export file and click the OK 
button to start the export.
12.4. MARK A SELECTION

The list of available events contains the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Event number, starting with 0.</td>
</tr>
<tr>
<td>Type (Event type)</td>
<td>The following types are defined: ASYNC (framing, parity, break), DATA (data channel A), DATB (data channel B), LLS (logical change of a line) VLS (validity of a line changes).</td>
</tr>
<tr>
<td>Time (μs)</td>
<td>Time stamp of the events as offset to the start of the recording in microseconds.</td>
</tr>
<tr>
<td>Data/State</td>
<td>contains either the data byte or the status of a single line ⇒1.</td>
</tr>
<tr>
<td>Error</td>
<td>in case of a transmission fault contains the kind of error like B (Break), F (Frame) or P (Parity).</td>
</tr>
<tr>
<td>dt same type</td>
<td>Time difference to the last event of the same type in microseconds.</td>
</tr>
<tr>
<td>dt last</td>
<td>Time difference to the last occurred event, i.e. the last recorded change in the line.</td>
</tr>
<tr>
<td>Date/Time(s)</td>
<td>Date and time of the event.</td>
</tr>
<tr>
<td>Original line</td>
<td>Is used for the quick exporting of all events as displayed in the event monitor. The graphical display is changed to a respective text string ⇒2.</td>
</tr>
</tbody>
</table>

CH1, CH2, CH3,...
- ⊥ correlating with -12V resp. mark or 'logisch 1'
0 is a invalid or inactive signal state
+1 correlating with +12V resp. space or 'logisch 0' ⇒3

[1] Data status
The content of this field is depending on the data type. If it is a transferred data byte it contains the data value in the lower 8 bits, the upper 8 bits are 0.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Unused</th>
<th>Data Byte</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

In case of a level change the upper 8 bits contain the logical state of the lines. The lower 8 bits contain the valid states. If it is '0' the line is in an invalid condition, that is a level of -0.7V to +0.7V.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Line State</th>
<th>Bit</th>
<th>Bit</th>
<th>Valid State</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td></td>
<td>6</td>
<td>7</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

CH1 CH2 CH3 CH4 BDIR BSG IO1 IO2 CH1 CH2 CH3 CH4 BDIR BSG IO1 IO2

[2] Text symbolism of the level conditions
Sent data and line states is different information and consist of a different number of fields. Data are represented as hex value with a respective ASCII or control name, while the lines are listed as eight status and transition sequences.
To reach the same number of columns for the CSV export the data as well as the conditions of the lines are embraced by "...".
The conditions and transitions of all lines are described by the following names:

- \(^\) : High level
- \(-\) : Invalid level
- \(\vee\) : Low level

A sequence of \(-\vee\) describes a change from invalid to low level, while a level change from high to low is described by \(^\vee\). The following extraction shall clarify this:

```
"Typ","Nummer","Zeit(s)","Daten oder Status"
LLS,[000000],0.000000s,"--CH1,--CH2,--CH3,--CH4,--BDIR,--BSIG,--vIO1,--vIO2* VLS,[000001],0.000000s,"--CH1,--CH2,--CH3,--CH4,--BDIR,^^BSIG,vvIO1,vvIO2* DATA,[000002],0.000985s,"00,<NUL>"
LLS,[000003],0.255893s,"^-CH1,--CH2,--CH3,--CH4,^^BDIR,^vBSIG,vvIO1,vvIO2* VLS,[000004],0.255997s,"^-CH1,--CH2,--CH3,--CH4,^-BDIR,vvBSIG,vvIO1,vvIO2* VLS,[000005],0.257247s,"v-CH1,--CH2,--CH3,--CH4,^-BDIR,vvBSIG,vvIO1,vvIO2* LLS,[000006],0.257352s,"^-CH1,--CH2,--CH3,--CH4,^^BDIR,^vBSIG,vvIO1,vvIO2* LLS,[000007],0.257456s,"^-CH1,--CH2,--CH3,--CH4,^^BDIR,^vBSIG,vvIO1,vvIO2* LLS,[000008],0.257560s,"^-CH1,--CH2,--CH3,--CH4,^-BDIR,vvBSIG,vvIO1,vvIO2* LLS,[000009],0.258289s,"^-CH1,--CH2,--CH3,--CH4,^-BDIR,vvBSIG,vvIO1,vvIO2* DATA,[000010],0.258337s,"01,<SOH>"
```

[3] Signal names during exporting
Please note, that the lines do not have to have the default names since you can rename them according to your application. The new names appear instead of the standard names. Compare the chapter signal names in the control program.

12.5 Measure time distances

Every event can be marked by right click the event line (item). Next to the right side of the type column a clock symbol is fade in and in the status line the time difference from the marked event to the current event at the cursor position is displayed.

A second right click onto the marked event removes the clock mark again.
12.6 The toolbar

The toolbar is used for a quick access to the most needed functions. Some are identical to other views, some are specific for the event monitor.

A **End**: Saves all settings and closes the window.

B **Display mode**: According to the mode the window either shows always the current (last recorded) event or locked or actualizes its content synchronous to the other windows.

C **New View**: Opens a new window with the same sector and settings.

D **Event dependent scrolling**: Jumps to the last or next event of the same type like the one at cursor position.

E **Event search**: Opens the level finder dialog for event search.

F **Goto...**: Opens the Goto dialog to select the visible section by event number or time specification.

12.7 Short commands

<table>
<thead>
<tr>
<th>Action</th>
<th>Short command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online Help for the event monitor</td>
<td>F1</td>
</tr>
<tr>
<td>Opens the search dialog (LevlFinder)</td>
<td>Ctrl + F</td>
</tr>
<tr>
<td>Open Goto dialog</td>
<td>Ctrl + G</td>
</tr>
<tr>
<td>Jump to the time marker</td>
<td>Ctrl + T</td>
</tr>
<tr>
<td>Select all Events</td>
<td>Ctrl + A</td>
</tr>
<tr>
<td>Clear selection</td>
<td>Ctrl + Shift + A</td>
</tr>
<tr>
<td>Save selection as region</td>
<td>F4</td>
</tr>
<tr>
<td>Jump to last event of the same type</td>
<td>Ctrl + Up arrow</td>
</tr>
<tr>
<td>Jump to next event of the same type</td>
<td>Ctrl + Down arrow</td>
</tr>
</tbody>
</table>
With the analysis of protocols you enter the next level of communication. The seemingly arbitrarily occurring data are sorted and grouped according to your rules. Output functions allow you to format and color data sequences individually.

The exchange of data between two or more communication partners generally happens depending on a protocol, which defines the format of the transferred data together with their content and meaning. The smallest data unit is called Datagram. While the data monitor displays the transferred data in the sequence of their occurrence without any interpretation (which sometimes has advantages) now the analysis of protocols and datagrams is the next level for understanding the communication.

The data stream, captured by the analyzer, has to be split into single data sequences or datagrams to display them on screen. Since there are no defined rules (resp. a lot of different standards) for the definition of datagrams a lot of different practical realizations are known. They vary from simple end-of-string characters (EOS), start (STX) and end (ETX) marks to the usage of timeouts, run time lengthcodes and other definitions. The protocol monitor puts you in the position to interactively create so called protocol templates which are directly applied on your recorded data.

13.1 User Interface
To make the selection and adaption of a protocol to your application as easy as possible the protocol monitor offers, besides the display of the protocol sequences, an additional template editor. The input of templates is interactive and directly effects the display of the recorded data, even while the logging is still running.

This capability of the protocol monitor to define sequences using an own template language exceeds the normal use of a fixed selection list. Admittedly this demands a certain learning process concerning the syntax and may sometimes be a bit hindering for trivial problems.

Therefore the user interface of the protocol monitor offers both. Often used templates, e.g. different combinations of carriage return and linefeed characters or changes in the data direction are simply to recall from a selection list. These templates are unchangeable but can be easily added as a copy to the list and then edited by the user. The following picture shows a typical application.
CHAPTER 13. THE PROTOCOL VIEW

In the upper part of the monitor the recorded data are displayed according to the definition of the chosen template. In this case a simple communication between a modem and a PC. Each line (sequence) is closed by a carriage return/linefeed.

The lower expanded window contains the description how the single sequences shall be splitted and displayed, separated for each data direction. The Protocol View comes with some predefined templates for common use. By a simple click you can copy the template for modification according to your application, without losing the original.

At the beginning you probably will change only small things: the color of the data or the definition of the line end. With F5 or a click onto the cogwheel symbol you directly apply your changes onto the logged data. The predefined templates offer an easy start for own concepts. All new templates are automatically added to the list and saved (see chapter 13.3).

13.2 Protocol Display

The Protocol monitor displays every sequence (or datagram) in a single line.
Each datagram is provided with a date/time information which by default is your local time. Alternatively you also can add the display of the duration of the datagram and the time distance between the datagrams (See section 13.5).

**Changing Date/time format**
You can change the time format under Options → Preferences.

At the left border a red (Data channel A) or blue (Data channel B) dot marks the data direction. Since all sequences do not always occur alternately every sequence is counted separately for each data direction. Uncompleted sequences, that means datagrams whose end condition is not yet reached (e.g. no end character received), are marked with a dotted arrow.

**Synchronizing the display**
All MultiView programs have in common that they can synchronize or lock their displays or always show the last recorded data (auto scrolling). That also applies for the protocol monitor. Leftclick onto the desired datagram to switch all the other views to the display of the marked datagram. The current datagram is framed. Likewise the protocol display is sensitive to synchronization from other views. The sequence which is part of the synchronization is marked with a green arrow at the left border.

**Choosing a range**
With the export or copy and paste function you can process further any segment of the protocol in other applications. For that you first have to mark the desired area. The selection is done like the file selection in your operating system. Place the cursor onto the first cell of the desired sector and click the left mouse key. After that shift the visible section to the end of the range and mark the end of selection with a left-click together with a pressed shift key. The field will become gray.

If you want to mark all lines press Ctrl+A.

**Copy and Paste**
Copy and Paste copies the selected range as text into the clipboard. Every Sequence is stored with all information like data direction, chosen time information and according to its protocol definition.

```plaintext
```

At first mark the desired sector and press Ctrl+C to copy the selection into the clipboard. With Ctrl+V you can insert the clipboard content into other applications which support the copy and paste functionality.
CHAPTER 13. THE PROTOCOL VIEW

Saving a protocol
You also can save the recorded protocol or a section of it directly as a text file (export). The text format is the same as the format of copy and paste. Press Ctrl+E to open the export dialog. If you did not select any lines the program recommends the export of all lines. The default export file is protocol.txt on your desktop.

13.3 Protocol Templates
The protocol monitor contains a list of predefined templates for the representation of simple protocols. By clicking onto the selection button of the template toolbar you can chose another template at any time. The adaption of the protocol view is directly done.

The predefined templates are a fixed part of the monitor. As they are samples for your own template definitions they can neither be deleted nor be altered. To distinguish them from others they are marked with a '*' in their names.

Define your own templates
For the handling of the templates the protocol monitor follows a double strategy. On the one hand it offers a list of ready-made templates, on the other hand their adaption for own applications shall be as easy as possible.
To change a template or to create a new one you have to open the template editor. Either click on the arrow symbol next to 'Template Definition' or press Ctrl+T. (A new click on the arrow symbol or press of Ctrl+T closes the editor window again).

As a next step you can copy any template from the list to the now active '+' button in the template toolbar to get a sample for a new template. The copy is new part of the selection list and has to have a unique name. The default name for the current template is a continuous number.
The new template can be modified in the editor window in any way. With F5 or click onto the wheel symbol in the toolbar the template definition is applied onto the data.
If the template is erroneous a respective message is fade in into the status line.

Apply the template
Please note, that a change in the sequence or datagram input format definition requires a new formatting of the recorded data and may take a while depending on the size of the recording. The modification of the datagram representation affects the display only and is directly visible.
13.4. TEMPLATE LANGUAGE SYNTAX

Modify an available template
As already mentioned the predefined templates are secured against deletion and modification. This is allowed for your own templates only. Simply select the favored template from the list and modify it according to your application. All changes are automatically saved and won’t get lost, even when you in between open another template from the list.

Rename template
Copy the template as new with from the list and name it. Then remove the original one from the list with click onto .

Saving templates
Every new defined template is part of the current session and is automatically saved. At the same time templates are part of a project and are saved together with the project. Saving manually is not necessary. Despite that you can store templates in own files and reload them later, e.g. if you want to export the recorded protocol or insert it with copy and paste into other applications together with a link to the template definition.

Saving
Press Ctrl+S or click on ‘save template’ in the file menu. Select a directory. The current template name is recommended as file name which can be changed.

Load
With Ctrl+O you open the Dialog for selecting the template to load. The loaded template is automatically added to the template list and applied to the data.

13.4 Template language syntax
Protocol templates describe the size of a datagram (data sequence) and its representation. Therefore the templates are split into an input part (in) and two output parts (aout) and (bout), separately for each data direction. The input part defines the rules for separating the single datagrams, while the output parts allow rules for the visualization of the datagrams like format or color of single data parts. Additionally it is possible to use alias names for any character, e.g. a <LF> for linefeed. And: Output rules can be grouped for a certain number of characters. That allows you to highlight defined positions within a sequence.

Splitting into datagrams
The protocol monitor offers some template functions which can be used for separating of single sequences. A simple separating could be into diagrams, which end with a linefeed. Often datagrams are started by a change in the data direction (end also ended). Or a send pause is defined as start and end of a datagram. All have in common, that datagrams obey a break condition which flags the end of the data sequence and at the same time the start of a new one. It is important for the understanding of the functionality of the protocol monitor.
CHAPTER 13. THE PROTOCOL VIEW

that for splitting the data you have to define a break or end condition, not a start condition for the datagrams.

Definition of a datagram

For the definition of a protocol template the question has to be: when does the datagram end and not when does it start since the end of the datagram is at the same time the start of the next one.

The functions for splitting the input data stream are started with the key word:

```
1  in = { ... } ;
```

The block, embedded into the curly braces contains at least one (end) rule, which can be extended by a logic OR (indicated by the square brackets). Here is the complete syntax where FNC is a synonym for any function, declared later:

```
1  in = { FNC( ... ) | | FNC( ... ) } ;
```

In case of splitting by linefeeds at the end of a sequence, that means that all datagrams are ended by a linefeed:

```
1  in = { EOS(0A) } ;
```

Please note the semicolon at the end of the line. Because a definition can range over multiple lines the semicolon is necessary to flag the end of the command.

The protocol monitor offers a lot of functions to define these end conditions. They can be combined by the OR Operator ‘|’.

Imagine a protocol where the end of the datagram is indicated by a linefeed or a carriage return. This can happen in the communication between partners with different operating systems. In this case you would enter the end condition as follows:

```
1  in = { EOS(0A) | EOS(0D) } ;
```

or to be written in a bit more readable way:

```
1  in = {
2     EOS(0A) |
3     EOS(0D)
4 } ;
```

Do not mistake the above with `in={ EOS(0A0D) }`; that case a datagram is defined which explicitly awaits a linefeed, followed by a carriage return.

You can combine multiple and or break conditions by the OR operator ‘|’.

Functions for splitting into datagrams

The following functions for splitting the data stream into datagrams are available. You can combine them with the OR operator ‘|’.

The function names are shown in capital letters to set them apart from the surrounding text. You do not have to regard this, you also can write the function
names in lower case letters.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;BCHG&quot;(S.83)</td>
<td>Ends a datagram or sequence when a change of the given bit or bits takes place.</td>
</tr>
<tr>
<td>&quot;CAB&quot;(P.83)</td>
<td>Ends a datagram or sequence when a change in the data direction takes place.</td>
</tr>
<tr>
<td>&quot;CNT&quot;(P.84)</td>
<td>Defines a datagram as a character series with a constant length.</td>
</tr>
<tr>
<td>&quot;CNTA&quot;(P.84)</td>
<td>Defines each datagram, received from Data channel A, as a character series with a constant length.</td>
</tr>
<tr>
<td>&quot;CNTB&quot;(P.84)</td>
<td>Defines each datagram, received from Data channel B, as a character series with a constant length.</td>
</tr>
<tr>
<td>&quot;EOS&quot;(P.85)</td>
<td>Defines any character sequence as end of datagram (End Of String).</td>
</tr>
<tr>
<td>&quot;RTL&quot;(P.85)</td>
<td>The first received data byte is interpreted as the length definition of the datagram.</td>
</tr>
<tr>
<td>&quot;TO&quot;(P.85)</td>
<td>The datagram is ended after a send pause with a minimum duration (timeout).</td>
</tr>
</tbody>
</table>

### BCHG

Starts a new sequence after alternation of a given bit in the data word. Certain bus protocols make usage of the most significant bit to distinguish between a normal command or address sequence. With BCHG (Bit Changed) you are able to mark the alternation of a special bit as the start of a new datagram.

**BCHG**(bitno)

- **bitno**: The number of the bit indicating a new sequence. The bit number count starts with 0 (LSB) until 8 (MSB bit in a 9 bit data word).

### Anmerkung

The following examples explains how to use the alternation of the most significant bit (bit 8) to apart the various sequences in a 8 bit data transmission.

**Examples**

```c
in={ BCHG(8) };
```

### CAB

Ends a datagram or character sequence when a change in the data direction takes place. This function is appropriate when the communication partners generally send alternating and no overlapping of the send/receive data can happen. It is not applicable for protocols with echo mode (terminal). The syntax is **CAB**() without parameter.

**CAB()**
CHAPTER 13. THE PROTOCOL VIEW

Examples

\begin{verbatim}
in={ CAB() };
\end{verbatim}

**CNT**

Defines each datagram as a series with a constant number of characters, which has to be entered in decimal notation.

CNT\((number)\)

- **number**: Quantity of data bytes or characters in decimal notation

**Hint**

Since any number of functions can be combined in the input part, CNT can also be used as a limit for the data sequences, when no other end condition is found. In the second example the sequence is ended when 256 characters are received OR when a linefeed was found in the data stream. Please regard the ‘|’ between both functions.

CNT has an effect on both data directions. Send and receive packets have the same length. For the definition of different lengths the following functions CNTA and CNTB are appropriate.

Examples

\begin{verbatim}
in={ CNT(16) };
in={ CNT(256) | EOS(0A) };
\end{verbatim}

**CNTA**

Defines each datagram, received at Data channel A as a series with a constant number of characters, which has to be entered in decimal notation.

CNTA\((number)\)

- **number**: Quantity of data bytes or characters in decimal notation

Examples

\begin{verbatim}
in={ CNTA(10) };
\end{verbatim}

**CNTB**

Defines each datagram, received at Data channel B as a series with a constant number of characters, which has to be entered in decimal notation.

CNTB\((number)\)

- **number**: Quantity of data bytes or characters in decimal notation
13.4. TEMPLATE LANGUAGE SYNTAX

Examples

in={ CNTB(20) };

EOS
Defines an end of string sequence. Simple protocols use one or more (but commonly not more than two) characters to indicate the end of a datagram. The communication between a modem and a PC uses a carriage return (hexadecimal 0D), a linefeed (hexadecimal 0A) or a combination of both. With the EOS function you can set any character sequence as end string.

EOS(string)

- **string**: Order of characters, which indicate the end of a datagram, entered in hexadecimal notation.

Examples

in={ EOS(0D0A) };

RTL
Run-Time-Length Function. The first received data byte is seen as the length definition of the datagram excluding the first byte. For instance: If the first byte contains the value 5 then 5 following bytes are awaited. The complete length is 6 bytes.

RTL(value)

- **value**: Optional positive or negative number which is added to the data byte to determine the final length of the datagram. The default is 0, in this case the parameter can be omitted.

Examples

in={ RTL() };
in={ RTL(2) };
in={ RTL(-1) };

TO
Imagine the communication uses a protocol where single datagrams are separated by a certain time duration, a defined pause in the communication. Or you want to force a break with a new sequence when a gap is detected in the data stream. In both cases the function TO is the appropriate solution.

TO(milliseconds)

- **milliseconds**: Number of milliseconds in decimal notation after which the data sequence is seen as finished.
CHAPTER 13. THE PROTOCOL VIEW

Hint
In the following example the datagram is ended when no byte is received at Data channel A or Data channel B for at least 1 second.
Important! The timeout is supervised individually for each direction with the same time value.

Examples

\[ \text{in} = \{ \text{TO(1000)} \}; \]

Individual display of the datagrams
The fixed templates display the data according to their direction and their range and function (control characters, ASCII or Hex). The data received from Data channel A are red, the data from Data channel B are blue (like colored in the data monitor)
Additionally the data direction is marked with a red dot (data from Data channel A) resp. with a blue dot (data from Data channel B).

