

MLab Tutorial

*MLab - Data Acquisition and Control Program for Windows
Tutorial with example applications for MLab*

This manual has been prepared to the best of our knowledge. The information and data contained in these manuals are given without guarantee and may change without notice. The software described within these manuals is delivered in conjunction with a software license agreement.

This documentation may not be reproduced or relayed, in part or in whole, regardless of the manner or medium, electronically or mechanically without express written permission from the copyright owner.

© Copyright 2000 by STIEGELE Datensysteme GmbH
Conrad-Eberhard-Straße 5
91541 Rothenburg ob der Tauber
Germany

eMail: info@stiegele-systems.de

Internet: www.stiegele-systems.de

Author Tutorial: Emanuel Winter

Trademark

Microsoft, MS-DOS, Windows are either registered trademarks or trademarks of the Microsoft Corporation in the USA and other countries.

Mention of third-party products is for informational purposes only and does not represent a breach of copyright.

Contents

Part 1	Before getting Started	9
	MLab: Overview	10
	The Facilities	10
	The Concept of this Tutorial	12
	<i>User Groups</i>	12
	<i>The Chapter Sequence</i>	13
	Preparation: MSetup	14
	Hardware Setup: Differences between Windows 95 and NT	14
	What you need to know before the Configuration	15
	The Base Address	16
	Input Voltage Range	16
	Running MSetup	17
	Adding and Configuring the Hardware	18
	Details on Data Recognition: I/O Address, DMA Channel etc	21
	Details on Data Usage: Measurement Range and Amplification	22
	Setting up Two Cards of the Same Type	25
	Starting MLab for the First Time	26
	Input Wizard	26
<hr/>		
Part 2	First Measurements	29
	Basic Setup for a Measurement	30
	1st. Step: Defining Hardware	30
	Pre-setting the Hardware	30
	2nd. Step: Clock Source and Sampling Rate	31
	Pre-setting the Clock Source	32
	3rd Step: Defining Datasets	33
	Inserting Datasets	33
	Editing Datasets	35
	4th. Step: Defining Graphical Items	37
	Inserting Graphical Objects into a Graphic Page	39
	<i>Inserting an $x(t)$ Diagram</i>	40
	<i>Inserting a numerical Field (Display as Digits)</i>	43
	Starting a Measurement	45
	Possible Causes of Errors	47
	The End of the Chapter	47

Further Notes	48
<hr/>	
Part 3 Monitoring	51
Introduction	52
1st. Example: Storage above a limit value	52
Definition of a Standard Limit Monitor	53
2nd. Example: Bar Display with Extremity Marker	55
<i>Defining a Bar Object</i>	55
<i>Activating the Display of Extremity Markers</i>	56
<i>Defining a Key</i>	57
<i>Creating a "logical (event)" Type Variable</i>	57
<i>Defining a "Key" Graphical Object</i>	58
<i>Assigning the Function to the Key</i>	59
<i>How these Settings Work</i>	59
General Information about Variables and Action Modules	60
3rd Example: Display of Maximum Value in an x(t) Diagram	62
Definition of the two Analog Variables.....	63
Defining the "Extremity Marker" Action Module.....	64
Defining the Graphical Display	65
Inhibiting and Activating the Extremity Marker	66
4th Example: Range Display with a Switch	67
1st Step: Defining Datasets and Variables.....	67
<i>Inserting a Digital Output Dataset</i>	68
<i>Entering the Digital Device as an Additional Device</i>	68
<i>Inserting the Digital Channel as a Dataset</i>	69
2nd. Step: Setting up the Limit Monitor Module	71
<i>The Information Variables of a Limit Monitor Module</i>	72
3rd. Step: The Graphical Display Definition, a Switch.....	73
Possible Causes of Errors.....	74
The End of the Chapter	75
Notes.....	75
<hr/>	
Part 4 Storing Data	77
Introduction	78
Standard Data File versus "Storage Module" Action Module.....	78
<i>The Standard Data File</i>	78
<i>The "Storage Module" Action Module</i>	78
In which File Format will Data be Stored?.....	79
<i>Data Multiplexing</i>	79
<i>Storage in RMS Format</i>	80
<i>Storage in MDF Format</i>	80

The Examples in this Chapter	81
Example 1: Storing a Test Sequence	81
Creating a Standard Data File	81
<i>Entries in the Dataset List</i>	81
<i>General Definitions for the Standard Data File</i>	83
<i>Further Definitions for the Standard Data File</i>	84
Changing File with the "N" Key	85
A Text Box for Displaying the Current Filename	85
<i>The Information Variables for Storage Modules</i>	86
Example 2: Rapid Storage and Ring File	87
Setting up the Storage Module	88
Setting the Sampling Rate	89
Defining the Ring File	91
<i>Starting Ring File Storage Automatically at the Start of the Measurement</i>	92
Example 3: Storing Sections	93
Defining the Sections	94
Displaying and Using the Sections in MGraph	95
The End of the Chapter	96
<i>Possible Causes of Errors</i>	96

Part 5	Desired Values	99
	Introduction	100
	Types of Desired Values.....	100
	The Examples in this Chapter	100
	Example 1: A Continually Changing Load	101
	Registering the Demand Channel as an Additional Device	102
	Setting up the Signal Block Module	102
	Setting up the Online Graphical Display	104
	<i>The Information Variables from the Signal Block</i>	104
	<i>Displaying a BMP Picture</i>	104
	<i>Inserting an $x(t)$ Diagram to Match the Picture</i>	105
	Example 2: Specifying a Sequence	106
	Determining the Exact Sequence of Actions	107
	Defining the Steps as a Series	108
	<i>Entering the Series of Steps for the Rotational Speed</i>	108
	<i>Step Series for the Left Brake</i>	109
	<i>Step Series for the Right Brake</i>	110
	Example 3: Setting Desired Values with MWave	111
	The First Example in the MWave Tutorial	111
	The End of the Chapter	112
	Possible Causes of Errors	112

Part 6	Calculations	115
	Introduction	116
	Example 1: Permanent Calculations	116
	Calculating the Mean Value with a Calculation Dataset	117
	Logical Relations with a Calculation Dataset	118
	Logical Relations with a Logic Module	119
	The Layout of the Online Display	120
	Example 2: Action Lists: Calculations in Sequences	121
	Replication of the Previous Example with an Action List	121
	Coupling to the F4 Key	124
	Example 3: A Complex Calculation	126
	The Solution Plan.....	127
	Step 0: The Simulation Environment	127
	Step 1: Entering the Maximum Velocity.....	128
	Step 2: Monitoring the 80% and the 10% Thresholds.....	129
	Step 3: Constructing the Stop-Watch	132
	Step 4: Calculating the Mean Deceleration	132
	Step 5: Marking and Counting the Braking Manoeuvres	133
	The End of the Chapter	134
	Possible Causes of Errors.....	135
	What's Next?.....	135

Part 7	MLab and TestControl	137
	Introduction	138
	Linking TestControl to MLab	138
	Example: Change a Text Display and Restart	140
	Notes for Larger Test bed Projects	142

Before getting Started

Overview, Setup and using MLab for the first time

MLab: Overview

MLab is a universal Data Acquisition and Control Program for Windows. The word "universal" underlines the intention to fulfil many differing requirements for data acquisition. The range of applications stretches from a small, mobile, quickly-configurable measuring system with a graphical display through to a fully-automated test bed filling a whole room, including a complete data-processing system driven by MLab for a duration of several weeks. Such a data-processing system could include programs for checking desired values, performing online calculations, monitoring, executing corrective control and logging as well as performing triggered data acquisition distributed across multiple files.

The Facilities

Measurement data acquisition

MLab offers rapid data acquisition. The current maximum is a continuous data rate of 960 kHz. Many hardware systems, as well as timer-counters and PCM boards for data acquisition, digital and analog, are supported. Any amount of channels can be measured.

Desired Value Determination

In parallel with the measurement, MLab can generate desired values for control purposes. There are various drivers for the digital and analog hardware available. The number of channels is unlimited. There are mathematically-generated desired values (sinusoidal, square wave etc.), lists with target values (ramps) or slave tests (files). For desired values with especially complex interrelationships, there is a graphical editor for desired values (MWave) at your disposal.

Storage

MLab stores the data at an exact time (according to the sampling rate or at a lower frequency) and does this selectively, on demand or at or around borderline values of interest (pre and post triggering) The data can be stored in various files with different acquisition rates. One or more parallel ring buffers for post-mortem tests are also possible.

Freely-Configurable Online-Graphics

Many graphical elements can be set up in a graphical editor: $x(t)$ diagrams, characteristic curves, bars, numeric displays, buttons and static elements, such as text, lines, pictures (drawings, photos). Any number of these elements can be spread out over several pages which can be shown next to each other or alternately during the measurement.

Monitoring

The selective storage and control of sequences begins with monitoring. The measurement signals can be monitored in different ways: for fixed borderline values, for extreme values with running maximum and minimum indications or for sequences of values. For monitoring, there are further elements such as counters, which count, for example, limit exceedances, and timers, which start running, for example, at the point of exceedance and set a flag after a specified duration.

Event-Driven Routines

MLab's operation is event-oriented. This is the crux of the control of sequences. Sequences are always defined linguistically by "if... then..." relationships: If event A occurs then B should happen. Or more complicated: if A and (A2 or A3) occur, then wait 3 seconds and start B, calculate X with formula Y and output the value on channel 3.

Interactivity

Events originate not only from the monitoring modules, but also arrive via digital input channels (switching state of the device under test or test bed) and from the operator (via mouse and keyboard). The operator can also alter numerical values (for example limits, offsets, filenames) during measurement.

Calculations

Calculation datasets can be freely defined to calculate measurement values at each clock cycle or to execute calculations at other defined times. For data input, a scientific calculator is at your disposal. Further modules enable smoothing, extreme or mean value reductions and the various DIN statistical classifications, rain flow classification or FFT analysis. This all takes place online and the calculations can of course be incorporated into the data files.

Complex Sequences

If the available input masks for the definition of sequences, conditions and reactions are not adequate, then action lists are available which allow if/else branching with calculation and the setting of events.

TestControl

If all of this is not enough: the test bed language TestControl from the same company works with the same kernel as MLab and can be coupled to an MLab measurement configuration without difficulty. TestControl as a language embodies MLab. Because TestControl knows more parallel and sequential structures than MLab, it can be adopted for complex loops and sequences.

The Concept of this Tutorial

The combination of all these possibilities in one measurement project or one test bed can be a very complex matter. At the same time, an expert or even a beginner should be able to get going quickly on a new test bed.

The tutorial intends to introduce the user step-by-step to the possibilities of MLab, at the beginning using easy configurations and later on with more complex ones. The approach is constantly orientated towards examples. The order and choice of the examples is certainly somewhat arbitrary, but is based on real-life examples. In order to explain the concept, it will be differentiated between users as described below.

User Groups

One could separate the MLab-users into four different groups, whose requirements can be described in order of increasing complexity as follows:

The first group runs MLab on a mobile computer and expects the tasks to be fulfilled which used to be performed with a multimeter or oscilloscope. For example: connection of cables, selection of measurement ranges and viewing and/or storing values.

The second group uses MLab for diagnosis of a complex measurement problem. The device under test (for example a gearbox or brake) should demonstrate its behaviour under one more different stress situations. Various measured values are calculated together, data is acquired and stored automatically with high sampling rates and in accordance with selective criteria. The graphical display should show the procedure as informatively as possible.

The third group wants to define the stress level of the device under test themselves, in addition to the selective, complex measurement. Before starting the measurement, several parameters are set up, for which MLab provides the desired values. Parallel monitoring actions and digital communication with the test bed with regard to its settings and status come on top of this.

The fourth group wants a test bed which can work automatically for weeks without human intervention. On top of this, the test program should remain flexible. The correlation of the desired values has a high, mathematical complexity. During this continuous test it must be possible, to stop and resume the test routine.

MLab tries to satisfy all these users. For the simple applications the measurement should remain easy and clearly configurable. For complex applications every conceivable flexibility for handling data sets and variables has to be available. The more complex relations in MLab are hidden away through the use of Action Lists. These offer extensive parameters for solving tasks.

The Chapter Sequence

To introduce the specific and complex usage of MLab, this manual is split into two parts. The first part, the **Tutorial**, acquaints the user with the interaction of the program's various components through a series of examples. A second part, the **Reference Handbook**, systematically discusses points of the program, in particular every Action Module, and describes which service is linked to which input field.

The explanations start by setting up a simple measurement (Part 2), then explain monitoring (Part 3), data files (Part 3), the generation of desired values (Part 4), calculations and Action Lists (Part 5), MLab and TestControl (Part 6). In addition, the last chapter goes into the planning and structuring of a major test bed installation.

The examples discussed are provided with the handbook and are available for trying out or for copying. They all work with the test drivers that simulate input and output signals. In this way, they can be used by any user regardless of the chosen hardware.

We are convinced that the program configurations which have to be undertaken at various points can only be kept in mind if a clear goal exists. By orientating the Tutorial heavily towards examples, this conviction is pursued. The examples, however, are born out of experience and so can often serve as easily-understandable suggestions for the user's own test bed installation. We recommend that the examples be replicated, in order to get a feeling for how things fit together.

Preparation: MSetup

Regardless of the actual measurement and its configuration, MLab must know which hardware it has to operate with. If you have received a pre-configured system then you can skip over this chapter; you only have to know the name of the setup file with an **"*.MHC"** extension which was set up for you.

The setup file is the starting point for every MLab project. Defining which setup file you want to work with is the first step when setting up a measurement with MLab. If, however, you never change your computer's plug-in card then it is possible to set the default values for this parameter file once and you will not need to consider these any further. (In the next chapter, you will discover how to do this.)

For users that are configuring their system from new or repeatedly modify it, the following chapter contains necessary information.

Hardware Setup: Differences between Windows 95 and NT

The registration and setup of hardware, i.e. plug-in cards and interface cards is at different stages for Windows 95 and Windows NT. While Microsoft has implemented Plug&Play technology into Windows 95, there is still no corresponding technology for Windows NT. It is planned, however, for the impending Version 5 (Windows 2000).

In order maintain clarity about this problem it makes sense, especially for instrumentation, to differentiate between two configuration levels for your hardware: *data recognition* and *data usage*. Defining data recognition is necessary so that the operating system - and with it the program as well - can actually "see" the data, i.e. that they know where the data can be found. Data usage for measurement is defined by the user. The following table explains the various components.

Data Recognition	<ul style="list-style-type: none">● I/O Address● Resources used for the card: DMA channel, interrupt channel or memory
Data Usage	<ul style="list-style-type: none">● Wiring (single-ended or differential)● Polarity and measurement range● Amplification factors● possible calibration of the pre-amplifier and masking

In the future, the settings for **data recognition** (card recognition), i.e. basic access to the cards, will increasingly be defined via the operating system. For Windows 95, this is already the case. Here, you use the **"Hardware"** point un-

der your computer's "Control Panel" to define the data recognition. If Plug&Play is functioning with Windows 95 (depends upon the card type and the Windows 95 version) then this point is opened up automatically when the card is detected for the first time. If not, then you must call it up automatically from the Start menu. For Windows NT computers, this point does not currently apply and is dealt with instead by MSetup.



Note: a separate guide is available for installing instrumentation and control hardware with the title "**MLab driver installation**". This guides you through the various steps for Windows 95 and Windows NT.

Further Note: under Windows 95, for cards which must be configured for data recognition using switch and jumper settings, the following procedure should be adhered to:

- To begin with, keep the card removed from the computer.
- Register the card via the "Hardware" field in the Control Panel and see which Windows 95 addresses and resources are available to you. It is possible, for example, that the desired base address 300 hex is already occupied and that Windows 95 recommends 320 hex.
- Complete the hardware setup, shut down the computer and switch it off.
- Configure the card as recommended by Windows 95. Install the card and start the computer.

The setup for **data usage** remains unaffected by these points. Under both operating systems, your settings need to be defined once and saved to be used. This is done in all cases with MSetup. For Windows NT, the settings for data recognition (card recognition) are made in MSetup as well. For Windows 95, these are merely displayed.

What you need to know before the Configuration

Before you define the equipment settings (Setup) for your measurement rig, you need to know all the parameters introduced above and the names of the plug-in cards and interface cards. A few explanations concerning this point now follow:

The next section deals in particular with the subject of the base address. As mentioned above, under Windows 95, an available base address will be allocated for you. With Plug&Play cards, (PCMCIA cards or ISA cards with Plug&Play drivers) the address will also be harmonised with the card by software. With other cards, an adjustment must be made on the card by hand using switches.

The Base Address

The hardware for your measurement, the plug-in cards or interface cards are connected to the computer's I/O bus when they are fitted. Each card located in the computer is accessed by the processor via a different base address. These base addresses are mostly to be found in an area between 100 hex and 700 hex. Two rules apply to these base addresses:

- Each base address may only be used once in the system.
- The space between the base addresses should be 10 hex. The permitted addresses can be found in the handbook for your plug-in card.

If these rules are ignored, the processor cannot identify the cards or the data located on them.

Input Voltage Range

Most plug-in cards offer variable usage of the signal connections. Examples of this are:

- Different input voltage ranges, e.g. +/- 10 Volt, +/- 5 Volt, 0-10 Volt etc.
- Bipolar (+ and -) or unipolar (+) input.
- Amplification factors: e.g. 1, 2, 4, 8 or 1, 10, 100, 500.
- Single-ended or differential inputs.

On some cards, these options are set using switches (or jumpers) on the card in the same way as the base address. With more modern cards, these settings can be made via software. This is then achieved using the MSetup program. If the settings are made with jumpers then this must be done both on the card and in MSetup because recognising these via software is often not possible. Whatever the case, you should be aware of the manner of use when configuring the parameters in MSetup.

Running MSetup

Start the MSetup program via the menu bar. You will see a small menu and a tree with three branches: **hardware**, **channels**, **characteristics**.

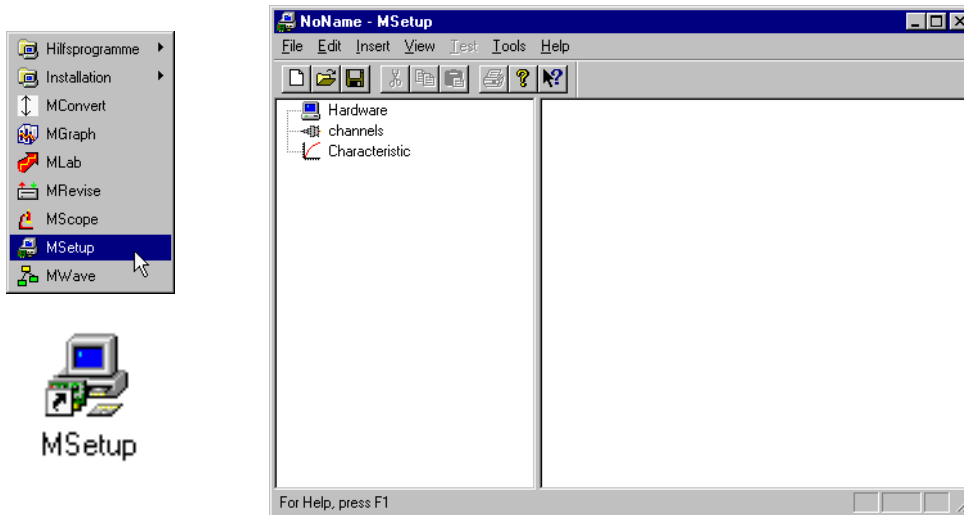


FIGURE 1: MSetup without parameter file

- | | |
|------------------------|---|
| Hardware | Here the plug-in cards and interface cards are added. Below each card the corresponding hardware channels are listed. |
| Channels | Pseudo-channels derived from hardware channels. For bit masking and additional special calibrations. (Up to now not included, can also be resolved in MLab by inserting the channel twice). |
| Characteristics | Definition of characteristic curves to linearize the channels. (Up to now not included, can also be resolved by MLab's linearization module). |

Adding and Configuring the Hardware

In the following, you will insert the plug-in cards and interface cards which are in your computer under the branch **Hardware**:

Open the context menu by clicking with the right mouse button on **Hardware**. Select the menu item **Insert**.

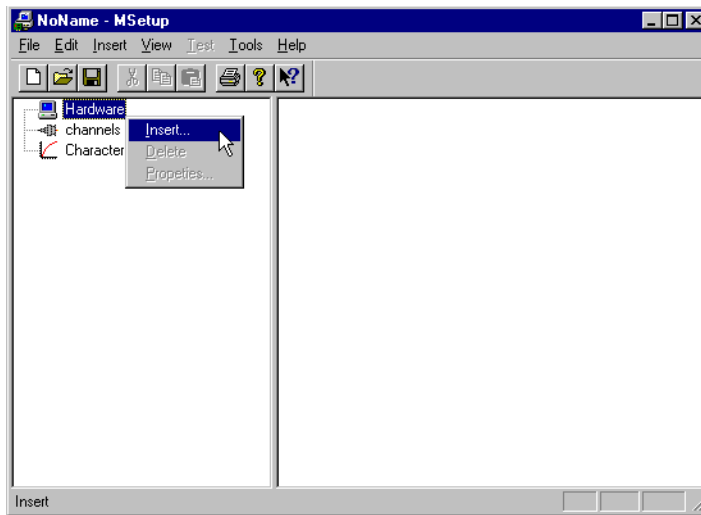


FIGURE 2: MSetup: Inserting new hardware devices via the context menu.

A selection of all cards which can be used with MLab appears. They are split into categories.

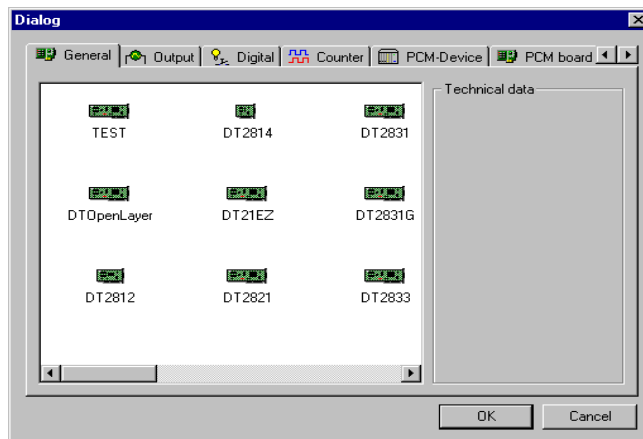


FIGURE 3: MSetup: Selection of cards and devices: Here the acquisition cards

The following list gives an overview of the card drivers which are offered via the window's various property pages:

General	<p>Input cards (acquisition, measurement values):</p> <ul style="list-style-type: none"> ● Test drivers: for data simulation without hardware ● Data Translation cards (DT2801,...) ● Meilhaus (ME300) ● National Instruments (DAQ700, DAQ1200) PCMCIA-cards ● Realtime Device (DM406 for MicroRomulus) ● Analog Device (RTI 800,...)
Output	<p>Output (presets, desired value)</p> <ul style="list-style-type: none"> ● Meilhaus: ME66K8, ME66K12 ● Analog Device: RTI802
Digital	<p>Digital in- and output signals: switch settings and feedback</p> <ul style="list-style-type: none"> ● Meilhaus: ME14A, ME14B ● Intel: I8255 Chip
Counter	<p>Counter:</p> <ul style="list-style-type: none"> ● Meilhaus: ME15A ● Advanced Micro Devices: AM9513 Chip
PCM devices	<p>Devices, which are accessed via a PCM interface card. These devices are only shown when inserted below a PCM board.</p> <ul style="list-style-type: none"> ● CAESAR Datensysteme: MOPS, WDac ● Volland: Compact256 DIN and IRIG ● Johne & Reilhofer: 8K13 ● Kraus Meßtechnik: KMT D- 2/4,..., KMT PCM4,..., MC16....) ● Johne & Reilhofer: Meas PCM
PCM board	<p>PCM = Pulse Code Modulation: continous data stream (variable number of channels)</p> <ul style="list-style-type: none"> ● CAESAR Datensysteme: IF16, IFAT ● Eigner Meßtechnik: ECIA 100 (often referred to as "MOPS card")
PCM output	<ul style="list-style-type: none"> ● Intel: i8255 PCM Out
Special	<ul style="list-style-type: none"> ● PC Speaker ● LPT Port

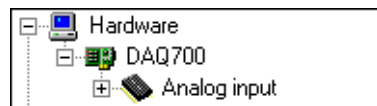
Field bus controller	<ul style="list-style-type: none"> ● GW Instruments: instruNet200 ● Phoenix Kontakt: Phoenix Interbus S ● Open Control
Field bus devices	--- (on request)
IEC controller	--- (on request)
IEC devices	--- (on request)

With the exception of the PCM boards, all cards are defined just by selecting them in one of the property pages. In contrast, the PCM boards are only interface cards. They are only ever used in conjunction with the connected device. To each of the PCM boards, therefore, the device must be added. The following examples explain the user inputs for a simple acquisition card (DAQ700) and then the more complicated inputs for high tech measuring hardware MOPS from CAESAR Datensysteme GmbH:

Example 1: Inserting the DAQ700 card

If you want to insert a normal plug-in card (for example a DAQ700):

- Click on **Hardware**, select **Insert** from the context menu (or double click) and select the **General** property page. Select the DAQ700 card. In order to see it, you have to move the display slightly to the right with the aid of the lower scroll bar.
- The card is inserted into the tree under the **Hardware** menu item using the **OK** button. Click on the small '+' sign next to the word **Hardware** in order to open the tree. The card has been inserted with all of its 16 channels ("analog inputs").

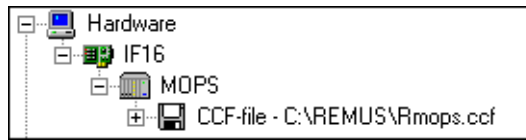


Example 2: Inserting the CAESAR MOPS device

If you want to insert a MOPS device:

- Click on **Hardware**, select **Insert** from the context menu (or double click) and select the property page **PCM boards** and from there your interface card: IF16, IFAT oder PCMCIA MOPS-CARD (= ECIA 100).
- The card is inserted into the tree. Extend the tree, so that you can see the card. Click on it with the right mouse button, so that you can see its context menu. Select **Insert** again.
- You will see the same dialog window as before. Select the **PCM devices** property page. Select "MOPS" and confirm with **OK**.
- The MOPS device is now displayed in the tree below the PCM board. (Or extend the tree via the '+' sign).
- Now click on the **MOPS** field with the right mouse button, so that its context menu is visible. Select **Insert** again.
- Now you will get a small dialog window, where you have to enter the CCF file. The MOPS is a configurable device and the chosen configuration is saved in a CCF file.
- Use the **Browse** button or type the path and file name yourself. Con-

firm with **OK**. The image should look like this:



The configuration of MSetup for standard cases is now complete. The setup has to be saved in a file whose name you can choose. The file gets the **extension "MHC"**. You can now switch to MLab.

The points for data recognition and data usage discussed above should be inspected and adjusted, if you are not sure how they have been set up.

Details on Data Recognition: I/O Address, DMA Channel etc.

The individual settings for data recognition can be found in the Properties window for each card inserted in the tree. Extend the tree, so that you can see the name(s) of the card(s). Click on it with the right mouse button, so that you get the context menu. Select the **Properties** menu item. These properties are different for every card. The following figure shows the properties of the RTI834-card.

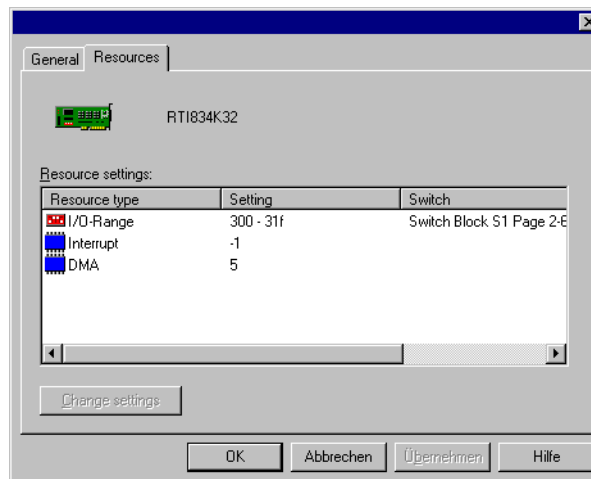


FIGURE 4: MSetup: General properties of the RTI834K32 card

The I/O range is the area above the base address used to transfer data. Here, the base address has been set to 300 hex. It can be changed if you click on the word "I/O-Range" in the window and then use the **Change settings** button which now becomes active. Parallel to this modified setup, the switch setting on the card must be changed on the specified switch block "S1". This is described on page 2-6 in the card manual.



Note: As already mentioned above, **Windows 95** assigns the available address during the driver installation. In this case you cannot change anything here. The setting shown in this window is then only informative. This item can only be modified under (the current) Windows NT.

Details on Data Usage: Measurement Range and Amplification

The parameters for the signal connections always have standard settings. Please check that these settings correspond to your usage and change if necessary. The settings can be split into two groups:

- Settings which are valid for the whole card.
- Settings which are valid for the individual channels.

Card settings

Open the Properties window for the card. To do so, extend the tree, so you can see the channel groups. For an acquisition card, the channel group will be called **Analog Input**. The RTI834K32 card has two additional digital channels. Click on the group name (**Analog Input**) with the right mouse button and select the **Properties** menu item. You will see the following image for the RTI834K32 card.

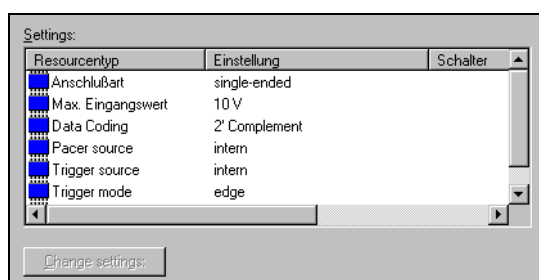


FIGURE 5: Common properties of the analog inputs of the RTI834K32 card.

In order to change the parameters, select the appropriate item in the **Resource Type** column, at which the lower button **Change Settings** is activated. Then use the button to change the values.

Settings for an individual channel

In order to inspect and modify the settings for an individual channel, double click on the chosen channel. Its properties window appears. The settings which are possible here again depend on the type of card and perhaps (with PCM boards) on the connected device. Once again, the example for the RTI834K32 is shown. First, select the second property page called **Settings**.

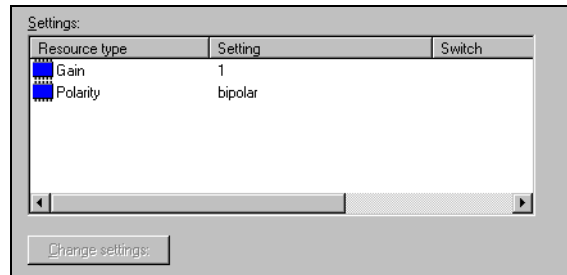


FIGURE 6: MSetup: Settings for an individual channel (here: RTI834K32)

For this card you can adjust the amplification and the polarity for each channel. Let us assume, you have set an input voltage range of +/- 5 Volt (in the card settings above (**Max. input value**)), which corresponds to a bipolar signal. If your sensor for this channel has a voltage range of 0-5 Volt, than you should switch here to *unipolar*. If, additionally, you know that your signal only varies within a range of 0-2.5 Volt, then you should set the amplification to 2 in order to achieve the optimum resolution.

The first property page of the same dialog window, which is called **General**, shows the following parameters.

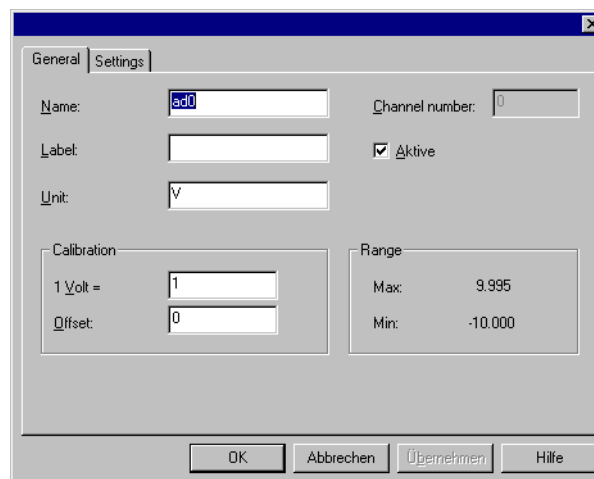


FIGURE 7: MSetup: General parameters for a channel

An example with pre-amplifier: Temperature measurement with the 5B module

In most cases, nothing has to be entered here. It is possible to define default settings for channel names and captions later in MLab. Additionally, you can carry out a special pre-calibration. This pre-calibration is intended for pre-amplifiers such as the 5B module, where two calibration steps occur, once for the measurement amplifier and once for the actual measured value. The following example explains an application:

The temperature should be measured with one channel (e.g. ad0). The setup looks as follows:

- Temperature range from 0 to 300 degrees Celsius
- A temperature sensor converts 0-300 °C to 0-100 mV
- A 5B module amplifies 0-100 mV to 0-5 Volt
- The temperature should be displayed in MLab. For example, as a vertical bar with a scale from 0 to 300 degrees Celsius

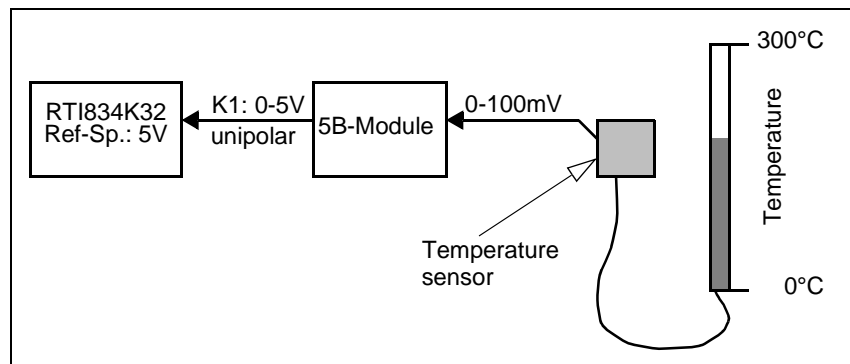


FIGURE 8: Example: Connecting a temperature sensor.

In order to set this up, you have to take the following steps:

1. A reference voltage of 5 Volts is set in the **Analog Inputs** properties window for the RTI834K32 card. The parameter is called **Max. Input Value**, is selected and set to 5 Volts using the **Change Settings** button.
2. In the channel's properties window, on the **Settings** property page, the **Analog Mode** entry is set to *unipolar*.
3. Changes to the 5B module are made in the **General** property page. 1 Volt corresponds to 20 mV (5 V correspond to 100 mV). MSetup requires the value for 1 Volt at this position. Therefore, the value entered has to be 20. The next figure shows the correct settings:

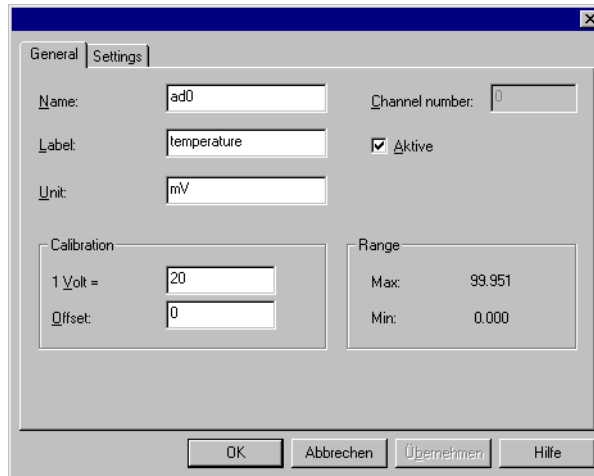


FIGURE 9: Settings for the amplifier calibration

4. In MLab, you set up the actual units used, in this case the temperature. MLab will indicate an input voltage range of 0-100 mV to you for this channel. For this, you enter a measuring range of 0-300 degrees Celsius in the channel's calibration window.

With "pre-calibration" of this sort, make use of the **Label** and **Unit** options. These are the default settings for the same values in MLab.

Setting up Two Cards of the Same Type

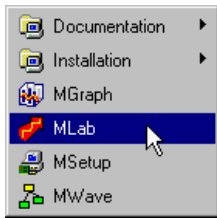
If you use two cards of the same type next to each other, in order to get double the number the channels, the two following points should be considered:

1. The base addresses for the cards have to be different.
2. The cards must have different names.

Both items are properties of the card. Insert both cards under **Hardware** one after the other into the project tree. Then open the cards' properties windows one after the other. On the **General** property page, give the cards different names (if you are using an ME300 card, call the first one e.g. "ME300_1" and the second one "ME300_2"). Check in the second property page of each card, the **Resources** property page, that the base addresses differ.

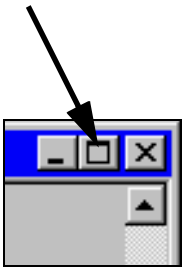
MSetup usually numbers the channel names automatically. You will need the different names of the cards when you enter the clock source and additional devices in MLab.

Starting MLab for the First Time



The installation program for MicroEdition automatically creates a sub-menu in the Windows Start Menu called MicroEdition. The program is started by selecting the **MLab** menu item.

Should shortcuts have been created on the desktop during the installation, then the program full-screen display can be started directly by double clicking on the program icon (see left).



Full-screen display

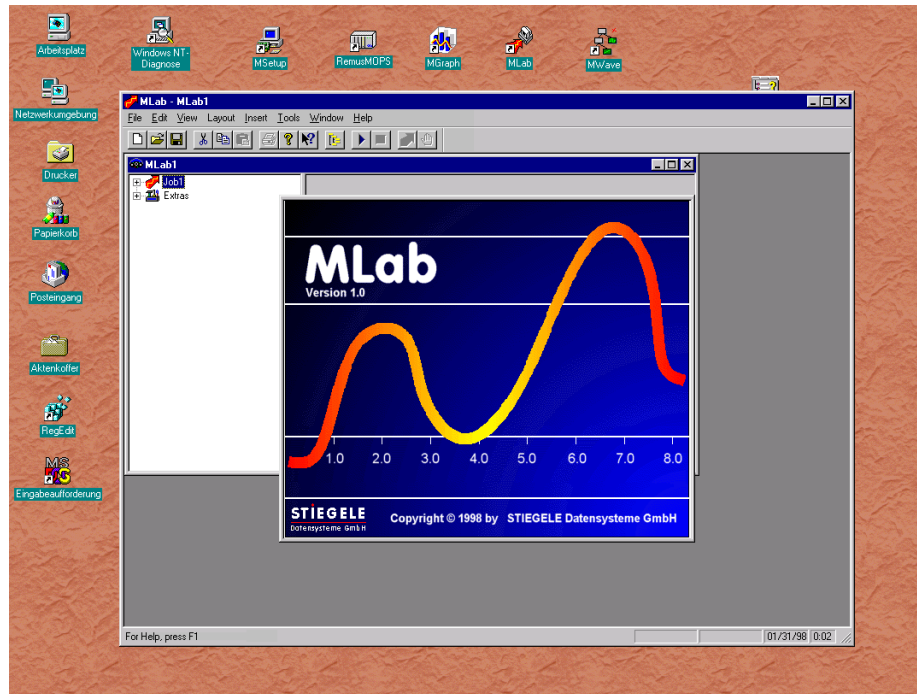


FIGURE 10: MLab program start.

With small screens, it is advantageous to switch to full-screen display in order to make it possible to display several windows at the same time.

Input Wizard

When starting MLab for the first time, a Setup Wizard will start up if required. This will offer to set defaults for the hardware and clock sources. You can also start up the wizard at any time later via the **Tools/Wizards/Hardware** menu item. There now follows a brief description of how you can use it.

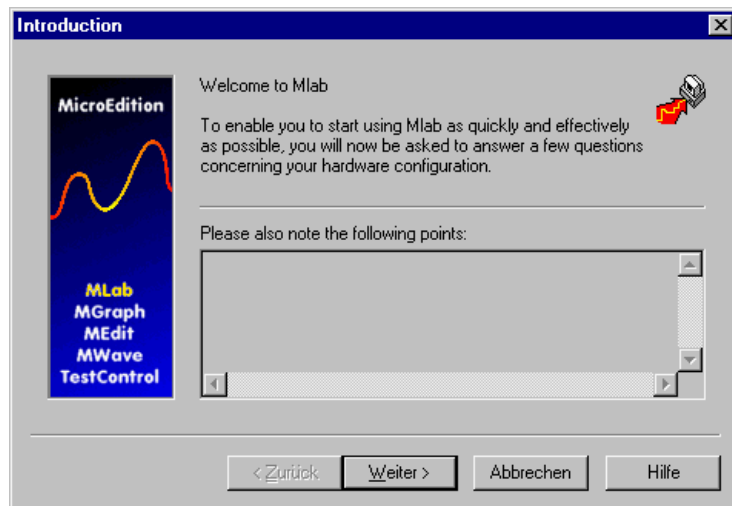


FIGURE 11: First page of the Hardware Wizard.

On two subsequent pages, you will be asked for the name of your setup file (*.MHC) and the card for this setup that should operate as the clock source.

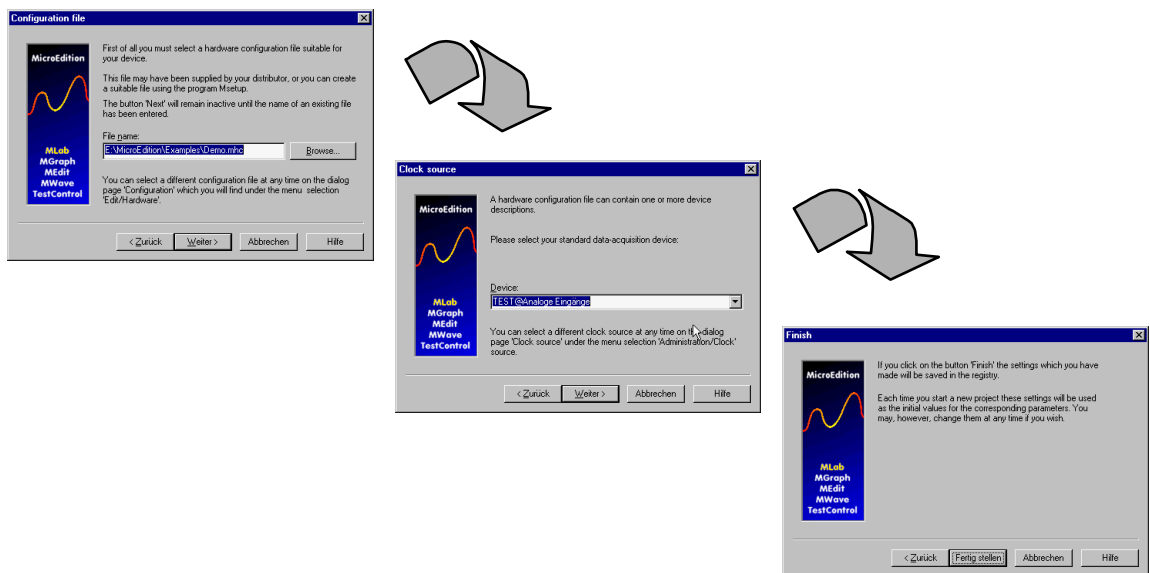


FIGURE 12: Further pages of the Hardware Wizard.

If at this stage you don't know what to enter here, use the defaults offered. In the next chapter you will be shown how to perform the settings yourself.



First Measurements

Hardware, clock rate, datasets, graphical displays

Basic Setup for a Measurement

In the following chapter basic steps are discussed, which you need to take in order to use MLab for a measurement. This basic setup shows how you can display and store all available measured signals. The following steps are necessary for this basic setup:

1. Define the hardware
2. Define the clock source and sampling rate
3. Define datasets
4. Set up graphical displays

1st. Step: Defining Hardware



MLab's project tree shows you two main branches: **Job1** and **Extras**. To extend **Extras**, you have to click on the small plus sign in front of the word. Double click the **Hardware** field. An alternative method for selecting this window is via the menu item: **Edit/Hardware**.

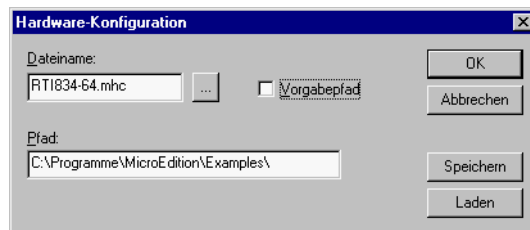


FIGURE 13: Declaration of Setup File

In this window, select the setup file which you defined in MSetup. It describes which cards are installed in the computer, how they are set up and which datasets will be available. Via the small "(...)" button you will get a dialog window, with which you can search for the file on your hard disk.

Pre-setting the Hardware

Use the **Save** button in the window above if you want to define this setup file as the standard. The hardware configuration for new projects will then always be pre-set as it is seen here now. In this event, you will no longer have to set up this main item of the basic setup for a measurement. From now on you will only have to set up 3 further main items.

The **Load** button is used to reload the saved configuration, if you used a different setup file temporarily.

2nd. Step: Clock Source and Sampling Rate

Your largely analogue measuring signals will be translated into digital number series with the aid of the acquisition card or acquisition devices. The signals are sampled at a selected clock rate and converted into numbers depending upon the resolution (mostly 12 or 16 bit). The number series which arise from each channel are called datasets.

Here you have to define for your MLab project which acquisition card should set the sampling rate and how high this sampling rate should be.

*Defining the
Clock Source*

The following window is opened by double clicking the **Job1** field or via the **Administration/Sampling Rate** menu item. Change to the second property page called **Clock Source**.

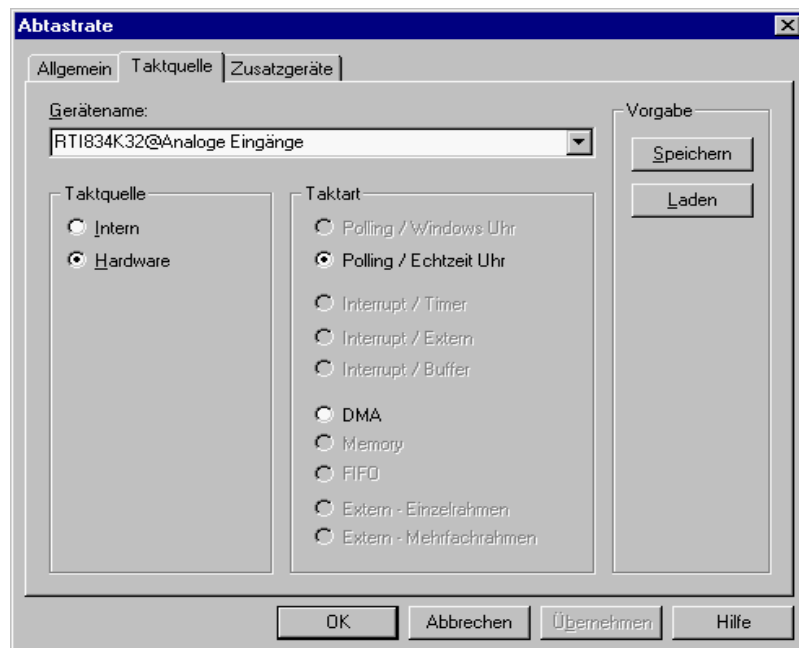


FIGURE 14: Setting the Clock Source



At first, the device defined as standard is shown here as **device name**. Open up the selection box for the **Device Name** field. It will display all devices from your setup file and their ranges of use. For the basic setup, select the first visible option which contains the name "Analog Inputs". If, due to your setup file, there are several sub-devices in this selection box, i.e. other analog inputs, digital inputs, analog outputs or digital outputs are listed, the following must be observed: all the other sub-devices - except "**Analog Inputs**" - which should be clocked as well, have to be named individually on the window's next property page (**Additional Devices**).

For the basic setup of the measurement, leave the other parameters of the fields **Clock Source** and **Clock Type** as they are suggested to you. These setup items are discussed in more detail in the chapter concerned with the "Storage" topic.

Pre-setting the Clock Source

Use the **Save** button in the window above to define the selected device name as standard. The device name for new projects will then always be pre-set as it is seen here now. In this event, you will no longer have to set up this main item of the basic setup for a measurement. From now on you will only have to set up 2 further main items.

As this setup goes together with your setup file you should only do this if your setup file (see Hardware Configuration above) remains the same. The **Load** button is used to reload the saved configuration, if you selected a different device temporarily.

*Defining the
Sampling Rate*

Now select the window's first property page. The parameters offered here, depend upon the hardware installed and therefore differ according to device and mode of operation.

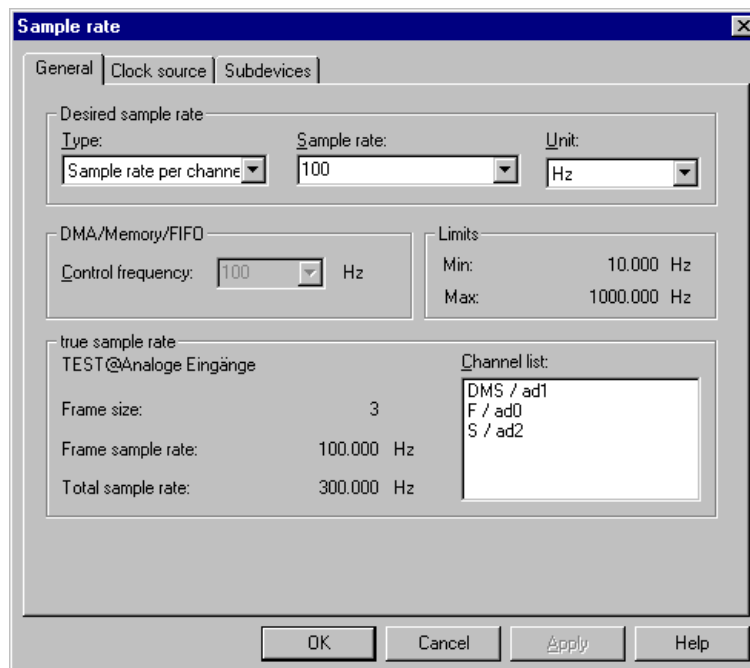


FIGURE 15: Dialog for general settings for the sampling rate.

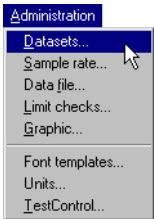
To set up the sampling rate exactly for very high and also for very low sampling rates, it is necessary to make an estimation concerning the frequencies of the signals which have to be measured, the amount of data expected to be stored and the possibilities offered by the acquisition hardware in use. You will find more about this subject in the chapter dealing with the "Storage" topic.

At this stage, we are principally concerned with measuring the signals without considering high frequency components. For this, we recommend a sampling rate of 100 to 1000 Hz per channel. This is neither a problem for the acquisition hardware nor for the program.

3rd Step: Defining Datasets

Every dataset which should be available for the measurement project must now be entered into the dataset list. Hereby, they are given a meaningful name and calibrated in such a way, that the signal actually being measured (i.e. at the sensor) appears in the correct units.

Inserting Datasets



Via a double click on the **Datasets** field in the project tree or via the **Administration/Datasets...** menu item, you get the corresponding dialog window. Here, data sets can be created, modified and deleted.

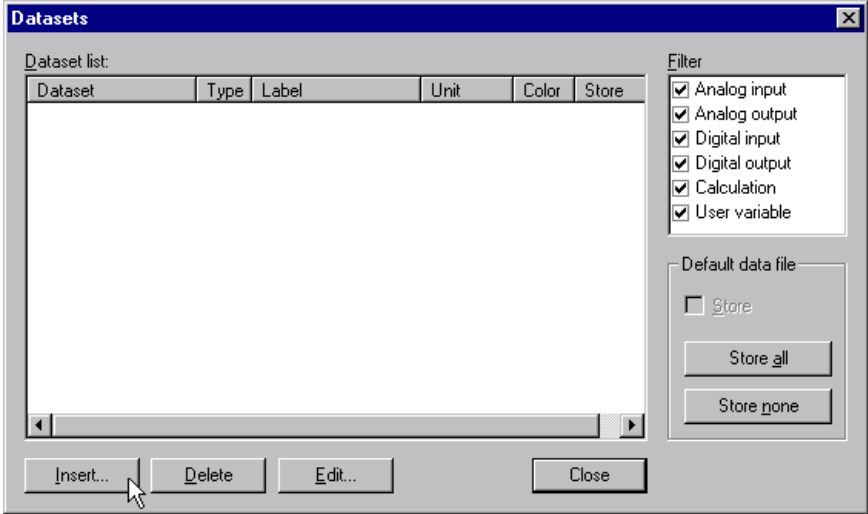


FIGURE 16: Dialog with an empty list of the defined datasets.

In order to define new datasets, you have to activate the **Insert** button. A dialog window opens, in which the available channels are listed. Normally, **Analog input** is active as category and the datasets which are ready to be inserted are shown. Datasets from other categories can be inserted as well. We will modify this window often in later chapters.

To begin with, select the necessary channels from the list and then click the **OK** button. With the aid of Shift or Control keys, several channels can be selected at the same time. All of them can be selected with the aid of the **Set** button in the **All markers** group.

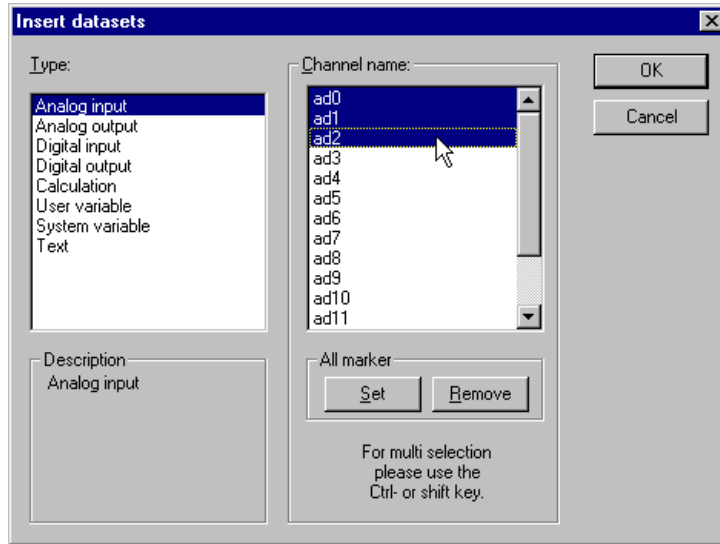


FIGURE 17: Dialog to insert datasets.



Tip: If you want to insert a lot of datasets, you should have a look at the **Extras/Options** menu item. There, the option **When creating a new object, open Edit window** should be deactivated. If not, this Edit window appears for every inserted channel. These calls cannot be aborted. It is recommended to re-activate this option once the datasets have been inserted because, for all other insertable objects, such as graphical items or action modules, it is very useful.

The new datasets are inserted into the dataset list. Each dataset is automatically allocated a synthetic name during creation. Here, analog input datasets get the prefix "X_" and analog output channels a "W_". This naming convention is usual with test beds. To edit, select the name of the dataset and click the **Edit** button.

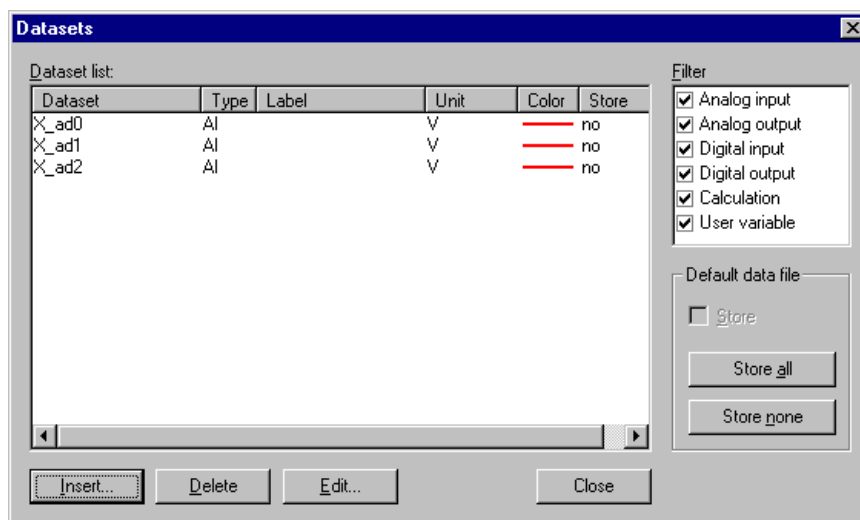


FIGURE 18: Dataset list with three datasets defined.

For the basic setup of a measurement, all datasets should be selected for Storage. During data storage, they go into the standard data file and are available for analysis.

Editing Datasets

Editing Dataset Properties

You can now edit the properties of a dataset individually, apply a different name, caption or calibration. To do this, select the name of the desired dataset from the left column and then use the **Edit...** button. The **Dataset Settings...** dialog window will open up, in which the dataset's properties are displayed on three property pages.

You can close the dataset list and modify the datasets later. Every dataset has its field in the project tree, in the **Datasets** branch. You can open its properties by double clicking on the desired dataset and will get the following dialog window:



FIGURE 19: Properties Window for a Dataset.

The **General** property page contains the dataset's name, caption, unit and the name of the hardware channel which is linked to the dataset. The **Calibration** property page is used to change the calibration range. Here, a physical measurement value is allocated to the input value (normally voltage V). The dialog window above describes a dataset which should measure the force on a cross-rod. The input signal of +/-10 volts corresponds to a force of +/-100 kN.

In order to describe a dataset, the following 3 fields are at your disposal:

- Dataset Name** The name of a dataset has to be unique and can only appear once in a project. It has to start with a letter and may only contain digits, letters and underscores. Spaces, points etc. are not allowed. This name is used everywhere in the project when referring to the channel, for example with calculations, monitoring modules, graphical fields etc..
- Caption** The caption can be chosen at will. Spaces and special character can be used. It is used to label graphical displays on the screen.
- Unit** The interpretation of the dataset values and the way they are displayed should be held here. The calibration of the channel applies to this unit.

Dataset Calibration

For the calibration, select the window's second property page. The calibration converts the input range of the channel on the acquisition card into the actual measured range in the units indicated above (for example Nm, bar, kN, mm, rpm, Degree C, etc). The following picture shows a calibration for a range of +/- 100 kN.



FIGURE 20: Calibration example.

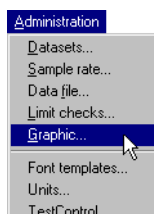
Because it is known how the input signal range relates to the physical value, the range option is selected. There are 3 ways of entering the information at your disposal:

- Range** This is the easiest way if you know to which range the input voltage range corresponds.
- Simple** If you know the linear conversion yourself, you can insert the factor and constant directly.
- Key Points** If you don't know the relation between input voltage and measurement value exactly, then measure the voltage for two measurement values (key points) which you know exactly. Enter both pairs of values here.

By returning to the first property page, the calibration values are accepted and are displayed as the range. Confirm the changes made with the **OK** button.

The definition of the datasets is now complete and the **Datasets** dialog window can be closed by clicking on the Close button. The inserted channels appear in the project tree under the **Datasets** branch.

4th. Step: Defining Graphical Items



The **Graphical** dialog window, in which all graphic pages of the present project are listed, is reached via a double click on the **Graphical** field in the project tree or via the **Administration/Graphical...** menu item.

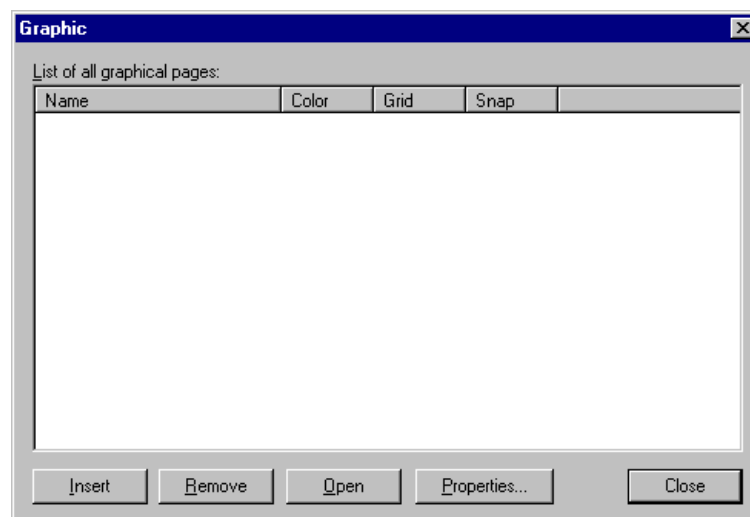


FIGURE 21: Dialog with a empty list of the defined graphic pages.