That may be not that what you would like to have. At this time we discussed the possibilities to extract single datagrams from the input data stream. Now we turn to the functions to represent the datagrams on screen.
For the description of the way of presentation the protocol monitor also uses a couple of functions, just as for the definition of the datagram separation. They can be combined in any way.
To differ between data received at Data channel A and data received at Data channel B two different output blocks are provided for formatting. The complete syntax is:

\[ \text{aout} = \{ \text{FNC(...)} \} [ + \text{FNC(...)} ] \}; \]
\[ \text{bout} = \{ \text{FNC(...)} \} [ + \text{FNC(...)} ] \}; \]

Every block has to be closed with a semicolon.
Combinations of functions are also possible, shown by the functions embedded in square brackets. But now these functions are not combined by the OR operator, but by the AND operator `+`.
Why not the OR operator? Well, that is simple. While it makes sense to define some different possibilities to end a datagram (e.g. the occurrence of carriage return OR linefeed), it is not very helpful to display a sequence in yellow OR green. Simply because both is impossible and a character or letter is not yellow and green at the same time.

Output functions
The protocol monitor defines only five output functions. By their combination a broad range of formatting are possible.
To keep the learning process as low as possible the output functions look very similar.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 13.4. TEMPLATE LANGUAGE SYNTAX

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ALIAS&quot;(P.87)</td>
<td>Defines any character string, which is shown instead of a certain character/byte.</td>
</tr>
<tr>
<td>&quot;ASCII&quot;(P.87)</td>
<td>Defines a character range which is displayed as normal text (ASCII-character).</td>
</tr>
<tr>
<td>&quot;CTRL&quot;(P.88)</td>
<td>Activates the display of the ASCII control characters.</td>
</tr>
<tr>
<td>&quot;HEX&quot;(P.89)</td>
<td>A range of characters, which are displayed in hexadecimal notation.</td>
</tr>
<tr>
<td>&quot;MASK&quot;(S.89)</td>
<td>Limited the representation of each data to the given bits.</td>
</tr>
<tr>
<td>&quot;REPEAT&quot;(P.90)</td>
<td>Repeats the last command for a given number of characters.</td>
</tr>
</tbody>
</table>

All functions can be combined by the AND operator `+` in any way. At functions with identical parameters these are arranged in the same order. Partly the parameters can be omitted, they are set to default values. These optional parameters are embedded in square brackets. The single parameters have to be separated by a comma. The evaluation of the functions is from left to right.

#### ALIAS
Assigns a name to any character, written in hexadecimal notation.

**ALIAS**(hex value, alias [,text color, background])

- **hex value**: code of the character in hexadecimal notation.
- **alias**: character string to be displayed instead of the character. Besides normal letters, digits and space the following characters are allowed: `< > _ - ; : . % & * " ! ?`. All characters are allowed except the comma.
- **text color**: Optional color for the string. Default is black.
- **background**: Optional color of background of the string. Default is white.

In the following example all characters received at Data channel A with the value hexadecimal 0A are displayed as red text 'LF' (for linefeed) on black background. The same characters from Data channel B are displayed with white text on blue background. In contrast to the first definition the colors in the second definition are defined in short RGB notation.

**Examples**

```plaintext```
aout= ALIAS( 0A, LF, red, black ) ;
bout= ALIAS( 0A, LF, #FFF, #00F ) ;
```

#### ASCII
Displays a certain character or a range of characters as ASCII letters.

**ASCII**(hex value1 [, hex value2, text color, background])
CHAPTER 13. THE PROTOCOL VIEW

- **hex value1**: Character code of the first or only character in hexadecimal notation.
- **hex value2**: Optional character code of the last character of the range to be displayed as ASCII.
- **text color**: Optional color for the representation of the ASCII letters. Default is black.
- **background**: Optional color of the background of the ASCII letters. Default is white.

**Hint**
If in this function the second parameter (hex value 2) is omitted it is automatically set to the value of the first parameter (hex value 1).

In the following example all characters in the range from hex 20 (corresponding to blank) to hex 7F (last character in the ASCII table) from Data channel A are displayed in ASCII with red text. The background is not set and therefore the default color white is used.
The same range from Data channel B is displayed in blue.

**Examples**
```
aout={ ASCII( 20, 7F , red ) };
bout= ASCII( 20, 7F, blue ) ;
```

**CTRL**
The output of control characters (range hex 00 to hex 1F) takes place according to the standard ASCII control code names. So hex 02 is replaced by the string 'STX', hex 07 by 'BEL'. You will find an overview of the 7 bit ASCII characters in the Appendix. With this function you save yourself a lot of ALIAS definitions, as long as you are interested in the standard codes only.

**CTRL**/**text color, background/**
- **text color**: Optional color for the display of the control code string. Default is black.
- **hintergrund**: Optional color for the background of the control string. Default is white.

In the following examples all characters from Data channel A in the range from hex 00 to hex 1F are replaced by their respective standard code in orange color, all control characters from Data channel B are displayed in navy blue.

**Examples**
```
aout= CTRL( orange ) ;
bout= CTRL( navy ) ;
```
HEX
Displays a certain character or a range of characters in hexadecimal notation.

HEX(\text{hex value1} [, \text{hex value2}, \text{textcolor}, \text{background}])

- \text{hex value1}: Character code of the first or only character in hexadecimal notation.
- \text{hex value2}: Optional character code of the last character of the range to be displayed as hexadecimal.
- \text{text color}: Optional color for the representation of the ASCII letters. Default is black.
- \text{background}: Optional color of the background of the ASCII letters. Default is white.

Hint
If in this function the second parameter (\text{hex value 2}) is omitted it is automatically set to the value of the first parameter (\text{hex value 1}).
In the following example all control characters from Data channel A (the first 32 characters, defined as a range from \text{hex 00} to \text{hex 1F}) are displayed in red and all control characters from Data channel B in blue on light blue background.

Examples
\begin{verbatim}
aout={ HEX( 00, 1F, red ) };
bout= HEX( 0, 1f, blue, light blue ) ;
\end{verbatim}

MASK
Fades out bits which are not relevant for the representation by masking the valid representation range. This is mainly used for 9 bit protocols whose data exceeds the allowed value range of 0 to 255.

MASK(\text{bitmask})

- \text{bitmask}: Appropriated bits, used for the displaying.

Hint
You probably will use this function only if the upper bit or bits of the transmitted characters have a special functionality which shall not effect the representation. Or if you want to narrow the display of a character value to a certain value range.

Examples
In a 9 bit protocol the characters \text{hex (100)} to \text{hex (1FF)} shall be displayed as normal 8 bit ASCII characters since the upper bit (MSB, most significant bit) only distinguishes between data and address commands. Without this limitation all data above \text{hex(FF)} would be displayed as dots since the (extended) ASCII character set is specified for 8 bit data only.
**CHAPTER 13. THE PROTOCOL VIEW**

1. \texttt{aout=} \\
   2. \texttt{MASK( ff ) +} \\
   3. \texttt{HEX(80, ff, blue) +} \\
   4. \texttt{ASCII(20, 7f, blue)} \\
   5. \texttt{);}

**REPEAT**
Repeats all preceding output formatting commands and applies them to a given number of sequenced data bytes.

**REPEAT**(\texttt{number})

- \texttt{number:} Number of repetitions of the output functions, standing at the left of the repeat output command.

**Hint**
The repetition is valid either for all preceding functions or until the last preceding \texttt{REPEAT} command.

**Examples**
The first two data bytes shall be displayed as hexadecimal characters with white color on red ground (line 2,3). The following two are also displayed as hex, but with blue color on white ground (line 4,5). The rest of the data bytes are displayed in ASCII, as they are in the range of hex 20 to hex 7F. Their representation is by default black text on white background.

Note: The example is for data from Data channel A. For all data from Data channel B a completely independent display format can be created.

1. \texttt{aout=} \\
   2. \texttt{HEX(0, ff, white red) +} \\
   3. \texttt{REPEAT(2) +} \\
   4. \texttt{HEX(0, ff, blue, white) +} \\
   5. \texttt{REPEAT(2) +} \\
   6. \texttt{ASCII(20, 7f)} \\
   7. \texttt{);}

Please note that the display is depending on the sequence of the function definition. So the following example:

1. \texttt{aout=} \\
   2. \texttt{ALIAS(0A, Newline) +} \\
   3. \texttt{CTRL()} \\
   4. \texttt{);}
shows a \texttt{\textbackslash LF} and not a ‘Newline’ because the \texttt{CTRL()} call \texttt{\textgreater\textless} overwrites \texttt{\textless\textgreater} the \texttt{ALIAS} definition.

It would be correct to call the \texttt{ALIAS} function after the setting of the general control characters by \texttt{CTRL()}. That means that simply the order of the functions has to be exchanged.

**Comments**

A comment starts anywhere with a double hyphen (\texttt{--}) and runs until the end of the line.\footnote{If you would like to ask now: Why just double hyphens? The script language Lua defines this as a comment specifier. And Lua will play an important role in the future versions of the protocol view.}

You are invite to document your templates like the common usage of it or some tricky lines. Last not least comments are very useful to comment out some code for testing purpose. For example:

```lua
1    -- modem.msbhtml
2    -- All sequences end with a carriage return/linefeed
3    in=[EOS(0D0A)]
4    }:
5    out=
6    -- All ASCII characters received at data A are displayed in red
7    ascii(20,7f,red) +
8    -- all control characters are orange
9    ctrl(orange)
10   }:
11   bout=
12   }
13
14   ascii(20,7f,blue) + \texttt{-- the same with data B (just in blue)}
15   -- the following line is commented out
16   -- hex(0,1F,black)
17   ctrl(navy)
18   }
```

### 13.5 Changing Date and Time

The protocol monitor precedes each datagram with a line containing the date and time of the received data string. The time is set according to the occurrence of the first character of the datagram.

**Date format**

By default the time specification is in seconds, the date specification is local time according to the setting of your operating system. Depending on the problem and application it may make sense to change this format:

- **Local time**: The display of date and time is done with the local time information, e.g. \texttt{Mo 18 Aug 2008 10:46:53 CEST}
- **Relative to the start of recording**: Only the time since start of the recording is displayed in seconds, e.g. \texttt{8.269604s}.
- **Date and time according to ISO8601**: This is an international format for date and time of the following kind: \texttt{2008-08-18 10:46:53}
CHAPTER 13. THE PROTOCOL VIEW

Time resolution
You can increase the resolution of the time specification if you add the milliseconds and microseconds to the second display. The split seconds are displayed as an offset with a preceding +, e.g.

Duration and distance
Beside the exact time of the occurrence of a datagram often the time is of interest, which was used for the transfer of the datagram or sequence. This means the time distance between the first and the last data byte. The transfer duration of the datagram as well as the elapsed time since the last datagram can be displayed as additional information.

1 8.276218s Abstand=4.534ms Dauer=521us
2 <CR><LF>
3 8.27726s Abstand=521us Dauer=1.562ms
4 OK<CR><LF>
5 13.311445s Abstand=5.032623s Dauer=2.601ms
6 ATE0<CR><LF>
7 13.312002s Abstand=OVERLAP Dauer=4.635ms
8 ATE0<CR><LF>

Overlapping sequences are marked with an OVERLAP (line 7). This means that a new datagram was received although the current one was not yet finished.

Please note, that all information are regarded while copy and paste or exporting. This is important if you process this time information by external programs, which require a certain time format.
If you are interested in the datagrams only you can fade off the display of date and time completely.

Change the font
Besides the date and time representation of each sequence you also can alter the font type, e.g. to use a font with letters of equal width instead of the default proportional font or to adapt the letter type and height.
Click on Settings→configure protocol monitor to open the settings dialog.
The chosen font type is automatically stored.

13.6 The Toolbar
The toolbar serves for a fast access to the most used functions. Some are identical in all monitor windows, some are specific for the protocol monitor.
13.8. SOME EXAMPLES...

A  **End:** Saves all settings and closes the window.

B  **Display mode:** According to the mode the window either shows always the current (last recorded) event or locked or actualizes its content synchronous to the other windows.

C  **Data direction:** The protocol monitor can display both data directions (Data channel A and Data channel B) combined or separately to display them in different windows.

D  **New view:** Opens a new window with the same sector and settings.

E  **Apply template:** Applies the current template on the available data.

### 13.7 Short commands

<table>
<thead>
<tr>
<th>Action</th>
<th>Short command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online help for the Protocol view</td>
<td>F1</td>
</tr>
<tr>
<td>Apply template</td>
<td>F5</td>
</tr>
<tr>
<td>Save current template</td>
<td>Ctrl+S</td>
</tr>
<tr>
<td>Load template from file and apply</td>
<td>Ctrl+O</td>
</tr>
<tr>
<td>Open and close template editor</td>
<td>Ctrl+T</td>
</tr>
<tr>
<td>Select all protocol lines</td>
<td>Ctrl+A</td>
</tr>
<tr>
<td>Reverse selection</td>
<td>Shift+Ctrl+A</td>
</tr>
<tr>
<td>Copy selected lines into clipboard</td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Export selected lines</td>
<td>Ctrl+E</td>
</tr>
<tr>
<td>Save settings and close protocol view</td>
<td>Ctrl+Q</td>
</tr>
</tbody>
</table>

### 13.8 Some examples...

The ‘theoretical’ description of the protocol monitor is now complete. Now it is time to put this knowledge into practice. Since the input of the template (the separation of the datagrams as well as their output) can be done in an interactive way (you immediately see, how template changes have effect on the representation) we now turn to your first example.

You simply can read the following chapter or additionally start the analyzer program and load the appropriate sample recording `modem.msblog` in the folder `examples/ProtocolView`. 
CHAPTER 13. THE PROTOCOL VIEW

If you just like to see the running protocol template, open the already finished project file `modem.msbprj`.

Modem Communication with echo

This record is the matter of a simple communication between PC and modem. Open the protocol monitor and select the template change data direction from the template list. The monitor now interprets all data received at one port or channel as a sequence. Each datagram starts and ends with a change in the data direction (line 1).

Data from Data channel A are red colored (line 3 to 5), data from Data channel B are blue (line 8 to 10). In addition the ASCII control characters are displayed with their name and a slightly different color (lines 4 and 9). All data in the range from hex 80 to hex FF are displayed in hexadecimal notation (lines 5 and 10). The respective template is named:

```c
1 in={CAB());
2 aout={
3 ASCII(20,7F,red) +
4 CTRL(orange) +
5 HEX(80,FF,red)
6 };
7 bout={
8 ASCII(20,7F,blue) +
9 CTRL(steel blue) +
10 HEX(80,FF,blue)
11 }
```

At first the modem the modem works in a echo or terminal mode and sends all received data bytes back as echo. A division of the datagrams based on changes in the data direction makes no sense, because all send and receive data are mixed as can be seen in the picture at the side.

The first step is to define the end of the datagram. In this case it is easy. Each sequence is closed by a carriage return / linefeed (hex 0D0A). So we replace the current template by the carriage return / linefeed template from the selection list.

That looks much better and is sufficient for the most cases. But as this is a matter of an example we want to play a bit around with the visual appearance of the output.

The selected template is a predefined and unchangeable definition. So expand the template editor and you will notice, that the template text doesn’t show any caret to mark, that it can not be edited.

So at first you have to add a copy of the current template. Click on the button next to the template list and enter the name `Modem`.

This copy is a new template in the list and can be changed at any time. We now use this feature to do some adoptions.

Expand the protocol editor with Ctrl+T and click into the white background to activate the text cursor.

---

94
13.8. SOME EXAMPLES...

Instead of the Alias names `<CR>` and `<LF>` you like to have a 'cr' and 'lf'. In addition both characters shall be a bit more highlighted. End characters, send to the modem, shall be on red background (Data channel A), received ones from the modem (Data channel B) shall be displayed on blue background.

With the ALIAS (P.87) function you can assign every data byte any character string. The lines 4,5 and 10,11 show the respective modification of the original template.

```plaintext
1  in=(EOS(0D0A));
2  aout={
3       ASCII(20,7F,red) +
4       ALIAS(0D,cr,white,red) +
5       ALIAS(0A,lf,white,red) +
6       CTRL(orange) +
7       HEX(80,FF,red)
8  };
9  bout={
10      ASCII(20,7f,blue) +
11      ALIAS(0D,cr,white,blue) +
12      ALIAS(0A,lf,white,blue) +
13      CTRL(steel blue) +
14      HEX(80,FF,blue)
15  };

Press F5 to apply the template!
If you did not make an writing error you see - nothing, no change !
And that is correct. Please review the definition. After both ALIAS functions a CTRL call follows which overwrites our ALIAS functions and sets the default ALIAS definitions.

Since in this example no other control characters other than CR and LF occurs the CTRL function can be totally removed or has to be set in front of the ALIAS definitions (line 4 and 11).

With this the predefined control characters are activated with their names and then replaced by our own Alias and color selection.

```plaintext
1  in=(EOS(0D0A));
2  aout={
3       ASCII(20,7F,red) +
4       CTRL(orange) +
5       ALIAS(0D,cr,white,red) +
6       ALIAS(0A,lf,white,red) +
7       HEX(80,FF,red)
8  };
9  bout={
10      ASCII(20,7f,blue) +
11      CTRL(steel blue) +
12      ALIAS(0D,cr,white,blue) +
13      ALIAS(0A,lf,white,blue) +
14      HEX(80,FF,blue)
15  };
```
S-Record data transmission

In this example it is less about the partition of the single diagrams but about the formatting possibilities for the representation of the data sequences. Open the srecord.msblog record from the folder examples/ProtocolView and open the protocol monitor. You will find the already defined template as project file srecord.msbprj. The record contains a log of a S-Record data transmission in a microcontroller and consists of single in this format defined data sequences, finished with a linefeed.

As the original template we choose the predefined Linefeed* template from the selection list. The S-record format can be found with a good description at wikipedia. Therefore we restrict to the most important information. A S-record consists of a sequence of datagrams which have the following structure:

```
S 0 6 0 0 0 4 8 4 4 5 2 1
S 1 1 3 0 0 2 8 5 F 2 4 5 F 2 2 1 2 2 2 6 A 0 0 4 2 4 2 9 0 0 0 8 2 3 7 C 2 A
S 1 1 3 0 0 1 0 0 0 2 0 0 0 8 0 0 8 2 6 2 9 0 0 1 8 5 3 8 1 2 3 4 1 0 0 1 8 1 3
S 1 1 3 0 0 2 0 4 1 E 9 0 0 0 8 4 E 4 2 2 3 4 3 0 1 8 2 3 4 2 0 0 0 8 2 4 A 9 5 2
S 1 0 7 0 0 3 0 0 0 1 4 4 E D 4 9 2
S 5 0 3 0 0 0 4 F 8
S 9 0 3 0 0 0 0 F C
```

Every data set starts with the start code ‘S’ and a number ‘0’ to ‘9’ which represents the record type. Then two number follow with the length of the data set in hexadecimal notation. Then a 4 digit address, also in hex. The rest of the data set consists of the data and a checksum byte at the end. Our aim is to display the recorded sequences appropriately colored.

With the selection of the linefeed* template the record is correctly divided into the single datagrams. An visual partitioning is not yet done. Every data sequence it acknowledged by the receiver with a <ACK> (Data channel B). We restrict to the stream, recorded at Data channel A.

At first we copy the linefeed* template by clicking onto the icon next to the selection list and call it srecord.

S-record files do not contain other control characters than the line feed at the end. Therefore we can remove the HEX function to simplify the template.

```
1  in = {EOS(0A)};
2  aout = {
3      ASCII(20,7F,red) +
4      CTRL(orange) +
5  };
6  bout = {
7      ASCII(20,7F,blue) +
8      CTRL(steel blue)
9  };
```

With the exception of the start code all characters are in the range '0' to '9' and 'A' to 'F'. To highlight the start code it is sufficient to insert a simple ASCII call. We replace the ASCII definition in line 3. #FFC corresponds to a light yellow in short RGB notation. Likewise all characters are reset to the default black text color.
13.8. SOME EXAMPLES...

Record type, data length and address can be differed from each other not by their value but by their position in the data stream and are a matter for the \texttt{REPEAT} command (Page 90). \texttt{REPEAT} allows any repetition of the preceding output functions. To color the S-record format we extend the template as follows:

\begin{verbatim}
1 in={EOS(0A)};
2 aout={
3 ASCII(53,53, black,# ffc ) +
4 CTRL(orange) +
5 };
6 bout={
7 ASCII(20,7f,blue) +
8 CTRL(steel blue)
9 };
\end{verbatim}

As the \texttt{REPEAT} command starts counting from the first character of the data-gram the first \texttt{ASCII} command in line 3 also has to be followed by a \texttt{REPEAT}. Otherwise both \texttt{ASCII} commands would be repeated just one time and only color the ‘S’ because the second \texttt{ASCII} call regards only the value ‘0’ to ‘9’. The principle should now be clear. To highlight data length and address we insert two further \texttt{ASCII} calls with the appropriate \texttt{REPEAT} command (lines 5,6). Line 7 defies the colors for the rest of the data and is not repeated, especially as the length is not constant.

Line 8 supplies colors to the linefeed. Now the template is ready - nearly..

\begin{verbatim}
1 in={EOS(0A)};
2 aout={
3 ASCII(53,53, black,# ffc ) + \texttt{REPEAT}(1) +
4 ASCII(30,39, black,#fcc) + \texttt{REPEAT}(1) +
5 CTRL(orange) +
6 };
7 bout={
8 ASCII(20,7f,blue) +
9 CTRL(steel blue)
10 };
\end{verbatim}

Scroll to the end of the record and you will find a single linefeed, which is displayed as a dot only. What happened?
CHAPTER 13. THE PROTOCOL VIEW

Look at the template. The repeat command is ‘greedy’ and steps forward in the data sequence with every repetition, independent if data are available or not. The sequence 82 (received at Data channel A) contains a single linefeed only where the repeat command does not fit.

To include this case we have to add another CTRL command in preceding to the first REPEAT command in line 3:

```plaintext
1 in = {EOS(0A)};
2 aout = {
3     CTRL(black, coral) + ASCII(53, 53, black, #f3c) + REPEAT(1) +
4     ASCII(30, 39, black, #f3c) + REPEAT(1) +
5     ASCII(30, 46, black, #c6c) + REPEAT(2) +
6     ASCII(30, 46, black, #c6c) + REPEAT(4) +
7     CTRL(black, coral) +
8     ASCII(20, 7f, blue) +
9     CTRL(navy) +
10    }
11 };
```
The MSB-RS485 Analyzer samples all signals with up to 16 MHz. The result displays the signal monitor. Analogous to a digital scope you can select any sector and examine in different magnification levels.

For analyzing of serial data streams it is sometimes not sufficient to watch the transmitted data bytes.

Especially bus systems need a smooth interaction of all components. This requires a correct parameterization and strict observance of the protocol specifications by all bus participants. The possible error reasons are manifold.

By wrong settings or a malfunctioning hardware invalid data can be set onto the bus.

Occurring data collisions by multiple simultaneously active senders (Bus blocking/unblocking) or invalid data sequences (frames), caused by wrong timing, can not be judged only by recording the data. The same applies for the hardware handshakes.

All by the MSB-RS485 recorded lines are displayed in parallel, whereby each line signal can be individually switched on and off and the sequence of their display can be varied. The function of the signal client is that of a 8 channel digital scope. In the opposition to a scope the recording depth (Record depth) and the duration of the recording is limited only by the disc capacity and computing power of your PC.

By opening of multiple signal clients you can check the recorded signals at different places with different time resolution. Aside that the signal client is also well suited to judge the response time of sent data bytes. In the easiest case the signal client shows level changes of an active data connection and provides important hints for the function or malfunction of the connection.
14.1 Signal representation

The signal display is divided into 3 sectors. Directly below the toolbar the cursor bar is located (look Cursor) and the timeline. The timeline provides you with the exact position and resolution of the visible signal section. To ease the readout all times displays are shortened by removing unnecessary prefixes. So 0.012570s changes to 12.57ms.

Below the timeline the signals are displayed. For all signals the same sector and the same resolution applies (time base). To examine a signal at different positions simply start a new signal client. You can duplicate the actual client by pressing the 'Clone' button in the toolbar. By this a new signal client will be started which has exactly the same settings like the actual one. Or you start a signal client with default settings from the control program.

Compare different signal sections

Different signal regions can be examined through multiple signal clients. Just open as many as you wanted.

Each visible signal is described by its name at the left border. The signal name is set before the recording in the preference dialog of the control program. The sequence in which the signals are displayed can be set in each signal client individually. That makes it possible to arrange important signals directly on top of each other. Unimportant signals can be fade out. All signal specific settings are done in the Settings dialog, which is opened from the Options→Preferences menu.
14.2. NAVIGATION

**Individual signal settings**
Signal sequence, visibility, resolution and color are set in the signal dialog!

The visible signal region is defined by its position as time difference from the beginning of the recording and its visible sector. The visible sector is derived from the size of the window counted in screen pixel and the time base, that means how many microseconds are displayed per Pixel. The greater the time base the bigger is the time window of the signal sector.

To get a complete overview over the recorded events select 'View total' from the 'View' menu or press Ctrl+Pos1. The **Timebase** is automatically choosen so that all events can be seen at the same time. Depending on the selected time base multiple events could have occurred in one screen pixel. The signal client draws a vertical line instead of a single pixel to mark this time. The following image shall make this behaviour clear:

![Timebase 0.5ms](image1)

![Timebase 50µs](image2)

![Timebase 10µs](image3)

All three pictures apply to the same signal, but displayed with decreasing time resolution.

14.2 Navigation

The scroll bar below the signal windows represents an overview over the position and size of the displayed sector in comparison to the complete signal. The slider size of the scroll bar represents the size, the slider position the offset of the displayed sector.

Beside that the scroll bar allows to navigate through the complete signal. The arrows at the left and right border scroll the signal sector in grid or in 10 grid steps. Or the sector can be moved with the slider. Also the signal can be moved with the arrow keys, look Shortcuts.

Position and zooming (Time base) are displayed in the two left status boxes. Below the scroll bar you see a slider, with which you can vary the signal hight of all signals. Normally you will not need this function. It is useful if you can not read the displayed name of a Region when it is hidden by the signal. In this case simply decrease the signal hight.
CHAPTER 14. THE SIGNAL VIEW

Navigation by mouse wheel
The navigation by mouse wheel offers a way to shift the signal. Keep the Ctrl key pressed while turning the wheel. Depending on the turning direction the signal is shifted to the right or to the left in 10 grid steps. If the ctrl key is not pressed the signal is zoomed or unzoomed.

Shift with the hand cursor
The hand cursor allows the pixelwise shifting of the signals. Click on the hand symbol in tool bar. The cursor changes to a hand symbol. To move the signal to the right or to the left simply grip the signal by pressing the left mouse key and drag the signal in the desired direction. Keep the mouse key pressed. While gripping the signal the cursor has the look of a griping hand.

14.3 The time base
The time base correspondends to the magnifying for the represented signal. The smallest time base is 10µs, that means 10 microsecs per raster and means 1µs per displayed pixel. One raster grid is 10 pixel wide. The signal is magnified, for a screen resolution of 1024 it is about 1 millisecond (if you have maximized the signal monitor window).

If the level changes are in the milisec or second range you will choose a higher timebase to watch a larger section of the signal. By clicking of the two magnifying glass symbols in the toolbar the time base is set to the next higher or lower value. The same can be done by using the key combination Ctrl+Up Arrow and Ctrl+Down Arrow.

You also can magnify a certain sector of the signal by selecting the sector with a pressed left mouse key. Move the mouse cursor to the beginning of the sector and press the left mouse key. Hold the key pressed and move the cursor to the end of the section. A rectangle marks the current selection. As soon as you release the mouse key the section is displayed magnified.

14.4 Undo and Redo
All magnifications can be taken back (undo) or redo after an undo. By that you can magnify an interesting signal section, for example to place a cursor exactly, and go back to the original view, simply by clicking on the undo symbol in the toolbar or entering Ctrl+Z.

The original view before an undo is recalled by redo. Both symbols are marked as inactive if no further undo/redo steps are possible. Undo and redo are used only for section magnifying. A normal increasing/decreasing of the signal is possible at every time so that no undo redo is necessary.

14.5 Settings dialog
In the setup dialog you can adapt the signal display to your needs in broad range. This applies to the number of displayed signals as well as to the sequence and color. Aside that the performance of the display can be influenced. From Version 2.1.1 you can additionally fade the recorded data bytes (receive and send data) in into any signal to compare them with the physical signals.
14.5. SETTINGS DIALOG

The signal dialog
In the signal dialog all available signals are listed and their actual settings are displayed. The signal with blue background is the actual selected one. First you have to select a signal with your mouse then you can change its attributes.

Visibility and grid can be switched on or off by clicking the respective symbol. To change the colors of a signal click the signal symbol on the right side. In the opening color dialog you can select signal color, background color and grid color from 72 colors.

Choose your favored color and click on the corresponding button for paper, signal and grid to actualize the color. The changes are directly actualized in the signal client so that you can see immediately the changes you have made. If the changes are ok accept with ‘Ok’ while ‘cancel’ restore the old settings.

Save your color settings
Don’t worry about your color settings. All properties will be saved automatically after close the dialog and will be present also for future sessions.

Signal inverting
Each signal can individually be inverted. By default the representation is as recorded by the analyzer, symbolized by a buffer symbol. Click with your mouse onto this symbol to invert the display. The signal is now mirrored horizontally. In the table now an inverted driver symbol appears.

Please note, that this inverting applies to the display only.

Signal sequence
The Signal sequence is freely definable. At first select the signal whose position you want to change. Then move the signal with the keys ‘Up’ and ‘Down’ of the signal dialog. The setting directly apply to the signal client so that you can check the settings immediately.

The key ‘Defaults’ cancels your changes and resets the colors to default values.

Fade in the transferred data
Independent of the recording of the physical signals you can fade in the transferred data bytes into any signal. You only have to activate the recording of
CHAPTER 14. THE SIGNAL VIEW

RxData and TxData in the control program. The display is done as a data block with the hex value of the data byte.

Grafical effects
The display of the selected range and fade in region was made as background color. This had some disadvantages.

Depending on the selected display colors the selected range or region was hardly to see. Additionally the start and end of overlapping region could not be seen.

With introduction of transparent ranges for selection and regions the display is now independent of the colors for signal, grid and background. Selection and regions now appear transparent and allow a more user friendly inspection and marking of signals.

However, transparent display has a disadvantage. They are not for free and cost additional performance. That should not be an issue on modern PCs and is limited to the display of regions only. If the performance degree is too obvious you can switch the transparent display of in the setup dialog (effects), separated for selection and regions. You also can adjust the transmittance of the transparent ranges.

14.6 Cursor operating
Every signal client owns 2 Cursors I and II which can be moved arbitrarily over the signal inside the visible range. To move a cursor click on the respective cursor symbol, an upside down triangle above the timeline, and draw the cursor line to the wanted position. If both cursors are on the same position you can see only the last activated one because it overlays the other cursor. But in this case always Cursor I will be activated. To move the second cursor keep the SHIFT button pressed while clicking the cursors. Now you can move cursor II while Cursor I stays at its place.

Cursor selection
With pressed SHIFT key the second cursor is activated if both cursors are at the same position!

Placed cursors keep their signal specific position even if you choose another signal view. Cursor outside the visible signal window are displayed at the left or right border. Their actual position can be read in the status line. c1 means cursor I and c2 cursor II.

In addition to the position of each cursor their time difference is fade in in the status line. So a time difference measurement is easily posible, e.g. the
14.7. SYNCHRONIZING

duration of an active line.
To compare multiple sectors you can assign the marked signal sectors to a region. Click the ‘Add Region’ Button in the toolbar. A maximum of 8 regions can be defined. The range between both cursors gets colored. Read more about regions in chapter Regions.
You can also move both cursors at the same time, for instance to compare the duration of two signal changes which did not occur at the same time. Both cursors are connected by pressing ‘c1+c2’ in the toolbar.
As long as this key is activated both cursors are moved simultaneously, all the same which one you move.

Signal selection
The sector between both cursors represent the current selection at any time. You do not have to make another selection. Since both cursors do not change their position in relation to the signal, the position and distance between them stays the same, even if the signal view is changed. Both are displayed in the status line.
All operations, which are related to the current selection, always concern the signal range between both cursors.
To define a region click on the ‘+’ Symbol in the toolbar or press Key F4.

Regions
Analogous to the other monitors regions have a colored background.
The picture shows two regions, where the light blue one is framed by the two cursors and assumedly selected by them. Because regions are superordinated and valid for all analysis windows, selected signal sectors can be marked and examined with different tools at the same time.
The red-yellow triangle in the cursor bar is the current Synchronizing Event, received from another analysis window.
You can find more information at P.109.

14.7 Synchronizing
Each analysis window can synchronize its view with others.
How the signal monitor acts on receiving of the sync signal from other analysis tools depends on the sync selector in the toolbar, identical for every analysis tool.
By default, the display of the signal monitor is locked, it does not react on change commands from other tools. Click on the ‘Sync’ symbol and a red-yellow triangle appears in the cursor bar which marks the position of the current synchronizing event.
If you switch on the ‘Scroll’ Symbol the signal monitor always shows the last event resp. level change.
The signal monitor not only reacts on sync-changes from other tools but can also trigger a sync event itself... For that click in the signal view the right mouse key (context menu) to open the sync. menu. The entries are more or less self explaining.
1 Synchronizing on Cursor 1
Synchronized is on the first event after the cursor 1 position.
CHAPTER 14. THE SIGNAL VIEW

2 Synchronizing on Cursor 2
Synchronized is on the first event after the cursor 2 position

3 Synchronizing the display
As the synchronizing event the current signal sector is used. That means the first level change seen from the left border.
Why is the first event after the cursor position used and not the cursor position itself?
Synchronization is only on events not on a certain time in the signal. As the cursor can be between two events (in contrary to other tools) the next following event has to be taken for synchronization.

14.8 The toolbar
The toolbar serves for a fast access to the most used functions. Some are identical in all monitor windows, some are specific for the protocol monitor.

A End: Save all settings and close the signal monitor window.
B Display mode: According to the mode the window either shows always the current (last recorded) event or locked or actualizes its content synchronous to the other windows.
C New view: Opens a new window with the same sector and settings.
D Mouse control: Optionally the mouse can be used to zoom the selection or to move the signal (hand symbol).
E Signal zooming: Magnifies or demagnifies the visible section in 1, 2, 5 multiplicative factors by choosing the next lower or higher time basis.
F Undo/Redo: Undoes the last change of the visible section or restores it respectively.
G Add region: Saves the range between both cursers as a new region.
H Interlock Cursor: The cursors can be selectively moved singly or together (combined).
I Show region dialog: Opens the region dialog, e.g. to fade in or off regions, to delete them or to name them.
## 14.9 Short keys

<table>
<thead>
<tr>
<th>Action</th>
<th>Short command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online help for the Signal View</td>
<td>F1</td>
</tr>
<tr>
<td>Undo last zooming operation</td>
<td>Ctrl + Z</td>
</tr>
<tr>
<td>Redo last undo operation</td>
<td>Ctrl + Shift + Z</td>
</tr>
<tr>
<td>Add range between cursors as new region</td>
<td>F4</td>
</tr>
<tr>
<td>Move view 1 raster towards signal end</td>
<td>Right arrow</td>
</tr>
<tr>
<td>Move view 1 raster towards signal start</td>
<td>Left arrow</td>
</tr>
<tr>
<td>Move view 10 raster towards signal end</td>
<td>Shift + Right arrow</td>
</tr>
<tr>
<td>Move view 10 raster towards signal start</td>
<td>Shift + Left arrow</td>
</tr>
<tr>
<td>Upsize view</td>
<td>Ctrl + Up arrow</td>
</tr>
<tr>
<td>Down size view</td>
<td>Ctrl + Down arrow</td>
</tr>
<tr>
<td>Complete view</td>
<td>Ctrl + Home</td>
</tr>
<tr>
<td>Open in a new window</td>
<td>Ctrl + Shift + N</td>
</tr>
<tr>
<td>Save settings and close signal view</td>
<td>Ctrl + Q</td>
</tr>
</tbody>
</table>

**Short commands**

for the most important functions
To save and quickly recover interesting areas in the recorded data these areas can be marked as regions. Regions are present in all views so that a region, marked in the data monitor is shown in signal monitor too. Regions can directly accessed.

Regions are selected Ranges which are shown in all analysis windows. Each window can define a selected range as a region and make it available to other windows. Since the different analysis tools represent different kinds of data views each region can be defined in a different way. A region can be defined in the signal monitor as the selection of a certain signal sector, in the data monitor as a certain data sequence or as the occurrence of single characters. This interaction is interesting if you want to examine a defined section in a different view, for example the physical signal state (signal monitor) of a data sequence (data monitor).

As soon as you add a selected range as a region this one is attached to the list of available regions. Regions are part of each recording and therefor are saved with them self-acting.

The MSB software allows the definition of 8 regions. Every region can be individually named and optionally switched on and off. With exception of the line state monitor you open the region dialog with View→Region dialog in every analysis window. An always opened region window will be put in front of all windows automatically.

The picture above shows a region dialog with all together 6 regions where region nr 2 with the name first request is switched off and therefore not shown in the other analysis tools (indicated by a closed eye).
CHAPTER 15. REGIONS

15.1 Switch regions on/off
Each region can be individually fade in or out. This is reasonable if some regions overlap in the display and make an assignment difficult. To alter the visibility condition of a region simply click onto the eye symbol at the very left side. An open eye symbolizes a visible region, a closed eye a fade out region respectively.

15.2 Remove a region
Delete deletes the current in the region dialog selected region. A further enquiry is not made but you can undo the deletion each time as long as the region dialog is open. Just click the Undo button in the toolbar.

15.3 Rename a region
By default regions have no name, but you can choose a remarkable name for each one. Mark the wished region with the left mouse key and click on the edit name field in the toolbar. The name of a region allows any character except for the colon ':' and is limited to 64 characters. Enter adopt the name to the selected region.

15.4 Move regions into view
Certain segments of the recording are marked as region because they are important parts. Of course you want to bring them fast and easy into the visible part of your analysis windows, e.g. the signal or data monitor. Possibly you want to compare two regions. Therefore the region dialog supports the same mechanism for synchronization like the other analysis windows with the exception that it can initiate the synchronization only. Select the appropriate region and click on the start value to bring
15.4. MOVE REGIONS INTO VIEW

up the start of the regions in all analysis windows with activated synchroniza-
tion. Or click on the end value to bring up the right limits of the region.
Please note: Only those analysis windows will react which have the synchro-
nization active.

Fetch regions into views
Regions can easily be brought into the visible range of an analysis window by
a click onto the start or end value with the left mouse key.
Lua is one of the fastest scripting languages in the world. Because of it's small and simple design it's also easy to learn. Lua contains a few but also more powerful concepts which makes it the first choice to add the benefits of a scripting language to the analyzer software. This chapter will give you a first glimpse of the language and how it fits with your analysis.

16.1 Getting started

The most descriptions of a new computer language just begins with the traditional "Hello World". To keep the tradition, our first script will do the same.

Open the Data Monitor and expand the Watch expressions on the bottom of the program view. The Watch window contains eight (at first empty) entries. Each one shows the result of the according Lua script. To start with our little "Hello World" example, double click the first line in the watch list which opens the integrated editor. All you have to do is to input the following two lines:

1 require "dv"
2 dv.watch("Hello World")

Now let’s go and see what happens. The internal Lua engine will translate this expression into it’s byte code and execute it after you switch to the watch list again by click on the Watch tab. Later you will see, that each expression will also be executed at every cursor movement, receiving synchronize trigger or after update a find request. But for this time we keep it as simple as possible. If you don’t have made any typo the first watch items should display:

--> Hello World

Don’t worry, if you made some mistake. In this case, the watch window will show you some informational message about the error.
CHAPTER 16. A QUICK START WITH LUA

So what's going on here?
The origin Lua doesn't know anything about the analyzer or the watch window of the Data Monitor. Therefore we need some 'glue' to put both things together. One of this glue is the dv (Data View) library or module. It provides all necessary output functions to print the results of your scripts in the watch list or to mark the data in the grid window.

To load a module, you simply call require"modname". After that you can access each function provided by the module by prefix the module name with a dot like dv.watch.

In normal case you always have to load a module before you can use it. But for all analyzer specific modules and also the basic libraries coming with Lua we decide to 'preload' them into the Lua engine for an easier handling. So the require"dv" is obsolete and we will dispense with it in the following examples.

You can add as much arguments to the watch function as you like. Because the arguments will be displayed one after another you should also add some separator between them. So let us modify the example above to:

```
1 dv.watch( 1+1 , " is less than ", 10/3 )
```

--> 2 is less than 3.3333333333333

As you can see, we are now calling the dv.watch function with three complete different arguments separated by a comma. The first is the result of the addition of two integer numbers. The second is a text string identified by its enclosing double quotes. And the last one is the real number 3.333...

But you don't have to bother about the 'type' of each variable. Lua is a dynamically typed language and handles each type automatically\(^1\).

That's nice, but you may ask what's the benefit for the Data Monitor? Read on!

16.2 Accessing the Data Monitor

With Lua you can compute every data displayed by the Data Monitor. It is the aim of the Data Monitor module to provide you with functions to query the cursor position and also the information about each cell in the grid. The next little script shows the current position of the cursor:

```
1 n = dv.cursor()
2 dv.watch( n )
```

The function dv.cursor() returns the current cursor position starting with 1 (the upper left cell in the grid) and goes up to the bottom right cell in the data grid. As soon as you move the cursor or click into any cell, the watch entry is updated with the new position.

\(^1\)Lua supports numbers (double-precision floating point), strings like our "Hello World" and so called tables (which are a very mighty concept to realize almost all typical data structures such as arrays, sets, lists, and records. We will discuss the tables later)
After this little introduction it’s time for a really important function:

dv.cell(index) returns all necessary informations about the given grid entry like the recorded data byte, the timestamp, the direction and something more in a table.

Don’t worry about the item ‘table’. For a first explanation consider a table as a container with several cases. Each entry is accessible via its name.

The following example shows what we mean:

```lua
1  c = dv.cursor()
```

For a better output, we have to put an additional description in front of each information.

Now move the cursor in the data grid with the arrow keys of your keyboard or click on any data cell. If you haven’t closed the watch window you will note that every time you change the cursor position, the watch item is updated with the new information computed by this little script.

Line breaks play no role in Lua’s syntax. Therefore we can arrange the both arguments of the dv.watch function in separate lines to make it more readable.