*Inserting a new
Graphic Page*

In order to generate a new graphic page, click on the **Insert** button. The graphic page then generated is added to the list.

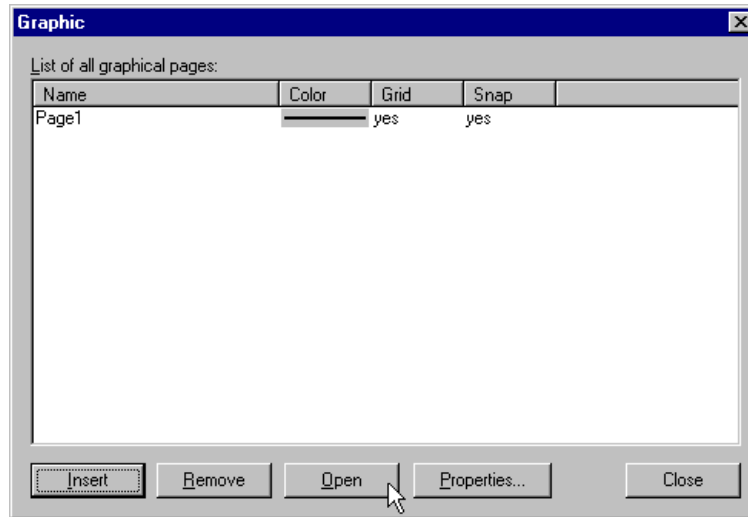


FIGURE 22: Dialog with one graphic page defined in the list.

*Properties of a
Graphic Page*

Each graphic page should have a unique name to make it easier to select it from the list later on. The name can be changed in the properties dialog. As in previous dialogs, select the corresponding entry in the list with the mouse and then click the **Properties...** button.

In this dialog, settings can be made for the grid size and capture as well as various colour settings.



Tip: A quicker way to define new graphic pages is as follows: Open the **Extras/Options** menu item and activate the field **When creating a new object, open Edit window**. Now click in the project tree on the **Graphical** field and select **New** from the context menu. The new page now opens up directly in the editing view.

Inserting Graphical Objects into a Graphic Page

Opening the Graphic Page

The newly-created graphic page is initially empty. In order to fill it with graphical elements, this graphic page has to be opened. To do this, select the graphic page from the list and then click on the **Open** button in the context menu. At this, the dialog closes and the editing window for this graphic page opens.

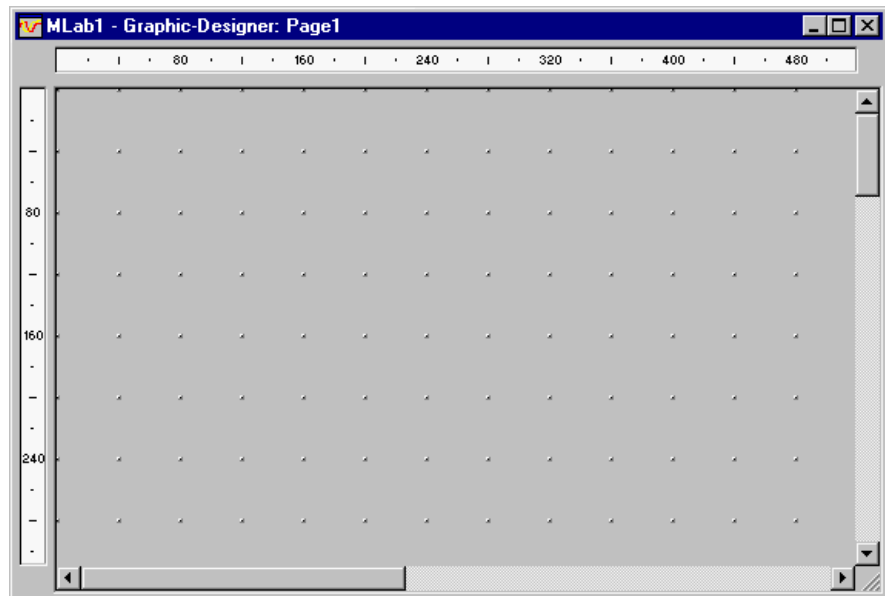


FIGURE 23: Empty graphic page, as opened after it is created.

Several tool bars for defining the graphical display are activated for this graphical window in editing mode.

Object Types

Basically, there are three types of graphical objects which can be used in a graphic page:

- Dynamic Objects:** These are objects which are linked to datasets whose values and properties are displayed. Examples of these are *x(t) diagrams*, *bars*, *numerical values* and *switches*.
- Static Objects:** Fixed objects such as *rectangles*, *texts* and *images* belong to this group. These are mainly used to create an attractive and informative layout.
- Input Objects:** These serve to make inputs possible during the measurement. They include *buttons*, *input numbers* and *input texts*.

For the basic setup of a measurement, *x(t) diagrams* and numerical displays (display as digits) are usually necessary.

The following shows how a display is defined with these two graphical object types. The number of graphical objects is only limited by space. It is possible to define several graphic pages next to each other. During the measurement, it is possible to switch between the graphic pages.

Inserting an x(t) Diagram



1. First click on the x(t) diagram button (see left) in the **Graphical Objects** tool bar which, in the basic setup, is visible on the left-hand side of the program window. An alternative is the **Insert/x(t) Diagram** menu item.
2. Now move the mouse pointer on the graphic page to the position where the top left corner of the diagram should be roughly.
3. Next drag out the diagram to the desired size by **holding down** the left mouse button.
4. After releasing the mouse button, the diagram appears with 8 selection points which can be used to change the size of the object.
5. By double clicking on the graphical field, the properties window opens up (as an alternative you can select the **Properties** field via the **context menu** for the right mouse button).

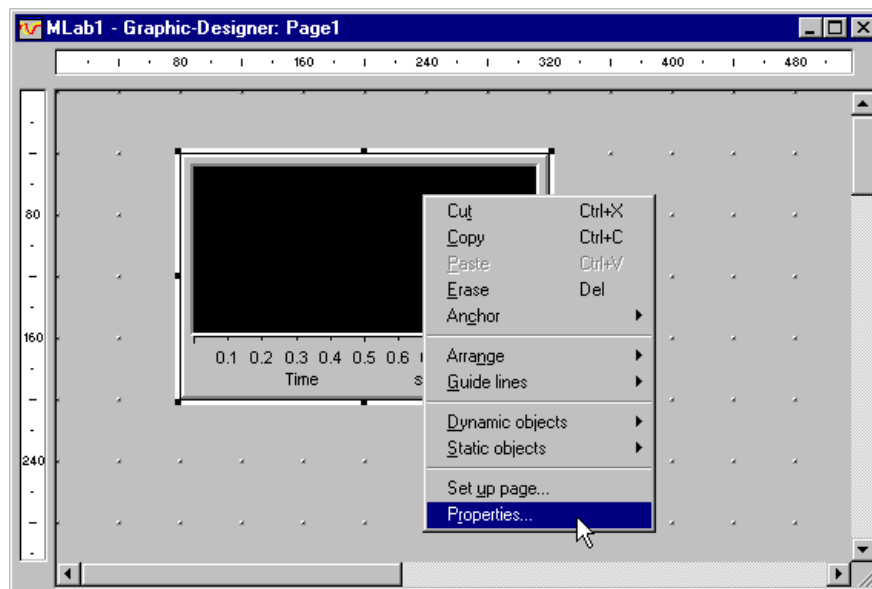
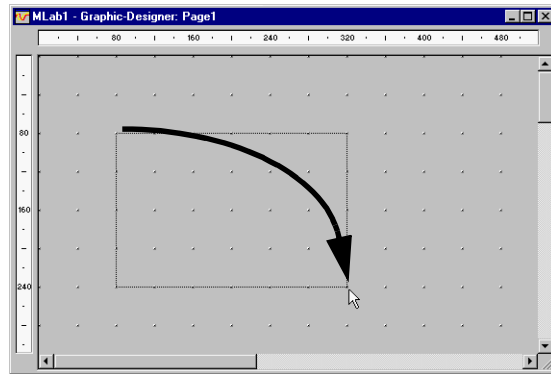


FIGURE 24: Context Menu for a graphical object.

The Properties of a graphical object vary from object to object. Therefore, the following explanations are only valid for the $x(t)$ diagram. In the **General** property page, the major properties of the $x(t)$ diagram can be modified.

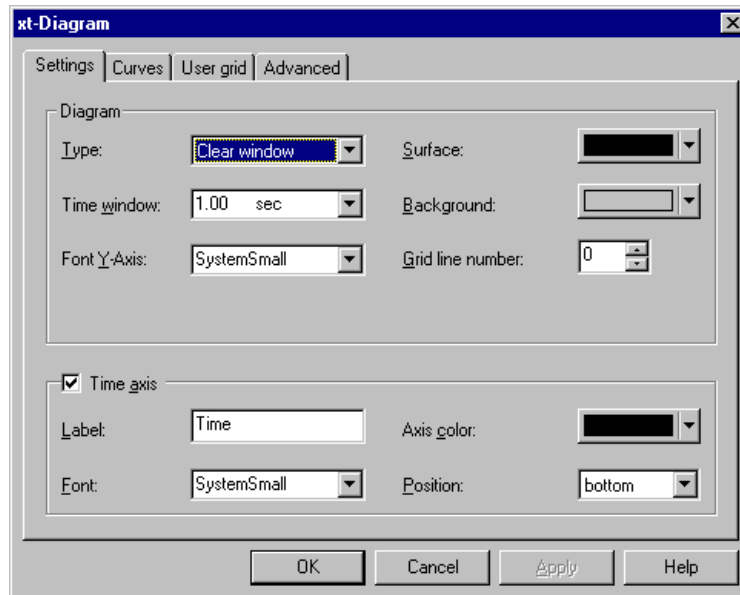


FIGURE 25: Dialog with the general properties of a $x(t)$ Diagram.

The dialog above contains the following settings for the $x(t)$ diagram:

- Type:** Manner in which the diagram is refreshed in the online mode. **Clear Window** dictates that the contents of the window are deleted whenever the signal reaches the right. The signal display then starts again on the left edge of the window.
- Time-slice:** Size of the windows x-axis, i.e. the time span which is shown within the diagram.
- Font Y-Axis:** Font for the diagram's y-axis. Master fonts, which can be extended and modified at will (see the **Tools/Fonts** field in the project tree) are at your disposal.
- Diagram Area:** Background colour of the drawing area.
- Background** Background colour of the window frame.
- Number of Gridlines:** Maximum number of vertical grid lines in the diagram.

In the dialog shown below, the time axis can be parametrized. This is only displayed if the appropriate check box is activated.

In order to insert a curve definition, select the **Channels** property page. This shows the list of curves which will be displayed.

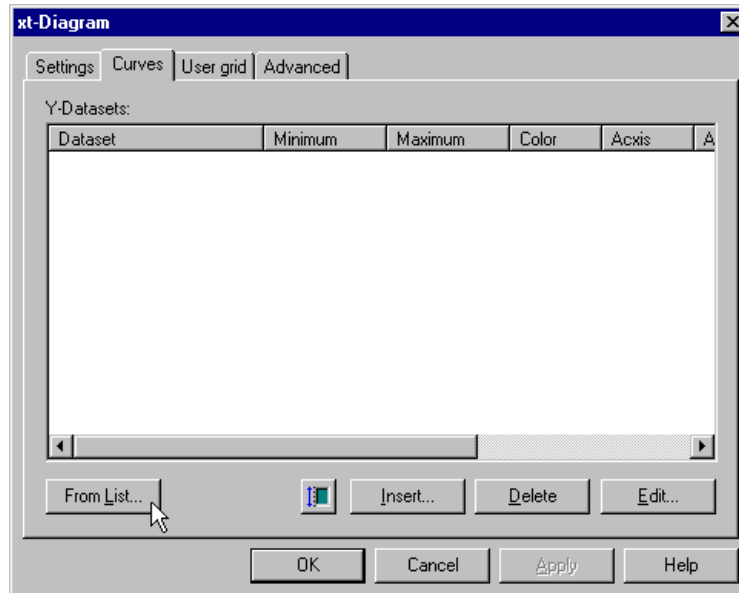


FIGURE 26: List of displayable datasets (Y-Channels)

Individual curve definitions can be created using the **Insert** button. If several datasets should be displayed, it is faster to use the **From List...** button. In the dialog which then appears, select all datasets which should be displayed in the object. After clicking the **OK** button, the new curve definitions are inserted into the list. Each curve is automatically assigned a different colour.

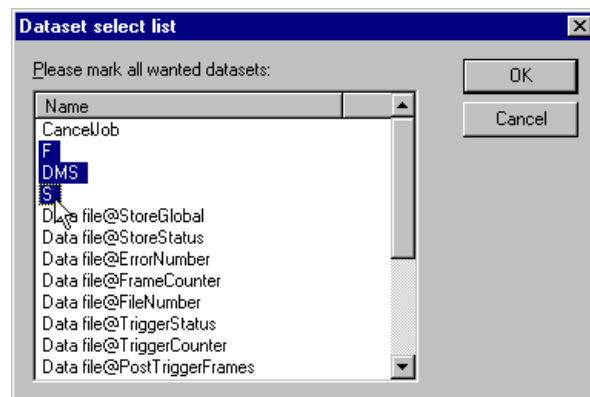


FIGURE 27: Dialog with dataset selection list.

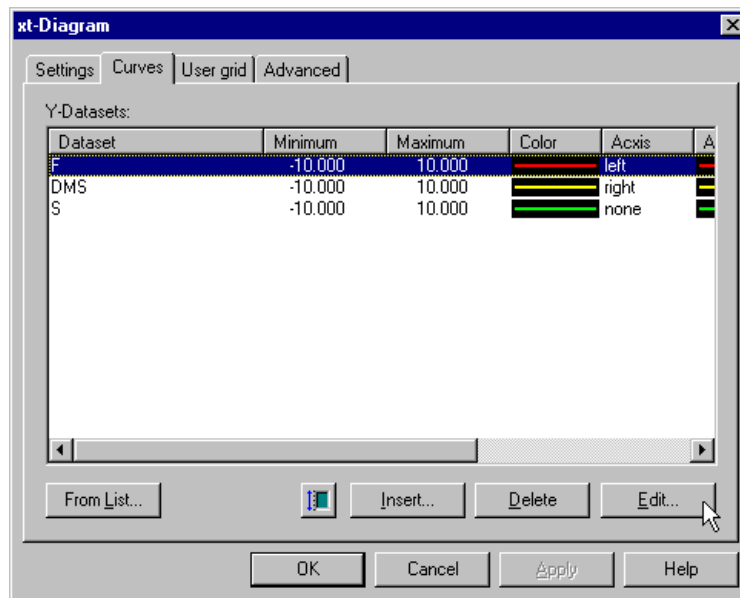


FIGURE 28: Dialog with list of the defined channels of a $x(t)$ diagram.

The colour, the axis scale and other properties can be modified using the properties dialog. To do this, select an entry from the list as in previous dialogs and press the **Edit** button.

Inserting a numerical Field (Display as Digits)



In order to insert a numerical field, select the appropriate button from the tool bar or the **Insert/Numerical Field** menu item. Drag open the field in the same way as already described for the $x(t)$ diagram. The field now appears on the graphic page. By double-clicking on the field, you will get the associated properties window.

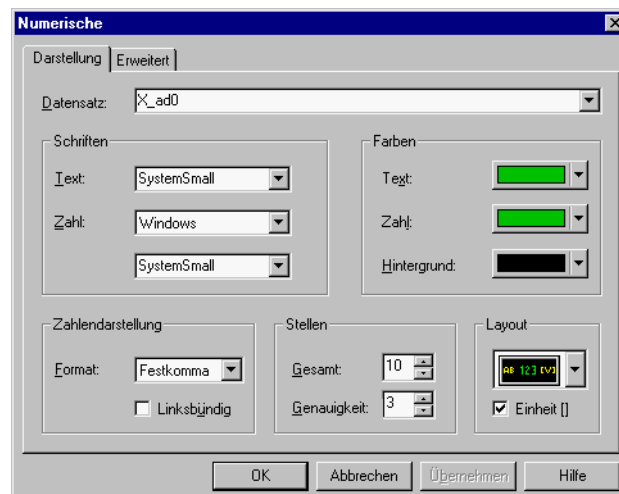


FIGURE 29: Properties window of a numerical field.

The software offers many display options; it is best, you try these out for yourself to see what effect they have. When selecting fonts, make sure that the display has enough space in the field.

The most important option is the dataset. In the dataset list, you can see very many datasets - many more than you have inserted yourself. MLab maintains many additional datasets in the background which are also available for display. If you have set the presettings to begin with "X_" then your datasets will appear right at the bottom of the list. The list display reacts to the initial letter. If you type an 'x', the list will already open up at this position.

Copy of a Field

If you are satisfied with the setup of a field and the entries repeat themselves for further channels, then you can copy a field in the usual "windows manner". Simply select the field, CTRL+C to make a copy and CTRL+V to paste onto the page.



Pay attention to the screen width !

Please note that a ruler is displayed on the edge of the graphic page. It displays the pixels. Be aware of your screen's pixel width when constructing your page. If your screen can only display a resolution of 800 x 600 pixels, you will not be able to see graphical elements outside of this region in the on-line display. You can, however, plan out the project for a different computer with a higher resolution.

Together with a numerical value object, the graphic page could appear as follows:

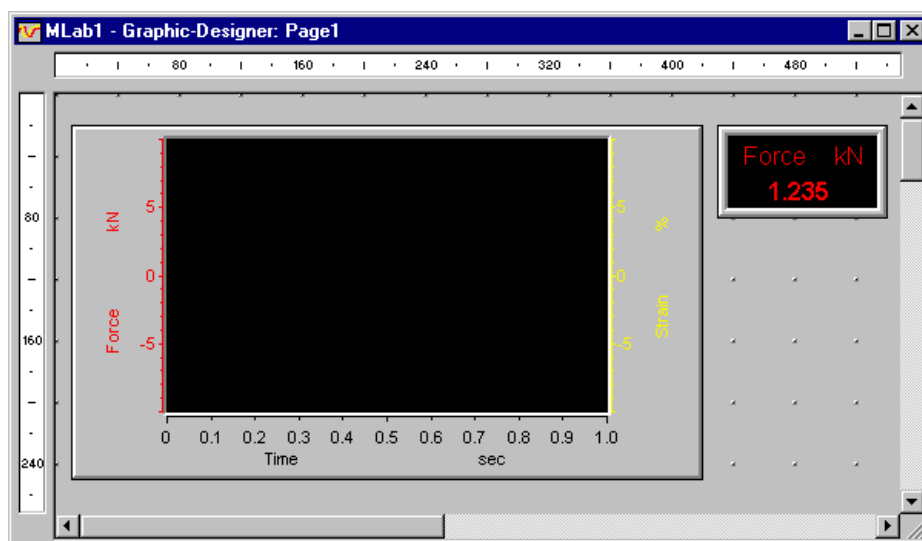


FIGURE 30: Example of a graphic page with an $x(t)$ object and a numerical object

Closing the graphic page



In order to close a graphic page, click the **Close** button (see left) in the menu bar of the window. You will see, that the menu changes and that the graphic toolbars disappear. A soon as you open an new graphic page, the graphic menu reappears.

Starting a Measurement



Starting a Measurement

Having made all definitions, the measurement can now be started. This can be done in three different ways:

1. Via the **Acquisition/Start** menu item
2. Via the CTRL+F2 key combination.
3. Via a mouse click on the associated button (see left) in the **Standard** toolbar.

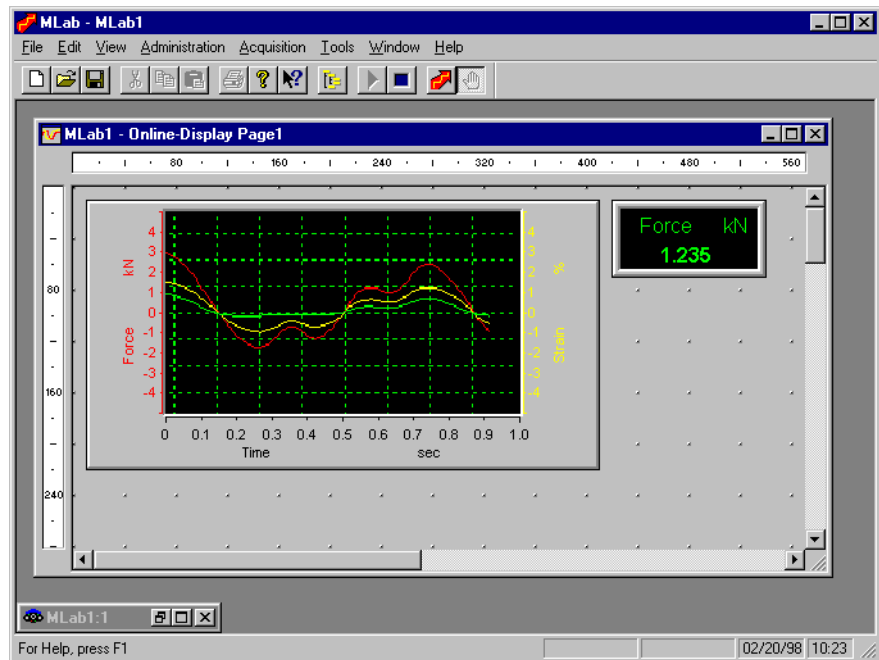


FIGURE 31: Example of an MLab online display.



Starting data storage



The data storage is started with the mouse via a button in the **Standard** toolbar.



Note: Only datasets whose storage status is active are stored in the standard data file. If no dataset of the dataset list has this status, the button stays deactivated.



Stopping data storage

Data storage is stopped with the mouse via the adjacent button.



Starting and stopping data storage can be repeated at will.



Stopping the measurement

As when starting the measurement, there are also three ways to stop the measurement:

1. Via the **Acquisition/Stop** menu item.
2. Via the CTRL+F4 key combination.
3. Via a mouse click on the associated button (see left) in the **Standard** toolbar.



The measurement can also be stopped if data is still being stored. Storage will be terminated normally.



Saving the project



Note: Save your current project status when leaving the program and after making extensive entries in MLab. This can be done very quickly using the Save button (see left) in the Standard toolbar. The project gets a name - which you can choose - and is saved in a file with the extension **".MLB"**. When you start the program the next time, you can continue your work on the project or start immediately with the measurement.

Possible Causes of Errors

MLab generates error messages for incorrect entries which you have to confirm. After this an error window opens up and shows the problems which arose when switching to active measurement. Sometimes the cause is obvious, e.g. if no dataset is available at all or if no graphic page was selected. Correct your entries and start the measurement again.

A short list of error messages follows, for which the cause or explanation is perhaps not so easy to find:

- **This program was terminated because of an invalid operation:** These are often problems with the name of the datasets. You might have a graphical object without a channel to display (e.g. an $x(t)$ diagram). Or there is a typing mistake in the dataset name which is therefore invalid.
- **Error when starting a Job:** This message appears if you use the test driver (for example when testing the program) and, when entering the clock source and clock type, "polling / real-time clock" was selected. Close MLab and start the Task Manager (CTRL+ALT+DEL). Close all programs with the names "MSetup", "MKKernel" and "MKTestDriver". Start MLab again and use the clock type "polling / windows clock" clock type.
- No error message appears if the name of a dataset or a variable in a numerical field is written wrongly. There will simply be no value displayed. If the format with caption and unit was selected, then the words "Caption" and "Unit" are written in the field.

The End of the Chapter

You have learned how to set up a complete measurement. A default setup should always exist parallel to your own - possibly very complex - measurement project. There should be a "reference" project, that means an MLB-file, for large measurement installations or test beds in particular. This should display all channels but perform no other function. You can always resort to this project file if in doubt.

The example "**Reference.mlb**" shows such a reference project for the test driver.

Further Notes



Reference.mlb

- In the following chapters you will find notes on setting up and parameterising other graphical objects (see Index under "Graphic").
- Graphic pages don't have to be created anew for every project. In the context menu of every graphic window you will find the option to save the page as a **graphical template**. In a new project you can load such a page as a whole. For this, you will find the **Load** menu item in the context menu of the **Graphical field**. The example "**Reference.mlb**" uses such a template for the x(t) diagram page and for the page with the numerical displays. You can use these templates in your project; you only have to ensure that the dataset names used are modified.

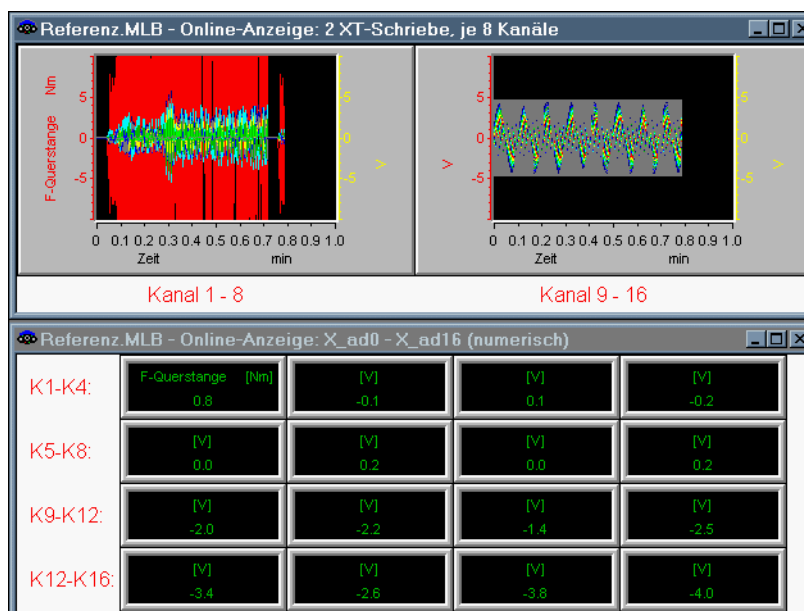


FIGURE 32: Example of a reference display (Reference.mlb). Using two graphical templates: "XT_16_Channels" and "NUM_16_Channels".

- The setup of the data file was not described in this first chapter. As default, it is always called "Test.dat" + "Test.rms" and can be found in the "examples" directory. The setup can be modified via the **Administration/Data File** menu item. You can find further information in the chapter "Storage".



Monitoring

Checking Measured Signals

Introduction

A measurement is conducted in order to examine the behaviour of significant values for a device under test - whether or not permitted limits are reached or exceeded. Often, it is necessary to react whenever limit values are exceeded. Monitoring functions serve this purpose.

In the following, various types of monitoring functions are dealt with using four examples:

1. Standard limit monitoring: store when above a particular value, when below do nothing.
2. Graphical checking: bars with extremity markers. Extremity markers reset by key press.
3. Displaying extreme values in an $x(t)$ diagram.
4. Range-checking: as long as the signal is within the permitted range, the switch is shown in green. If the level leaves the permitted range, a red warning indication is displayed.

1st. Example: Storage above a limit value

The aim is as follows: Whenever the value of the dataset "X_ad1" is above 3 volts, it should be saved, but not below.

In order to achieve this, a limit check is used. In MLab there is a small and a large limit check module. The small one, the **Standard Limit Monitor**, is sufficient for our example. The large one, the **Limit Monitor** action module, is dealt with in this chapter's fourth example.

Definition of a Standard Limit Monitor

Use the **Administration/Limit Monitor** menu item. A window opens up with a list in which the limit monitors can be entered.

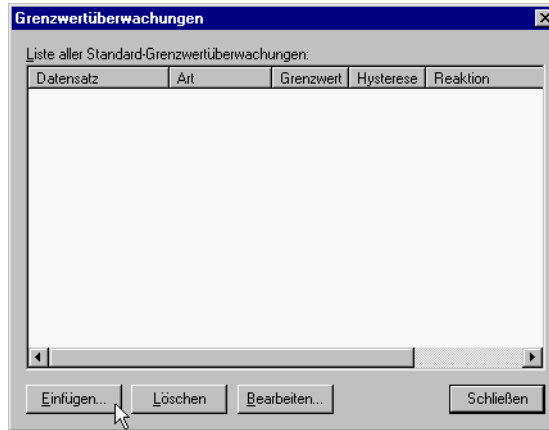


FIGURE 33: List of the standard limit monitors

Click on the **Insert** button. Normally a window would now open up, in which the limit parameters can be defined. Whether or not this window opens by itself depends on the **Open ... during Creation** field in the window of the **Extras/Options** menu item. If the window does not open automatically, then open it by double-clicking on the limit's row in the list shown above.

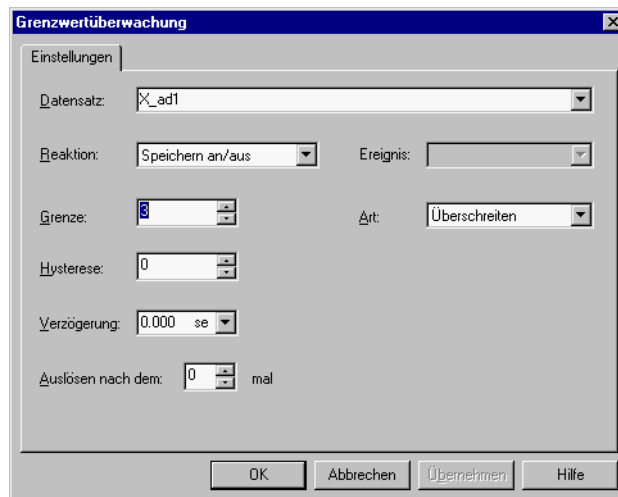


FIGURE 34: Edit window for the parameters of a standard limit monitor

Now enter the following parameters:

1. The dataset "X_ad1", or otherwise the dataset you want to monitor. If the list is empty, close the window with the **OK** button and re-open it. Now the list is filled.
2. Set the limit to the desired 3 volts.
3. Set the "Type" item to overshoot.
4. Set the reaction to "Storage on/off".

A list of the most important reactions follows:

Storage on/off	When exceeding the limit, "Storage on"; below the limit "Storage off".
Storage on	When exceeding the limit, "Storage on"; the store operation remains active afterwards.
Storage off	Once the limit is exceeded, an active store operation is stopped.
Trigger	The "\$StandardTriggerEvent" is switched on above the limit (below it is off). Used for storage <i>around</i> the overshoot occurrence, i.e. before <i>and</i> after it with the use of a data buffer (see the standard data file's pre and post trigger time).
Event on/off	For this, a logical variable has to be defined (such as the F5 key from the next example). This can be switched on and off with the limit.
Abort measurement	When the limit is exceeded, the measurement is terminated. (For this, the "TerminateJob" system variable must first be defined.)

Like all other objects, the standard limit monitors are also to be found in the project tree and can also be defined and edited there.

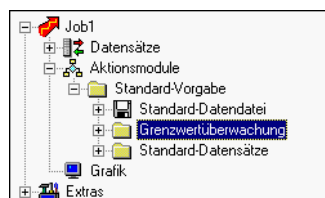


FIGURE 35: Standard monitors shown in the project tree.



Note: For each limit monitor, the program requires two variables. One to be monitored (usually the acquisition dataset) and one where the result of the monitoring is held (MLab calls this result variable "Event"). This variable is not required for switching the storage operation, as the existing standard variable "\$StandardStoreEvent" carries out this function. This is the reason why the event field goes grey when "Storage on/off" is selected. The event variable is fixed, namely as "\$StandardStoreEvent". The "Trigger" reaction uses such a standard dataset ("\$StandardTriggerEvent") as well. If you open the branch "Standard Datasets" in the previous picture, you will see the 4 standard datasets which MLab maintains permanently. You will learn more about using these in the chapter about the standard data file.

2nd. Example: Bar Display with Extremity Marker

A dataset ("X_ad0") is to be displayed as a bar. Extremity markers in the bar show the maximum and minimum values last reached. The extremity marker is to be reset using the F5 key.

To achieve this, the following steps are necessary:

1. Define a bar object.
2. In this, activate the display of extremity markers.
3. Define a button with which to reset the extremity marker.

Defining a Bar Object



Open the graphic page on which the bar should be displayed. The graphic tool bar opens at the same time. Select the button for the bar and draw the object out to the desired size.

1. A bar appears on the graphical display. Open its properties window (by double-clicking or via the context menu). Enter the name of the dataset in the **General** property page.

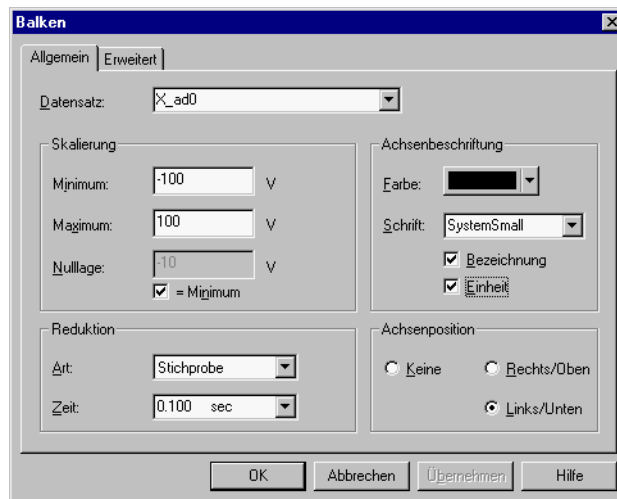


FIGURE 36: Property window of the bar.

2. Enter the correct scale values and, if desired, select the display of caption and unit. Confirm your entries with the **OK** button.

Activating the Display of Extremity Markers

Go to the second property page, **Extended**. Here you can see further settings for the bar: amongst these, the colour definition. Select the extremity markers, as shown in the picture.

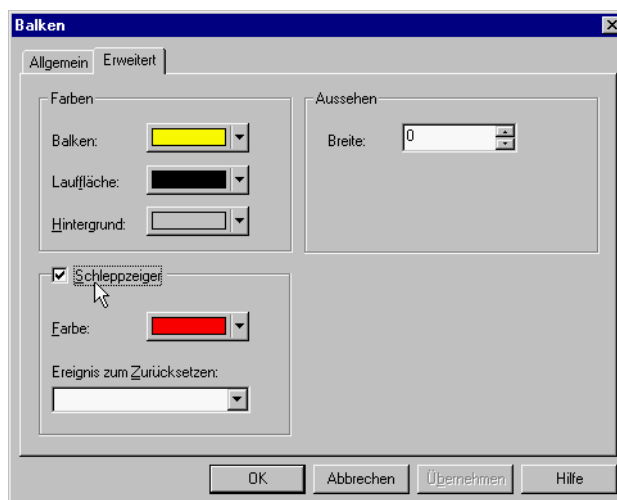


FIGURE 37: Further bar properties

Examine the "Reset Event" field. Such an event does not yet exist. For this, we first create a key object.

Defining a Key

A **key** function demands two steps in MLab:

1. A variable which stores the key status: key pressed or released.
2. A graphical key field, in which it is defined which key causes this to change.

Creating a "logical (event)" Type Variable



Select the **New** menu item from the context menu for the **Datasets** field in the project tree.

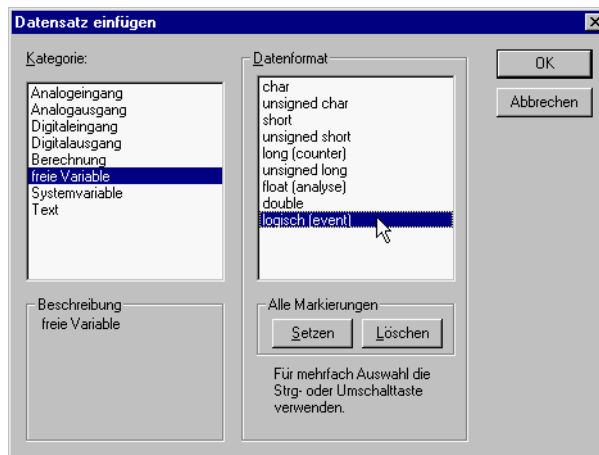
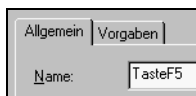


FIGURE 38: Inserting an event variable

The window shows the various categories of datasets and variables, which are at your disposal. The variable which we use for a key can represent two different states: key *pressed* (down) or key *released* (up). One could call it a binary variable. Further names with the same meaning are "logical" or "event". In this window, you must select the **Free variable** field under Category and **logical (event)** as the data format. Confirm with the **OK** button¹.

Call your variable something meaningful. As we intend to reset the extremity marker with the F5 key, we will call our variable "KeyF5". As with all dataset and variable names, the name may not contain any spaces or special characters. The other properties of the variable (caption, unit etc.) are not important for our application.



1. Usually, the Edit window for the variable now opens automatically (see **Extras/Options** menu item). If not, open the **Datasets** branch in the project tree. Your new variable can now be found in the list of the datasets. It has a default name "Variable1", "Variable2" etc. Double-click on the variable in order to open the Edit window.

Defining a "Key" Graphical Object



Switch to the graphic page and select the **Key** button from the tool bar. Draw open a field to any size you like. The size can be changed later at any time. The key field is now displayed on the graphic page. The object layout is initially set up for the F1 key. Open the properties window for the key (double-click on the field or use the field's context menu).

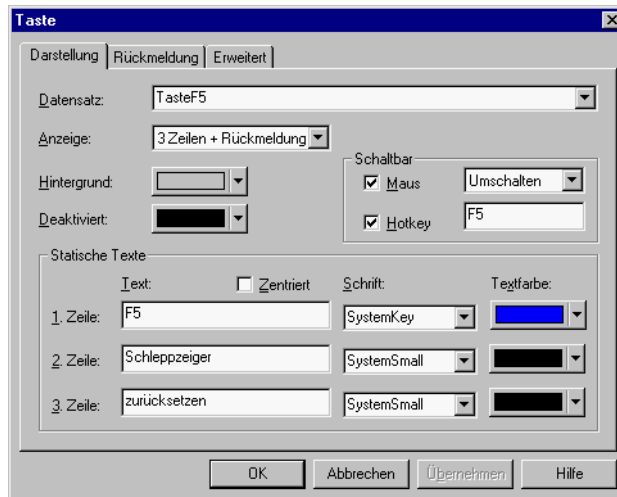


FIGURE 39: Properties window for the key

In the following, the most important parameters are explained:

1. Of most importance is the entry for the **Dataset** to be toggled. Our dataset is the variable "KeyF5". Open the list box and select this variable. If you find the list is too long, type a "k" and the display will jump to this initial letter.
2. The second most important parameter is the key's instigator. Which (physical) key should be used for toggling? This will be the **F5** function key. Go ahead and mark the "**Hotkey**" field. Then place the cursor in the adjacent field and press the F5 key! Do not type out the letters "F" and "5", press **F5** instead. In the same way, you can also enter SHIFT+CTRL+F5 by simply pressing this key combination.
3. The third most important parameter is the **Toggle** setting. This is already pre-set correctly. This means, that every time the key is pressed, the variable will be toggled (on, off, on, off, etc.) If you want to toggle the variable status with the mouse by clicking on the field during the measurement, then mark the **Mouse** parameter.
4. The remaining parameters relate to the graphical display and are therefore secondary for the key functionality. It is advisable, however, to keep the display informative. This means in our case: the name of the key "F5" and its later function "Reset Extremity Marker". In order to fit in the long words, you can use the small fonts.



Assigning the Function to the Key

This point can now be dealt with quickly: the F5 key should reset the bar's extremity markers. The variable "KeyF5" will be entered as the event in the **Extended** property page from the bar's properties window.

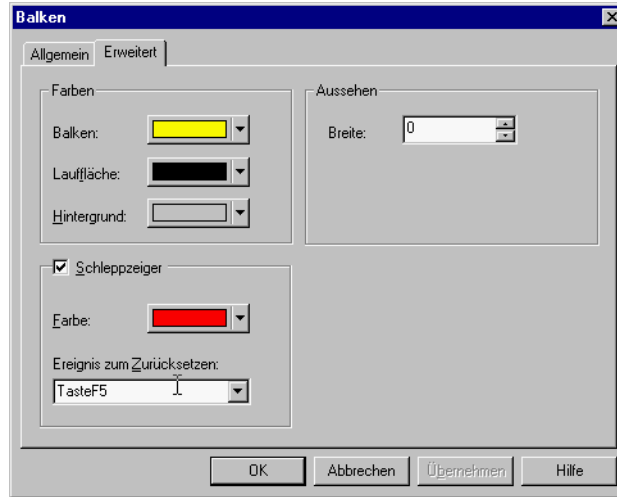


FIGURE 40: Reset extremity marker with F5 key

How these Settings Work



ExtremityMarker1.mlb

Now you can make use of your resettable extremity marker. The example "ExtremityMarker1.mlb" shows how this is done. Initially, the extremity marker starts running normally and indicates the maximum and minimum reached so far as a red line in the bar. Now press **F5**. The extremity marker is switched off and indicates the current value of the channel. Now press **F5** once more. The extremity marker starts working again.

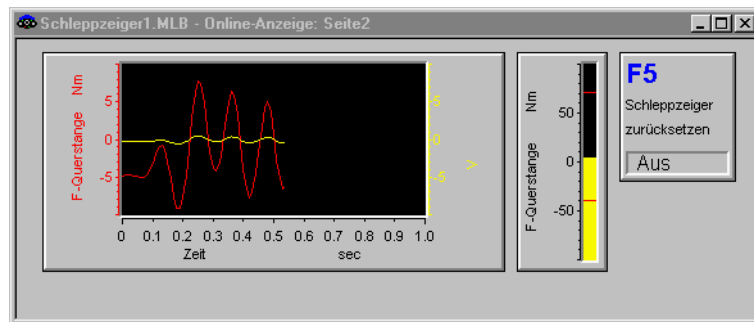


FIGURE 41: Online graphical display of the "ExtremityMarker1.mlb" example

This example shows the extremity marker for the bar object. Yet another action module generates an extremity marker. In this case, both extremities run as fully analog variables. These extremity markers can be reset as well. With these, it is also possible to reset and restart the extremity markers with a single key stroke ("pulse" setting).

General Information about Variables and Action Modules

The first two examples showed simple limit monitoring modules. In the last example, you already defined a variable for the key field. In the upcoming examples, you will regularly define variables. This is a consequence of the way that MLab fundamentally operates.

MLab projects consist of several small operational modules. Whether these are graphical objects, limit monitors or calculation modules, they are always small action units which perform something on datasets or variables. They generate graphical displays, change the indications or monitor the measurement.

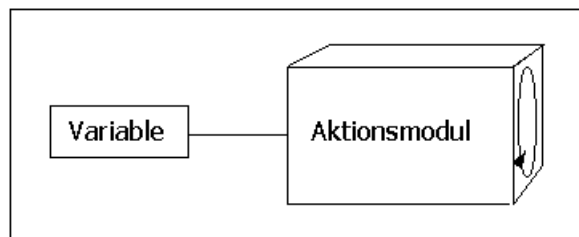


FIGURE 42: Each action module requires a variable

MLab refers to modules which execute a declared "action" as **Action Modules**. In principle, the other parts of the graphical display or the so-called standard area are no different. For this reason, this explanation applies to all of them and therefore to the above picture as well.

The rotation arrow indicates that each action module operates once per clock rate. In order to operate, an action module requires variables or datasets for it to process. The user has to implement these beforehand. In the previous example, it was necessary to implement the "KeyF5" variable for the "key" graphical module. In the previous chapter, no variables were necessary as only datasets were operated upon. These datasets had already been entered using the same selection window at the beginning. The so-called standard modules (standard data files and standard limit monitors, as well as most of the dynamic graphical objects) are created in a manner such that hardly any variables have to be implemented. These are prepared in the background. For other more complex actions, we need powerful, customised action modules, for which you have to define the variables yourself.

The declared action modules often need more than one variable. The following illustration shows the operation of a monitoring module with 3 variables:

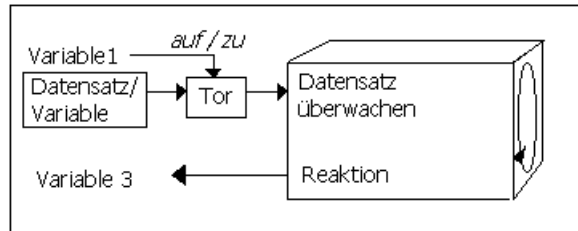


FIGURE 43: Schematic illustration of the "Limit Monitor" action module.

A dataset or variable is to be monitored. Variable 1 opens and closes the input gate and hence determines whether or not the action module should process current values. There is a gate like this for some action modules - it is optional and does not have to be used. At the beginning of a test procedure, such a gate is often useful in order to only start specific limit monitoring once the test object is "up to and running". Variable 3 obtains the information about the reaction. The reaction is carried out as soon as the limit is exceeded. How the reaction variable should then behave is defined in the module settings.

Variable 3 - and with it the whole module - now awaits its usage. Elsewhere, the variable will either be displayed or used as a switch for another action. In other words, it is an input variable for a different module. Please note:

1. **The variables link the separate modules of an MLab project together.**
2. **Hence, you define the action modules along with the variables that they use.**

To set up an action module you should have a flowchart that determines which variables and datasets will be processed and influenced. The next two examples show how action modules can be created.

3rd Example: Display of Maximum Value in an x(t) Diagram

What follows is a further example on the subject of the **extremity marker**, with the use of the extremity marker action module. This module can monitor a list of datasets or variables for minimum, maximum and mean value. The following task has to be fulfilled:

The maximum and minimum values of the dataset "X_k1" occurring up to now should be displayed in an x(t) diagram and also numerically.

For this, the following steps are necessary:

1. A variable for the minimum and the maximum value has to be defined.
2. The extremity marker module has to be parameterised correctly.
3. The graphical display has to be defined.

A remark based on practical experience: The example specifies the extremity marker for "X_k1", meaning the first channel of the acquisition card. The user would usually number a series of 16 channels from 1 to 16. In the digital world, though, we number from 0 to 15. Consequently, MLab initially assigns the channel numbers with numbers from 0 to 15. You do not, however, have to agree to this. One simply changes the channel numbers in the setup file. MLab will then keep to this. For the following example (and all subsequent ones), the **Setup file "TestChannel.mhc"** was created, which gives every channel an appropriate name such as "Channel 1" etc. The complete solution for this example is shown in the file "**ExtremityMarker2.mlb**".

Definition of the two Analog Variables

The definition of analog variables is performed in a similar manner to the definition of logical variables already described. They only differ as a result of different data types, which have to be selected accordingly. Select the **New** menu item from the context menu of the **Datasets** branch. You will be presented once more with the following, familiar choice:

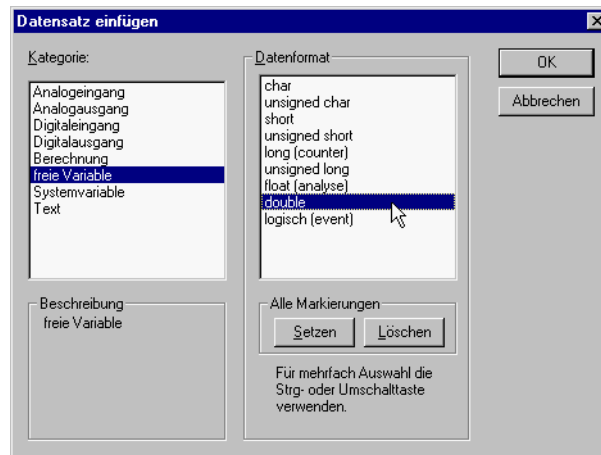


FIGURE 44: Adding a free variable of type "double"

For an analog value such as the extremity marker, select the **double** type (8 bytes) or the **float (analysis)** type (4 bytes). For the current example, it does not matter which. An accuracy of 4 bytes is sufficient and saves space in the storage file. Confirm with the **OK** button and then give the variable a meaningful dataset name ("X_k1_max") and caption ("Max.(Kanal 1)").

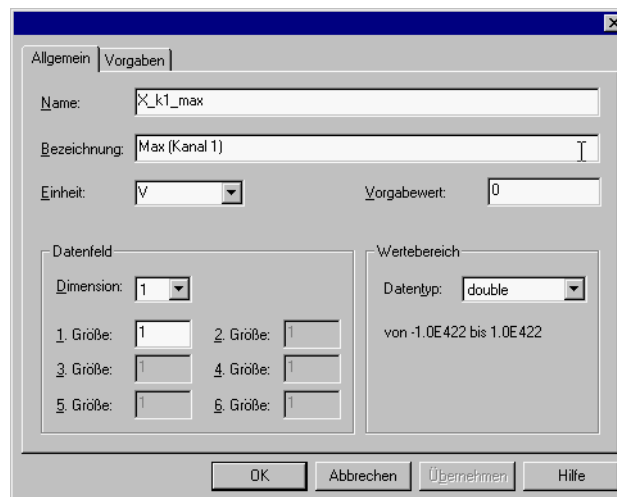


FIGURE 45: Parameters for the "maximum of channel 1" extremity marker

The second variable for the minimum is defined similarly.

Defining the "Extremity Marker" Action Module

Now we are going to create the first customised action module. Go to the **Action Module** branch in the project tree and select the **New** menu item from its context menu. The following selection will now appear:

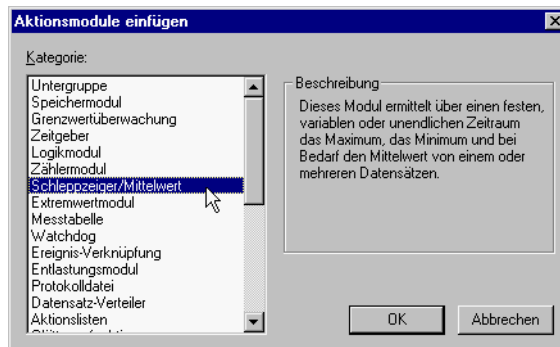


FIGURE 46: Selection of the "Extremity Marker/Mean Value" action module

Select the action module "**Extremity Marker/Mean Value**" and press the **OK** button. The module will be inserted. Call up the module's properties window (may open up automatically). You will see the following dialog:

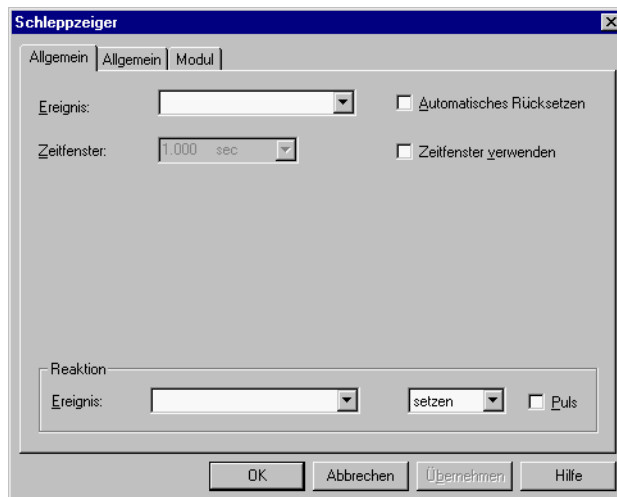


FIGURE 47: First property page of the extremity marker module

You can see the input gate ("Event") and the output gate ("Reaction"). Here you can leave all entries empty.

Switch to the second property page, in which a datalist can be seen. The module can modify several extremity markers at the same time. In this example, we want to restrict ourselves to the definition of a single extremity marker. Click on the **Insert** button and enter the values as shown in the following figure:

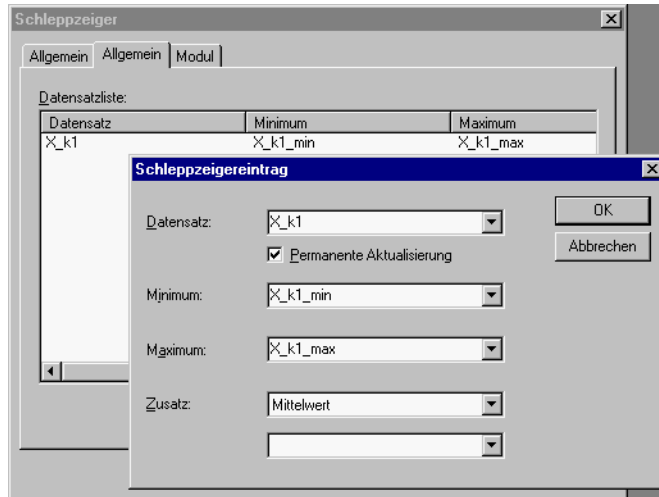


FIGURE 48: Correct extremity marker entry for min. and max. of "X_k1"

If you also want to know what the current mean value is, you can enter another variable for this in the bottom field.

Defining the Graphical Display



ExtremityMarker2.mlb

All that remains now is the graphical display. For this you define an x(t) diagram and three numerical fields: one for each variable. The next picture shows what this could look like (example: "ExtremityMarker2.mlb")

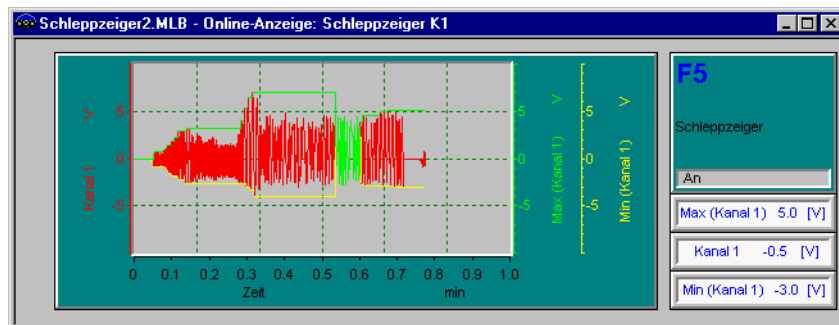


FIGURE 49: Online graphical display of the example "ExtremityMarker2.mlb"

Inhibiting and Activating the Extremity Marker

The picture shows the F5 key next to the x(t) diagram with the three channels "X_k1", "X_k1_min" and "X_k1_max" and the three numerical indications. Its usage is equivalent to the first example in this chapter. One implements a variable and a key to toggle it with. In order to use it to switch the extremity marker activity on and off, this will be entered as the event in the extremity marker's first property page.

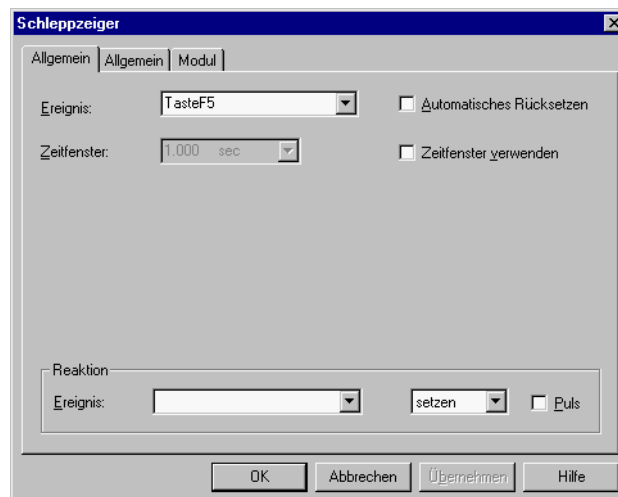


FIGURE 50: Extremity marker with "KeyF5" as reset event.

An entry in the **Event** field has the following effect on the extremity marker module:

- empty** Without an entry, the extremity marker module is always active.
- with Variable (=1)** The extremity marker values will be set to the current value of the originating reference dataset.
- with Variable (=0)** The extremity marker function is activated.

So, the input event acts as an **inhibit!** If KeyF5 is "on" then the extremity marker is "off", if it is "off" the extremity marker is operating. If the extremity marker is off, the min. and max. variables run with the dataset being monitored. They then always take its current value. In this way, the extremity markers will immediately start indicating the minimum and maximum from the moment of the next activation onwards.

4th Example: Range Display with a Switch

The following setup is often used:

A signal (for example "X_k1") has a measurement range of +/- 100 Nm. Everything is OK as long as it is within a scope of +/- 50 Nm. As soon as it goes beyond the limits, a switch (in red) should show a warning. Peaks can often occur when the installation is switched on and, consequently, monitoring should not start at the same time as the measurement. Instead, it should only start after the F3 key has been pressed. In order to ensure that the alarm is not set off prematurely by a stray value, the alarm should only result after more than 3 out-of-limit values. In addition, it is desired that the same signal should demand a shutdown (or rundown) from a connected SPS-System.

For such a problem, the **Limit Monitor** action module is ideal. The following steps for input are necessary:

1. Set up datasets and variables: The "X_k1" channel is calibrated to +/- 100 Nm and displayed, for example, as a bar or x(t) diagram. A switch is defined in the usual manner with the "KeyF3" variable. A digital output is provided.
2. The **Limit Monitor** action module is set up and is inhibited or enabled with the "KeyF3".
3. A **Switch** indicates the monitoring result ("OK" / "Shut Off").
4. In order to fulfil the additional requirement, we redirect the result to a **digital output**.

1st Step: Defining Datasets and Variables

This example is a further development of the previous example; therefore, one can load this project as a basis. The extremity markers used there can be reused. The following steps must now be taken:

- "X_k1" is calibrated to +/- 100 Nm and shown in an x(t) diagram.
- The variables "X_k1_min" and "X_k1_max" are given the unit "Nm" and used further in an extremity marker module. The variables are displayed in two numerical fields.
- The F5 key becomes a button for the F3 key and is given the name "Enable" for the display.

A new item is the digital output, which should relay the monitoring result to the outside world.

Inserting a Digital Output Dataset

A requirement for a digital output is, of course, a suitable digital output card. Another requirement is that this card has been entered into your setup file. Assuming these requirements have been fulfilled, these are the steps that follow:

1. You enter the digital device as an **Additional Device** in addition to the clock source.
2. You insert the digital dataset into your dataset list and select the bit on which the output should lie.

Entering the Digital Device as an Additional Device

The acquisition card with the analog channels remains your measurement's clock source. The program has to be informed that the digital output card should be driven as well. MLab knows from the setup file, that such a card exists and what the channels are called, but not that it should be driven. So, open the **Administration/Sampling Rate** menu item or double-click on the **Job** (Normally, the default name that MLab gives it to a new project is "Job1". You can rename it via the context menu.) In the **Sampling Rate** window, select the **Additional Devices** property page.

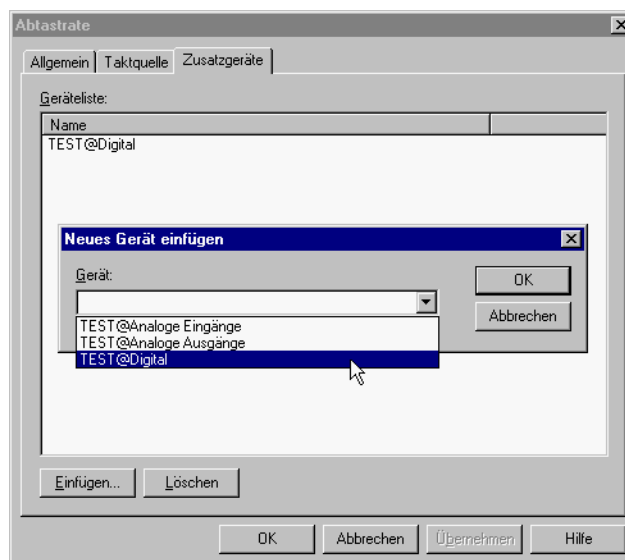


FIGURE 51: Inserting additional devices

As with the other channel groups, the digital channels are listed under **Device**. Use the **Insert** button and select your digital channels from the device list.

Inserting the Digital Channel as a Dataset

Select the **Administration/Datasets** menu item and use the **Insert** button. As you might have noticed by now, you will get the same dialog via the **New...** context menu item for the **Datasets** field. Now select the **Digital Outputs** category and to the right the desired channel (port).