```lua
1  c = dv.cursor()
2  dv.watch(
3      "Data: ", dv.cell(c).Data,
4      "Time: ", dv.cell(c).Time
5 )
```

As mentioned above the function dv.cell returns all informations in one table. The entry ‘Data’ contains the recorded data byte at the given cursor position, the entry ‘Time’ the timestamp of this data event in seconds.

The syntax for accessing table entries is similar to the modules, i.e. a dot following by the name of the item.

You will find a more detailed description about the data table and all its fields in the following chapter 17.12. For now we continue with another very useful function of the dv module.

Take your colour pencils, we are going on and colorize the grid.

### 16.3 Mark sequences in the data grid

Imagine you can visualize one or more specific data sequences. For instance to highlight a protocol frame or a range of data starting by the cursor position. All you need to know is the function: dv.mark(index, length, colour)

For the first we want to show the next four data beginning by the cursor as a 32 bit integer (big endian) and also mark them in the grid.
Voila - that's all. As you can see the next four data cells directed by the cursor are highlighted and the result of our little computing is shown in the watch window.

The `dv.mark` function is independent of the current cursor position. Therefore we have to call the function explicitly with the index of the cell where the cursor was located. This will give you the big advance to mark also cells without pending on the cursor movement. The following few lines mark all cells with an error in a warning red background.
17

Lua beginners guide

In the last chapter we gave you a quick overview what's possible with Lua without any deeper details about the Lua language syntax and all its mighty qualities. Lua is a programming language that offers a very impressive set of features while keeping everything fast, small and simple. So let's go to learn a little bit more about this amazing scripting language.

Each programming language comes with its own ingredients like operators, keywords, functions and last but not least some rules how you put this things together. This is called the programming language syntax. The language syntax declares, how a program has to been written correctly. In this chapter we will give you a short overview about the Lua language, the supported operators, keywords and some helpful additional modules (libraries) we have integrated in the embedded Lua by default.

Please note! Each time we need an output of some Lua script computing we are using the Data Monitor and its dv (Data View) Module. If you like to give each example a trial just start the MSB-RS485 program, open a Data Monitor and enter the code in a watch entry.

17.1 Lua is case-sensitive
First of all: Lua is a case sensitive language. while is a reserved word (a so called keyword), but WHILE or While are two other identifiers denote a variable or function. Because this is the common use in the most modern languages it shouldn't bewilder you much.

17.2 Whitespaces and line ends
Lua ignores any whitespaces (like the space or tab characters) if they aren't part of a string constant (see 17.4). It also doesn't worry about the indentation like Python, therefore you can format your code for your own purpose (or just make it more readable).
Lua doesn't use any special line end and line breaks play no rule in the Lua syntax. The Lua interpreter detects the end of a statement automatically therefore a line can contain more than one statement and a statement can also be split into several lines.
CHAPTER 17. LUA BEGINNERS GUIDE

If you write several statements in one line, you can use the semicolon as a separator.

```lua
1 x = 1  y = 2  --> not very readable but ok
2 x = 1;  y = 2  --> better
3 z = x
4 +
5 Y  --> z = 3
```

17.3 Comments

A comment in Lua starts anywhere with a double hyphen -- and runs until the end of the line. It’s also helpful if you want to exclude some lines from execution.

More than this. Lua provides also a block comment which starts with --[[ and runs until the corresponding --]]. It makes it very easy to comment or uncomment several lines as we will show in the following:

```lua
1 x = 1
2 --[[
3 x = 10
4 --]]
5 dv:watch(x)  --> 1
```

To uncomment the block, just add a single hyphen to the beginning comment. The starting and closing comment identifiers are now just like other commented lines and the statement between them will be executed as normal.

```lua
1 x = 1
2 --[[
3 x = 10
4 --]]
5 dv:watch(x)  --> 10
```

17.4 Types and values

Lua is a dynamically typed language. You don’t have to specify the type of a value, because each value carries its own type. Lua supports eight basic types but we contemplate only the following ones:

- number
- boolean
- string
- nil
- table
- function

It is common use to define most of the types also as a ‘constant’ value. A constant is a ‘hard coded’ value in your program which isn’t a result of any computing. Constants are numbers (integer and floating point numbers, whereat Lua
17.4. TYPES AND VALUES

doesn’t distinguish between them), strings and the boolean values false and true.

**Numbers**
Lua simplify the use of different numbers like integer, single float, double float by using only one kind of type for each numbers. Numbers in Lua are always double precision floating point numbers and were converted automatically.

```
1  dv.watch( 1 )   --> 1
2  dv.watch( -12 )   --> -12
3  dv.watch( 100000000000 )   --> 100000000000
```

Notice that the numbers are never rounded into integers to. Hence:

```
1  dv.watch( 10 / 3 )   --> 3.3333333333333
```

**Hexadecimal constants**
Despite the fact, that Lua compute exclusively with floating points you sometimes want to use other number bases like hex.

```
1  dv.watch( 0x1234 )   --> 4660
```

**Floating point constants**
Lua can understand also exponent types for expressing numbers. Therefore you can write numeric constants with an optional decimal part and an optional decimal exponent like:

```
1  dv.watch( -0.05 )   --> -0.05
2  dv.watch( 10E-2 )   --> 0.1
3  dv.watch( 1.25E+6 )   --> 1250000
```

**Booleans**
A boolean data type according to the classical logical state and is either true or false. If a boolean value isn’t true, it has to be false and reversely. Boolean values are used to represent the result of logical or conditional operations.

```
1  dv.watch( 2 > 1 )   --> true
2  x = 2 < 4
3  dv.watch( x )   --> false
```

**Strings**
Strings in Lua has the common meaning, a sequence of characters. But Lua is, in opposition to other languages, eight-bit clean which has the great advantage: Strings contain characters with any numeric code, also a null byte (in C the string terminator). With other words: You can store any binary data in a string without an exception.

Strings can be defined using single quotes, double quotes, or double square
Why so different ways to specify a string? It allows you to enclose one type of quotes in the other. And: Double brackets have a few other properties like to suppress escape sequences as seen above.

**Escape sequences in strings**
Lua strings can contain the following escape sequences:

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>bell</td>
</tr>
<tr>
<td>\b</td>
<td>backspace</td>
</tr>
<tr>
<td>\f</td>
<td>formfeed</td>
</tr>
<tr>
<td>\n</td>
<td>newline</td>
</tr>
<tr>
<td>\r</td>
<td>carriage return</td>
</tr>
<tr>
<td>\t</td>
<td>horizontal tab</td>
</tr>
<tr>
<td>\v</td>
<td>vertical tab</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>'</td>
<td>single quote</td>
</tr>
<tr>
<td>\ddd</td>
<td>character with its numeric value ddd</td>
</tr>
</tbody>
</table>

The following examples show their use:

1. `dv.watch( "It\'s your code" )` ---`It 's your code`
2. `dv.watch( 'He says:"Hi"' )` ---`He says: "Hi"
3. `dv.watch( [[Hello
World]] )` ---`Hello\nWorld`

You can also specify each character in a string by its numeric decimal value through the escape sequence `\ddd` as mentioned above. For instance the binary sequence of the bytes 0...3 comes as: `"\000\001\002\003*`.

**nil**

`nil` is a special type and indicates a non-value. Each variable has a `nil` value before its first assignment by default.

1. `dv.watch( x )` ---`nil`

More than: Lua uses `nil` to specify the absence of a useful value (it doesn't exist anymore). By setting a variable to `nil` you can delete a variable.
Table 17.4. Types and Values

One of Lua's mightiest built-in datatypes is an associate array, which defines one-to-one relationships between keys and values. Key and values can be of each type. And more than this: Because functions are also just some kind of value, you are able to realize some object orientated behaviour with tables too, but this go beyond the scope of this chapter.

Tables has no fixed size and grow up as necessary. If you haven't use of a table anymore, you can throw it away with assigning \texttt{nil} to it.

Ok, that's enough for the first. Let's go on with a few examples to bring more light in this matter. At first we will create a simple list containing three line states as strings:

\begin{verbatim}
1 linesates = { "mark", "space", "invalid" }
2 dv.watch( linesates[2] ) —> space
\end{verbatim}

This statement in line 1 will initialize the first entry in the table \texttt{linesates[1]} with "mark", the second \texttt{linesates[2]} with "space" and the third with "invalid". Please note, that in Lua indexes start with 1 and not with 0 (like in C). The table here behaves like a simple list. You can append a new element to a table using the \texttt{table.insert(table,value)} function.

\begin{verbatim}
1 t = { 1, 2, 3, 4, 5 }
2 table.insert( t, 6 ) —> 1, 2, 3, 4, 5, 6
3 dv.watch( "Count: ", #t ) —> Count: 6
\end{verbatim}

The statement in line 3 uses Lua's internal length operator \#, see also section 17.8. If you like to insert a new item somewhere in between the list without having to shuffle the other elements around you can use the same function with an additional position parameter \texttt{table.insert(table,position,value)}.

\begin{verbatim}
1 t = { 1, 2, 3, 4, 5 }
2 table.insert( t , 4 , 44 ) —> 1, 2, 3, 44, 4, 5
3 dv.watch( "Count: ", #t ) —> Count: 6
\end{verbatim}

To remove an element from the table (or list) use the call: \texttt{table.remove(table,position)}.

\begin{verbatim}
1 t = { 1, 2, 3, 4, 5 }
2 table.remove( t , 4 ) —> 1, 2, 3, 5
3 dv.watch( "Count: ", #t ) —> Count: 4
\end{verbatim}

You can always replace one element with another one just simply by overwriting it. Each element in the list is accessible with the index operator \[]. To rewrite the second element with 200 \texttt{t[2]=200} will do the job. You can also query the value at a given position conversely with: \texttt{v=t[2]}. If there doesn't exist a value at the index position, a \texttt{nil} will return. On the other hand: If you try to overwrite a value at an invalid position it will append to the list.

In the examples above the keys of the associated array are set by default (as numeric) and only the values are given. But you can choose any desired index or key value too. Imagine a data type representing a point:

\begin{verbatim}
1 point = { x = 5 , y = 10 }
2 dv.watch( point.x , point.y ) —> 5 10
\end{verbatim}
A table storing data with a key/value relationship is sometimes called a dictionary.

Functions
We will discuss functions in this paragraph only according to their role as values. For detailed information about functions and their definition please take a look at section 17.10.

In Lua, functions are assigned to variables, just like numbers and strings. If you are bothering with the long term `dv.watch` assign it to the internal Lua `print` (the latter is without any use in the analyzer environment and can therefore be overwritten).

```lua
1 print = dv.watch
2 print( "Hello World" ) —> output in the watch window
```

Furthermore a function is a variable referencing the code of the according function and you can overwrite it with any other value.

17.5 Identifiers
In computer languages identifiers are names referencing some kind of variable or a function. Some identifiers are reserved by the language itself as so called keywords. (We already know the boolean keywords `true` and `false`). Others are built in functions like `print`.

Names (or identifier) in Lua can be any string of letters, digits and underscores, not beginning with a digit\(^1\). Valid names are:

```
x y ABC t1 _nm
aVeryLongVariableName the_last_result
```

Invalid names throw an error

```lua
1 dv.watch( 2n ) —> malformed number near ‘2a’
```

17.6 Keywords
The following keywords are reserved and cannot be used as names:

```
end false for function if
in local nil not or
repeat return then true until while
```

Please remember: Because Lua is case-sensitive, `and` is a keyword, whereas `And` and `AND` are just two other and different identifiers!

\(^1\)Because the analyzer software and Lua itself too reserves the starting `_` for some language supplements we recommend to start a variable name without an underscore.
17.7 Variables

Variables are like a named box that can store any kind of value. In Lua variables can cover a single number as also a million characters or a container of key-value pairs. The name of the variable has to be a valid identifier (see above). You don't have to declare a variable before the first use. As soon as the Lua interpreter finds a new variable it will create it automatically.

Assignment

Note! Before the first assignment to a variable, its value is nil. Assignment is the general procedure to set or change the value of a variable (or a table field).

```
if x == nil then
  x = 1
end

-->
```

As mentioned before: Lua is a dynamically typed language. You don't have to define the type of a variable because each value carries its own type. And: The type of a variable is an object of change. Every time you assign a new kind of value to a variable it change its type again.

```
x = 1
x = "Hello World"
```

```
-->
Hello World
```

Lua also supports multiple assignment which means: A count of values is assigned to a count of variables in one step. We will discuss this very nice feature in a later section in the context of functions with multiple results. For the curious reader here a little code example exchanging the values of two variables without any additional temporary variable:

```
x = 5
y = 10
x, y = y, x
```

```
-->
10 5
```

Global and local variables

There are three kinds of variables in Lua: Global variables, local variables and table fields (we discuss tables later).

By default each variable is a global one which means: It is accessible during the complete runtime. Global values resides in a 'global' space (in detail in a global table). Beside this local variables is only valid in the context or block where they are declared.

```
y = 10
if x == nil then
  local y = 5
  x = 1
end
```

```
-->
1 10
```

---

123
CHAPTER 17. LUA BEGINNERS GUIDE

It's a common strategy to use local variables wherever you don't like to access a global one. For instance if you need some variables only in a function, declare them as local.

17.8 Operators
Operators are symbols, which activate calculation, when using them in combination with variables, values or results from expressions. Lua supports arithmetic, conditional and logical operators. In addition a very helpful string concatenation operator.

Arithmetic operators
Lua supports the usual arithmetic operators: the binary + (addition), - (subtraction), * (multiplication), / (division), % (modulo), ^ (exponentiation) and unary - (negation).

```
+ - * / % ^
```

Please note: In Lua Numbers are always represented as real (double-precision floating-point) numbers.

Conditional operators
Conditional operators always result in true or false. Lua provides the following conditional (or relational) operators:

```
< > <= >= == ~=
```

The == operator tests for equality, the operator ~== is the opposite of equality. You can apply all operators to any two values, numbers and strings (all condition operators also dealing with strings). If the both values have different types, Lua handles them as not equal.

Please note: The value 0 isn't a false test condition as you may suspect from other languages.

```
1 dv.watch("abc" < "def") --> true
2 dv.watch(0 or true) --> 0
3 dv.watch(false or true) --> true
```

Logical operators
Lua provides logical operators for use in statements. They are: and, or and not. The logical operators behave in a common way. They always evaluate to either true or false. In a special case the value nil will be considered as false. and and or use a short-cut evaluation, means: They evaluate their second operand only when necessary. For instance:

```
1 dv.watch(4 and 5)         --> 5
2 dv.watch(4 or 5)          --> 4 (short-cut evaluation)
3 dv.watch(false and true)  --> false (short-cut evaluation)
4 dv.watch(a and 1)         --> nil, because a wasn't specified
5 dv.watch(not false)       --> true
```
String concatenation operator
The two dots .. denote the concatenation operator in Lua. The operator takes two strings (numbers will convert by Lua in strings) and combines them in one. Please note! If the first operand is a number you have to insert a space between the number and the .. operator. Otherwise Lua misinterprets the first dot as a decimal point and throws an error. Hence:

1. `dv.watch("Hello .. "World")` --> `Hello World`
2. `dv.watch("100 .. "sec")` --> `100sec`

The concatenation operator always creates a new string and leaves the operands behind without modifications.

The length operator
The length operator is denoted by #. The length operator returns the count of bytes in a string or the items in a table if the table doesn't have any gaps.

1. `dv.watch(#"Hello World")` --> `11`

Precedence
The following is a list of all Lua operators and their order of precedence. The operators are listed highest to lowest.

```
^ not # -(unary) *
/ % 
+ - .. < > <= >= ~= ==
and or
```

If in doubt, use explicit parentheses. It makes your code more readable and prevents you from any additional look in this manual.

17.9 Control structures
Control structures tell the program which way to proceed in the code (or script). They are integrated part of each language and something like the traffic police in Lua scripts.

Lua provides the following set of control structures, the if for conditional executions, for, repeat and while for iteration. All of them, except repeat, needs the explicit end terminator. repeat has to be closed with until.

if then else
The if statement tests a condition and executes depending to its result the then section or the else section. The later one is optional.
CHAPTER 17. LUA BEGINNERS GUIDE

1 if x < 0 then
2 x = 0
3 else
4 x = math.sqrt( x )
5 end

You can put small condition tests in a single line like:

1 function max( a, b )
2 if a > b then return a else return b end
3 end

Lua doesn’t have any switch statement. Therefore the following if, elseif chains are common.

1 if a >= 100 then
2 exp = 2
3 elseif a >= 10 then
4 exp = 1
5 else
6 exp = 0
7 end

while
The while statement executes a block as soon as the while condition is true. As usual the condition is tested first. The block will never execute if the first test results in false.

1 local x = 0
2 while x < 10 do
3 x = x + 1
4 end

repeat
On the contrary the repeat statement repeats its body until the condition is true. Because the test is done after the block, the block is always executed at least once. Please note the different terminator until.

1 local x = 0
2 do
3 x = x + 1
4 until x < 10

Numeric for
Lua provides two for statements but we confine ourself to describe only the first and more comprehensible numeric for. The numeric for has a variable with a starting assignment, an end value and an optional step value. The latter one is 1 by default.
17.10. FUNCTIONS

Every computer language has functions, even the simple ones. Lua is no exception. A function can perform a specific task and/or compute and return values. If you notice the plural values you are right - Lua functions are able to return multiple results.

In both cases you have to give the function a list of arguments enclosed in parentheses. If the function doesn’t need any argument, you still give it an empty list specified by ()

Function call
Simply said a function is called just by its name and an optional count of arguments as a list. You can invoke a function with more than the specified arguments whereas only the first are handled but if you try to call a function with fewer parameters you get an error. You learned about some already defined functions like the `dv.watch(...)` or the `math.sqrt(x)`. Most of this functions are part of some module, others are defined by the analyzer.

Function definition
Function are conventionally defined with the keyword `function`.

```lua
1 function fnc( arg1, arg2, ... )
2     --> the function body
3 end
```
For example a maximum function returning the greater value of two given numbers is defined as:

```lua
function max( n1, n2 )
  if n1 > n2 then return n1 else return n2 end
end
```

Function doesn’t have to return a value. In this case you can just omit the `return` statement or leave the `return` without any following value(s).

As mentioned above, a function in Lua can also return multiple results. This is a big advantage, because you don’t have to collect the results in some container and don’t run the risk of side effects by set up a global (outstanding) variable. Several predefined functions in Lua return multiple values like the `math.modf(x)` (one result is the integral part of x and the other one the fractional part of x). For instance:

```lua
dv:watch( math.modf( 5.125 ) ) −−> 5 0.125
```

The definition of a function with multiple results is as easy as of each other function. An example shows the differences. Imagine some function to convert coordinates of a plane polar system (radius and angle) into the cartesian system (x and y).