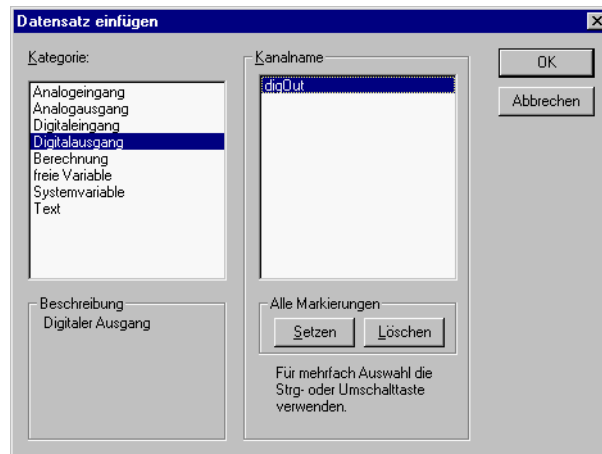


FIGURE 52: Inserting digital outputs

If, at this stage, no digital outputs are displayed on the right side of your window, this is because one of the previous steps was not carried out properly or in full. Check the following:

- The digital card is registered in your setup file, i.e. in the file named under **Extras/Hardware** in the project tree. The various digital ports (port = 8 commonly-addressed I/O bits) will be listed below it.
- The digital port you want to use has been set to **Output** in its properties window (for digital cards a port can be defined as output or input).
- The digital card has really been entered as an **Additional Device** - as described above - in addition to your clock source. The presets for the clock source *do not* hold the entered additional devices or the chosen sampling rate. These have to be set up anew for each new MLab project.

After you have inserted the digital channel as dataset, choose a meaningful name for it and select the bit number. For the planned example, the following setup is chosen:

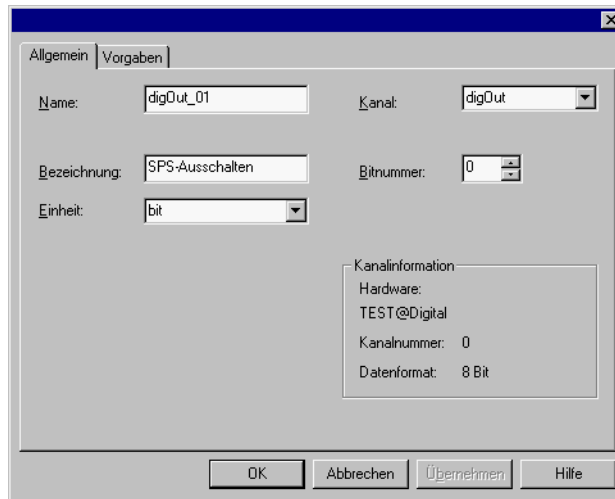


FIGURE 53: Setup for digOut, used as output for SPS

When a digital output is selected, the entire 8-bit port is initially inserted. With **Bit number**, you actually select the exact digital output you want to use. If you also want to use a different bit, you must insert the digital output once more, but give it a different dataset name and a different bit number.

Before defining the actual limit monitor, you should have the following datasets and variables at hand:

- Dataset "X_k1" calibrated to +/- 100 Nm.
- Dataset "digOut_01" set to bit number 0.
- Variable "KeyF3" with associated graphical object "Key".
- Optional: the extremity marker variable "X_k1_min" and "X_k1_max", so you can check the reactions in MLab.

2nd. Step: Setting up the Limit Monitor Module

Insert a new action module of the **Limit Monitor** type and give it a meaningful name using the **Rename** context menu item. The display in the project tree could appear as follows:

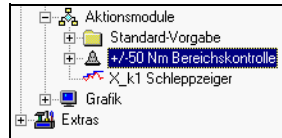


FIGURE 54: Display of the new module in the project tree. The extremity marker below it was adopted from the previous project.

Now open the properties window of the module and enter the following parameters:

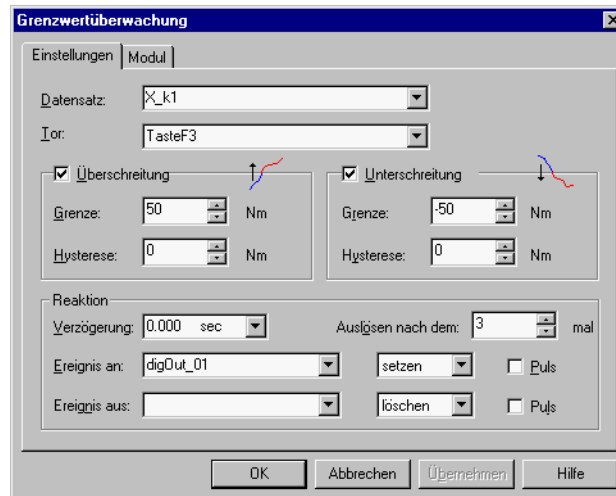


FIGURE 55: Limit monitor module settings

The "X_k1" dataset will be monitored. The data input gate for the module can be opened and closed with the "KeyF3" button. The KeyF3 releases the module when *set*, when *not set* the monitoring is off. The value of "X_k1" is monitored at the upper limit of + 50 Nm as well as for undershooting the lower limit of -50 Nm. The reaction signal, "digOut_01", will only be set if the limit is violated a third time.

The reaction field offers a range of further options. The reaction can be triggered after a certain period of time only. Sometimes just a single pulse is desired as reaction. In this case, the signal is only set for one clock period of the sampling rate and then cleared again afterwards.

The Information Variables of a Limit Monitor Module

In addition to these features, the limit monitor module places four further information variables at your disposal which are used for display purposes. You can find these variables in the project tree under the names of the modules. They have the following names and functions:

StatusUpperLimit	Is on above the upper limit, off below it.
Overshoots	Counts the number of overshoots of the upper limit, independent of the reaction properties.
StatusLowerLimit	Is on below the lower limit, off above it.
Undershoots	Counts the number of undershoots of the lower limit.

These variables always appear in the dataset selection lists prepended with the name of the module to which they belong. An "@" character ("at") sits between the module name and the variable name. Consequently, the first variable for our example is called: "+/- 50 Nm Range@StatusUpperLimit".

Note: It is intentional, that these variables bear this complicated name. They only exist for information purposes, and cannot be saved or used in the customised action lists. If the content of such a variable is to be saved, it must be copied into another variable with a "normal" name via an action module.



A hint: If the limit monitoring module should run continuously, without being controlled with the enable button, then you can leave the **Gate** entry empty. However, if you want the module to operate right at the beginning and be disabled later, then you can pre-set the KeyF3 variable with the value 1. It is then on at the start of the measurement and can be switched off during the measurement. You can find the entry for the **pre-set value** in the variable's properties window.

3rd. Step: The Graphical Display Definition, a Switch

An x(t) diagram can be used to display the signal form for "X_k1". Additionally, we mentioned the key field for the F3 key and both numerical displays for the maximum and minimum of "X_k1". What's still missing is a switch object to display the warning. Open the graphic page, select the switch symbol from the toolbar and drag open a field. You will obtain its properties window by double-clicking on the field.



FIGURE 56: The Switch object's settings

The state of the digital dataset "digOut_01" is the value which we need to toggle the switch indication. As long as the value is "off" it should display an OK on a green background; as soon as it is switched "on", the text "Shut off!" on a red background should appear.

If the signal is wired up as required, then it will lead to a shut down of the SPS control. The display for the Tutorial's example file "Range.mlb" shows the following online display:



Range.mlb

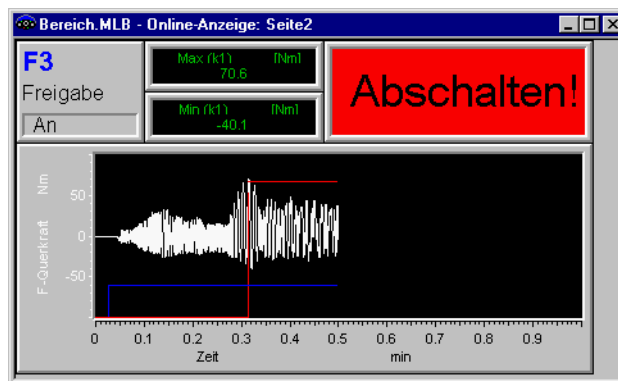


FIGURE 57: Online graphical display for the example "Range.mlb"

If you could see this picture in colour, you would see the sequence of activities of the F3 key and the signal "digOut_01" in the x(t) diagram. The F3 key enabled the monitor module after about 6 s; the shut off signal was triggered after about 18 s.



Digital datasets and variables can be displayed in the x(t) diagram and stored for analysis just like analog values. However, it should be pointed out that in MLab, the value for the "on" state depends on the data type. Hence, the following rule always applies: The "off" state always corresponds to the value 0. The "on" state corresponds to a non-zero value. In the current example, this means the value 255 for "KeyF3" and for "digOut_01", the value 1. This can be verified with a quick test using a numerical display of the variables.

Possible Causes of Errors

MLab recognises setup faults for the extremity marker or the monitor module as part of the parameter checks before starting the measurement and issues a warning. For example, a dataset name was perhaps written wrongly:

- "Dataset KeyF51" was not found"

or there was none entered at all:

- "Module contains no dataset"

The last message appears frequently, if unknown action modules were simply inserted and then not used. In such a case, check the list of the action modules in your project. Give the modules used names - then it is easier for you to recognise them again and find unknown (empty) modules faster.

The End of the Chapter

At the core of this chapter is the information about the co-operation between variables and action modules. Variables have to be inserted so that they can be read from or written to in order for action modules to operate. The user has to create such variables. An exception to this are the modules from the "Standard Definitions" branch. For these, the "Standard Datasets" displayed here as well are used, in order to be able to use the standard files and monitors without having to define further variables. More about this in the next chapter.

Notes

This chapter could not show you all the possibilities for the limit monitoring. Another important module is the **Extreme Value Module**. Here the local minima and maxima of a dataset are monitored and can trigger events. Further possibilities are offered by the action modules of the **Action List** type. Here datasets can be monitored in comparison with variable limits. More about this subject in the "Calculations" chapter.



Storing Data

Files and Sampling Rates in MLab

Introduction

Data storage enables us to analyse the measured data later off-line. In order to do this, you must first decide which datasets or variables you want to store; secondly, when and for how long and thirdly, with which frequency. All datasets and variables in a single file are stored with the identical frequency. However, several files with differing frequencies can be stored in parallel.

MLab offers two versions of the storage module (Standard Data File and Storage Module) along with several data formats (e.g. RMS, MDF,...).

Standard Data File versus "Storage Module" Action Module

The standard data file is a minimised version of the "normal" storage module. It is easier to set up and - as with the standard monitoring module - is automatically available in each new project as a **standard default**.

The Standard Data File

The standard data file stores all datasets and variables for which the **storage status in the dataset list** is active. Storage start and stop are controlled during the measurement using the toolbar buttons. Fixed standard variables are available for event-driven storage, triggering or file name switching. These can immediately be routed to other modules such as a key field or a monitor module without any further definitions. The storage frequency cannot, however, be chosen freely; it corresponds to the acquisition frequency.

The "Storage Module" Action Module

The regular storage module has its own property page, in which the datasets and variables to be stored are entered. These settings are independent from those in the dataset list. Here, all variables can be stored - including the internal informative variables for the various modules used (i.e., the variables with the "@" character (the "at" symbol)). In addition to the possibilities provided by the standard data file, the storage module permits the **storage of section information**. When storing the section information, tags are placed in the header file to enable you to display an isolated measurement phase during off-line analysis without having to perform a tiresome search. A **ring file** is used, for example, to investigate an unexpected measurement termination ("post mortem"). In this case, data is written out continuously. The length of the file is limited to a certain duration. As soon as the end of the file is reached, the file is overwritten again from the beginning. This way, the last seconds or minutes of the test are always available regardless of when the measurement was terminated.

In which File Format will Data be Stored?

In practically every format, information about the data stored (referred to from now on as dataset descriptions) is written to the storage medium in addition to the actual data. Examples are: calibration, label, unit etc. Depending on the file format, this additional information will be stored in the same file as the data or in a separate header file.

Data and dataset description in a common file

Examples of this are: NCode[®] (DAC), Somat, DASyLab[®] (DDF), TurboLab[®] (DAF), Text (ASC, TXT)¹.

With these data formats, the dataset description is normally located at the beginning of the file. This area is called the data header.

Data and dataset description in separate files

Examples are: Remus (RMS), MicroEdition (MDF), Diadem[®] (DAT).

The dataset descriptions are written into a separate file (header file). Normally, the header file and the data file are named identically apart from the file ending. With almost all formats, however, the header file contains a reference to the associated data file in, so this does not always apply.

Data Multiplexing

If not set up differently, the measurement frequency (operating cycle) is used as the storage frequency. In each cycle, an up-to-date value is available for each dataset or variable which has to be stored - MLab ensures this. These values are written into a data file in a fixed sequence. At the next cycle, each dataset and each variable again has an up-to-date value. Again, these are written into the file, and so on. The analysis program reads the values back out of the file using a fixed initial offset and a fixed increment. If, for example, the datasets "k1" and "k2" and the variables "v1" and "v2" are stored, then the beginning of the data file would look like this:

1. DASyLab[®] is a registered trademark of DATALOG GmbH, Diadem[®] is a registered trademark of GFS GmbH. Mention of third-party products is for informational purposes only and does not represent a breach of copyright.

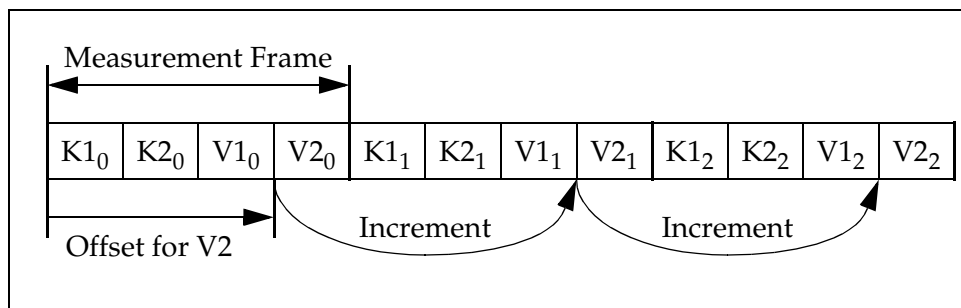


FIGURE 58: Multiplexed data in the file. Offset and increment for V2

Here the channel "k1" has an offset of 0, "k2" an offset of 1, "v1" of 2 and "v2" of 3. They all have the same incremental step, 4, in order to locate their next value. This is a simple and rapid approach to the read and write processes for recording as well for analysis.

In its basic version, MLab supports the RMS and MDF data formats, which record in accordance with the principle described. They differ in their individual features.

Storage in RMS Format

Storing in RMS format creates two files:

Filename.dat	the data file
Filename.rms	the header file

The data can be read and processed by analysis programs such as **Remus-Graph**, **RemusView**, **MGraph** and others. Its limitations - when compared to MDF - concern, firstly, the possible data formats (only 16 and 32 bit integer and 32 bit real data are allowed) and, secondly, the limited possibilities for storing additional information.

Storage in MDF Format

MDF format also creates two files:

Filename.dat	the data file
Filename.mdf	the header file

The analysis program **MGraph** can read and process the data. All the features of the MLab data storage can be used, variables of all types can be stored as well as section tags.

The Examples in this Chapter

The examples in this chapter should demonstrate the various possibilities for data storage.

1. A measurement with the standard data file. A sequence of tests should be conducted one after the other. For each test a new file should be created. All data should be written into a test directory. When beginning the measurement, the storage should start automatically.
2. A measurement with the storage module: Data should only be stored above a certain level. The number of overshoots should be stored as well, so that parts of the storage can be recognised. The storage should be performed at a sampling rate of 1000 Hz per channel. Simultaneously, a ring file should run in order to record all relevant values.
3. The storage of sections: The various steps of a test run should be marked.

Example 1: Storing a Test Sequence

A sequence of tests is to be conducted. Each test will be written into its own file. The following list applies:

- The data directory "C:\Testruns"
- The file name: "Test_001", "Test_002", "Test_003" etc.
- Measured channels: all 16: "X_k1" to "X_k16"
- Stored channels: "X_k1", X_k5", X_k6"
- Test step-on with "n" key

Creating a Standard Data File

Entries in the Dataset List

First of all, the 16 input channels are inserted as datasets in the usual manner. Then, the 3 named datasets are marked for storage.

The dataset list then appears as follows:

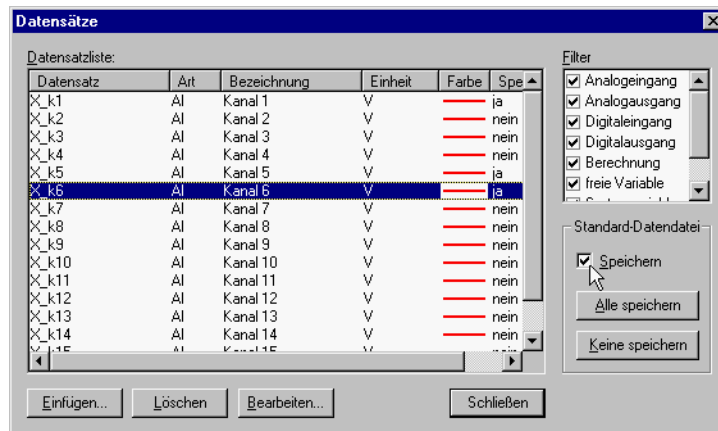


FIGURE 59: Display of dataset list. For 3 datasets, the "Store" checkbox is active

All 16 channels will be acquired as datasets, but only 3 of these are to be stored.

Here, once more, an explanation of the use of language:

Definition of terminology: channel, signal, dataset, variable

By *channel*, we mean the input to the acquisition card, at which the *signal*¹ to be measured is received. Being sampled at the sampling rate, this results in one value per cycle per channel. A series of such values is known as a *dataset*. Only the channels in the above list are translated into a dataset. Other channels, which might be present in hardware terms, are ignored. In addition to a series of data, a dataset possesses some general values: the label, the unit and the calibration. This transition from a channel to a dataset is called *data acquisition*. The *storage* is a further, separate step. All recorded datasets are available for display, monitoring etc. However, of these, only a selection is stored and a further selection of *variables*, i.e. other values generated during the measurement. The storage criteria which still have to be defined dictate when and with which frequency the storage takes place.

1. Within the signal, one differentiates again between the real signal, i.e. the actual information we want, and the interference in the line, the noise.

General Definitions for the Standard Data File

Next, the standard data file is set up. In order to do this, use the **Administration/ Data File** menu item. Alternatively, in the project tree, use the **Standard Data File** field in the **Standard Presettings** on the **Action Module** branch. You will be presented with the following dialog window:

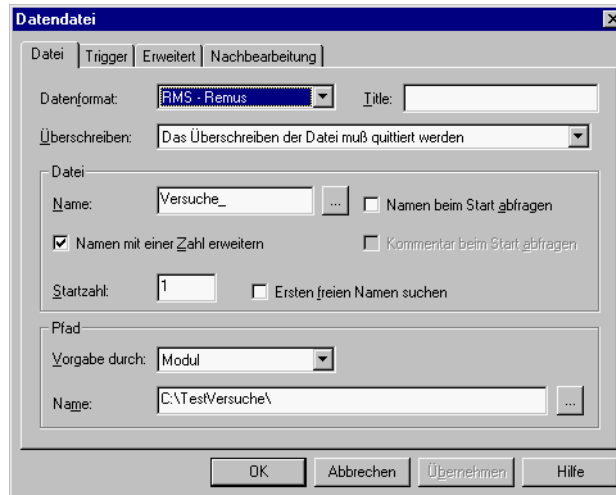


FIGURE 60: Setup for the standard data file

The following points must be considered:

- | | |
|-----------------------------------|---|
| Filename | Because the name will be extended with a number, the fixed part of the name is entered here: "Test_". The first file pair to be created will be called "Test_001.dat" and "Test_001.rms". |
| Search for first free name | Here, we will leave this option out, so that the measurement starts every time with the initial number (1) and can only be incremented during the measurement. If this first file already exists, a user query will appear - as defined above - and ask if you want to overwrite it. This option should be selected, however, if you want to resume a test sequence which has already been started. Then, if there have already been 23 tests, MLab resumes with the number 24. |
| Path | The data path can be chosen at will (module) or MLab's (program) default directory can be used. The default for the program path is set using the Extras/Options menu item.
The data path must exist! It can be created as a "new folder" in the file selection window or in the Windows Explorer. |

Further Definitions for the Standard Data File

Furthermore, the filename should be changed using the "n" key. To do this, we will examine the setup parameters of the standard data file on the second property page, **Extended**.

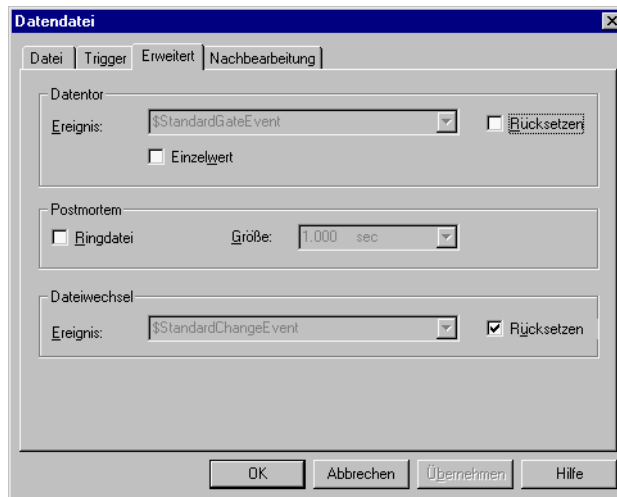


FIGURE 61: Extended settings for changing file

These parameters exist for every storage module, for the standard data file as well as for the superior storage module. The switching variables are already set up for the standard data file; they have fixed names and just need to be applied.

The 4 standard variables and their functions:

Standard variables

\$\$StandardStoreEvent	Storage on/off
\$\$StandardTriggerEvent	Determines triggered storage. If this variable goes "on", the last x seconds before this moment (pre-trigger) and the next y seconds after this moment (post-trigger) are stored.
\$\$StandardGateEvent	Opens and closes the flow of data into the data file. Specifically for single value storage.
\$\$StandardChangeEvent	Invokes the change of file when the variable changes from "off" to "on".

For this example you use the "\$StandardChangeEvent". In the next step, this variable will be switched by the key which still has to be created. Mark the "Reset" selection as shown in the picture; the variable will then be set back to 0 directly after the file change and will be ready for the next file change.

Changing File with the "N" Key

You construct a key in the usual way. However, unlike before, you don't have to create a variable (as earlier with KeyF3) to hold the switch state. As we are using the standard data file, this variable already exists: "\$Standard-ChangeEvent". The settings appear as follows:

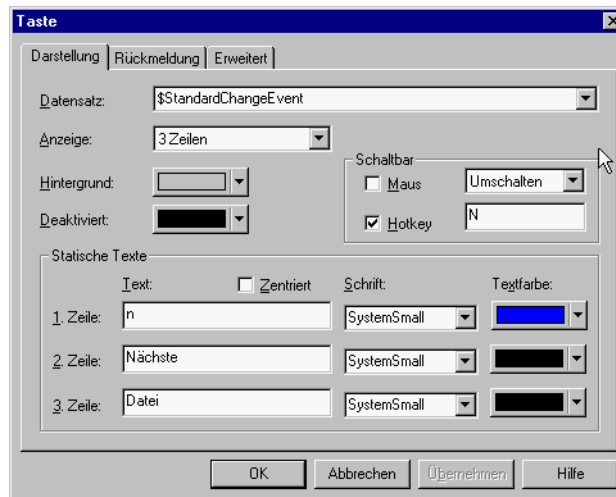


FIGURE 62: File change switched with the "N" key (=n or N)

A Text Box for Displaying the Current Filename



TestTrials.mlb

As the filenames should change from one test to the next, it make sense to have the current filename displayed. In order to do so, you create a **static Text Box** (see left picture), where you enter the text "filename" or "current filename".

Then, you create a **dynamic Text Box** (see the figure on the right), in which you display the information variable "Standard-DataFile@Filename".

The Information Variables for Storage Modules

The storage modules, i.e. the standard data file and the **Storage Module** action module, have several information variables at their disposal (12 in total). For most online displays, the following variables can be recommended:

- StorageStatus** Status variable for continuous storage. The store operation is driven by the storage event.
 off: Data will not be stored.
 on: Data will be stored.
Beware! The variable only displays store operations which have been initiated by the storage event. If storage was triggered by the trigger event, it will be indicated by the **TriggerStatus**.
- TriggerStatus** Shows whether the trigger event has been set. Is on as long as the post trigger time is running.
- FrameCounter** Number of measurement frames stored up to now. This corresponds to the number of values per channel.
- FileNumber** Last number resulting from file changes (ChangeEvent). In our case, this is also the test number.
- Filename** The complete name of the data file (with path), in which the data is currently being stored.

Note: This example can, of course, also be accomplished using the normal storage module. It is only more tiresome. The storage module has to be created especially, the datasets which are to be stored must be entered into the storage module and the variables for the file change must be created manually.

The example can be tried out using the "**Testtrials.mlb**" file.



Testtrials.mlb

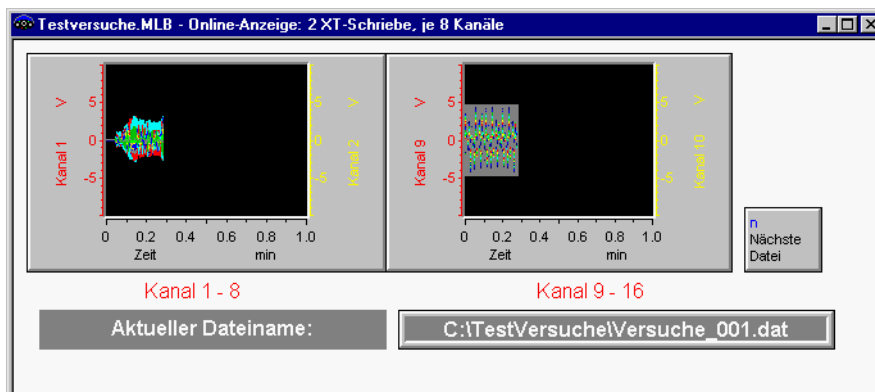


FIGURE 63: Online display of the example "Testtrials.mlb".

Setting up the Storage Module

You generate a **storage module** using the context menu for the **Action Modules** field. The first page corresponds to the dialog as you know it from the standard data file. This time select the MDF format as an information variable must also be stored.

Now we will select the datasets which are to be stored. This selection is *completely independent* of the "Store" settings in the dataset list. Those settings only relate to the standard data file and not to the storage module. For our example, no datasets at all need to be marked in the dataset list. Here in the storage module, however, the datasets must be entered.

Switch to the second property page **Datasets**. Here, select the 8 channels and the variable "X_k2>3Volt@Overshoots" which will count the overshoots. "X_2k>3Volt@StatusUpperLimit" will be used as the switching event for starting and stopping (and interrupting and resuming) the store operation.

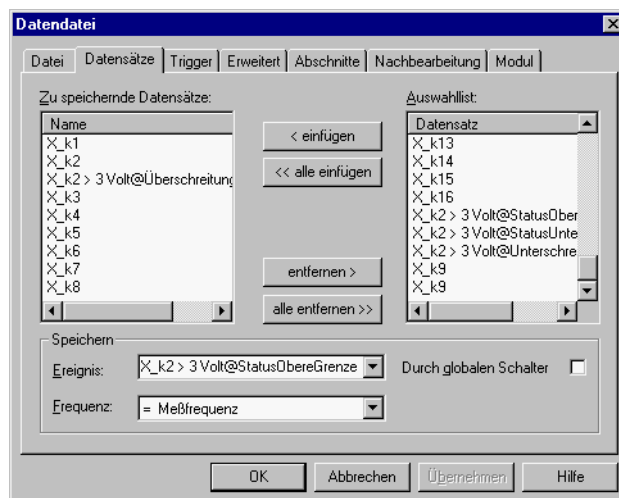


FIGURE 65: Settings for datasets and for the storage event

The "use global switch" field enables storage to be controlled additionally (OR relation) using the button on the toolbar together with the standard data file. As we do not require this in our example, we will deactivate this option.

Three options are provided when selecting the origin of the storage frequency:

- = Measurement frequency** Storage frequency corresponds to the measurement frequency (sampling rate).
- <= desired frequency** If this setting is chosen, an input box appears in which the desired frequency must be entered. MLab calculates the nearest divisor for the storage frequency which ensures that this frequency is attained. The sampling rate (measurement frequency) is not affected by this.
- = Measurement frequency/ Divisor** The storage frequency corresponds to the measurement frequency divided by a divisor that must be entered in the adjacent field.

The "= Measurement frequency" setting is correct for the example. The storage frequency is the set sampling rate.

Setting the Sampling Rate

The default setting for the sampling rate is 100 Hz. This setting works for nearly all acquisition hardware. Which higher sampling rates are attainable depend upon the acquisition hardware being used and the number of channels being measured.

Select the **Administration/Sampling Rate** menu item. The parameters which are offered on the **General** property page depend upon the hardware installed and consequently vary depending upon device and mode of operation. Some examples follow:

- Polling operation** The program polls the card's input channels at the sampling rate. The sampling rate ranges from 1 Hz to 1000 Hz per channel. This mode of operation can be used, for example, with the following cards:
DT2814, DT2812, RTI800, RTI834/5, ME300, DAQ700, CIO.
- DMA operation** The card samples the channels autonomously and writes the data directly into the computer's RAM (**D**irect **M**emory **A**ccess). Here, the sampling rate is heavily dependent upon the hardware and ranges from 0.01 Hz to over 300 kHz total sampling rate. Representatives of this group are:
DT2801, DT282x-Serie, DT283x-Serie, RTI834/5, CAESAR-PCM-Interface IFAT.
- IRQ operation** In this mode of operation, the card also samples the channels itself and stores the results in a buffer memory on the card. Via an interrupt request, it informs the program that data can be collected. Sampling rates from 0.01 Hz to over 500 kHz are possible. The value depends upon the size of the buffer memory on the plug-in card.
- Memory operation** The card's direct access to memory via the I/O Bus (and not via a DMA chip). The sampling rate is more than 500 kHz, depending upon the card, e.g.:
DT3000-Serie, CAESAR-PCM-Interface IF16.