```lua
function polar2cartesian( radius, angle )
  x = radius * math.sin( math.rad( phi ) )
  y = radius * math.cos( math.rad( phi ) )
  return x, y
end
```

Instead of build some container for both results we just return them as a multiple result.

### 17.11 Modules

A module is a package of functions for a special purpose. You already know the modules `math` or the `dv` module belonging to the analyzer software. From the Lua point of view each module is a table which contains functions (functions are a special kind of value as we mentioned before), global module values, module constants etc. Therefore each module function is called like a table element with the prefixed table (module) name and a dot.

#### Standard Modules

The following standard modules are supported by Lua in the analyzer environment, see section Limitations 17.13.

In normal case you have to load a module first with the `require("modname")` statement before you can access any module function. But for the analyzer environment we decide to preload the following standard modules automatically for an easier use.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>math</td>
<td>The mathematical library. It provides access to the mathematical functions defined by the C standard.</td>
</tr>
</tbody>
</table>
string

The string library provides a lot of generic function for string manipulations, searching and extracting substrings and pattern matching with regular expressions.

You will find a short (but very good) reference paper about Lua and its supported modules as one PDF file at: http://lua-users.org/wiki/LuaShortReference

The paper is also contributed by the MSB-RS485 software. Take a look in the doc directory of your installation folder.

Analyzer Modules

The MSB-RS485 software offers some additional modules to connect the capabilities of the analyzer with the Lua language. Most of them are fitted as best to the according view.

- bit Module
- dv Module
- record Module

The bit module provides you with binary bit operations which are not implemented in Lua.

The second module you have get to know is the Lua support for the Data Monitor.

With the record module you can query informations about the current record like the used protocol, the start time of the record and the names for each signal.

Bit Module

Lua does not know different number formats but processes all numbers as floating point values with double precision. Therefore bit operations are not provided in the standard implementation of Lua.

On protocol or data level you will sometimes face the task to evaluate single bits or to modify data bytes bit-wise (e.g. in the context with check sum evaluation).

The bit module expand the integrated Lua interpreter with the following functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>band(x1,x2)</td>
<td>returns the bitwise and of its arguments x1 and x2, for instance <code>bit.band(0xFF,0x01)</code></td>
</tr>
<tr>
<td>bor(x1,x2)</td>
<td>returns the bitwise or of its arguments x1 and x2, for instance <code>bit.bor(0xFF,0x01)</code></td>
</tr>
</tbody>
</table>
**CHAPTER 17. LUA BEGINNERS GUIDE**

<table>
<thead>
<tr>
<th>bxor(x1,x2)</th>
<th>returns the bitwise xor of its arguments x1 and x2, for instance <code>bit.bxor(0xFF, 0x0F)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>bnot(x)</td>
<td>The result is the logical negation of the single bits (also ones complement). Each 1 is replaced by a 0 and vice versa. For instance <code>bit.bnot(0x55)</code></td>
</tr>
<tr>
<td>lshift(x,n)</td>
<td>returns bitwise logical left-shift of its first argument x by the number of bits given by the second argument n. For instance <code>bit.lshift(0x100, 2)</code></td>
</tr>
<tr>
<td>rshift(x,n)</td>
<td>returns bitwise logical right-shift of its first argument x by the number of bits given by the second argument n. For instance <code>bit.rshift(0x1FF, 1)</code></td>
</tr>
</tbody>
</table>

The sample project `9bit.msbprj` shows the use of the bit module based on the evaluation of a LRC check sum. You will find it in the `examples/DataView` sub-directory of the installation directory.

**Data View Module**

The Data Monitor (or Data View) module `dv` provides you with functions to query all information about the recorded data as displayed in the grid. For instance you can access each cell via its index, the current cursor position and the amount of the cells in the grid.

Leave it to Lua to compute the data from specific cells and output it in up to eight independent entries in the Watch window. For example validate a checksum or convert the data of different cells into floating point numbers etc.

Beside this you are able to mark a specified cell or a sequence of cells with a given colour. And you can dye the complete data field or just a small number of cells relative to the cursor position.

Last but not least: If you are unhappy with the information shown in both fields of the statusbar, don’t worry. The Data Monitor modules provides you with a function to put in any information you like to see.

Examples for the use of the Data View Module can also be found in the sub-directory `examples/DataView` of the installation directory. `errors.msbprj` colors all data bytes containing a frame or parity error or occurred breaks. `srecord.msbprj` visualizes a recorded S-Record transmission.

**The Data View (dv) module**

| cell(index)       | returns the log event table from the cell with the given index. The first cell of the Data Monitor is indexed by 1, the last with `dv.size()`. See also 17.12 |

---

130
17.12. ANALYZER SPECIFIC DATA TYPES

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor()</td>
<td>returns the current cursor position in the grid as index. The index starts with 1, the last possible index is dv.size().</td>
</tr>
<tr>
<td>mark(pos,n,col)</td>
<td>marks the next n data cells from position pos (starting with the first cell as 1) with the given colour col</td>
</tr>
<tr>
<td>size()</td>
<td>returns the index of the last visible cell in the Data Monitor grid.</td>
</tr>
<tr>
<td>statusbar(f,args)</td>
<td>displays each of the given arguments args in the left field (f=1) or right field (f=2) of the statusbar</td>
</tr>
<tr>
<td>watch(args)</td>
<td>displays each of the passed args to the according watch entry in the Data Monitor</td>
</tr>
</tbody>
</table>

**Record Module**

The `record` module offers you some additional functions to query important informations about the current record. For instance: the used baudrate, wordlen, parity setting, stopbits, all signal names and the start time (date) of the recording.

The functions in detail:

### Das Record Modul (record)

- **protocol()**
  - returns the baudrate, count of databits, parity setting `None`, `Odd`, `Even` and used stopbits as a value list. For example:
    ```python
    baud,databits,parity,stopbits = record.protocol()
    ```

- **signalnames()**
  - returns all eight signalnames as a list from `Signal1` to `Signal8`. For instance:
    ```python
    s1,s2,s3,s4,s5,s6,s7,s8 = record.signalnames()
    ```

- **starttime()**
  - supplies the start time (date) of the current record as the so called Unixtime (represents the number of seconds elapsed since 00:00:00 on January 1, 1970, Coordinated Universal Time (UTC)).

#### 17.12 Analyzer specific data types

The analyzer stores each detected change as a special record. This record contains all the associated information like the time in microsecs, the kind of alteration, the state of each line and other. With a `dv.cell(dv.cursor())` you are able to request the event from the current cursor position (or relative to it) as a table and compute the information for your own purpose.

Each field in the table represents one specific information about the event. Valid table fields are:
### The LogEvent Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dv.cell(i).Data</code></td>
<td>Returns the data (up to 9 bit) of the given cell <code>i</code>.</td>
</tr>
<tr>
<td><code>dv.cell(i).CountAll</code></td>
<td>The current data event number (all data channels A+B).</td>
</tr>
<tr>
<td><code>dv.cell(i).CountThis</code></td>
<td>The current data event number (only for this direction).</td>
</tr>
<tr>
<td><code>dv.cell(i).Error</code></td>
<td>Returns the error (if exist) of the given cell <code>i</code> or 0. Errors are: 1=Frame, 2=Parity, 3=Break.</td>
</tr>
<tr>
<td><code>dv.cell(i).Position</code></td>
<td>The event number of the given cell <code>i</code>. The event counting starts with 0 and includes all selected events.</td>
</tr>
<tr>
<td><code>dv.cell(i).Sig?</code></td>
<td>Query the logical signal state (line level) at the time of the given data event (cell). ? denotes the signal number 1...8. The result is: +1, −1 or 0 (inactive).</td>
</tr>
<tr>
<td><code>dv.cell(i).Source</code></td>
<td>Returns the data source of the given cell <code>i</code>. 1: Data channel A, 2: Data channel B.</td>
</tr>
<tr>
<td><code>dv.cell(i).Time</code></td>
<td>The time stamp of the given cell <code>i</code> in seconds as a floating point number.</td>
</tr>
<tr>
<td><code>dv.cell(i).Valid</code></td>
<td>Returns the valid state of the given cell <code>i</code>. Empty cells (marked as XX) returns false, otherwise true.</td>
</tr>
</tbody>
</table>

### 17.13 Limitations

The original Lua comes with a lot of additional modules. Not all of them are for any use in the analyzer environment and therefore are excluded from the embedded Lua implementation. Not available modules are:

- I/O Library
- OS Library

To avoid a slow down of the analyzer software by busy or overloaded computing scripts, for instance a long running or endless loop, the internal Lua VM (virtual machine) doesn’t allow to out run a specified quantum of operations. In this case, the VM aborts the execution of the script and throws out an informational message.

Just try the following:

```lua
local x = 0
while true do
    x = x + 1
end
```

--> [string "local x = 0..."]:-1: overrun of allowed executions

---

132
17.14 Further information
This chapter can’t replace any good introduction to Lua. It only covers the necessary information you need to undertake the first steps with Lua in the MSB-RS485 software. It also gives you a small outlook of all the language features Lua comes with. For more information about Lua please visit the Lua website at http://www.lua.org. You will find a very good tutorial at http://lua-users.org/wiki/TutorialDirectory too.
Synchronize two analyzers

You have two connections (RS232 and/or RS422/485) which you want to watch or examine in parallel, for instance IN and OUT data of a protocol converter, different bus segments or generally interdependent data transmissions. How you have to proceed and what is to be regarded is described in this chapter.

For a simultaneous recording of two separated connections you need two MSB analyzers. But this is only one requirement. To compare two recorded data files the data have to be in a precise time relationship. Without this relationship you can neither decide about the chronological sequence nor check the synchronicity of certain events.

For example when was a data byte or data sequence sent in relationship to the data of another connection. What happened in both connections at a defined point in time.

18.1 Technical requirements

One of the outstanding features of the MSB-Analyzer is the exact time measurement and visualizing of the time behavior in microsecond resolution. This precision is necessary to deliver correct results even for higher baud rates and is also valid for the common analysis of two connections. What does this mean?

Imagine two agents which shall enter a secured building and have to watch and record the way of the guards at different positions. Before they begin they compare their watches. This is done within a difference of a typical one second divergence. Both agents have watches which differ not more than one second per day. It doesn’t bear contemplating if both watches would disperse after some minutes. Now each agent proceeds to his position.

Each one notes the point in time (seconds precision) of the change of guards. As the watching takes some days both agents synchronize their watches repeatedly at midnight by a short radio pulse from one of the agents.

For the successful execution of their plan they have to know each step of the guards within a difference of one second.

The same procedure but with a far higher precision must be performed for the simultaneous recording of two connections. The time comparison at the
CHAPTER 18. SYNCHRONIZE TWO ANALYZERS

beginning is done by the exact simultaneous start of the recording, where simultaneous means a precision of one microsecond.

Of course the clocks of both MSB-Analyzer are more precise than the watches of the agents, but nevertheless they also differ from each other because of small natural differences of the crystal oscillators. They have to be synchronized in regular intervals. The MSB-Analyzer uses for both, the synchronous start and the regular timing of the clocks an additional synchronizing connection.

For this purpose each MSB-Analyzer offers a so called ’MSB-Link’ jack in RJ45 design. To synchronize two analyzers simply connect them with a standard network 1:1 cable. Please regard that although standard network cables are used you must not connect the analyzer to another data network. The signals are not compatible and the analyzer could be damaged.

It doesn’t matter if both analyzers are connected to the same or different PCs. The PCS also do not have to share a common network. The only restriction is the length of the synchronizing cable between both devices\(^1\). The synchronizing affects only the start of the recording and the precise keeping of the common time basis. That means that the time stamps of both analyzers are comparable on millionth second.

Furthermore both analyzers work fully independent. That means that you can record completely different protocols and events (baud rate, data format a.s.o.). Moreover you can connect a MSB-RS232 and a MSB-RS485 analyzer to simultaneously analyzer RS232 and RS485/422 connections, for example in interface converters.

18.2 Master Slave operation

It makes no sense to start the recording of the synchronized analyzers separately. Especially if both devices run at different PCs which may be at different locations. Start, Pause, Stop of the common recording are operated from a before as ’Master’ defined analyzer. This one is freely selectable. The second analyzer, connected via the synchronizing cable, is automatically set as ’Slave’ and controls its own recording synchronous to the Master with microsecond precision.

Both synchronized analyzers can be configured in the way that the recorded data are automatically stored to a predefined storage location when the record is stopped. This can be a local drive of the PC where the respective analyzer is connected to. It can also be any other drive, for instance a network drive. So both analyzers can store their data on the same drive but in different files.

The recorded data files are named with the serial number of the analyzer and the date/time of the start of recording. Additionally you can place any character string at the beginning (prefix).

\(^1\)Tested was a CAT6 network cable with 100m length.
18.3 Establish a synchronous record

Now you have a rough idea of how the synchronous recording works. Let's come to the practical part. Imagine you have two RS232 connections you want to record commonly. To simplify matters the recording is done at only one PC, where both analyzers are connected to.

At first connect both devices via a standard network cable. We recommend cable of category CAT-6, but for the most applications cables of category CAT 5 are sufficient.

Warning!
Please note that the analyzer must NOT be connected to a PC network through the MSB Link jack. This will probably result in a damage of the MSB-Analyzer.

Start the Analyzer software with the desktop icon. If multiple analyzers are connected to the same PC you have to select the wanted analyzer from a list (select connected analyzer). Repeat this step for the second analyzer. The same procedure applies even if the analyzers are connected to different PCs. Place both control programs (each connected to a different analyzer) on the screen.

Still both devices work independently of each other. You can start, stop or pause the analyzers individually. They also work on different time bases.

Since both analyzers can record different connection protocols or types (RS232 or RS485) you first have to configure each analyzer according to the requirements of the examined connections. This is done just like the recording of a not simultaneous recording. Set all the required parameters in the settings dialogs of the analyzers.

By default both analyzers store their data on the desktop as soon as the recording is stopped by the Master (by you). You can also set the place to any other location by selecting another directory in the settings dialog ‘Auto store’.

The file name is set by the analyzer program itself to avoid errors for repeated storing by already available files. It also makes it possible to assign the file to the analyzer exclusively.

The file name consists of the following parts, here a sample of the analyzer with the serial number MSB01060, started on 16th of April 2014 at 15:32.17.

MSB01060-20140416153217.msblog

Additionally you can place any character string in front of the name as a prefix, e.g. MASTER or SLAVE.

After having configured both devices you only have to set the Master for synchronous recording (assumed both analyzers are connected via network cable).

Activate the device which shall be the Master device. This is done in the settings dialog under ‘Analyzer is Master’. In the program display the word Master
CHAPTER 18. SYNCHRONIZE TWO ANALYZERS

is shown above the running recording time. At the same time the analyzer, which is connected to the Master, displays the word 'Slave' in its program display and the buttons and menu entries to control the recording are deactivated.

As soon as you uncheck the Master entry both analyzers are autonomous devices again. The same applies if you disconnect the link cable.

Close the settings dialog and click in the control program of the master on the start button. Both devices change to the record mode, indicated by the respective button display and the red LED at the analyzers themselves. Click the Pause button of the master to hold the recording. After clicking the stop button of the master the recording of both analyzers are finished. They automatically store their data on your desktop or any other specified directory.

You can repeat this procedure any time. As soon as you click the start button both analyzers will start a new recording and and after clicking on stop they will store them as two new data files. This way of operation does not differ if both analyzers are connected to different PCs which may also be placed in different rooms. The only requirement is the connection via the link cable.

18.4 Analyse a synchronous record

The MSB-Analyzer software is optimized to visualize a single recording by multiple different views. The loading of multiple record files is not possible because two or more records with different settings makes no sense within the application. For instance a RS232 and a RS485 recording need different displays and dialogs. But how can two records be analyzed at the same time?

The Analyser software consequently extends the already available communication between the views of a single application to multiple parallel running applications. That means that like the the signal monitor follows the cursor of the data view now all views of the separately running analyzer programs are synchronized to the cursor. That has a number of crucial advantages:

- Comparing analysis of differing recordings (baudrate, protocol, type of communication, ..).
- Synchronous moving and parallel display of certain ranges in both records (e.g. search for events in record A and showing the respective signal sequence in record B).
- No new operating scheme, no new menus.

Therefore the analysis of two synchronized recordings is not different to the analysis of a single recording. Instead of starting only one analyzer application you

\[^2\text{In the end a running MSB analyzer program application corresponds to ONE recording. This is the same as for Audio or Video applications.}\]
now start two different programs for the Master and the Slave recording.

For the evaluation in conjunction you do not need a connected analyzer. The examination can be done as is usual in the offline mode. Click on the master and slave recordings one after another. Both applications make the accustomed access to the respective data. The views of each application synchronize their windows if the synch. Mode is activated in their tool bar.

To synchronize the views between BOTH running applications you first have to enable this feature. By default the synchronization from external sources is disabled.

The enabling is done for all Views of an application centrally in the control program at ‘common settings’. Activate ‘allow external synchronization’.

By enabling the external synchronization the control program receives the mouse clicks or events (search results, region selection, etc.) from a parallel running analyzer application and passes them to its opened views. Each view with active synch. setting reacts on these events and actualizes its display.

In this way you can watch the slave recording at any time point in the master recording and vice versa. Both applications keep their views synchronous to one defined time stamp.

18.5 Conclusion

The comparing recording or analyzing of two separate connections requires a high precise reference to set the recorded data and events in relationship to each other.

These chapters showed you why this is necessary, which technical requirements have to be fulfilled and how such a recording has do be done with two MSB analyzers.

Here come the necessary steps again without ballast.

**Synchronous recording**

1 Connect both analyzers which shall be synchronized via a standard network cable.

2 Connect both analyzers to one or two PCs.

3 Start a separate MSB analyzer program for both devices.

4 Set up individual connection parameters for both analyzers.

5 Check if the automatic storage after record stop is activated and specify a storing location if necessary.

6 Define one of the devices as Master in the set up dialog of the appropriate application program at ‘Record’.

7 Start the synchronous recording at the master control program.

8 The recording is also stopped by the master whereas both records are automatically stored separately.
CHAPTER 18. SYNCHRONIZE TWO ANALYZERS

Synchronous analysis
For the evaluation of two synchronously logged records you do not need a connected analyzer, but both MSB analyzer programs have to run on the same computer because a synchronization of the views is not possible through the network cable in opposite to the synchronization of the recordings themselves.

1. Double click on both (master and slave) recordings resp. start two MSB-Analyzer programs. Load the files into the control program.
2. Activate in both programs under ‘General’ the entry ‘allow external synchronization’
3. Place both control programs and the wanted views on the screen.
4. Navigate as used through both recordings. The views in synchronous mode will automatically align their content to the examined time period.
You want to automatize the recording of a data connection and process the recorded data in your own application, or to store respectively output them?
A long recording should saved as several sequenced files or splitted afterwards.
You like to control the analyser from within your application.
The MSB-Analyzer software offers a series of powerful tools which we will describe in this chapter.

After installation of the analyzer software you will find some helpful other tools in the installation directory beside the programs for operating the MSB-RS485 and visualizing the recorded data.
All these programs are based on command lines and might be used as part of batch files or shell scripts. According to the Unix philosophy ‘Do only one thing but do it well’ each of these programs has only one function. By their capability to read from the standard input and send their results back to the standard output these programs may be combined in any way (program tool chain).
Further more: You can combine them with a lot of other programs which are able to handle data via standard input/output.

The command line programs in overview:

- **msb_record**
  This tool controls the analyser and writes all received data to the standard output or in a given file.

- **msb_format**
  Output the analyser data read from standard input in a user specified format.
CHAPTER 19. COMMANDLINE API

- **msb_filter**
  Filters the analyser data passed from standard input to output by user defined rules.

- **msb_split**
  Reads data or a record file from the standard input and splits the output into smaller record files.

- **msb_trigger (in planning stage)**
  Checks the data from the standard input against some given trigger conditions and start or stop passing the data to standard out according to the result.

### 19.1 Combine the programs as a tool chain

You can simply put the tools mentioned above together to work as a processing chain. Thereby each program processes data of the former program and forwards it to the next tool in the queue.

The processing (tool) chain always consists of a data source and a data sink. The single programs can be linked with the '|' operator which is identical for Windows and Linux.

```
DATASOURCE | MANIPULATOR1 | MANIPULATOR2 | ... | DATASINK
```

**Data source**

Each tool chain starts with a data source. The data source provides the following programs with the necessary input, here most of all the tool *msb_record*. But the output of a already existing analyser record file via *type* (Windows) respectively *cat* (Linux) works just as well. For instance:

```
type recordfile.msblog or cat recordfile.msblog
```

**Manipulators**

A program which modifies the data during the forwarding is called a manipulator. A typical manipulator might remove special parts of the read data before it pass it on the next link in the chain or change the read data in another format specified by the user.

Therewith you can extend or complete the processing of the data simply by inserting any number of manipulators in the processing chain. One manipulator might remove unwanted data before the next tool converts the remaining data into another format, for instance with the *msb_format*.

**Data sink**

A data sink specifies the end of the processing chain. A typicall sink is the screen output (of a command line window) or a file storing the data. But a data sink might be also your own application which read the resulting data and processes it for your own purpose, for instance a LabView application.

The *msb_split* tool is a representative data sink. The program doesn’t forward the data to other tools but stores it as serveral files on your hard disk.

**Some examples**

The program folder of the MSB-Analyzer software will be added automatically to the search path for executables during installation. For a first try, you just have to open a command shell (console). We will use the example records coming with the software package as a data source. Therefore you don’t need
19.2. RECORD DATA WITH MSB_RECORD

Go to the example directory and pipe a record into the `msb_format` program like:

```bash
type DataView\9bit.msblog | msb_format
```

Linux users have to use the `cat` command instead of the `type`. `type record` works as a data source like an active recording with `msb_record`. `msb_format` is the manipulator tool in the processing chain and forwards the result to the command line window which stands for the data sink and just displays the result on your screen.

Now use the `msb_split` tool to split the same record file in several little pieces.