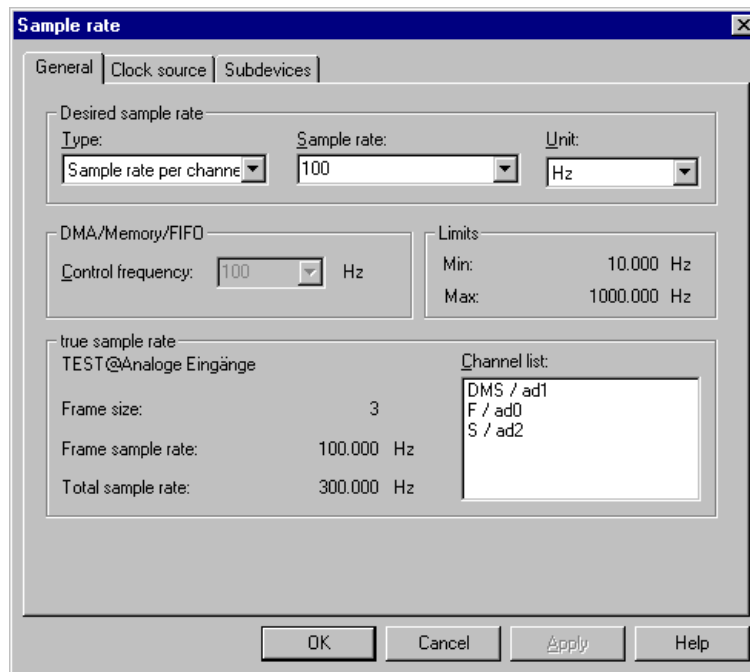


FIGURE 66: Dialog for the general settings for the sampling rate

Sampling Rate per Channel and Total Sampling Rate

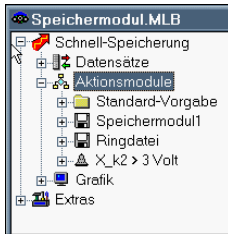
Note: For the polling mode of operation, the maximum **sampling rate per channel** is given here. This is because the number of channels which are going to be sampled has little influence on the polling frequency. This maximum frequency is not very high anyway. By contrast, for higher sampling rates, the total sampling rate is usually given. The **total sampling rate** is the data transmission rate¹. If only one channel is to be measured, the total sampling rate is exactly the same as the sampling rate per channel. With two channels, the sampling rate per channel is only half as big. With n channels, the channel rate is equal to the total sampling rate divided by n (e.g. total sampling rate 128 kHz and 32 channels means sampling rate per channel is 4 kHz).

At higher sampling rates, it should in each case be clarified, with which acquisition card and processor type the measurement can be performed. The interest in higher sampling rates is targeted upon the high frequencies occurring in the signal. If these are to be made available for analysis then the sampling rate must be at least double this (with regard to the highest occurring frequency), in accordance with *Shannon's Sampling Theory*. In practice however, a factor of 5 - 10 for acquiring the signal frequency is recommended.

1. The data transmission rate in words, i.e. at 128 kHz, 128000 data words are transmitted per second. With a dataword comprising 2 Bytes, this is a transmission rate of 256 kBytes/s.

Defining the Ring File

To define a ring file, a further **Storage Module** will be created. The ring file is a dedicated file, completely independent of any other storage. Choose a new name for it, e.g. "Ring" and then select all 16 channels for storage. As can be seen in the picture to the left, there are now two storage modules. In order to maintain an overview of the project tree, one gives the modules meaningful names.



The way in which the ring file will perform is set up in the **Extended** property page.



StorageModule.mlb

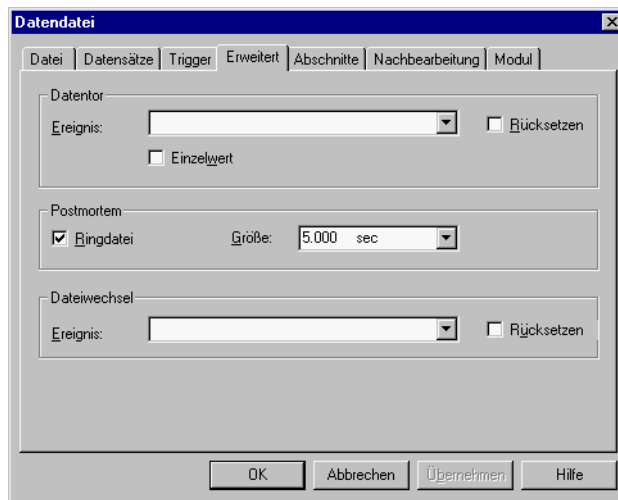


FIGURE 67: Extended settings: ring file for so-called "post mortem" investigations.

The ring file is set to 5 seconds. In this way, the last 5 seconds of the measurement are always available. If the measurement must be aborted for any reason, one can investigate the events leading up to this using all available channels that were being measured. It could emerge that the suspected channel "X-K2", for which overshoots were being stored, was not responsible for the abortion at all.

These "Post Mortem" investigations are especially important if an automatic test was aborted, e.g. due to a limiter that set the "JobAbort" system variable.

Starting Ring File Storage Automatically at the Start of the Measurement

The first storage module will be started automatically the first time "X-k2" exceeds its limit. How will the ring file be started? To enable this, one creates a custom variable (of logical type) and presets its value to 1. This is then a switch that is initially "on" at the start of the measurement. It makes sense to name this variable "MeasStart, for example.

FIGURE 68: Variable which is immediately "on" at the start of the measurement (preset value = 1)

One defines this variable as the storage event for the new ring file storage module.

The example can be tried out using the "StorageModule.mlb" file.



StorageModule.mlb

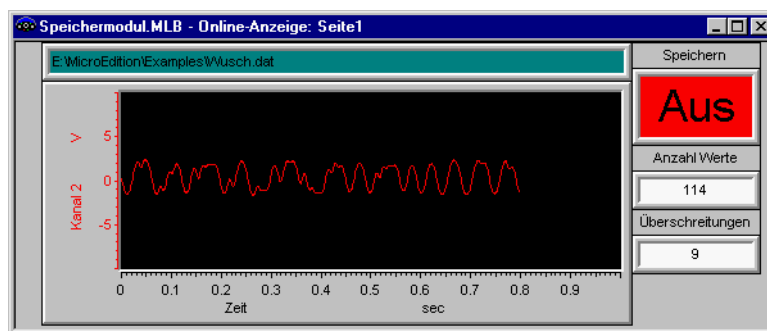


FIGURE 69: "Page 1" of the online display of the MLab project "StorageModule.mlb"

Example 3: Storing Sections

Section information serves to mark areas or events whilst storing data in the data file. In this way, it is possible to access these areas very quickly during subsequent offline analysis.

The sections are given numbers and descriptions. The status of an event variable defines when the area begins and when it ends. This can be realised, for example, by a external digital signal, by a key or by a reaction from another action module (e.g. monitor module).

Often, one performs a test in order to carry out several investigations on a component in parallel (perhaps the test can only be performed once because the test object will only survive one test). In such a situation, one constructs certain monitors and uses these to mark particularly significant areas in the files (e.g. as long as the torque exceeds 200 Nm, the marker "Torque critical" should be set). Later, one can target these areas for display in MGraph.

In order to make the construction of the example simple, the sections will be marked using the F9 key. The test will be measured and data will be stored permanently. Whenever an interesting area starts, one presses F9 (F9 on) and at the end of the interesting area F9 once again (F9 off).

You can try setting up MLab using the example file "**Section1.mlb**" file:



Section1.mlb

1. A normal display will be constructed for the measurement channels. There are four channels in the example.
2. A variable called "KeyF9", a key field for F9 which uses the "toggle" mode and a further variable called "MeasStart" preset to a value of 1 (as in the previous example) will be defined.
3. A **Storage Module** action module will be defined and will use the **MDF** file type. You will store four channels and the "KeyF9" variable. "MeasStart" is the storage event. The global switch will not be required (but does not do any harm either).

Defining the Sections

Open the properties window for the storage module and go to the **Sections** property page. There, start by entering the "KeyF9" event variable, with which - using positive logic - the beginning (0->1) and end (1->0) of an area will be marked.

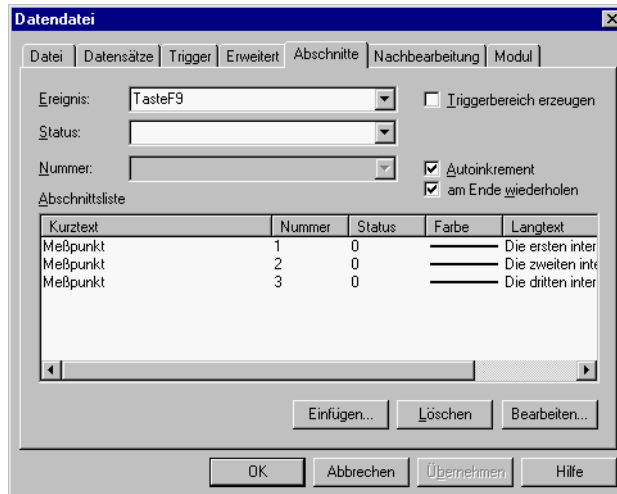


FIGURE 70: Section settings

As the illustration shows, we will simply enter three sections for this example. For each, a number, a status value and an explanatory text will be defined. For this example, the number will be incremented at each new section (KeyF9 phase). The "Autoincrement" field has been activated for this reason. The "repeat at end" field causes MLab to start from the beginning again if F9 is pressed more than three times.

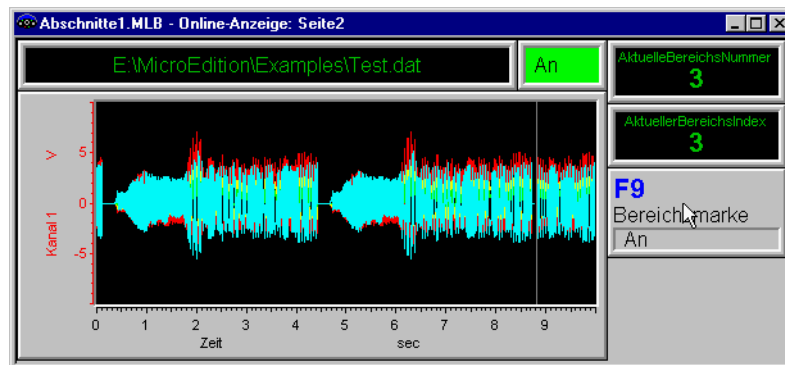


FIGURE 71: Online display of the "Section1.mlb" example

In addition to the $x(t)$ diagram, the F9 key, the filename and the storage status, two other storage module information variables are displayed.

- CurrentAreaNumber** The number entered in the section row.
- CurrentAreaIndex** The index number of the section from the list.

At the next section, the area index will increase to 4, the area number is 1 once again. The section that follows has the index 5 and the number 2, etc.

Displaying and Using the Sections in MGraph

This example does not explain every setup parameter for the window. It shows a simple, meaningful example. The example "Full-braking5.mlb" in the "Calculations" chapter shows a further variation of marking sections. Now, we want to consider what one sees of the sections during analysis in MGraph.

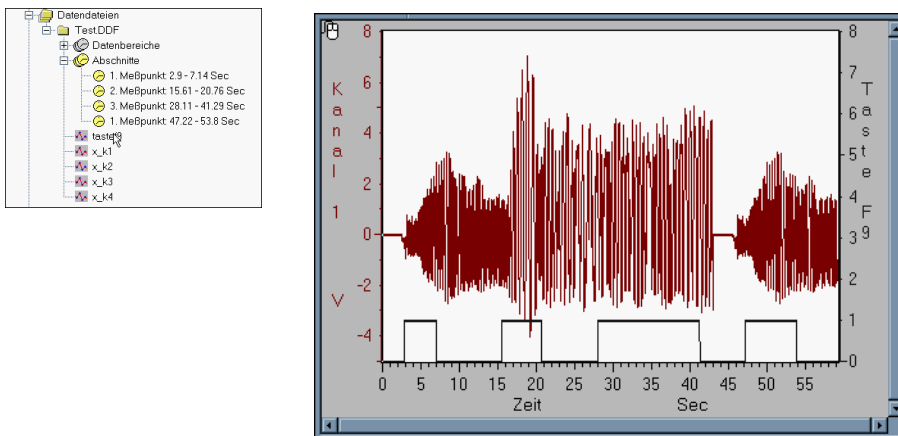


FIGURE 72: Display of channel 1 together with KeyF9, the section marking event

You can see the four marked sections in the MGraph project tree. Because we also stored the F9 key, we can see clearly when the sections were taken by viewing a channel entirely (here Channel 1). In addition, the exact time can be read from each section.

You can look at the individual sections in isolation (i.e. "zooming") by grabbing the section in the project tree with the mouse and dragging it onto the desired window. The section information is a precise definition of the x-axis for the graphical display. In MGraph, the adjustments to the display take place automatically. When dealing with large files, you will find the section markers to be very helpful as they save time searching through data.

The End of the Chapter

This chapter showed how to use the storage module. The simplest version of this module, the standard data file, is used in order to record a measurement quickly and simply. The storage module is used for sophisticated storage operations and offers advantages whenever the datasets are to be distributed across several files and perhaps with different storage frequencies.

At the conclusion of this chapter, we leave the standard applications: monitoring and data storage. In this chapter and the previous chapter, you became familiar with both the standard modules and the action modules associated with each of these subjects and consequently with the approach to *working with event variables*. MLab can generate more than just digital events. It can also generate analog values. The next two chapters explain self-generated datasets (desired values) and calculated variables.

Possible Causes of Errors

When setting up storage modules, one might, of course, forget certain setup parameters. MLab will then generate a fault list. Here are a few hints:

Path does not exist

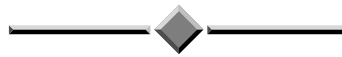
- The message "File C:\MyPath\Test.dat cannot be created" appears if the path "C:\MyPath" does not exist. Check this in the Windows Explorer and create a new folder with this name. Currently, MLab cannot create paths itself.

Tip: There is no problem leaving Explorer open in parallel with MLab. After the path has been created, switch back to MLab using ALT+TAB and start the measurement.

MKernel resolution too low

- "The selected sampling rate of 1000 Hz is too high. With the selected resolution in MKernel of 10 msec, it may only be 100.0 Hz. Should the sampling rate be adjusted to 100 Hz?" This message - or something similar - arises with polling cards. Sampling cannot be faster than the kernel. The **kernel** (MKernel.exe) is MLab's core (the "workhorse"). It carries out operations during the measurement. Its basic raster is normally 100 Hz (= 10 ms). Its maximum frequency is 1000 Hz (= 1 ms). If you want to realise a sampling rate of 1000 Hz with a polling card, set this to 1 msec. The field for this can be found under the **Extras/Options** menu item on the **Device Driver** property page. The field is called **Resolution**. Exit the message box with OK along with the next two messages that MLab then displays. Change the resolution setting, save your project, close MLab and then restart it.

Remark: The MKernel resolution corresponds to the operating raster during measurement. The action modules work in this raster. For other acquisition types (DMA, memory, interrupt), it is generally not necessary to have a higher MKernel resolution at higher sampling rates. In this case, the hardware deals with the fundamental signal sampling and buffers the data. The MLab Kernel then processes more than one data element per cycle.



Desired Values

Creating Datasets in MLab

Introduction

By passing demands for digital or analog values *from* the computer *to* the test bed, the installation moves into a **different category**. Up until now, the computer had simply recorded values passively. Consequently, the installation itself remained unaffected by a computer failure or incorrect numerical values in the computer. This is not the case when demanding desired values. Incorrect desired values can provoke uncontrolled test bed reactions. Hence, whenever giving out demands for desired values, monitoring units should always be defined as well. In general, therefore, the actual value for a desired value should be monitored to see whether the test bed really approaches the value intended.

These monitors are not only needed to switch off or shut down the installation, but can also be used for process control. An example of this would be to automatically perform certain recovery phases at a certain load (e.g. a cooling-down period or a reduction of the brake pressure), i.e. make changes to other desired values.

Types of Desired Values

Following this hint on equipment safety, we will now be introduced to the numerous possibilities for generating desired values in MLab.

- Continual and continually-changing signals
- Signal series. Procedures
- Correction to desired values (depending on actual values)
- Complex signal groups
- Digital procedures

In this chapter, you will get to know these action modules: **Signal Block**, **Multi-Step Test** and **MWave Output Block**. The latter inserts a desired value group, created with the MWave program, into an MLab Project. Other typical desired value action modules such as **Random Amplitudes**, **Recovery Modules** or **Road Signal Test** cannot be presented here.

The Examples in this Chapter

The examples in this chapter will demonstrate the basic possibilities for generating desired values:

1. Example 1: A continual, sinusoidal load on a metal rod
2. Example 2: Input drive and output drive for a brake. A load sequence.
3. Example 3: A complex load change sequence with MWave.

Most of the requirements for generating desired values can, of course, only be tested and performed on the test bed. As this is everywhere slightly different, the Tutorial must simulate a fictional test bed. MLab's test drivers offer predefined datasets. With these, it is possible to demonstrate the manner in which the various MLab modules function.

Using the test drivers provides the possibility for a theoretical simulation. The real application is usually more complicated. You should regard the examples in this chapter as practical exercises. The application of desired values to the test installation must later be realised and tested slowly and gradually.

Example 1: A Continually Changing Load

Imagine a metal rod which we want to test for bending. To do this it will be clamped on one side. The other side will be moved by a hydraulic cylinder. This load is similar to the load on a vehicle axle caused by the behaviour of the wheels.

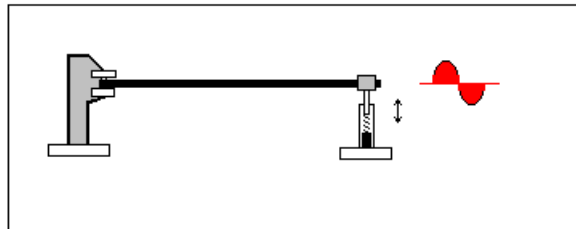


FIGURE 73: Test arrangement for Example 1 ("Sine.mlb")

The rod should be subjected to a sinusoidal load with an amplitude of ± 30 mm and a signal frequency of 2 Hz. The load will be stopped and resumed using the F10 key. A counter for the load transitions indicates the progress of the test.

For this, you will need the following steps:

1. Setting up the dataset to hold the demands. Definition of an additional device for the desired values.
2. Setting up the graphical display: $x(t)$ diagram, KeyF10 variable and key field.
3. Setting up the **Signal Block** module.

The first and third steps contain new aspects. They will be dealt with in the following.

Registering the Demand Channel as an Additional Device

When outputting a digital signal in the fourth example from the chapter on monitors, we already demonstrated that MLab groups the channels by device. Whenever analog input and output channels are located together on one acquisition card, these will nevertheless be considered as separate devices. When performing measurements, the acquisition card (the "analog inputs") is generally the **Clock Source** and the output devices (the "analog outputs") is an **Additional Device**. Any digital devices used are inserted into the list of additional units.

The **Test Driver** provides 3 separate "devices" for the simulation; "Analog Inputs" (16 channels), "Analog Outputs" (4 channels), "Digital Inputs/Outputs" (each of these can be an input or an output). In the case at hand, the "Analog Outputs" will be registered as an additional device.

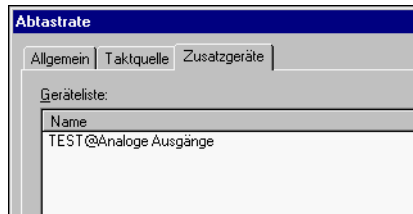


FIGURE 74: Inserting the "Analog Outputs" as an additional device



Note: The analog outputs only become available in the dataset list once the additional device has been defined.

Setting up the Signal Block Module

Before you define the **Signal Block**, you must take the following steps:

1. Insert an analog input, give it the dataset name "X_CylinderDisplacement" and calibrate it to units of +/-100 mm.
2. Insert an analog output, give it the dataset name "W_CylinderDisplacement" and calibrate it to units of +/-50 mm. (This means that actual value and desired value have different resolutions. This can occur and depends upon the measurement sensors and the signal converters.)

Now insert an action module of **Signal Block** type and set its parameters as follows:

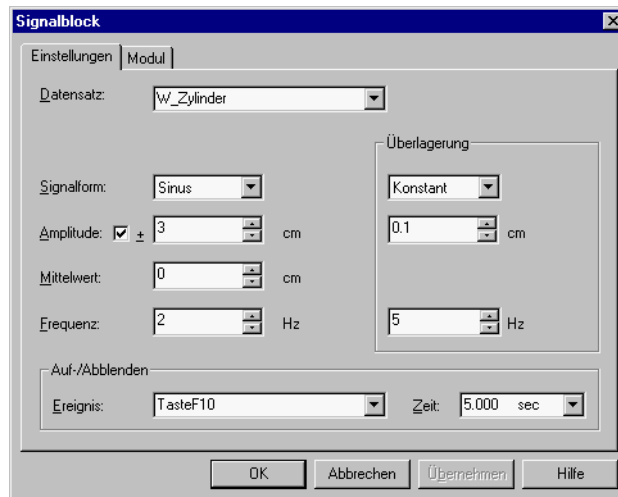


FIGURE 75: Defining a signal block

"W_Cylinder" will be the dataset created or modified. It should generate a sinusoidal signal with an amplitude of 30 mm and a frequency of 2 Hz. The window offers three groups of parameters:

- Signal Form** The Sine/Sawtooth/Square Wave signal forms can be set up with amplitude, mean value and frequency as usual. The Constant signal form sets the value to the specified mean value.
A constant offset for the base signal can be achieved using the "Mean Value" parameter.
- Modulation** The base signal can be modulated with another signal. It is possible to select one of the signal forms - Sine, Sawtooth or Square Wave - once again for the modulation. If "Constant" is displayed, this has the same meaning for the modulation parameters as the word "None".
- Fade In/Out** If the signal is to begin immediately with the full amplitude then the event must be set to 1 (or no variable defined) and the time must be set to 0 sec. Otherwise, when the measurement is started or the event variable changes from 0 to 1, the amplitude will be increased to its full value over the specified time. After a change from 1 to 0, the signal will be faded out in the same time.

Setting up the Online Graphical Display

The Information Variables from the Signal Block

The **Signal Block** action module provides two information variables. These report on the status of the module:

Time	The active time up to now as dictated by the event variable, including the fade-in and fade-out times.
Oscillations	The number of oscillations up to now, including the fade-in and fade-out times.

The number of oscillations will be displayed in a numerical field for this example.

Displaying a BMP Picture

As the small schematic drawing of the test arrangement shown in the text above exist in the form of a file, one can also display it in the online graphical display. It was quite simply created with the Windows accessory program MS-Paint, which saves pictures in the Windows BMP format. MLab can display these files. The BMP format demands a lot of memory, because it stores every pixel in the picture individually, but it is easy to generate and can be displayed rapidly. Pictures and photos of the test bed or of the test object which are available in PCX, GIF or JPG can be converted into BMP format by nearly every drawing program.

The size of the picture can be freely defined in MLab. You insert the picture using the button shown here in the margin and name the file to be displayed.



Inserting an x(t) Diagram to Match the Picture



The x(t) diagram will be positioned next to the picture and will display the "W_Cylinder" dataset. Because the dataset is calibrated to +/-50 mm, the initial setting for the Y resolution of the x(t) diagram is not correct. In order to apply the dataset's maximum value, open the diagram's properties window and switch to the **Channels** property page. Mark the dataset in the list and click on the maximising button (shown here in the margin).

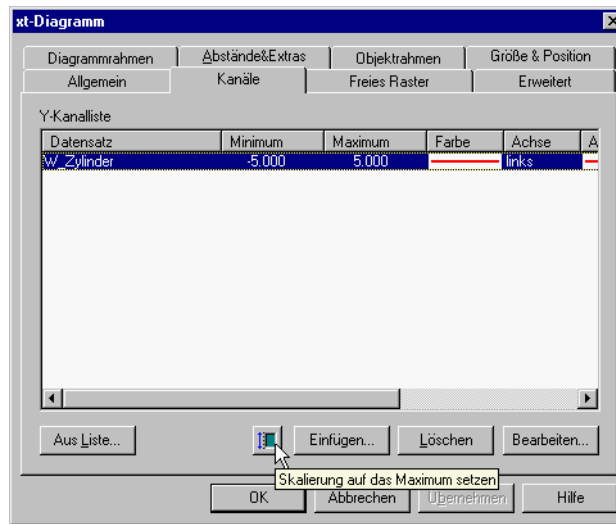


FIGURE 76: Display for the desired value in the x(t) diagram (+/-50mm). In the lower row is the maximise button.

The completed online display then looks like this:

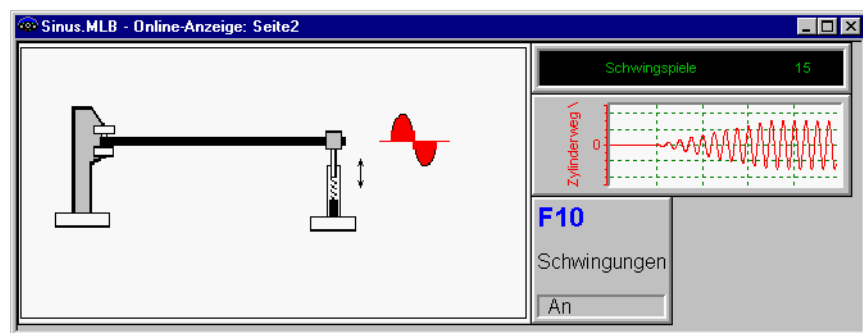
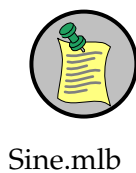


FIGURE 77: Online display for the "Sine.mlb" example

Example 2: Specifying a Sequence

A commonly-occurring test case is that of a set value having an actual signal level which is exposed to various loads. We then have two or three successive desired values. Many of these test cases arise in passenger or heavy vehicles; the desired values on the *input drive side* are subjected to loads from the desired values on the *output drive side*. A typical case is the test of a transmission with three analog desired values:

1. input drive speed
2. left brake pressure
3. right brake pressure

The situation is often even more complicated: the desired braking value is not the brake pressure, but a certain braking torque. This braking torque must then be converted into a set pressure with the use of a calculation module. The control of the braking torque must perhaps be readjusted. In addition, a transfer ratio between the desired and actual side of the braking torque can arise, depending upon the gear engaged. Further dynamic set values for the installation, i.e. desired values, may perhaps include switched levels (digital), certain transverse forces (analog) at the wheels and/or the steering angle (analog).

The tests are often durability tests and are intended for investigations into material fatigue. The **Multi-Step Test** is the basic MLab action module for these purposes and delivers a desired value for each of several steps.

Determining the Exact Sequence of Actions



Brake.mlb

It is best to draw a picture of the sequence of actions for such a multi-step test, especially if several channels are involved. In the following example, we are going to work with the MLab online display "Brake.mlb". The x-axis resolution has been set up so that exactly one sequence can be seen. A test run lasts 75 seconds.

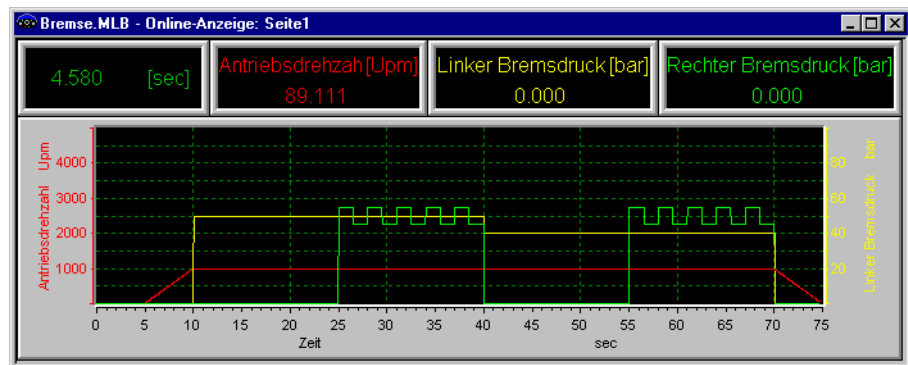


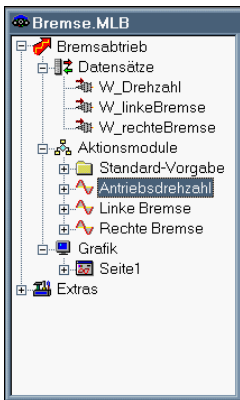
FIGURE 78: Online graphical display of the example "Brake.mlb"

- 1.) The input drive speed has the following phases:
 1. 5 seconds warm up to 10 revolutions per minute (rpm)
 2. 5 seconds ramp up to 1000 rpm
 3. 60 seconds test time to 1000 rpm
 4. 5 seconds ramp down to 0 rpm

- 2.) The left brake has 2 braking phases during the test period (1 min).
 1. 30 seconds at 50 bar
 2. 30 seconds at 40 bar

- 3.) The last half of each of the left brake's phases (15 seconds) is accompanied by a change from 55 to 45 bar at the right brake, i.e.:
 1. square wave with 3 Hz, mean value 50 bar, amplitude +/-5 bar (15 sec).

Defining the Steps as a Series



For every desired value, create a **Multi-Step Test** action module. This is because each multi-step test only relates to one desired value dataset. Give the module a meaningful name (context menu: **Rename**). The project will look approximately like the picture to the left.

Now start by defining the desired value for the input drive. To begin with, you will run the input drive speed alone in order to check that the transmission behaves as you expect. Then, in a second step, define and test the brakes. With some installations, the drive motor or the transmission can be damaged if the brake pressure is exerted at insufficient rotational speeds. In this case, a monitor module is required to inhibit the module's switch event if the speed is too low (see below).

Entering the Series of Steps for the Rotational Speed

Open the action module for the input drive speed. Start by giving general values.

Dataset	The desired value dataset "W_Speed" to be run in this series of steps.
Repetitions	How often the series of steps defined on the next property page is to be repeated.
Fade In/Out	The signal value can increase slowly, as with block signals. In the case at hand, this is not used. No switch event is entered and the time is set to 0 sec.

The entries on the first page look like this:

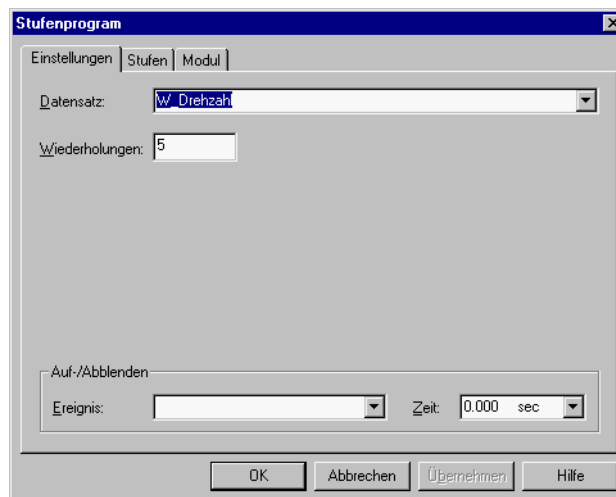


FIGURE 79: General settings for the Step Series for the "W_Speed" dataset

The actual series of steps now follows. You have the possibility to define periodic signals or linear motion (ramps). Only ramps are needed for the intended speed curve. These will be appended one after the other. If a ramp component is forgotten when making the inputs, this can be entered subsequently and set at the correct position by using the position arrows (top right in the window).

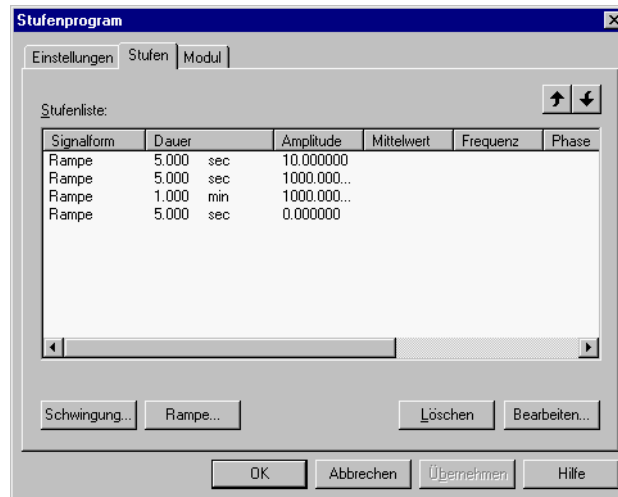


FIGURE 80: Step series for the rotational speed

Step Series for the Left Brake

The general parameters are identical for all three desired values. They should all be repeated five times and start without any fade-in phase. It must be ensured that the series of steps is of the same duration. When co-ordinating the beginning and end of a phase for periodic desired values with the states and phases of other desired values, a pocket calculator assists with the conversion. The MWave program is at hand as a solution for these problems (see Example 3).

When making demands for a desired brake pressure, it is necessary that the pressure is present immediately. For this reason, the ramp for the pressure change will be made as short as possible.

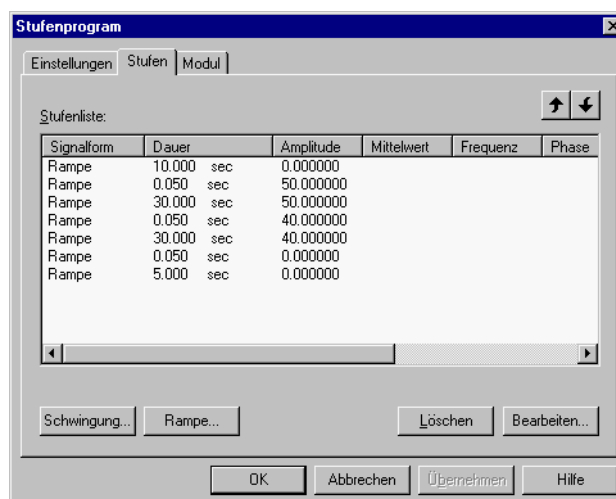


FIGURE 81: Step series for the left brake

The series of steps can be separated into four phases: 10 sec no pressure, 30 sec 50 bars, 30 sec 40 bars and, to finish with, 5 sec no pressure. The pressure jumps will be defined as short 0.05 second ramps.

Step Series for the Right Brake

The right brake should contribute with load changes between 55 and 45 bars twice at the end of the left brake's braking phase. These load changes will be realised using a square wave function having a mean value of 50 bars and an amplitude of 5 bars. For this example, it has been specified that the duration should amount to 15 seconds and within this time 5 load changes should take place. The consequence is a frequency of 3 Hz.

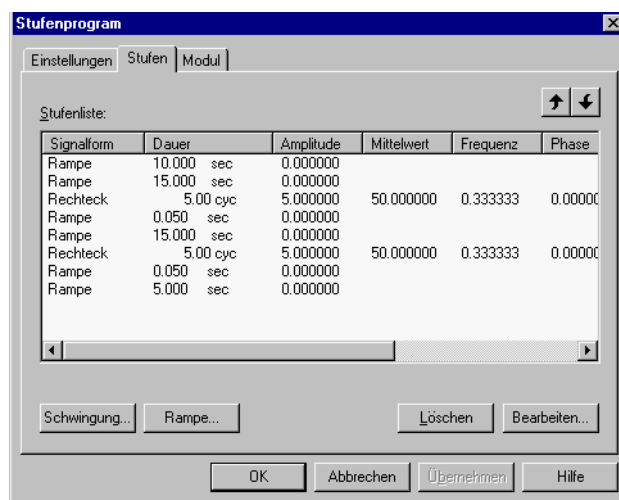


FIGURE 82: Step series for the right brake

To complete the graphical display, one adds the information variable **@Time** from one of the three modules, so that the time axis is also visible numerically. Additionally, in a real case, it would be necessary to construct the same display for the three actual values. All values, desired values and actual values can be stored (see storage module).

Example 3: Setting Desired Values with MWave

There are several further methods for generating desired values in MLab. One is the road signal test for measured data that has been recorded during a test drive and should be rerun later in the laboratory. A second method is more for programmers: the action lists and the TestControl language with which datasets and variables can be modified as desired using mathematical formulae and procedural structures. A third method is the action module for outputting MWave test files.

The First Example in the MWave Tutorial

The small MWave Tutorial shows two examples. In the first example, an axle is subjected to a load by two hydraulic cylinders. The second, more complex example is a test for an articulated shaft using a test matrix.

Building these demands into MLab is done as follows:

1. One creates an action module of the type **MWave Output Block**. There, one enters the file created in MWave (*.MWT) as the **Test File**.
2. The association between MWave and MLab is achieved via the dataset names. You must ensure that the names used in MLab are **exactly the same dataset names** as used in MWave.



MWave
Example 1.mlb

Assuming one had entered Example 1 from the MWave Tutorial, saved the parameters as described as an MWW file and created an MWT file with the values. Then, the setup for the **MWave Output Block** action module would look like this - whereby, only the **Test** parameter must have a setting (see "MWave Example 1.mlb").

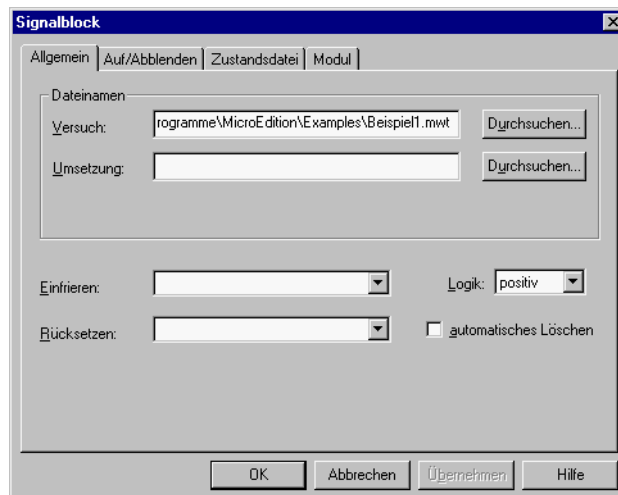


FIGURE 83: Specifying the MWT file as the test file in the MWave output block.

The MWave example uses two datasets "LeftForce" and "RightForce", which are each calibrated to +/-10 kN. The signal sequences defined in MWave will be generated for these two signals, if they have been included in the MLab project.

The online graphical display for the example shows the two signals on top of each other. The 180° phase delay between the two signals, which was defined in MWave, can also be easily recognised.

The End of the Chapter

The chapter's examples were kept simple and concentrated upon the generation and display of desired values. None of the important checks on actual values, which were mentioned at the beginning, were demonstrated. This is somewhat difficult without a test bed. The signals that the test driver generates are fixed and would not serve well as an example. The next chapter ("Calculations"), however, will make use of the fact that, for MLab, all data fields are mathematically the same and that the non-present actual values can therefore be simulated (see Example "Full-braking0.mlb").

However, although the Tutorial sets a bad example, please be warned: desired values are a serious issue. Construct the monitors before you construct the desired values and test again and again!

Possible Causes of Errors

MLab is rather economical with error messages to do with desired values. For this reason, check that any MWave file being used is correctly written and that the datasets used therein are the same as those entered in the dataset list. If an incorrect entry is made, MLab will not generate any error messages.



Calculations

Types of Calculation and Action Lists in MLab

Introduction

MLab has numerous capabilities to do calculations, as seen with sine functions under the generation of desired values. The same sinusoid which is calculated for an output dataset can, of course, also be calculated for a variable. A formula generator is integrated into MLab, which enables you to calculate formulae of any length, whereby all available datasets and variables can occur in these formulae.

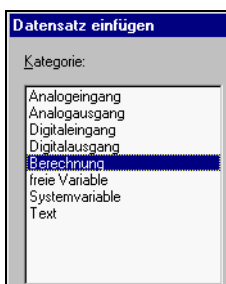
The simplest type of calculation is the continual, **permanent calculation**. A simple example of this is the calculation of the mean value between two channels. For each pair of measured values from the two channels, the mean value will be calculated online (and stored).

Sometimes an additional requirement arises: a calculation should only be performed at a certain time and use different formula depending on the event. One could also call these calculations **process-controlled** or event-controlled calculations. For this purpose one uses the so-called **action lists**.

This chapter will introduce you in steps to the possibilities made available by calculation datasets and action lists:

1. Example 1 shows a permanent calculation with calculation datasets and logic module.
2. Example 2 shows a calculation with an action list and makes these calculations dependent upon conditions.
3. Example 3 shows a somewhat more complex calculation which should be performed at a certain moment of time in the measurement.

Example 1: Permanent Calculations



The list of dataset types shows the **calculation dataset** before the variables. The calculation dataset is nothing more than a custom variable of *double* or *logic (event)* type with a formula field connected. One could also define a calculation by creating a custom variable and allocating a formula to it in an **action list** (see 2nd example). The calculation dataset will now be demonstrated in the example at hand. This type can process mathematical as well as logical formulae. In MLab, there is a special module for logical formulae: the **Logic Module**. In certain cases, the services provided by the modules overlap. As you can see, calculations are so important in MLab that you are given several possibilities for defining formulae.

In the following, a measurement with the two channels "X_k1" and "X_k2" and two custom logical variables "KeyF2" and "KeyF3" is to be set up. This measurement should perform three calculations:

1. Formula1 = (X_k1 + X_k2) / 2 (Calculation of the mean value)
2. Formula2 = (KeyF2) AND (KeyF3) (Logical AND relation)
3. Logic1 = (KeyF2) AND (KeyF3) (Logical AND relation)

The first two datasets "**Formula1**" and "**Formula2**" are executed as calculation datasets. The third dataset "**Logic1**" is a custom variable of *logical (event)* type and will be calculated with a logic module. "Formula2" and "Logic1" must, of course, always take the same value. All values are stored in the standard data file (of file type MDF and - if you wish - analysed in MGraph).

Calculating the Mean Value with a Calculation Dataset

You create a calculation dataset by choosing the type from the category list and pressing the **OK** button. The following dialog, which resembles a pocket calculator, then appears.

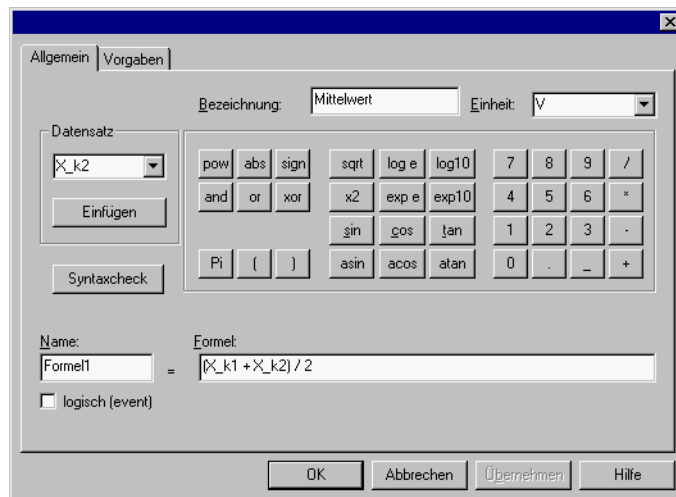


FIGURE 84: Definition of a calculation dataset

In the following, the dialog's most important parameters are listed:

- The **Name**, **Caption** and **Unit** fields belong together and define the dataset name (here "Formula1"), its caption ("Mean value") and its unit ("V"). As always, the fields **Caption** and **Unit** are optional and are only for indications in the on-line graphical display and/or offline display later (MGraph, RView etc.). The **Dataset** field is unimportant here and is only used for entering the formula (see below).

- The **logical (event)** checkbox in the bottom left corner of the dialog defines the formula's data type and is important. As long as it is not marked, "Formula1" is a *double* type - if it is marked, "Formula1" is *logical (event)* type. In the first case, it has an accuracy of 8 bytes, in the second case only 1 byte and - as far as the value range is concerned - just 1 bit (0 or 1). The formula must also comply with this setting! For a logical variable, no mathematical operations and, for a mathematical variable, no logical operations (AND, OR etc.) are allowed. You can test this with the **Syntax Check** button.
- The third and biggest area of the window, i.e. **all fields in the central area**, is dedicated to entering the formula. If you know the dataset names and operators, you can also type the formula by hand into the input field at the bottom right. Otherwise, select the datasets or variables from the **dataset** list box and insert these with the **Insert** button. Use the digits and operators as you would do on a pocket calculator. The sequence in which the operations are carried out follows the usual priority rules.

Our formula calculates the arithmetic mean of "X_k1" and "X_k2".

```
Formula1 = (X_k1 + X_k2) / 2
```

Some might be confused by the title "calculation dataset". Basically, it is a "calculation variable" if you assume that the word "dataset" is only used otherwise in the direct acquisition area (inputs, outputs and their drivers). In MLab, the word "dataset" is often a short form for datasets as well as for variables (e.g. with list boxes).

Logical Relations with a Calculation Dataset

For the second formula, create a calculation dataset and select the "**logical (event)**" data type in the large calculation field by marking the small field on the bottom left. Then, enter the formula. For the example, brackets have been used. In this case, they serve to make it easier to understand and could equally be left out. The following picture shows the corresponding part of the calculation dialog.

The screenshot shows a dialog box with two input fields: "Name:" containing "Formel2" and "Formel:" containing "(TasteF2) and (TasteF3)". Below these fields is a checkbox labeled "logisch (event)" which is checked.

FIGURE 85: Entering a logical formula

The dataset in the example is called "Formula2". You could also use a different name. You can always change a formula later using the dataset's properties window.

At this point, we assume that you created both logical variables "KeyF2" and "KeyF3" beforehand, along with a graphical key field for each as usual. You

can also create these afterwards if you take care with the names. It is best if you carry out a **Syntax Check** for the calculation window before starting the measurement.

Logical Relations with a Logic Module

You can construct the same logical formula with a custom variable and a **Logic Module** type of **Action Module**. In order to do this, carry out the following steps:

1. Create a custom variable of logical (event) type and name it "Logic1" for this example.
2. Create a **Logic Module** type of **Action Module** and insert an AND relation between the "KeyF2" and "KeyF3" for "Logic1".

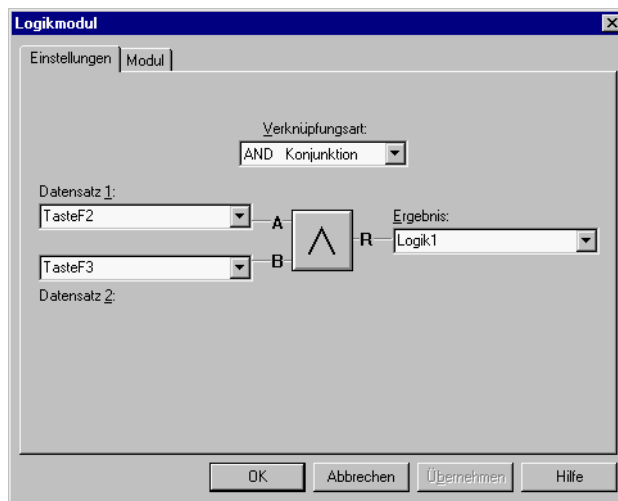


FIGURE 86: Entries for the logic module

For a simple, logical relation, the logic module is probably the faster and simpler version. The logic module offers a whole range of logical relations including Flip-Flop and Toggle. Use the example file ("**Formulae.mlb**") and your online graphical display in order to understand those relations with which you are not familiar.

Note 1: You are allowed to use an input variable simultaneously as an output variable. For example, you can assign "KeyF2" to the R input and to the output of an RS flip-flop and "KeyF3" to the S input. Then you can set F2 using F2 itself but reset with F3. Try it yourself!

Note 2: As soon as you need a logic module with 3 inputs, you must define two logic modules one after the other and link these. With action modules, by the way, the order in the project tree determines the processing sequence (see **Layout/Up/Down** menu item). **Alternatively**, you can resort to a calculation dataset. The following formula can be quickly defined there:

$$\text{Formula2} = (\text{KeyF2}) \text{ and } (\text{KeyF3}) \text{ and } (\text{KeyF4})$$

The Layout of the Online Display



Formulae.mlb

The online display for the example ("Formulae.mlb") shows several new graphical layout features. The (orange-coloured) separating bars are drawn as fixed rectangles. In addition to this, the option **Display Extended Dialog Pages for Graphical Objects** has been selected from the **Graphical** property page under the **Extras/Options** menu item. With this option selected, the property windows of the graphical objects contain several additional property pages. Among other things, you will have the possibility of changing the **diagram frame** and the **width** of various elements. A special field frame for Formula2's result field has been laid out using these.

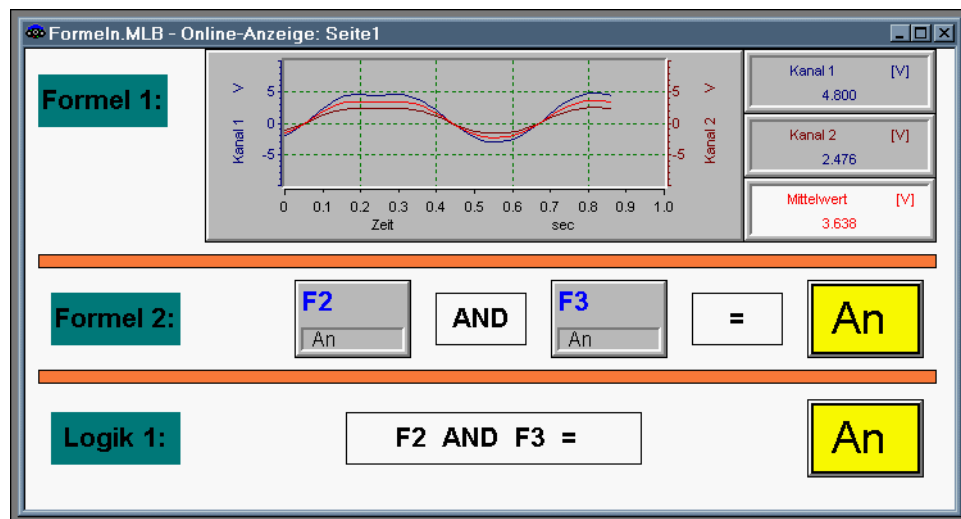


FIGURE 87: Online graphical display of example "Formulae.mlb"

This graphic page displays all three of the formulae created, one below the other. In the graphic page's context menu, the **Save** menu item is available for saving complicated graphical definitions like this. With this, the page construction is stored as a template (Filename "Formulae.von"). The next example shows how this template can be used in (or loaded into) a different project.

Example 2: Action Lists: Calculations in Sequences

The previous example can also be solved with an action list. With action lists, though, one can achieve a lot more. For example, calculations can be made at a certain time and under certain conditions. Wherever one gains more design freedom, however, the form of input usually becomes more abstract. This is also the case here.

As a first step for this example, the same two formulae from the previous example will be reconstructed (replicated) and calculated continually. In a second step, the calculation of the formula will be made dependant upon the state of the F4 Key.

Replication of the Previous Example with an Action List

You can, of course, load the entire previous example and modify it. Here are brief instructions on how to build the example from scratch:

1. The custom variables "Formula1" (*double* type) and "Formula2" (*Logical* type) are created. Furthermore, KeyF2 and KeyF3 are created (at first, without a key field).
2. You activate the **Graphic** field in the project tree and select the **Load** menu item from the context menu (right mouse button). You load the display template from the last example called "Formula.von".
3. The loaded graphic page will be modified slightly. The bottom line is deleted, as we do not have the variable "Logic1". The F2 key is copied (CTRL+C and then CTRL+V) and changed to the F4 key. The custom variable "KeyF4" is then created.
4. An **Action List** type of action module is now inserted.

The action list has a simple and easy-to-learn syntax. The most important points are the following:

1. The action list accepts every formula which is also permitted in the formula field of the calculation dataset. The formula "A = B + 5" is written exactly so:

A = B + 5;

The most important difference is the semicolon at the end of the line.

2. Additionally, the action list accepts "If...then...else" constructions.

```
if (KeyF2)
{
    A = 5;
}
else
{
    A = 1;
    B = 2;
}
```

These lines mean: If the F2 key is on, the variable "A" is assigned the value 5. If not, then the variable "A" is assigned the value 1 and "B" the value 2. It is not defined what value "B" takes if F2 is on. If F2 is on, "B" retains its current value.

There are situations when it is very useful to be able to make such constructions. You have every freedom to do this in an action list. You can nest the calculations and if/else constructions as often as you like, i.e. within the curly brackets of an if/else branch, another if/else construction is possible. Make sure you keep track of the order and number of the brackets!

Note 1: The else branch is optional of course.

Note 2: Logical variables are switched with "on/off" and not with 0 or 1 (otherwise a syntax error results). One writes, for example, so:

"ev1 = on;" or "ev1 = off;"

Note 3: Spaces and new lines only serve to improve clarity. The above example could also be written in two lines:

```
if (KeyF2)    { A = 5; }
else         { A = 1; B = 2; }
```

The freedom given by the action list has the disadvantage that there is no guidance for the user during entry. The variable names, the semicolons and the {} brackets must be placed correctly. In order to test this, the syntax check is activated. This displays a message for each mistake. If the syntax is valid, no messages appear. The syntax check does *not* distinguish between capital and small letters.

We will enter the following for our purposes:

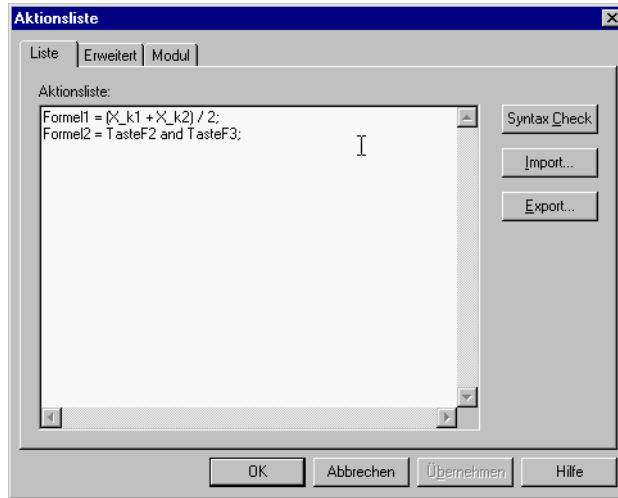


FIGURE 88: Both formulae in the action list

You must switch to the second property page - called **Extended** - so that the action list can actually be executed:



FIGURE 89: "Permanent Execution" mode of operation

The mode of operation defines when the action list is executed. If the **Permanent Execution** entry is selected, then the complete action list is executed once per cycle.

Test your project settings. The on-line graphical display should be the same as in the previous example.

Coupling to the F4 Key

The execution should now be made dependant upon the F4 key. There are two situations which could be thought up here. In a first variation, both formulae should be calculated as long as F4 is on. In the second variation, both formulae should be calculated each time F4 is pressed.

The first variation can be solved as follows:

```
if (KeyF4)
{
  Formula1 = (X_k1 + X_k2) / 2;
  Formula2 = KeyF2 and KeyF3;
}
```

The calculation is executed once per cycle whenever KeyF4 is on and is not executed otherwise.

The same can also be achieved, however, with the second property page:

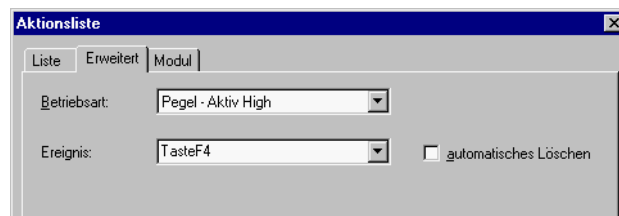


FIGURE 90: Level setting with the F4 key

This setting takes into consideration the level (= status) of KeyF4 when executing the action list. As long as F4 is on, the complete action list is processed.

The following figure also shows directly how the second variation is realised:

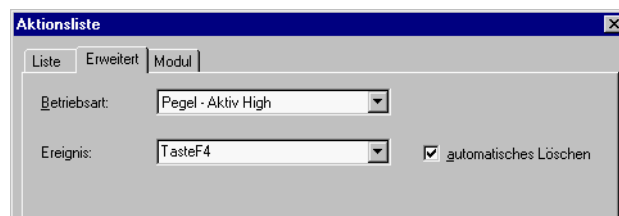


FIGURE 91: KeyF4 will be reset immediately

Here, the calculation only takes place when Key F4 is set to "on", i.e. at the instant it is pressed, because Key F4 is reset immediately after each execution of the action list.



ActionList.mlb

The situation at hand determines which variation is required. With its online graphical display, the example "ActionList.mlb" demonstrates the last variation:

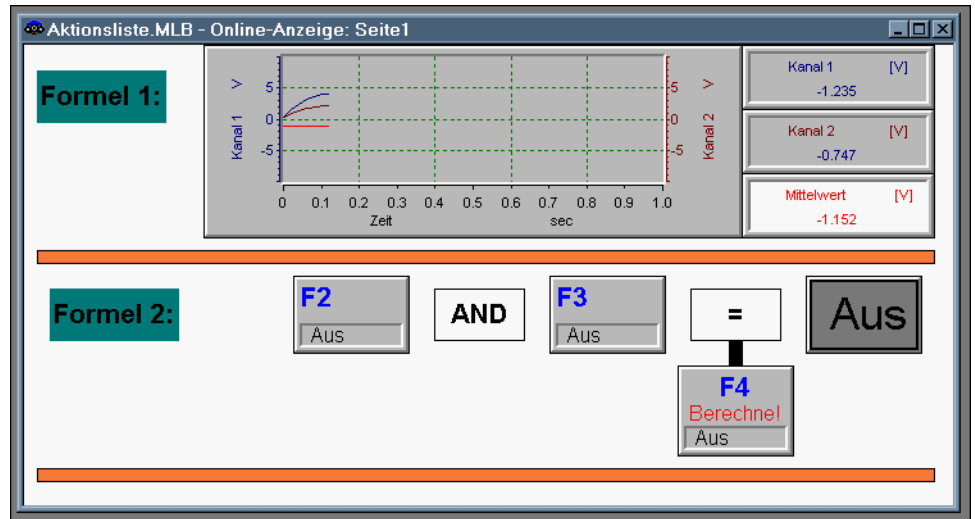


FIGURE 92: Online graphical display for the example "ActionList.mlb".

Formula1 and Formula2 are only calculated at the instant F4 is pressed.

Example 3: A Complex Calculation

The following example should now be realised: for a variety of HGV brake construction (HGV: Heavy Goods Vehicle), the *mean braking deceleration* in a full-braking manoeuvre should be determined. To calculate the average, the values at the start and end of a braking manoeuvre should be used. According to the requirements, braking starts at 80% of the maximum speed and ends at 10%.

The following plot shows the test sequence:

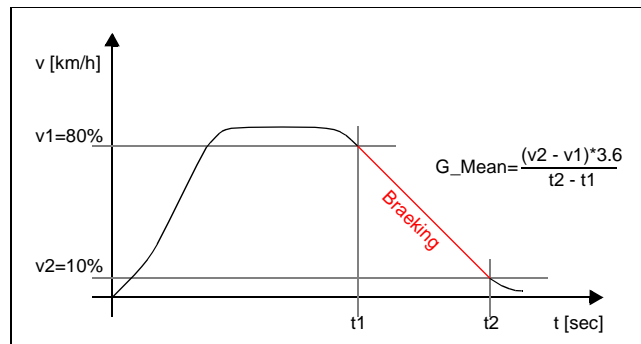


FIGURE 93: Test sequence for full-braking

The plot shows the velocity curve. The HGV will be accelerated up to a speed of over 80%. At some point, full-braking will be initiated. The computer must recognise when the velocity drops below the 80% level and start a stop-watch. After this it must then recognise when the velocity falls below the 10% level and stop the stop-watch. At both points (t_1 and t_2) the velocity must be registered and at point t_2 the specified formula must be calculated, because the result must be available immediately.

A further complication is that our user does not want to specify what the maximum velocity is. It is agreed that he will enter the maximum speed at the beginning of the measurement and that this will then remain constant unless it is modified. The user has also declared that he wants to perform several tests one after the other and store these all in one file, whereby the test number should be stored with these as well. It is then agreed that each actual braking period will be specially marked between t_1 and t_2 (MDF sections) so that, for example, the seventh braking manoeuvre can be examined at the click of a mouse during analysis.

The only actual value is the velocity.

The velocity sensor converts +/-100 kmh into +/-10 volts.