```bash
type DataView\9bit.msblog | msb_split -n1000
```

Without any arguments `msb_split` simply creates the following two files in the current directory named as xaa.msblog and xab.msblog.

You will find more detailed information about these tools in the program relating sections below.

### 19.2 Record data with msb_record

As the name indicated this program controls the operating and recording of a connected MSB-Analyzer. At the same time `msb_record` functions as a data source for all other tools.

Called without any further arguments `msb_record` searches for a connected analyser, transfers the firmware if needed and starts a new record of all transmitted data bytes with 115200 baud and 8N1 protocol as default.

If the tool doesn’t find any device or detects more than one analyser, it will give you an appropriate message. In the last case you can select the proper analyser by passing the serial number of the analyser to the program.

`msb_record` writes the recorded data directly to the standard output to make them available for the other tools. We will illustrate this with the following example, input directly on the command line window:

```bash
msb_record | msb_format
```

All recorded data are forwarded with the pipe operator `|` to the next program in the tool chain, here the `msb_format`. The latter reads all data received from its standard input, makes some transformation and put the result to the standard output again. In this case - unless there are more programs in the queue, it writes it just as a simple informal list.

```
00000001 4.383965 <A> 'h'
00000002 4.384051 <A> 'e'
00000003 4.384138 <A> 'l'
00000004 4.384224 <A> 'l'
00000005 4.384311 <A> 'o'
```

1. You can of course execute the `msb_record` tool standalone. But because the outputed data is in a binary format this doesn’t make any real sense.
CHAPTER 19. COMMANDLINE API

Press ‘Ctrl+C’ to abort the program.

In general you will use `msb_record` either in a combination with other command line tools as shown above or to write the output directly into a file. There are two ways to save the recorded data as a file. You can redirect the output like:

```bash
msb_record > output.msblog
```

Or you pass a file name as an additional argument:

```bash
msb_record -o output.msblog
```

**Connection settings and events**

We didn’t bother about the connection properties so far and called the record tool implicitly with its default settings. We restricted ourself also to record only the transmitted data bytes and ignored the change of the line states. Imagine you have a serial connection working with 38400 baud, 7 databits and an even parity. The MSB-Analyzer doesn’t worry about the number of stop bits, but nevertheless we will assume 2 stop bits here. Beside the raw data bytes we like to analyse the line level change of both data transmission lines (here RxD, TxD) as well as the handshake control lines RTS and CTS. The call of the `msb_record` tool is described as follows:

```bash
msb_record --baudrate=38400 --protocol=7E2 --logsignals=2,3,6,7
```
or in a short form:

```bash
msb_record -b 38400 -p 7E2 -l 2,3,6,7
```

We will show another way to pass the parameters via a configuration file later. For now and most important:

Don’t separate the arguments from the associate program! In the command line above all `msb_record` parameters have to specified before the pipe operator.

```bash
msb_record -b 38400 -p 7E2 -l 2,3,6,7 | msb_format
```

This applies for all tools, programs belonging to the MSB-Analyzer and also other ones.

**Usage in your own application**

Maybe you thinking that’s all pretty interesting, but how can I use these tools within my own application? Your application only has to fulfil the following requirements:

1. Execution of any command from within your application.
2. Read of a file opened/written by another process.
That sounds worse than it is. The most programming languages come with a special function to execute an external command. For instance: C has the `system` and `popen` function, LabView offers a System Exec VI. You can call an external command mostly in two ways:

First: The caller (your application) waits until completion of the command. We don't recommend this, because it would block your application.

Second: The command chain is executed as a so called detached process. In this case the function (with the command chain) returns immediately to the caller and the command chains works parallel to your application.

So far as good, but how can you get the results of the command chain? Your tool chain can write the results in a file where you read them back from within your application. Or: You fetch the results directly from the tool chain output. Which of one fits you best depends on your application language.

The detached tool chain process will be stopped and closed automatically at the end of your application. But there is another option to control the data collection of a parallel running `msb_record`.

**Remote control**

The `msb_record` tool contains a simple and easy to access inter process communication method which works for both platforms (Linux and Windows) equally.

Sending a command to a running background process is done by calling the `msb_record` program with the parameter `-r command` or executing the command from your application via system call.

Just open two command shells and start a record in one of them with:

```
msb_record | msb_format
```

The connected MSB-Analyzer is initialized and the recording is started, indicated by a permanent lighted red LED1. It doesn't matter if there aren't any data available.

Now switch to the second console (command window) and stop the recording with:

```
msb_record -r stop
```

The execution of the stop command can be checked by watching the analyzer red LEDs. They are blinking alternatively again.

To start or resume the recording repeat the call but now with the command 'start':

```
msb_record -r start
```

The command `msb_record -r quit` ends the process respectively the tool chain and closes the output channel/file.
msb_record program parameters
Call the program with:
msb_record [OPTION]...

[OPTION] can contain one or more of the following program parameters. If no parameter is set the default parameters are used. The following parameters can be sent to the program at start. All recorded data will be written to the standard output by default.
To send a command to a running background process the remote parameter indication `-r` has to be entered followed by the command (start, stop, quit).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-b rate</code></td>
<td>Baudrate of the recorded connection, default is 115200.</td>
</tr>
<tr>
<td><code>--baudrate=rate</code></td>
<td></td>
</tr>
<tr>
<td><code>-c</code></td>
<td>Uses the settings specified in the given config file.</td>
</tr>
<tr>
<td><code>--config-file=file</code></td>
<td></td>
</tr>
<tr>
<td><code>-C</code></td>
<td>Creates a new config file <code>msb_tools.config</code> in the current directory.</td>
</tr>
<tr>
<td><code>--create-config-file</code></td>
<td></td>
</tr>
<tr>
<td><code>-d</code></td>
<td>use the analyser at the given port exclusively.</td>
</tr>
<tr>
<td><code>--device=port</code></td>
<td></td>
</tr>
<tr>
<td><code>-h</code></td>
<td>Help. Output of all program parameters.</td>
</tr>
<tr>
<td><code>--help</code></td>
<td></td>
</tr>
<tr>
<td><code>-i</code></td>
<td>Transfers the firmware to the analyzer, even it is already loaded.</td>
</tr>
<tr>
<td><code>--initiate</code></td>
<td></td>
</tr>
<tr>
<td><code>--io1=operation</code></td>
<td>Use digital auxiliary channel IO1 (only MSB-RS485). The following values are valid: 0 : Input with pull down 1 : Output static 0 2 : Output static 1 3 : Output of the bus direction 4 : Output of the bus validness 5 : Output CHN1 validity 6 : Output CHN2 validity 7 : Output CHN3 validity 8 : Output CHN4 validity</td>
</tr>
<tr>
<td><code>--io2=operation</code></td>
<td>Use digital auxiliary channel IO2 (only MSB-RS485). Valid values see IO1 above.</td>
</tr>
<tr>
<td><code>-l</code></td>
<td>Specifies the signal lines which are logged by the analyzer. The lines are numbered from 1 to 8 as they are displayed in the analyzer control program (counted from left to right). For instance: <code>-l 2,3 oder --log-signals=2,3,6,7</code>.</td>
</tr>
<tr>
<td><code>--memory-test</code></td>
<td>Forces the analyser to executes an internal memory test.</td>
</tr>
</tbody>
</table>
### 19.2. RECORD DATA WITH MSB_RECORD

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-n serno</code></td>
<td>Use the analyzer with the given serial number serno.</td>
</tr>
<tr>
<td><code>--serno</code></td>
<td>Use the analyzer with the given serial number serno.</td>
</tr>
<tr>
<td><code>--output-buffering</code></td>
<td>Activate the internal output buffer, which increases the performance and avoids gaps in data records with high data transfer rates. Please note: With an active buffering the recorded events doesn’t occurs immediately in the following tool of the command chain, for instance if you like to see all recorded events in a console window via the msb_format tool.</td>
</tr>
<tr>
<td><code>-o file</code></td>
<td>Output file. Default is the standard output (console).</td>
</tr>
<tr>
<td><code>--output=file</code></td>
<td>Output file. Default is the standard output (console).</td>
</tr>
<tr>
<td><code>--paused</code></td>
<td>Starts the analyzer in paused state. The record begins only after the program receives a remote start command.</td>
</tr>
<tr>
<td><code>-p protocol</code></td>
<td>Protocol settings of the connection as combination of number of data bits (5 to 9), parity (N)one, (E)ven, (O)dd, (0)ff, (1)on and stopbits (1,2). E.g. 8N1 or 7E2. Default is 8N1.</td>
</tr>
<tr>
<td><code>--protocol=protocol</code></td>
<td>Protocol settings of the connection as combination of number of data bits (5 to 9), parity (N)one, (E)ven, (O)dd, (0)ff, (1)on and stopbits (1,2). E.g. 8N1 or 7E2. Default is 8N1.</td>
</tr>
<tr>
<td><code>-r Command</code></td>
<td>Remote. Sends the following command to an already running program. The following commands are supported: quit quits and removes the background process start starts or resumes a recording stop stops or pauses the recording</td>
</tr>
<tr>
<td><code>--remote=command</code></td>
<td>Remote. Sends the following command to an already running program. The following commands are supported: quit quits and removes the background process start starts or resumes a recording stop stops or pauses the recording</td>
</tr>
<tr>
<td><code>--showserials</code></td>
<td>Shows all available serial ports.</td>
</tr>
<tr>
<td><code>--showanalyzers</code></td>
<td>Shows all available (connected) MSB-Analyzer.</td>
</tr>
<tr>
<td><code>-t num</code></td>
<td>Transfer delay. Slows the firmware transfer down by the indicated number num. 0 is no delay (default), maximum is 100.</td>
</tr>
<tr>
<td><code>--time-delay=num</code></td>
<td>Transfer delay. Slows the firmware transfer down by the indicated number num. 0 is no delay (default), maximum is 100.</td>
</tr>
<tr>
<td><code>-u</code></td>
<td>Stores the recorded data in the current working directory in a record file with an unique name like YYYYMMDD-HHmMMmSSs.msblog, for instance 20110324-03h04m41s.msblog. This parameter is especially interesting in those applications where the record should start automatically after a (re)boot of the PC.</td>
</tr>
<tr>
<td><code>--unique=file</code></td>
<td>Stores the recorded data in the current working directory in a record file with an unique name like YYYYMMDD-HHmMMmSSs.msblog, for instance 20110324-03h04m41s.msblog. This parameter is especially interesting in those applications where the record should start automatically after a (re)boot of the PC.</td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Verbose, output of additional Information.</td>
</tr>
<tr>
<td><code>--verbose</code></td>
<td>Verbose, output of additional Information.</td>
</tr>
<tr>
<td><code>-V</code></td>
<td>Output of the program version.</td>
</tr>
<tr>
<td><code>--verbose</code></td>
<td>Output of the program version.</td>
</tr>
</tbody>
</table>
CHAPTER 19. COMMANDLINE API

-w wiring --wiring=wiring

Set the bus wiring (only MSB-RS485). The following values are allowed:

0 : 2-wire tapping
1 : 2-wire segment analysis
2 : 4-wire tapping
3 : 4-wire segment analysis (masterbus)

19.3 Formatted output with msb_format

The msb_format tool allows you to format the recorded analyser data for your own purpose. For instance if you like to see the data as CSV (comma separated values). Without any parameter you will get a list of the occurred events, each one with informations about the time, the kind of event, the data value or line state. This complies with the format specifier ’I’ which is the default setting.

To specify your own output format, call the program with the format parameter -F or --format=. All following characters are seen as format definition. A space character or all ’white space’ characters like Tab or Enter end the format string. If you want to define a space character as part of the output you have to quote it. How to do this is explained in chapter 19.3.

We restrict ourselves to the simple case of displaying the data bytes together with their time stamps. In every line the time in seconds and the data byte shall be listed, separated by a comma. The appropriate format string\(^2\) is: T,B

We will use an example record as data source so you can try the following samples without a connected analyzer. Please keep in mind, that it will be work the same way with the msb_record tool.

Open a command shell (again), go into the DataView example folder (i.a. msb-VERSION/examples/DataView) and input:

```
type modbus-ascii.msblog | msb_format -FT,B
```

The output looks like the following:

```
...  
5633.304127,048
5633.305162,070
5633.306197,057
5633.307226,013
5633.308261,010
```

The output can be changed from ASCII to binary representation. ASCII means that the decimal binary value is coded into ASCII numbers as a string (e.g.’104’ = hex binary 31,30,34) while binary means the value itself which is displayed according to the ASCII table (in this example as ’h’).

Binary mode makes sense if you want to write the data into a file and read in by another application. An inconvenient conversion of the ASCII representation into native program types as double or integer might be omitted.

The format identifier % activates the binary output while @ switches back to

\(^2\)A list of all format identifiers can be found in the format identifier table.
19.3. FORMATTED OUTPUT WITH MSB_FORMAT

ASCII (default). The following example displays the characters in their binary value:

```bash
type modbus-ascii.msblog | msb_format -FT,%B@
```

Please note that in binary mode no line feed is issued. Therefore we switch back to ASCII mode after each binary data output.

```plaintext...
5633.304127,0
5633.305162,F
5633.306197,9
5633.307226,6
5633.308261,1
```

Disable line feed in ASCII Mode

To make the output in ASCII mode more readable a linefeed is automatically attached after every output line. You can disable this behavior by calling the program with the parameter –disable-linefeed or by ending the format string with % (binary mode).

Output of any character

You want to insert a non-printable character or define a line feed, independent of the operating system. Use the format identifier `#ddd` to define any character which shall be output instead of this identifier. To separate the output of the data bytes by a tabulator enter the decimal value of this character. e.g.

```bash
type modbus-ascii.msblog | msb_format -FT#009B#009S
```

Or: Separate the values by a space char (decimal 032):

```bash
type modbus-ascii.msblog | msb_format -FT#032#032S
```

To generate a line line feed under windows with a single linefeed character you have to use its decimal value (010). To disable the standard system dependent line feed character sequence in ASCII mode you end the string with % to switch to binary mode:

```bash
type modbus-ascii.msblog | msb_format -FT#009B#009S#010%
```

The character value has to be entered with three decimal digits (0 to 9). Any other input leads to an error message.

File output

You also can redirect the output to a file. Call the program with the additional parameter `-o filename`.

Only the via format string defined outputs are send to the file, no status messages or auxiliary outputs which you might have enabled through program parameter.

A simple file output is done with:

```bash
type modbus-ascii.msblog | msb_format -FT#009B#009S#010% -o test.log
```

3Under linux all lines are ended with a single linefeed, while Windows uses a combination of Caridge Return/Linefeed.
CHAPTER 19. COMMANDLINE API

Format parameters
The following identifiers are defined as format parameters. Please note that not mentioned characters are output in the same way. Exceptions are the white-space characters that are all blanks, tabs and enter which are used to end the format string.

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>Binary Flag</td>
<td>Switches to the binary output for all following parameter.</td>
</tr>
<tr>
<td>@</td>
<td>Ascii Flag</td>
<td>Switches to the ASCII output mode for all following parameters.</td>
</tr>
<tr>
<td>#ddd</td>
<td>Character</td>
<td>Output of any printable or non-printable character, specified as a 3-digit decimal value. The allowed value range is 0 up to 255. e.g. the line ending carriage return character is #013</td>
</tr>
<tr>
<td>[...]</td>
<td>[format]</td>
<td>User defined date and time output, see table below 19.3.</td>
</tr>
<tr>
<td>b</td>
<td>Data-Byte</td>
<td>Data byte output as 8 bit value. In ASCII mode represented as 2-digit hexadecimal number with leading zeros, e.g. ’41’ is the character ’A’, ’0A’ the linefeed character.</td>
</tr>
<tr>
<td>B</td>
<td>Data-Byte</td>
<td>The same as ’b’ but with a 3-digit decimal number output in ASCII mode, for instance: ’065’ means the character ’A’, ’010’ the linefeed.</td>
</tr>
<tr>
<td>d</td>
<td>Date/Time</td>
<td>Timestamp output representation in the ISO 8601 format YYYY-MM-DD HH:MM:SS (ASCII mode). Output as 32 bit value containing the seconds since the Epoch (00:00:00 UTC, January 1, 1970) in binary mode.</td>
</tr>
<tr>
<td>D</td>
<td>Excel Date</td>
<td>Excel date as days from 1.1.1900. Output as 32 bit value (binary) or 8-digit decimal number with leading zeros (ASCII).</td>
</tr>
<tr>
<td>e</td>
<td>Error</td>
<td>Transmission error. Errors are outputed as a text string like ’frame’, ’parity’ or ’break’.</td>
</tr>
<tr>
<td>E</td>
<td>Error</td>
<td>Transmission error. Like ’e’, but the output is either a 3-digit decimal value (ASCII) or a 8 bit value (binary). Each coded as follows: 0 : no error, 1 : Framing error, 2 : Parity error, 3 : Break.</td>
</tr>
<tr>
<td>i</td>
<td>Info</td>
<td>shows all informations in a more human readable form. Only for testing purpose. Don’t use this parameter with others.</td>
</tr>
</tbody>
</table>
### 19.3. FORMATTED OUTPUT WITH MSB_FORMAT

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I</strong></td>
<td>Logic-Level</td>
<td>Output the current logic state of all 8 signal lines. A set bit correlates with a logic line level of ‘1’. The bit order is equal to the signal lines in the control program. Bit 0 is the first (left), bit 7 the last (right) signal. The state information is written as a 8 bit value in binary mode and as a 2-digit hex number with leading zeros in ASCII mode, e.g. ‘7F’ means all lines except for Signal 8 have a logical ‘1’ level.</td>
</tr>
<tr>
<td><strong>L</strong></td>
<td>Logic-Level</td>
<td>The same as ‘I’. In ASCII mode the output is written as a 3-digit decimal number with leading zeros, e.g. ‘255’ means all lines have a logical ‘1’ level.</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td>Milliseconds</td>
<td>Time stamp of the event in milli seconds as distance to 0h00 of the current day. The output is either a 8-digit decimal value (ASCII) with leading zeros or a 32 bit value (binary).</td>
</tr>
<tr>
<td><strong>P</strong></td>
<td>Position</td>
<td>Running event counter starting with the first output. The output is either a 8-digit decimal value (ASCII) with leading zeros or a 32 bit value (binary).</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>Source</td>
<td>Source or direction of the data byte. Data channel A=0, Data channel B=1. Output either as 3-digit decimal number with leading zeros (ASCII) or as 8 bit value.</td>
</tr>
<tr>
<td><strong>t</strong></td>
<td>event type</td>
<td>Output the event type as a character in ASCII mode: A (data received at port A/channel 1), B (data received at port B/channel 2), L (logical line state changed), V (valid line state changed) and as a 8 bit value in binary mode, range [0...3].</td>
</tr>
<tr>
<td><strong>T</strong></td>
<td>Time stamp</td>
<td>Microseconds precise time stamp of the event in relationship to the start of the recording. Output in seconds as floating point number with 6 digits after the decimal point (ASCII) or as 8 byte floating point number in double precision.</td>
</tr>
<tr>
<td><strong>u</strong></td>
<td>usec part</td>
<td>The microseconds fraction of the timestamp. You can use it to complete the normal date/time with the left microseconds like: -Fd.u results to 2012-04-11 15:57:40.184935.</td>
</tr>
</tbody>
</table>
CHAPTER 19. COMMANDLINE API

v Valid-Level
Output the current valid state of all 8 signal lines. A set bit correlates with a valid line level. The bit order is equal to the signal lines in the control program. Bit 0 is the first (left), bit 7 the last (right) signal. The state information is written as a 8 bit value in binary mode and as a 2-digit hex number with leading zeros in ASCII mode, e.g. ‘7F’ means all lines except for Signal 8 have a valid signal level.

V Valid-Level
The same as ‘l’. In ASCII mode the output is written as a 3-digit decimal number with leading zeros, e.g. ‘255’ means all lines have a valid signal level.

w Data-Word
A 9 Bit Data byte is output as a 16 Bit binary value 0 to 511. In ASCII this value is displayed as a 3-digit hexadecimal number with leading zeros, e.g. ‘105’.

W Data-Word
Like ‘w’, but writes the output in ASCII mode as a 3-digit decimal number. For instance: ‘261’ means the same as the hex value ‘105’ from above.

User defined date and time
A string enclosed between two square brackets is interpreted as a special date/time format. With it you can output the timestamps in your very own format, according to your application. An example:

type modbus-ascii.msblog | msb_format -F"[%d.%m.%Y %H:%M:%S],us"

results as:

27.08.2010 09:41:04,303098s
27.08.2010 09:41:04,304127s
27.08.2010 09:41:04,305162s
27.08.2010 09:41:04,306197s
27.08.2010 09:41:04,307226s
27.08.2010 09:41:04,308261s
...

Please note! The command shell (in both, Linux and Windows) uses white space characters (here the space between date and time) as a parameter separator. Therefore you have to insert the complete format string between two quoting marks.

The msb_format tool supports the following user defined date/time format specifier. Every parameters must start with a leading %-character.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a</td>
<td>the abbreviated weekday name</td>
</tr>
<tr>
<td>%A</td>
<td>the full weekday name</td>
</tr>
<tr>
<td>%b</td>
<td>the abbreviated month</td>
</tr>
</tbody>
</table>

152
### 19.3. FORMATTED OUTPUT WITH MSB_FORMAT

| %B  | the full month name         |
| %c  | the preferred date and time representation for the current locale |
| %d  | the day of month as a decimal number [01...31] |
| %e  | like %d, the day of the month as a decimal number, but a leading zero is replaced by a space ['1'...'31'] |
| %H  | the hour as a decimal number, range [00...23] |
| %I  | the hour as decimal number, range [00...12] |
| %j  | the day of the year as decimal number, range [001...366] |
| %m  | the month as decimal number, range [01...12] |
| %M  | the minutes as decimal number, range [00...59] |
| %p  | 'am' or 'pm' (according to the given time value) |
| %S  | the seconds as decimal number, range [00...59] |
| %T  | the time in 24-hour notation like %H:%M:%S |
| %U  | The week number of the current year as a decimal number, range [00...53], starting with the first Sunday as the first day of week 01 |
| %w  | the day of the week as decimal number [0...6], Sunday being 0 |
| %W  | The week number of the current year as a decimal number, range [00...53], starting with the first Monday as the first day of week 01 |
| %y  | the year as decimal number without the century [00...99] |
| %Y  | the year as decimal number including the century |
| %z  | the time zone, for instance CEST |
| %%  | the literal % character |

**msb_format program parameters**

Call the program with:

```bash
msb_format [OPTION]...
```

[OPTION] can contain one or more of the following program parameters. If no parameter is set the default format `-FI` is used and the output is written to the standard output channel.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>

153
CHAPTER 19. COMMANDLINE API

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-c</code></td>
<td>Uses the settings specified in the given config file.</td>
</tr>
<tr>
<td><code>--config-file=file</code></td>
<td></td>
</tr>
<tr>
<td><code>--disable-linefeed</code></td>
<td>Suppress linefeed in ASCII mode.</td>
</tr>
<tr>
<td><code>-F</code></td>
<td>Output format definition, see format table.</td>
</tr>
<tr>
<td><code>--format=formatstring</code></td>
<td></td>
</tr>
<tr>
<td><code>-h</code></td>
<td>Help. Output of all program parameters.</td>
</tr>
<tr>
<td><code>--help</code></td>
<td></td>
</tr>
<tr>
<td><code>-o file</code></td>
<td>Output file. Default is the standard output (console).</td>
</tr>
<tr>
<td><code>--output=file</code></td>
<td></td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Verbose, output of additional Information.</td>
</tr>
<tr>
<td><code>--verbose</code></td>
<td></td>
</tr>
<tr>
<td><code>-V</code></td>
<td>Output of the program version.</td>
</tr>
<tr>
<td><code>--verbose</code></td>
<td></td>
</tr>
</tbody>
</table>

19.4 Filtering data output with msb_filter

The filter tool will be your first choice when you have to extract a special part, certain events or a combination of both from a former record (*.msblog). For example: If you want to process only the transmitted data in a record without already existing signal events.

`msb_filter` reads the data from its standard input and write the filtered data to the standard output like each other tool. The program thereby works like a real filter between the input and output channel. You can specify the kind of data or events which are passed through the filter tool by several filter parameters.

```
type recordfile.msblog | msb_filter [Filterparameter] ...
```

Please note that you have to give at least one filter parameter because the tool only passes the data which were allowed by the program arguments. Without any filter parameter the program will block all data flow.

Filter data

The following tool chain filters all data bytes (received at port A and B) from the given file `modbus-ascii.msblog` in the directory `examples/DataView` and stores the result in the new record file `data-only.msblog`.

```
type modbus-ascii.msblog | msb_filter -A -B > data-only.msblog
```

You can also pass the data of the filter tool directly to the input of the formatter tool `msb_format`.

```
type modbus-ascii.msblog | msb_filter -A -B | msb_format
```

4 Linux user use the `cat` command instead of the `type` command.
19.4. FILTERING DATA OUTPUT WITH MSB_FILTER

**Filter certain signal events**
Beside the data you can also extract the recorded signal changes of every signal line. For instance if you have a record with all line changes but you are only interested in the transmitted data and the handshake signals RTS/CTS. The selection of the passed signals is given as a comma separated list and corresponds with the `--log-signals` parameter of the `msb_record` tool.

```
type modbus-ascii.msblog | msb_filter -A -B --pass-signals=6,7 | msb_format
```

The example above extracts the signal changes of the lines 6 and 7 (in RS232 connections the signals RTS and CTS) additional to the transmitted data and forwards the result to the output formater.

**Filter a given record part**
The tool `msb_split` described in the next section is able to split an existing record in several smaller record files. But imagine if you only need a 'certain' part of a record file. For instance: The first or last 100000 events? Or you want to analyze only the events in a specific time range.
The filter tool offers you two further parameters to define a specify record part:

1. `--pass-selection=pos1,pos2`
   
   `pos1` and `pos2` specifies the event position (number) of the first and last event which are passed through the filter tool. For example:
   ```
type modbus-ascii.msblog | msb_filter --pass-all --pass-selection=300,310
```

2. `--pass-time=time1,time2`
   
   `time1` and `time2` specifies the start and end of a record part in seconds. The time is input as a floating point number with the usual micro second precision.
   ```
type modbus-ascii.msblog | msb_filter --pass-all --pass-time=3.04,3.05
```

**A Non-blocking filter**
The filter tool 'blocks' all data by default. In case of a range selection you have to pass all allowed events as parameters. Or you disable (switch off) the filtering completely with the parameter `--pass-all`.

**msb_filter program parameter**
Call the program with:

```
msb_filter [OPTION]...
```

[OPTION] can contain one ore more of the following program parameters. You have to give at least one filter rule. Without any rule the program doesn't forward any data.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### CHAPTER 19. COMMANDLINE API

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td><code>--pass all</code> passes all data and signal events.</td>
</tr>
<tr>
<td>-A</td>
<td><code>--pass dataA</code> passes all data received at port A (MSB-RS232) respectively Channel 1 (MSB-RS485).</td>
</tr>
<tr>
<td>-B</td>
<td><code>--pass dataB</code> passes all data received at port B (MSB-RS232) respectively Channel 2 (MSB-RS485).</td>
</tr>
<tr>
<td>-c</td>
<td><code>--config file=file</code> Uses the settings specified in the given config file.</td>
</tr>
<tr>
<td>-h</td>
<td><code>--help</code> Help. Output of all program parameters.</td>
</tr>
<tr>
<td>-s</td>
<td><code>--pass selection=list</code> pass all events in the given range defined as comma separated event number from first to last. The following example passes all recorded events with the numbers 100 to 200: <code>--pass-selection=100,200</code>.</td>
</tr>
<tr>
<td>-t</td>
<td><code>--pass time=list</code> pass all events in the given time range in seconds as comma separated list with first time, last time. For instance: <code>--pass-time=1.257,10.231</code> passes all recorded events in the time range 1.257s until 10.231s.</td>
</tr>
<tr>
<td>-v</td>
<td><code>--verbose</code> Verbose, output of additional information.</td>
</tr>
<tr>
<td>-V</td>
<td><code>--verbose</code> Output of the program version.</td>
</tr>
</tbody>
</table>

#### 19.5 Split records with msb_split

When recording data with the MSB-RS485 analyzer large data quantities may arise. This happens if the searched error does not occur for days and recording in Fifo mode is not wanted for any reason.

*msb_split* reads a record file from the standard input and splits it into smaller record files. You can specify the size and name of the files by use of program parameters.

**Split existing record files**

You have a GByte large record file and want to split it into handy parts, especially as you are interested in the last events of the recording only.
19.5. SPLIT RECORDS WITH MSB_SPLIT

Open a console window (Windows command window) and change to the directory which contains your record file. enter the following command:

```
type record.msblog | msb_split -n1000000
```

With type the record file is sent to the standard output and fed into the standard input of the msb_split program by the pipe operator '|'.

Linux users use the cat command instead of the type program.

Depending on the size of the output file record.msblog msb_split divides them into multiple 1000000 * 24 + 3072 Byte files Each file (with exception of the last one) contains 1,000,000 events, each 24 bytes long plus a header of 3072 bytes. In the current directory a number of new msblog files are generated in the form:

xaa.msblog, xab.msblog, xac.msblog, ...

You can examine every file individually with the MSB-RS485 analyzer software by loading them into the program or double click onto it. By default the program enumerates all files alphabetically with a preceding 'x'. You can change this behavior by adding a respective parameter. For a 3-digit, numerical enumeration use the parameters -a and -d (see ??).

```
type record.msblog | msb_split -a3 -d -n1000000
```

As result you get: x000.msblog, x001.msblog...

You can substitute the preceding 'x' for your prefix by appending it as last parameter to the command line.

```
type record.msblog | msb_split -a3 -d -n1000000 Test
```

The resulting files now begin with: Test000.msblog, Test001.msblog, ...

This kind of enumeration does not mention the important time range of the single files. Alternative to the alphabetical or numerical naming you can also chose date and time of the first event for the name of the split files. The parameter is -D.

```
type record.msblog | msb_split -D -n1000000 Projekt-
```

The generated files have the following meaning:

```
Projekt-20110510_15h53m24s.msblog
Projekt-20110510_15h58m31s.msblog
Projekt-20110510_16h02m10s.msblog
...
```

If you don’t like any prefix, just append an 'empty' string as the last parameter
CHAPTER 19. COMMANDLINE API

(PREFIX):

type record.msblog | msb_split -D -n1000000 ""

With it you will get:

20110510_15h53m24s.msblog
20110510_15h58m31s.msblog
20110510_16h02m10s.msblog
...

Splitting the current recording from msb_record

As the msb_split program reads its data from the standard input you can use the output of the msb_record tool as data source to directly divide the recorded events into small portions. This may make sense if you plan long and large recordings to examine them later.

msb_record -b115200 -p8N1 | msb_split -a4 -d -n1000000

Program Parameter

Call the program with:  msb_split [OPTION]... [PREFIX]

[OPTION] can contain one ore more of the following program parameters. If no parameter is set the default parameters are used. [PREFIX] is an optional and freely selectable character string which precedes the file name. The default is the character ‘x’.

All parameters can be used in the short form (a character with a leading '-', first row) or in the long form (second row).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a length, --suffix=-length=</td>
<td>Number of usable characters for the enumerating suffix. Default is 2 characters.</td>
</tr>
<tr>
<td>-h, --help</td>
<td>Output of all program parameters.</td>
</tr>
<tr>
<td>-d, --numeric= suffix</td>
<td>The files are names numerically, the default is alphabetically.</td>
</tr>
<tr>
<td>-D, --date-time= suffix</td>
<td>The files names are extended with the date and time of the first occurred event in the format YYYYMMDD_HHhMMmSSs.</td>
</tr>
<tr>
<td>-n, --number=quantity</td>
<td>Quantity of the events per file. Each event occupies 24 bytes.</td>
</tr>
<tr>
<td>-v, --verbose</td>
<td>Output of additional information.</td>
</tr>
<tr>
<td>-V, --version</td>
<td>Output of the program version.</td>
</tr>
</tbody>
</table>
19.6. ONE CONFIG FILE FOR ALL

19.6  One config file for all
As yet we either used the default settings of the several tools or we handled our specifications to the respective tools via program parameters. Depending on the count of arguments this approach will lead to complex and - perhaps - buggy command lines.

Therefore all tools are also manageable by one configuration file which is given to the first msb_record program as a parameter. msb_record ensures that all further programs in the command (tool) chain receive the settings in this file\(^5\).

A configuration file isn’t part of the analyzer software but you can always create a new one just by the following command:

```
msb_record -C
```

respectively

```
msb_record --create-config-file
```

As a result the file msb_tools.config is written in the current directory. Open the file with your favorite editor (Windows user can use notepad, Linux users may choose between gedit, kate, or - of course vi, emacs, ...). The configuration file is well documented. You can simply adapt the parameters to your application and save the file under a new name. The latter makes sense because another call of msb_record -C overwrites the file without a warning.

You can - of course - create several individual configuration files, one for each application. File name and file extension are of no importance. As soon as you specify a configuration file to the msb_record program, all further tools in the command chain will take the settings in that file into account. For example:

```
msb_record -c meine-config-datei | msb_format
```

respectively

```
msb_record --config-file meine-config-datei | msb_format
```

Perhaps you are wondering about all the examples above which were using the output of a record file via type or cat as data source instead of the msb_record tool?

In this case you can specify the configuration file for each tool individually. Each analyzer tool understands the parameter --config-file or the simple variant -c. You do not need to change the configuration file. Just call the relating tool with the required file.

```
type examples\DataView\9bit.mablog | msb_format --config-file datei
```

\(^5\)This only works of course for the analyzer tools.
ASCII (American Standard Code for Information Interchange) is a form for the character coding, which, coming from teletype machines, now is established as the standard code for character representation.

The first 32 characters of the ASCII code (hex 00 to 1F) are non printable signs, reserved for control purposes. The main control characters are line feed or carriage return. They are used with devices which need the ASCII code for control purposes as printer or terminals. Their definition is caused for historic reasons.

Code hex 20 is the blank and hex 7F is a special character which is used for deleting.

<table>
<thead>
<tr>
<th>Code</th>
<th>...0</th>
<th>...1</th>
<th>...2</th>
<th>...3</th>
<th>...4</th>
<th>...5</th>
<th>...6</th>
<th>...7</th>
<th>...8</th>
<th>...9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0...</td>
<td>NUL</td>
<td>SOH</td>
<td>STX</td>
<td>ETX</td>
<td>EOT</td>
<td>ENQ</td>
<td>ACK</td>
<td>BEL</td>
<td>BS</td>
<td>HT</td>
<td>LF</td>
<td>VT</td>
<td>FF</td>
<td>CR</td>
<td>SO</td>
<td>SI</td>
</tr>
<tr>
<td>1...</td>
<td>DLE</td>
<td>DC1</td>
<td>DC2</td>
<td>DC3</td>
<td>DC4</td>
<td>NAK</td>
<td>SYN</td>
<td>ETB</td>
<td>CAN</td>
<td>EM</td>
<td>SUB</td>
<td>ESC</td>
<td>FS</td>
<td>GS</td>
<td>RS</td>
<td>US</td>
</tr>
<tr>
<td>2...</td>
<td>SP</td>
<td>!</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>3...</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td>4...</td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>5...</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td></td>
</tr>
<tr>
<td>6...</td>
<td>'</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td>7...</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>~</td>
<td></td>
</tr>
</tbody>
</table>

The upper table regards only 7 bits per byte, the first 128 characters. Extentions of the ASCII code use the next 128 characters for national language codings or graphical signs. They are very different in usage. So we will limit the description to the standard 7 bit version.
Baudrate measuring

The MSB-RS485 analyzer allows the setting and measuring of any baudrate in the wide range from 1 Baud up to 1 MBaud with the unique precision better than 0.1%

The measuring is performed eight times per second, thereby measuring and averaging the width of singular 0 or 1 bits. The more bits are available in the measuring frame of 125 ms the more precise the measuring becomes. A higher data quantity will lead to more precise and stable measuring values. The analyzer allows three kinds of baudrate measuring.

1. Automodus (UART A + B)
2. CH1 (UART A)
3. CH2 (CH3) (UART B)

In the auto mode either the data of the internal UART A (CH1) or UART B (CH2 respective CH3) is used for measuring, depending on which channel delivers the first data bit at the start of a 125 ms measuring frame. Therefore the measured baudrate can vary if both channels use different clock generators with slightly different baudrates. This mode is appropriate especially to detect different baudrates on the send and receive line.

To measure the baudrate of a certain channel the input must be explicitly set. This is done in the settings dialog of the control program.

The status window shows 2 baudrates, the set and the measured one. The latter with its percental deviation to the set rate. Deviations over 50% are indicated by Out!.

\[ dBaud = 100 \times \frac{Baudmeasured - Baudset}{Baudset} \]

A negative value indicates a lower baudrate, positive values indicates a higher baudrate (than the set one). Many bit errors can be explained from incorrect generated baudrates. The following can be taken as a rough guide value: Deviations of a maximum of ±3% can be accepted and compensated, higher deviations should be avoided.
APPENDIX B. BAUDRATE MEASURING

Baudrate tolerance
Avoid more than 3% deviation in the baudrate generation. This will result in bit errors.

Because of insufficient slew rates of the EIA-422/485 sender of a examined transmission line the measuring value can be too high for high transmission rates. This could also be a hint, that the EIA-422/485 drivers are not correct for the used baud rate when the baud rate is higher than allowed baudrate for the EIA-422/485 driver.
The MSB-RS485 analyzer software allows you to enter own color definitions at different places. A selection of predefined colors can be found here.

The input of color values can be done either in form of a color name (the following tables show an overview of the pre-defined color names) or by entering a RGB (red green blue) value as a hexadecimal number. Please note, that the names are generally in english, even if you use a German software version. Some colors consist of compound words as 'indian red'. The blank between is part of the name and has to be entered explicitely.

In the following list you find besides the color names also the RGB value, which can be entered alternatively. RGB values can be entered in short or in long form. The number of digits (3 or 6) determine the used format.

C.1 RGB short form
The short form #RGB reduces each color part to a value between 0 and 15 decimal (0 to F hexadecimal) where R,B,G is represented by this value 0 to F. That means that each part can be defined in steps of 1/15 of 100%. For instance red is #F00 and white is #FFF. For each part is valid that for 0 it is not contained and for F it is fully contained in the composite color. In the short form 16 x 16 x 16 = 4096 colors are possible.

C.2 RGB long form
The long form #RRGGBB extends the value range for the single color parts from 16 to 256, which simply is a higher resolution for each color. The resulting color range is 256 x 256 x 256 = 16777216 possible colors.

C.3 Predefined color names
The predefined color names are a selection from a list of standard colors used in web site displays. Besides the extended colors an input of ‘green’ should be much more intuitive than #0F0 of #00FF00. The basic colors like ‘black’, ‘white’, ‘red’ ... are easy to memorize.
### APPENDIX C. COLORS

#### Grey colors

<table>
<thead>
<tr>
<th>Name/Value</th>
<th>Color</th>
<th>Name/Value</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>black</td>
<td>#000000</td>
<td>dim grey</td>
<td>#696969</td>
</tr>
<tr>
<td>dark grey</td>
<td>#a9a9a9</td>
<td>grey</td>
<td>#bebebe</td>
</tr>
<tr>
<td>light grey</td>
<td>#d3d3d3</td>
<td>white</td>
<td>#ffffff</td>
</tr>
</tbody>
</table>

#### Basic colors

<table>
<thead>
<tr>
<th>Name/Value</th>
<th>Color</th>
<th>Name/Value</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>blue</td>
<td>#0000ff</td>
<td>green</td>
<td>#00ff00</td>
</tr>
<tr>
<td>red</td>
<td>#ff0000</td>
<td>cyan</td>
<td>#00ffff</td>
</tr>
<tr>
<td>magenta</td>
<td>#ff00ff</td>
<td>yellow</td>
<td>#ffff00</td>
</tr>
</tbody>
</table>

#### Extended colors

<table>
<thead>
<tr>
<th>Name/Value</th>
<th>Color</th>
<th>Name/Value</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>medium spring green</td>
<td>#7fff00</td>
<td>forest green</td>
<td>#228b22</td>
</tr>
<tr>
<td>lime green</td>
<td>#32cd32</td>
<td>dark green</td>
<td>#006400</td>
</tr>
<tr>
<td>aquamarine</td>
<td>#70db93</td>
<td>spring green</td>
<td>#00ff7f</td>
</tr>
<tr>
<td>medium aquamarine</td>
<td>#66cdaa</td>
<td>sea green</td>
<td>#238e6b</td>
</tr>
<tr>
<td>medium turquoise</td>
<td>#70dbdb</td>
<td>dark turquoise</td>
<td>#00ced1</td>
</tr>
<tr>
<td>steel blue</td>
<td>#236b8e</td>
<td>sky blue</td>
<td>#3299cc</td>
</tr>
<tr>
<td>slate blue</td>
<td>#007fff</td>
<td>light steel blue</td>
<td>#b0c4de</td>
</tr>
<tr>
<td>cornflower blue</td>
<td>#6495ed</td>
<td>navy</td>
<td>#2328e5</td>
</tr>
<tr>
<td>medium blue</td>
<td>#0000cd</td>
<td>dark slate blue</td>
<td>#483d8b</td>
</tr>
</tbody>
</table>
### C.3. PREDEFINED COLOR NAMES

<table>
<thead>
<tr>
<th>Name/Value</th>
<th>Color</th>
<th>Name/Value</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>medium orchid</td>
<td>#9370db</td>
<td>medium slate blue</td>
<td>#7f00ff</td>
</tr>
<tr>
<td>blue violet</td>
<td>#8a2be2</td>
<td>dark orchid</td>
<td>#9932cc</td>
</tr>
<tr>
<td>purple</td>
<td>#b000ff</td>
<td>orchid</td>
<td>#db70db</td>
</tr>
<tr>
<td>violet red</td>
<td>#cc3299</td>
<td>orange red</td>
<td>#ff007f</td>
</tr>
<tr>
<td>maroon</td>
<td>#b03060</td>
<td>salmon</td>
<td>#6f4242</td>
</tr>
<tr>
<td>khaki</td>
<td>#f0e68c</td>
<td>wheat</td>
<td>#d8d8bf</td>
</tr>
<tr>
<td>medium goldenrod</td>
<td>#eaaad</td>
<td>pale green</td>
<td>#8fbc8f</td>
</tr>
<tr>
<td>medium sea green</td>
<td>#426f42</td>
<td>medium violet red</td>
<td>#db7093</td>
</tr>
<tr>
<td>turquoise</td>
<td>#aadea</td>
<td>cadet blue</td>
<td>#5f9ea0</td>
</tr>
<tr>
<td>light blue</td>
<td>#add8e6</td>
<td>midnight blue</td>
<td>#2f2f4f</td>
</tr>
<tr>
<td>pink</td>
<td>#bc8fe8</td>
<td>thistle</td>
<td>#d8bfd8</td>
</tr>
<tr>
<td>plum</td>
<td>#eaaede</td>
<td>violet</td>
<td>#4f2f4f</td>
</tr>
<tr>
<td>firebrick</td>
<td>#8a2222</td>
<td>brown</td>
<td>#a52a2a</td>
</tr>
<tr>
<td>orange</td>
<td>#cc3232</td>
<td>indian red</td>
<td>#cd5c5c</td>
</tr>
<tr>
<td>coral</td>
<td>#fff7f50</td>
<td>tan</td>
<td>#db9370</td>
</tr>
<tr>
<td>sienna</td>
<td>#8e6b23</td>
<td>gold</td>
<td>#fffd700</td>
</tr>
<tr>
<td>medium forest green</td>
<td>#6b8e23</td>
<td>yellow green</td>
<td>#99cc32</td>
</tr>
<tr>
<td>dark olive green</td>
<td>#556b2f</td>
<td>green yellow</td>
<td>#adff2f</td>
</tr>
</tbody>
</table>
Windows Trouble-Shooting

The driver, necessary for the operation of the analyzer, is automatically installed and the device is detected at start of the program. If this is not the case or other problems arise you will find some solutions here.

D.1 Windows doesn’t found the analyzer (Part I)

Description
The Analyzer was connected correctly with your system and both red state LEDs are on. After starting the software, the startup dialog still doesn’t detect it.

Solution
At first remove all USB devices except Mouse and keyboard from your PC. Connect The analyzer to your PC and look up the assigned COM port in the device manager. Usually a new COM Port should occur after attaching the analyzer.
If this is the case open a command shell (DOS Box) and change to the installation program. Usually this is:

```
C:\Programme\msb-3.2.5
```

Start the software manually by entering the following:

```
msb_serv.exe -pCOMxx -nMSByyyyy --force
```

COMxx is the port number as displayed in the design manager, MSByyyyy means the serial number of the analyzer. The number sticks on the bottom of the housing. For instance:

```
msb_serv.exe -pCOM12 -nMSB01234 --force
```

If the software starts with this parameter you can add this program parameter to the MSB start icon under preferences. You will find a description in the chapter 7.13 of the manual.
APPENDIX D. WINDOWS TROUBLE-SHOOTING

D.2 Windows doesn’t found the analyzer (Part II)

Description
The analyzer is correctly connected to your system and both LEDs light red. But the device manager does not display a virtual COM port number. (See also D.1).

Solution
Reinstall the drivers which are necessary for the operation of the analyzer. You will find the driver on your installation CD in the directory driver or on our website under https://iftools.com/msb-rs485/download.en.php. For the installation use the executable driver setup program (recognizable by the ending .exe).

Please help us with conflicting devices
The firmware loader uses the information which is collected in the USB enumeration procedure to detect all serial ports connected to a MSB-analyzer. Because of the manifold combinations of existing USB devices and drivers we can not exclude the possibility that in rare cases the program does not detect the analyzer correctly. To regard these situations in the further program development we need your active help.

Open a command shell (DOS box) and change to the installation directory. Enter the following command:

```
msb_record.exe --show-serials > serials.txt
```

The command lists all available serial devices including driver information, manufacturer, product etc. and writes it in the given file serials.txt. Just send the serials.txt file afterwards to support@iftools.com.

D.3 MSB-RS485 Device quit working unexpectedly

Description
The MSB-RS485 device loses its connection and stops working unexpectedly. The device may appear to work fine when Windows first starts up but later stops working.

Solution
Use the steps (below) to resolve this issue.

1. Click Start, and right-click My Computer.
2. Click Properties, then click Hardware.
3. Click Device Manager.
4. Double-click the Universal Serial Bus Controllers branch to expand it.
5. Right-click USB Root Hub, and then click Properties.
D.4. OTHER PROBLEM

6 Click Power Management.
7 Deselect Allow the computer to turn off this device to save power.
8 Repeat Steps 5 through 7 for each USB Root Hub.
9 Click OK, and close Device Manager.
10 Reboot your system

D.4 Other problem
Description
Your problem isn’t listed here.

Solution
In case of problems do not hesitate to send us a mail under: support@iftools.com
Please do not forget to inform us about your software and system (Windows version, Service Pack, 23/64 Bit System) as also a detail description of your problem.
New Linux kernels innately contain all what is necessary for the operation of the analyzer. Nevertheless you can be trapped by the lot of different Linux variants and their differing implementations which may make the correct functioning difficult. How you can bypass the problems are explained in this chapter.

E.1 Linux doesn’t found the analyzer (Part I)

Description
The Analyzer was connected correctly with your system and both red state LEDs are on. After starting the software, the startup dialog reports an error (MSB not found).

Solution
Within Linux you have to be member of the group for accessing the /dev/ttyUSBx device. Open a console and type:

```bash
~$ ls -l /dev/ttyUSB? 
crw-rw---- 1 root dialout 188, 0 2009-08-26 14:47 /dev/ttyUSB0
```

On Debian the group is dialout, SuSE uses the uucp group. Make your user account member of the group and try again. You can check your group membership with:

```bash
~$ groups
username dialout cdrom floppy audio video plugdev
```

Please note! You have to logout and new login first until the changes are available. A reboot is not necessary.

E.2 Linux doesn’t found the analyzer (Part II)

Description
You have the correct permissions for accessing the /dev/ttyUSBx device. Anyhow the analyzer wasn’t detected by the software.
APPENDIX E. LINUX TROUBLE-SHOOTING

Solution
Be sure, that you don't have an installed Braille driver. Disconnect and connect the MSB-RS485 again. Open a console and input `dmesg`.

If the output displays something like this:

```
~$ dmesg
```

a Braille driver is part of your system. If you have no need for a Braille device, please remove it from your system. On Debian

```
~$ apt-get remove brltty
```

should be work.

E.3 Linux doesn't found the analyzer (Part III)

Description
You have disconnect all serial USB devices (except of mouse and keyboard if used) from your computer. Unfortunately there doesn’t exist any entry like `/dev/ttyUSB0`.

Solution
Open a console window and input the following command:

```
~$ dmesg
usb 1-1: new full speed USB device using uhci_hcd and address 5
ftdi_sio 1-1:1.0: FTDI USB Serial Device converter detected
drivers/usb/serial/ftdi_sio.c: Detected FT232BM
usb 1-1: FTDI USB Serial Device converter now attached to ttyUSB0
```

The output of the command `dmesg` shows you, if the kernel recognize the analyzer as FTDI USB serial device and if the associated kernel module `ftdi_sio` was loaded by the kernel.

If not, your kernel doesn’t support any USB or the FTDI support isn’t part of the kernel. If you have compiled your kernel by yourself, please check, if you have enabled FTDI devices as module or part of the kernel itself. Take a look in your kernel configuration file (you will find it at `/usr/src/linux/.config`). There must be a line like this:

```
CONFIG_USB_SERIAL_FTDI_SIO=m
```

If not, you have to recompile your kernel with the line above. Also without the `ftdi_sio` module your kernel should register the connected analyser as a USB device if the USB support working correctly. You can verify this with the following command line:
The example above shows a analyzer MSB-RS485 (MSB-B) from IFTOOLS with the serial number MSB00001. You should get this output in any case or your kernel have some trouble with USB devices in general. Can you confirm, that your Linux system works with other USB devices? Please contact us with detailed information about your system, see section E.9.

Please help us with conflicting devices

The firmware loader uses the information which is collected in the USB enumeration procedure to detect all serial ports connected to a MSB-analyzer. Because of the manifold combinations of existing USB devices and drivers we can not exclude the possibility that in rare cases the program does not detect the analyzer correctly. To regard these situations in the further program development we need your active help.

Open a command shell (DOS box) and change to the installation directory. Enter the following command:

```
~$ ./msb_record --show-serials > serials.txt
```

The command lists all available serial devices including driver information, manufacturer, product etc. and writes it in the given file serials.txt. Just send the serials.txt file afterwards to support@iftools.com.

E.4 Recording doesn’t work

Description

The analyzer seems to be correctly detected and the firmware was loaded. After starting of a recording session no data are recorded.

Solution

The analyzer indicates its correct firmware initialization by an alternating blinking of both red LEDs. If both LEDs are permanently on after starting the software, the firmware was not correctly transferred into the device. The reason is the high firmware transfer rate which can cause errors (very rare) in some driver implementations.

You can reduce the transfer rate for the firmware initialization via program parameter. Call the analyzer software with the following additional parameter from the installation directory:

```
~$ ./msb_serv -r 20
```
APPENDIX E. LINUX TROUBLE-SHOOTING

The value of 20 correspondents to the reduction of the transfer rate. Allowed values are 0 (no reduction) up to 50. As soon as the analyzer firmware was correctly loaded you can add the used parameter to your msb-3.2.5 desktop icon.

E.5 Segmentation fault during installation

Description
The installation program crashes with a segmentation error. This happens mainly within a kde4 environment.

Solution
Open a console, make the installation file as executable

~$ chmod +x msb-2.4.0-linux-installer.bin

and start the program with the --mode xwindow like this:

~$ ./installation-file-linux.bin --mode text

E.6 The help menu Help → Content F1 doesn’t work

Description
You pressed F1 or select the Help → Content F1, but nothing happens or you get error message. If you open the manuals directly, you can read the operation manual.

Solution
The MSB-RS485 software needs an additional PDF viewer for displaying the according manual pages. Within Linux this is the xpdf program, because of it’s features to reference certain sections via so called named destinations. You have to install this program if it isn’t part of your system. For reading or printing the manual you can always use your favorite PDF viewer like kpdf, okular or GNOME application of course.

E.7 The software doesn’t run within a 64 bit Linux (Part I)

Description
The analyzer software doesn’t start via click on the desktop icon or crashed with a segfault.

Solution
The analyzer software is build as a 32 bit application. Therefore you have to install the according 32 bit shared libraries (ia32-libs) to use on amd64 and ia64 systems. On debian based systems just install the ia32-libs with:

~$ sudo apt-get install ia32-libs

You can - of course - use your package manager or software center.

E.8 The software doesn’t run within a 64 bit Linux (Part II)

Description
You have installed the ia32-libs but if you start the software only a small window appears (Ubuntu 11.04 or higher) or nothing happened.
E.9. OTHER PROBLEM

Solution
Ubuntu’s method of creating the 32bit versions of these packages for the 64bit ubuntu has a bug caused by a hard coded path where the gdk-pixbuf library looks for several modules loaded at runtime. We recommend the following workaround:
Open a terminal window and input:

```
  gedit ~/Desktop/msb-3.2.5.desktop
```

**Add the additional line:** Path=/home/jb/msb-3.2.5.

**Replace the line:** Exec=/home/jb/msb-3.2.5/msb_serv with:

```
  Exec=env GDK_PIXBUF_MODULE_FILE=/usr/lib32/gdk-pixbuf-2.0/2.10.0/loaders.cache msb_serv
```

Now the changed desktop file has to look like:

```
[Desktop Entry]
Type=Application
Version=0.9.4
Name=msb-3.2.5
Comment=Starts the msb program
Icon=/home/jb/msb-3.2.5/msb-48.png
Path=/home/jb/msb-3.2.5
Exec=env GDK_PIXBUF_MODULE_FILE=/usr/lib32/gdk-pixbuf-2.0/2.10.0/loaders.cache msb_serv
Terminal=false
Name[en_US]=msb-3.2.5
```

Save the file and start the software by click on the icon.

E.9 Other problem

Description
Your problem isn’t listed here.

Solution
We are very much interested that our software can be used under Linux without problems. Because of the large numbers of different Linux distributions this is not always easy. Therefore:
In case of problems do not hesitate to send us a mail under: support@iftools.com
Please do not forget to inform us about your software and kernel version, 32/64 Bit system, Linux distribution, desktop environment and a detail description of your problem.
<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV</td>
<td>Comma Separated Values, text file format in which the content of single data sets are stored in independent lines, separated by commas. 55</td>
</tr>
<tr>
<td>Datagram</td>
<td>Datagram is generic term for data frame, data packet, or data segment and describes a data sequence with a defined start and length. It is also called telegram in fieldbus systems. 77</td>
</tr>
<tr>
<td>ETX</td>
<td>End of Text, in the ASCII character set defined as hex 0x03. ETX marks the end of a message or datagram. 77</td>
</tr>
<tr>
<td>Firmware</td>
<td>Firmware describes the software contained in an electronic device which is responsible for its function. Firmware can be a fixed and unchangeable part of the hardware or can be loaded into the device before the first start. 27</td>
</tr>
<tr>
<td>FLEXUART</td>
<td>An IFTOOLS in-house developed UART allows high-precise setting and measuring of any, even non standard baud rates in the range from 1 Baud up to 1MBaud with 0.1% accuracy. 28</td>
</tr>
<tr>
<td>Full-Duplex</td>
<td>Shortened as HD or HDX. A full-duplex, or sometimes double-duplex system, allows communication in both directions, and, unlike half-duplex, allows this to happen simultaneously. 1</td>
</tr>
<tr>
<td>Half-Duplex</td>
<td>Shortened as HD, or HDX. A half-duplex system provides for communications in both directions, but only one direction at a time (not simultaneously). 1</td>
</tr>
<tr>
<td>Lua</td>
<td>Lua is a dynamically typed language intended for use as an extension or scripting language. By including only a minimum set of data types, Lua attempts to strike a balance between power and size. 52</td>
</tr>
<tr>
<td>Multi-Master</td>
<td>Bus nodes which are allowed to initiate a data transfer with other bus nodes are denote as active node or master (otherwise they are denoted as passive nodes or slaves). A bus with several masters is called a Multi-Master bus. 1</td>
</tr>
</tbody>
</table>
## Glossary

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multidrop</td>
<td>A Communication based on the Master-Slave principle whereby a master (sender) can speak to several receivers without expecting any answer (single direction).</td>
</tr>
<tr>
<td>Record depth</td>
<td>The number of maximum events or samples which are contained in the signal recording is called recording or storage depth and depends on the available storage medium.</td>
</tr>
<tr>
<td>RGB</td>
<td>RGB color model, each color is defined by an additive red, green and blue color quota.</td>
</tr>
<tr>
<td>RTF</td>
<td>A document file format developed by Microsoft for cross-platform document interchange.</td>
</tr>
<tr>
<td>RTS/CTS Handshake</td>
<td>A hardware flow control implemented by correspondent levels on the RTS or CTS lines. The RTS/CTS lines of both participants are cross-connected. By setting the RTS line to logical 1 the receiver requests a stop of the data transmission. Only a few UARTS handle the flow control in hardware, so that the software driver have to react fast to recognize the state and stop the transmission.</td>
</tr>
<tr>
<td>S-Record</td>
<td>The S-Record format is developed by Motorola and is an ASCII based data format for coding binary data. It is mainly used for transferring program code in microcontrollers and embedded systems.</td>
</tr>
<tr>
<td>STX</td>
<td>Start of Text, in the ASCII character set defined as hex 0x02. STX marks the start of a message or datagram.</td>
</tr>
<tr>
<td>Timebase</td>
<td>The time duration which corresponds to a grid (10 pixel). The lower the time base the higher the time resolution of the display. The lowest time base is 500ns, which corresponds to 50 ns per pixel.</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter. Electronic element to send or receive data over a serial data line.</td>
</tr>
</tbody>
</table>
Index

Absolute Time, 57
ALIAS, 87
Analyser
  multiple, 38
Analysis tools
  see Views, 35
ASCII, 87
BCHG, 83
Break
  display, 51
  search, 60, 68
CAB, 83
CNT, 84
CNTA, 84
CNTB, 84
Control display
  active lines, 31
  PC connection, 30
  recording capacity, 29
  toggle information, 29
Control program
  parameter, 40
  Short commands, 39
  Special parameters, 41
CTRL, 88
Data View, 51
  copy section, 54
  export section, 55
  Font, 57
  Goto address, 53
  save section, 54
  selection, 54
  Short commands, 63
  Show control chars, 56
Datagram
  Data direction, 83
  displaying, 86
  End of string, 85
  with constant length, 84
Datagramm
  Bit alternation, 83
  Run time length, 85
  Timeout, 85
Datenmonitor
  see Data View, 51
Displays
  see Views, 44
EOS, 85
Event View, 65
  export selection, 72
  select lines, 71
  Short commands, 75
  switch columns on/off, 66
Firmware
  Loading, 27
Framing
  display, 51
  search, 60, 68
HEX, 89
Ledtester, 49
  show level notation, 50
LevelFinder, 65
MASK, 89
Measure time distances, 74
MultiProcess architecture, 43
Parity
  display, 51
  search, 60, 68
Program settings, 48
  transferred, 48
Project
  last opened, 37
  load, 48
  save, 36, 48
Projekt, 47
Protocol
  autodetection, 29
  settings, 31
Protocol Monitor
  see Protocol View, 77
Protocol Templates
  default templates, 80
  define, 80
  splitting into datagrams, 81
Protocol templates
  language syntax, 81
  load, 81
  saving, 81
Protocol View, 77
  copy selection, 79
  Font, 92
  save selection, 80
  selection, 79
short keys, 93
Template comments, 91
Protocols, 78
Record
open, 37
pause, 29
save, 36
start, 29
starting automatically, 38
stop, 29
Record mode, 33
continuous, 33
Fifo, 33
Region, 109
move in view, 110
remov, 110
rename, 110
select, 54, 72, 105
switch on/off, 110
REPEAT, 90
ring buffer, 33
RTL, 85
S-Record, 96
Scope display, 100
Session
load, 37
see Project, 36
session, 47
Signal level
displaying, 100
search duration, 71
search level state, 67
serach for changes, 70
Signal line
selection, 33
Signal name
rename, 32
Signal View, 99
color settings, 103
Cursor, 104
Display data values, 103
Raster on/off, 103
Selection, 104
short keys, 107
Signal inverting, 103
Signal sequence, 103
size distance, 104
undo zooming, 102
Zooming view, 102
Signalmonitor
see Signal View, 99
String searching, 57
Taskbar
Hide view entries, 35
Telegram
see Protocol, 78
Terminal mode, 94
Time base, 101
Time distance
between data bytes, 57
Time distances
search, 59
TO, 85
Transmission errors, 57
search, 60, 68
Transparency, 104
User request
unsaved data dialog, 35
Views, 44
autoscroll, 44
copy, 46
locked, 44
save state, 46
synchronize, 43