The Solution Plan

First, a list of the requirements will be produced:

1. An input field for the maximum velocity and then, on the press of a button the calculation of the velocity values for the 80% and 10% limits.
2. Two monitors for the 80% and 10% limits. The monitors may only operate under braking and not when driving off.
3. The construction of a stop-watch and definition of the start and stop events.
4. The calculation of the formula shown in the drawing for the mean braking deceleration. The incrementation of a test counter.
5. The construction of a marking event for the section marking of the storage file. Goes *on* during active braking when dropping below 80% and then *off* below 10%.

That's all. Because we would rather test out the project in the office first, we need a simulation environment with which we can test whether or not the algorithm as such is correct. This can be done very easily in MLab. This means we will start with Step 0 and not with Step 1.

Step 0: The Simulation Environment

In the office, no measured signals from the HGV are available. The key measured signal is the velocity. For this however, the sketch above already provides us with the information in graphical form. We replicate the expected curve using a multi-step module and use our algorithm on this. When we then switch from the office to the HGV, we must only exchange the names of our simulation variables for the names of the real input datasets.

It has been agreed that the velocity will later have "X_v" as its dataset name ("v" for velocity and "X", as usual, for an actual value). Hence, our simulation value will be called "S_v".

1. We create a custom variable of type "float" (name "S_v"). For this, we create an x(t) plot which is scaled between 0 and 100 kmh.
2. We insert an action module sub-group and call this "Simulation". Such sub-groups only serve to maintain an overview, they have no function themselves. At the end of Step 6 of the example, we will have a lot of action modules. In such cases, it is good to group action modules that belong together. From the context menu for "Simulation" (not that from the menu for **Action Module!**), we select the **New** menu item and insert a **Multi-Step Test**.
3. Now we construct the planned test curve using a pair of part-ramps. For test purposes (after consulting the user), we will start by taking a maximum velocity of 60 kmh.
This results in: 80% = 48 kmh and 10% = 6 kmh.

4. A storage module called "Datafile" is then inserted. The measurement file will be called "Full-braking", be an MDF-type file and initially only store the "S_v" variable.
5. We then insert a nice picture of an HGV (such pictures can be found on the Internet and can be converted into the Windows Bitmap format required by MLab (BMP) by many drawing programs) and display the information variables for **@Filename** and **@StorageStatus**. Storage will be initiated at the start of the measurement with the "MeasStart" variable we met earlier.
6. In order for MLab to operate, it requires at least one real dataset. In the absence of any datasets, MLab has no real-time driver with which to define the operating raster. The user has given us much information about the actual values. We will add "X_k1" to the dataset list, rename it "X_v" as agreed and assume that this will later be the real velocity. We will not yet display it, however.



Our state of affairs now looks like this ("**Full-braking0.mlb**"):

Full-braking0.mlb

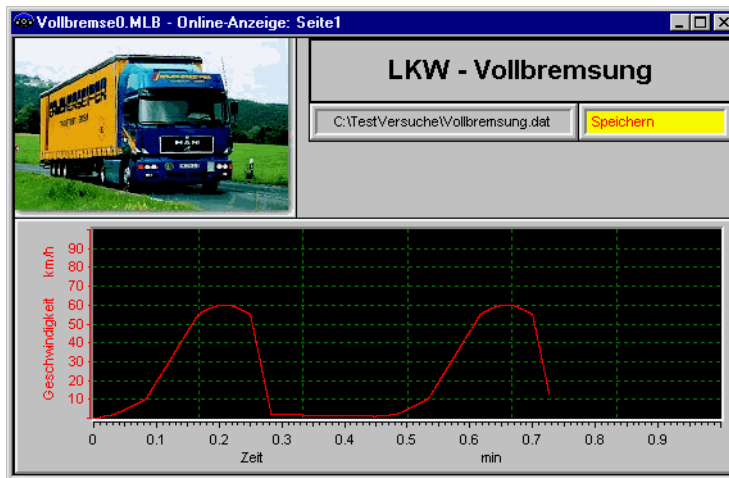


FIGURE 94: The simulation curve for velocity ("**Full-braking0.mlb**")

Step 1: Entering the Maximum Velocity

An input field for the maximum velocity must now be created and, in addition, three variables for the three velocities and a switch event "Apply" at which the maximum value will be adopted and the percentual values calculated.



1. You define three float variables "v_max", "v_80" and "v_10" and a logical "KeyF5".
2. You open the graphic page and insert a numerical input field for "v_max". You can either do this via the **Insert/Input Number** menu item or by using the button icon shown here in the margin. Set all the options

on the **Refresh** property page.

3. We require three text boxes for 100, 80 and 10 percent and then two numerical fields for "v__80" and "v_10" and a key field for F5.
4. You insert a **sub-group** with the name "Input" and, beneath this, an action list with the percent calculations. The **Mode of Operation** for KeyF5 (on the **Extended** property page) is set to **Level - Active High**. The **reset** field is activated.



Full-braking1.mlb

Now try out changes to the maximum velocity. Note that the cursor only appears after you have clicked in the field with the mouse. The two new values will be calculated as soon as you press F5. (Example "**Full-braking0.mlb**").

Step 2: Monitoring the 80% and the 10% Thresholds

Unfortunately, monitoring overshoots or undershoots of the 80% and 10% limits cannot be performed with monitor modules. This is because such monitors only operate with fixed limits. We must resort again, therefore, to the action list.

This action list will be the first complicated one. It has two cases:

1. **Brake readiness:** the HGV attains brake readiness as soon as it has exceeded the 80% limit after driving off. Our brake test makes no sense before this.
2. **Active braking:** as soon as brake readiness has been attained, braking should be registered as active once dropping below 80%. As far as we are concerned, braking is over below 10%.

The following approach can be recommended:

1. You create two logical variables: "evBrakeReady" and "evBrakeActive". It is good practice to always prefix a logical variable with the letters "ev", so that it is easily recognisable as an event in the action list.
2. Create switch fields for both variables to display their current status. For its "on" text, the first field is given the text "Ready" and is coloured green. The second field is given the text "Active" as its "on" text and is given a warning colour, yellow or red. The "off" text is simply "off" in each case and in grey.
3. Create the central **sub-group** "Braking". Insert an **Action List** called "Braking Recognition" into this group and set its mode of operation to "Permanent Execution". The **Event** field remains empty.
4. Now enter the algorithm.

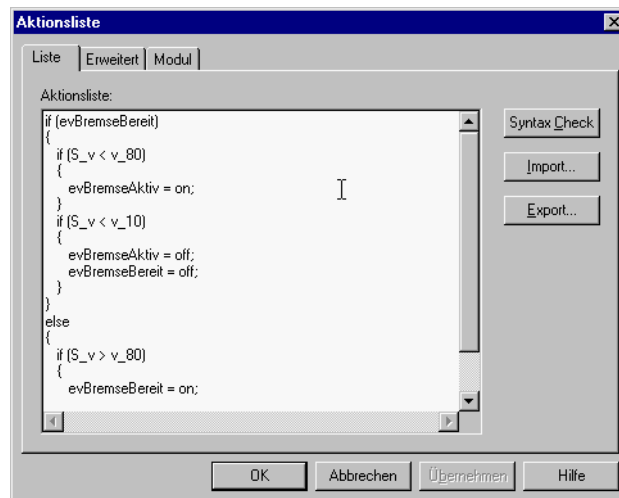


FIGURE 95: First algorithm for the braking recognition

Such an algorithm must, of course, be developed gradually. Two types of error can occur here. The first are syntax errors and incorrectly-typed dataset names. The second type are conceptual errors.

The *first type* of error can be detected using the **Syntax Check**. The number of errors declared should only be taken as a guide because all of the subsequent parsing errors are counted as well. If, however, the syntax check finds no errors, then your syntax is correct. The *second type* of error arises from conceptual faults. Here, the simulation curve from Step 0 can help. Display all the logically relevant states in your graphical display and then you will quickly find your conceptual faults as well.

As MLab offers the possibility to export action lists, the following approach to constructing more complex formulae is recommended for correct syntax.

- Enter the entire formula as you believe it should be.
- Save the text in a text file by using the **Export** button (export = save).
- Delete one or more of the suspected lines and carry out the syntax check until no more errors are detected. Then **Import** the entire formula again and modify it.

Lastly, test the entire formula for its logical correctness. Here, the online display and the simulation curve will help. When doing this, one will discover a further necessary step, Step 5:

5. Set the variable "v_max" to 60, "v_80" to 48 and "v_10" to 6. Otherwise, all values will take zero values from the beginning and your logic will not function.

The example "Full-braking2.mlb" shows the current state of the project.

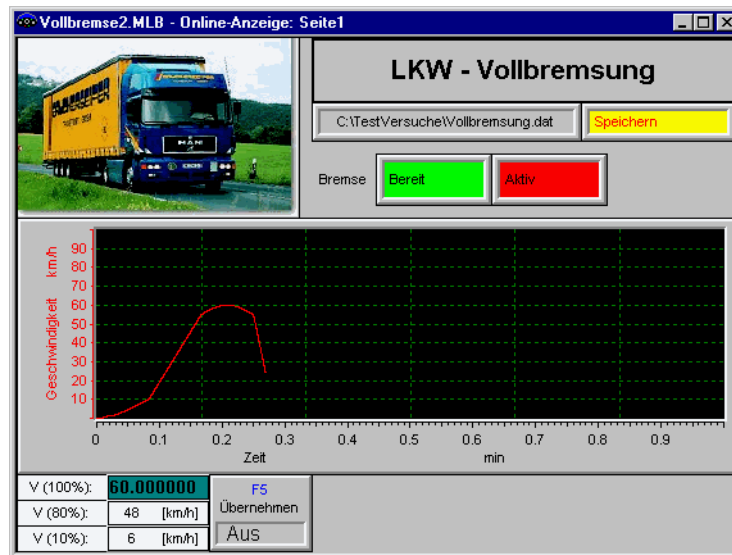


FIGURE 96: Online display with input fields and braking recognition

Step 3: Constructing the Stop-Watch

In order to calculate the time, we need a system variable to hold the **Step-Width** between one operating cycle and the next. This **System Variable** will be assigned a value by the system (MLab) and not by the user. It is the reciprocal of the sampling rate. As long as braking is active, the stop-watch should add the step-width to a variable called "BrakingTime" once per operating cycle.

1. Create a new variable of type "System Variable" and select the "Step-Width" in the "Type" field.
2. Create a float variable called "BrakingTime".
3. The new "Stop-Watch" **Action List** is added to the "Braking" group. It only has one calculation line, namely:

$$\text{BrakingTime} = \text{BrakingTime} + \text{Step-Width};$$
 The **Mode of Operation** is "Level - Active High" and is of the "evBrakeActive" event.
4. The braking time needs a numerical display.
5. The braking time must be reset at the beginning of the braking period for the next braking manoeuvre. Do this in the "Braking Recognition" **Action List** at the point where "evBrakeReady" is set to "on".

The example "Full-braking3.mlb" shows the project at this point. With our simulation curve, the braking time amounts to 1.58 seconds.

Try this out!

Step 4: Calculating the Mean Deceleration

Now we come to the part, which is the reason for all this effort. At the start of braking, the velocity must be stored "v(t1)" and then at the end "v(t2)". At the end, the formula for "G_Mean", the mean deceleration, must also be calculated. You can incorporate these actions into the "Braking Recognition" action list at the appropriate positions. We are going to take a different route in our example by using two further, small action lists.

1. Create three float variables: "v_t1", "v_t2" and "G_Mean".
2. Add two new action lists called "Time t1" and "Time t2" beneath the "Braking" group.
3. "Time t1" uses the rising edge of "evBrakeActive" (**Mode of Operation** is "Rising Edge" of the event "evBrakeActive"). The formula has just one line:

$$v_{t1} = S_v;$$
4. "Time t2" uses the falling edge. Once again the active velocity is taken using $v_{t2} = S_v$ and then the following formula is calculated:

$$G_Mean = ((v_{t2} - v_{t1}) * 3.6) / \text{BrakingTime} [m/s^2]$$
5. G_Mean must - just as for the braking time - be set to 0 at the start of the

next measurement. Do this once again in the "Braking Recognition" action list whenever "evBrakeReady" is set to "on".

- It is important to ensure that the stop-watch still adds the cycle time to the braking time. You can modify the order of the action modules using the context menu (Up/Down). In our case, the order should be: *Braking Recognition -> Stop-Watch -> t1 -> t2.*

The example "Full-braking4.mlb" shows the current state of the project. Our simulation curve results in a mean deceleration value of -0.010 m/s^2 . If you increase the maximum velocity from the preset value of 60 kmh to 70 kmh (and press F5 afterwards), you will obtain a value of -0.012 m/s^2 for the next braking manoeuvre. These values will not necessarily bear any resemblance to the value from the HGV later, as we have neither real measured values nor a real braking time.

Step 5: Marking and Counting the Braking Manoeuvres

Now, data storage should be marked and counted. In principle, marking can be tested with the aid of the "evBrakeActive" variable, as braking lasts exactly as long as this. However, after a few tests it will become clear that whenever the "evBrakeActive" marker is reset the value of G_Mean is difficult to see in the analysis display in MGraph. This is because G_Mean is calculated just at "evBrakeActive"'s falling edge and is, therefore, difficult to recognise displayed on the extreme right edge of the $x(t)$ plot. The marker should last another five values, for example so that G_Mean is calculated and is easy to recognise in an $x(t)$ plot. (More a question of aesthetics.) We require, therefore, two counters, one for the five extra display values and one for the current number of braking manoeuvres.

- Create two variables of type *long (counter)* called "ExtraValues" and "BrakingManoeuvres". In addition, create a logical variable called "evBrakeMarker".
- The "Braking Recognition" **Action List** must be extended. When the variable "evBrakeActive" ends, the counter "ExtraValues" is set to 1. At the end of the algorithm, a small IF sequence is added to increment the "ExtraValues" as long as this is greater than 0. As soon as the value is greater than 5 it is set back to 0 - as is "evBrakeMarker".

```

if (ExtraValues > 0)
{
    ExtraValues = ExtraValues + 1;
    if (ExtraValues > 5)
    {
        evBrakeMarker = off;
        ExtraValues = 0;
    }
}

```

- The counter "BrakingManoeuvres" is incremented at the start of braking readiness and a new numeric field displays the number of braking manoeuvres at the top right.

- The **Event** "evBrakeMarker" is now entered in the **Sections** property page in the **Storage Module**. One then inserts a section row with meaningful text for this section, e.g. braking manoeuvre. The entry's number and status value are 0. The **repeat at end** field is activated. The **Number** will not be incremented automatically, but will instead come from our "Braking-Manoeuvres" counter variable.

The example "Full-braking5.mlb" shows the completed project. This is what the graphical display looks like (here showing the end of the 6th. braking manoeuvre):



Full-braking5.mlb

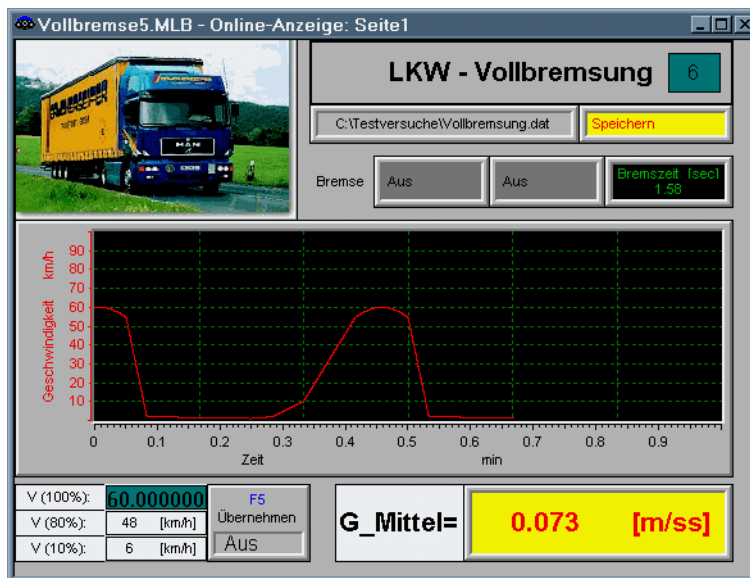


FIGURE 97: The completed project: the online graphical display after full braking manoeuvre number 6

The End of the Chapter

In this chapter, you have been introduced to examples of continual calculations and calculations positioned within the test sequence. You have seen that action lists are a very flexible and powerful instrument, but that one must have a clear and absolutely exact concept of how the measurements must run in order to be able to use these successfully. You should make a list of the variables, the action lists and the other action modules.

MLab has chosen this open system in order to master the complexity of real calculation. Test beds are complicated installations. The reality is tricky, but that is why we are making measurements in the first place!

In the example at hand, we will find that in the first test on the HGV, the velocity signal jitters rather a lot. The 80% and 10% thresholds will be crossed several times during one braking manoeuvre. We must then refine the algorithm for "evBrakeActive". Only the final 80% undershoot may set "evBrake-

Active" and the first 10% undershoot must reset "evBrakeActive" without going on again. The braking readiness can no longer be recognised so simply. Here, we must find a new criterion together with the user for a further braking manoeuvre (and, and, and, ...). MLab projects can always be easily modified and remodified. That is their strength.

Possible Causes of Errors

A common error message is the indication that an action list contains one or more syntax errors. This message appears if you do not monitor this yourself with the **Syntax Check**.

Another type of error message originates from your display and relates to its logical errors. The more information you pack into your display the more likely it is for logical errors to occur. It is important to differentiate between actions, associations and calculations which occur once, those which occur under certain logical conditions and those which occur continually. The faults often lie here.

What's Next?

A certain level of complexity has been achieved with the action lists. There are however, two interesting areas which go further. One is a functionality offered by the many action modules we have not yet mentioned (e.g. integration, spectral analysis, classification). For these we refer you to the Reference Manual. The other area is that of complex process control. For this, the expanded facilities resulting from the "TestControl" test bed language are available. The next chapter concerns itself with this subject.



MLab and TestControl

The Test bed Language TestControl completes MLab

Introduction

TestControl is a macro language with which all activities that are relevant for test beds can be defined¹. MLab alone covers a part of the facilities provided by TestControl. TestControl can be used to complement MLab where this is not adequate.

MLab and TestControl use the same *kernel*, meaning that part which does all the work during the measurement². What you see from MLab is basically only the smaller part, the windows and parameter definitions. The main performance and the main work lie in the kernel.

Linking TestControl to MLab

The procedure for linking is as follows: You work with MLab for as long as possible. If a certain extended service is needed from TestControl, you only have to add in this part - like an action list only with a more extensive syntax. All datasets and variables used in MLab are available in the TestControl processes. The complete setup of the online graphical display, of data acquisition and storage, and of much of the monitoring etc. remain with MLab.

-
1. In earlier versions TestControl was called TDAC or TorturDAC. Processes from TDAC Version 4.0 will, on the whole, be accepted by TestControl.
 2. For those who are familiar with this, a more detailed explanation: Up to now TestControl was DOS-based. MLab has always been 32-bit Windows-based. The two kernels, i.e. the executable parts, are growing together bit-by-bit. The services of the DOS TestControl have not yet all been integrated into the MLab kernel. This will be done in response to individual requests or needs.

How to proceed:

1. TestControl requires the following files:

<i>Filename</i>	<i>Content</i>
Tcc32.exe	The Compiler
Symbol.tcc	The list of TestControl's system variables
Syntax.tcc	The TestControl syntax
Keyboard.tcc	Special key codes (for function keys, CTRL combinations etc.)

TABLE 1: *Files needed for TestControl*

2. Create a separate directory for your processes (only to maintain an overview, not obligatory). You should copy the three TCC files into this directory. The compiler itself must only be accessible via the path.
3. Now write the processes. In order to do this, declare the MLab variable as "external" and then use it as normal in TestControl.
4. Then compile the processes. It is best if you use a Make utility. Create a Make file (*.mak) and then compile with the "/f" option.
5. The compiler generates an output file from all of the specified process files (PRC files). The standard name for this is usually OUTPUT.TC.
6. You enter this OUTPUT.TC (or rather its name) into the MLab project under the "Administration - TestControl" menu item. Additionally, you name the start process with which the process sequence should begin.

Example: Change a Text Display and Restart

Some of the things TestControl adds to MLab are changes to text variables (beyond that possible with the switch), log files which can be formatted freely and parameter files for storing the status of a test bed in the event of a test run having been aborted and having to be restarted from the same point.

This example should do the following: A message should change every 2 seconds between a first and a second text. During this procedure, a counter counts the changes of text in a short loop and in a second loop another counter counts the number of times both texts are run through completely. In other words, Counter 2 is half as fast as Counter 1. The actual counter readings are stored in a file called "Restart.sys" with each outer looping. When restarting the measurement after an abortion, the counters restart correctly from the last status.

1. An arbitrary measurement and $x(t)$ diagram are created in MLab.
2. In MLab, the text variable "Message_1" and two counter variables are created and, in addition to these, a text field and two numerical fields.
3. The following algorithm is written in an ASCII editor and stored under the name "Start.prc".



```

c:\tc\test1\start.prc Window 1
extern text Meldung_1;
extern counter zaehler_1;
extern counter zaehler_2;

realtime start
{
  // Anfang: Lese die letzten Werte
  read (filename = "c:\tc\test1\restart.sys",
        data = zaehler_1, data = zaehler_2)

  // Schleife
  label (l_anfang)
  pause (2)
  store ("Enge Schleife !" to Meldung_1)
  increment (zaehler_1)
  pause (2)
  store ("Weite Schleife" to Meldung_1)
  increment (zaehler_1)
  increment (zaehler_2)

  // Sichere stets den aktuellen Zaehlerstand
  write (filename = "c:\tc\test1\restart.sys", filestatus = new,
        data = zaehler_1, data = zaehler_2)
  goto (l_anfang)
}

----- end of file -----

```

FIGURE 98: TestControl process to solve the problem

In the process¹, the three variables used are declared as external at the top.

1. TestControl names every small independently-functioning procedural unit a "process". There are real-time processes and background processes. The former are invoked in every operating cycle and respond to a command; the latter run whenever there is time left over. Processes are started and checked with the start, stop, continue and call commands.

After this, the latest counter values are read (if "Restart.sys" exists, otherwise not). Then, both loops follow with wait times. At the end, the current parameter readings are stored and the process jumps back to the beginning of the loop.

The "Start.prc" file is compiled using the TCC32 compiler. The resulting file "Output.tc is (optionally) renamed as "Start.tc".

Back in the MLab Project, you enter this file under the **Administration/TestControl** menu item.

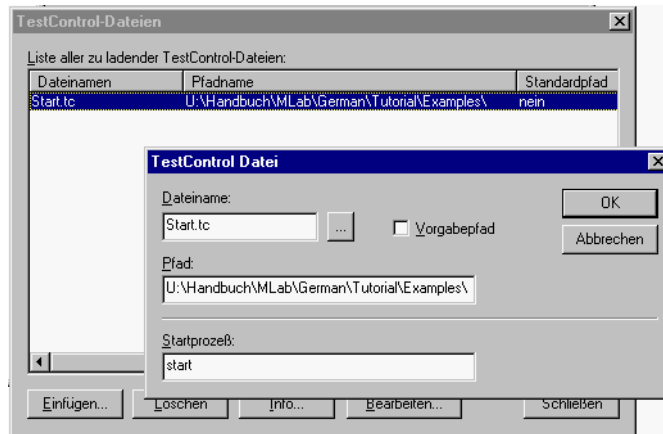


FIGURE 99: Inserting a TestControl file

From now on, the "Start" process now functions in the project like an additional, very high-performance action list. From within this process, you can successively call up or start all other possible TestControl Processes for your project. The project example shows the following graphical display. The measurement can be aborted several times and always restarts from the last counter reading.



Start.mlb

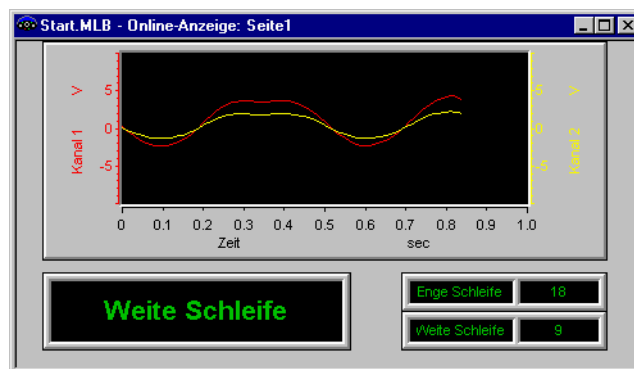


FIGURE 100: Online graphical display of the TestControl example "Start.mlb".

Notes for Larger Test bed Projects

Test beds generally have certain states or operational phases. During the planning and realisation of a test bed installation, it is necessary to be clear about the states and operational phases which can possibly occur. **Operational phases** are periods during which the desired values and the actual values change regularly, but in which the test bed ends up always doing the same, and/or finds itself in repetitive loops - however long these might be.

Often, the make-up of the phases is as follows:

Resting State	The test bed is on, the software is running. Certain parameters can be adjusted. Ready to be run up.
Running Up	The test bed is brought into its operating region (rotational speeds, pressures, positions).
Operational Phase ("Running")	The test bed works through the test programme. The test object is subjected to loads, the data is being stored (selectively).
Running Down	Abortion or end of a test. Return to resting state. Prevent the test or readiness to restart test.

Sometimes there are several very different operational phases, sometimes a variety of basic constructions are permitted. This list demonstrates the simplest of models.

When setting up a test bed, one can differentiate between the following phases of installation:

1. **The Basic Set-up:** Here the electrical communication and all inputs and outputs are checked out. At this stage, an MLab project should be created to clearly display all inputs and outputs. One will have to resort back to this reference project again and again, whenever any mechanical or electrical changes are made and it must be proved that the communication works without problem.
2. **The Resting State:** The resting state should verify the readiness for testing. Here, limits must be defined for all the relevant desired value / actual value pairings in accordance with the statements *"No start under these conditions..."*, *"The measurement must be aborted under these conditions"*. Here, one should differentiate between locks to protect the staff and the test bed and locks to ensure that only meaningful measurements are performed.

3. **Running Up** includes the definition of the schedule. We might say, for example, "first run up the rotational speed, then the pressure can follow". Here, special monitors must be devised to ensure the correct order and to abort if necessary.
4. The construction for **Running Down** precedes the construction for the running phase. Here, as well, it is usually necessary to keep to a schedule. Perhaps, logs are to be made (reason for abortion etc.). Before moving to the running phase, it must be clarified how to abort. It must also always be possible to perform an abortion manually. A special key (or key combination) on the keyboard is often recommended in addition to the usual emergency shut-off switch.
5. For the **Running Phase**, the following parameters must be defined: storage, timing control, reactions and control, logging and - for longer measurements - intermediate parameters for restarting after an abortion ("Rest Type").

Generally, many requirements only arise after some time. It should be planned to discuss the test bed once again after several test weeks. At this time, the algorithmic procedures are still fresh in the mind. The first MLab developments arose from the need to change and expand measurement equipment easily on site. With MLab you are using an open product; its projects "can easily develop by themselves".



Index

A

- Action List 116
 - Complex example 126
 - Example 121
 - Export 130
 - Import 130
 - Syntax Check 130
- Action Module
 - Action list 121
 - Extremity marker 62
 - Limit Monitor 67
 - Logic Module 119
 - Multi-Step Test 106
 - MWave Output Block 111
 - Signal Block 102
 - Storage Module 78
 - Sub-group 127
- Additional Device 68, 102

B

- Basic steps 30

C

- Calculation Dataset
 - Definition 116
 - Example 117
- Calculations 115
 - Permanent 116
 - Example* 116
 - Process-controlled 116
- Channel 82
- Clock Source 31
- Context menu 40

D

- Data Multiplexing 79
- Data Recognition 14, 21
- Data Usage 14, 22
- Datasets 82
 - Category 117
 - Editing 35
 - Inserting 33

- Desired Values 99
 - Multi-Step Test 106
 - Signal Block 102
 - Types 100
 - with MWave 111
- Digital Output
 - Bit number 70
 - Dataset 68

E

- Event 55
- Example
 - ActionList.mlb 125
 - Brake.mlb 107
 - ExtremityMarker1.mlb 59
 - ExtremityMarker2.mlb 65
 - Formulae.mlb 120
 - Full-braking0.mlb 128, 129
 - Full-braking5.mlb 134
 - MWave Example 1.mlb 111
 - Range.mlb 73
 - Section1.mlb 93
 - Sine.mlb 105
 - Start.mlb 141
 - StorageModule.mlb 92
 - Testtrials.mlb 86
- Extremity Marker 64
 - Inhibiting and Activating 66
 - Insert 65

G

- Graphic
 - Bar 55
 - BMP Picture 104
 - Dynamic Text Box 85
 - Input Number 128
 - Key 57
 - numerical Field 43
 - Static Text Box 85
 - Switch 73
 - Template 48, 120
 - x(t) Diagram 40, 105
 - Maximum value* 105

I

if/else 122

J

Job 68

K

kernel 96

L

Logic Module
Example 119

M

MKernel 96
Monitoring 52
Types 52
MSetup 14
Adding the Hardware 18
Example CAESAR MOPS 20
Example DAQ700 20
Running 17
Multiplexed data 79
Multi-step test
Example 127

O

Overview 10

P

Pre-setting the Clock Source 32
Pre-setting the Hardware 30

R

Ring buffer 10
Ring File 78
Example 91

S

Sampling Rate 31, 82, 89
Total sampling rate 90
Sampling rate
Per channel 90
Signal 82
Simulation Environment

Construction 127
Standard Data File 78
Example 81
Standard default 78
Standard Limit Monitor 52
Example 53
Standard variables 84
Start of the Measurement
Automatically storage 92
Starting a Measurement 45
Storage Format
MDF 80
RMS 80
Storage Module 78
Example 87, 128
information variables 86
Sections 94, 133
Sub-group 127
System Variable
JobAbort 91
Step-Width 132

T

Test beds 142
Operational phases 142
Planning 142
Test Driver 102
Simulation 102
TestControl
Compiler 139
Example 140
Linking 138
Menu Item 141
Process 140
Trigger 54, 84

U

User Groups 12

V

Variables 60, 82

