Measurement Technology beyond the decimal point

ELEKTRONIK GMBH

TestManager CE

Software Development Environment

for test sequences and test

TestManager CE V1.5.1.46 - Typ: 7GLUECKSPIEL - Zufallszahl - 01 Zufallszahl EC: 0.999
Typ Auswahl
Messwerte
Einstellungen
Editieren
Ablauf
Tools
Info
Intern
Hilfe
Exit
Start (Auf)
Typwahl (F2)
Setup
Type Edit
Test Edit
IP Edit
Passwort
Reset (Alt+F5)
Ende (Alt+F4)
Prüfergebnis
Kurven-Monitor
PASS
FAIL
Mode : Step-By-Step
TestManager CE V1.5.1.46 - Typ: 6NETSAMPLES - Sample .Net - 01 VariantDotNet EC: 0.999
Info
MCD Elektronik GmbH
TestManager CE
(c) 2001 - 2009 by MCD Elektronik GmbH
Version 1.5.1 Build 46 (2009-10-12)
Hompage: http:\\www.mcd-elektronik.de
Demo version
Close Info
TestCollection TestManager CE Setup
Willkommen beim Setup Assistenten von TestCollection TestManager CE
Der Setup Assistent wird TestCollection TestManager CE auf Ihrem Computer installieren. Klicken Sie auf "Weiter" um fortzufahren, oder auf "Abbrechen" um den Setup Assistent zu beenden.

TestManager Classic Edition

Test sequencer including a
development environment for test procedures.

MCD Elektronik TestManager CE • Software Development Environment for test sequences and test
First Edition, for TestManager, version 1.5.1.46

Content

Introduction

Have you ever worked in a software development environment, or with „Classic“ programming languages like Visual Basic, C #, C++ or any other C-variant?
Then you should be familiar with important fundamental concepts of programming languages, such as data types, loops or syntax. If you have not had any experience with programming and seem completely alien to the basic concepts, you will still find it moderately easy to create test sequences with the MCD TestManager.

MCD TestManager CE Software

The MCD TestManager provides a test sequencer including a development environment for test procedures. The environment allows you to create and manage family types and their variants, which can then be used for the audit process. The TestManager provides a wide range of possibilities, for example the connection to different hardware components. Common bus systems, such as LIN, CAN, TCP / IP and IIC- bus are also supported and a variety of tools and configuration options are offered to the user.

The Basic Idea of the Book

This book serves to facilitate the entry into the MCD TestManager development environment. The help of exercises improves access to the functions and ways to facilitate the TestManager CE. It cannot however be regarded as a general reference guide or a handbook. The focus of the book is its exercises that are closely aligned with practice. They are chosen so that the read information can be directly implemented and that the user obtains useful tips and further information.

As already mentioned, this book is not a general reference. It is more of a tutorial guide that covers the scope and simplicity of the topics at hand. However, should questions or problems arise, see Appendix, for further detailed sources of information.

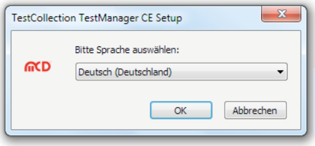
1. Introduction

Download TestManager CE

To use the book and perform the exercises effectively, you can download the MCD TestManager CE from the Internet. The software can be obtained on our website at **www.mcd-elektronik.de/tmce.html.**
Download the ➡ **trial version** of TestManager.

Installation

Once the download is complete, you will find a zip folder with the name Setup.zip where your downloads are stored. This contains the **SetupTestManagerCE.exe** that must be executed. When the installation window appears, first choose the preferred language and confirm with the ➡ **OK** button.

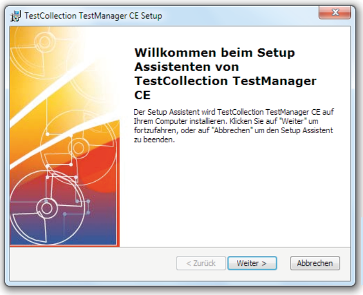

TestCollection TestManager CE Setup

Bitte Sprache auswählen:

Deutsch (Deutschland)

Figure 1-1
Language Settings

To confirm that you want to install the TestManager CE, click on ➡**Next.**

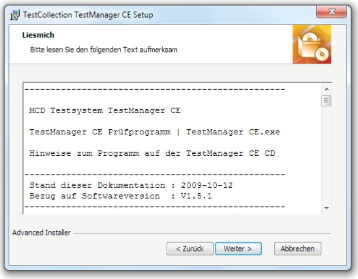

TestCollection TestManager CE Setup

Willkommen beim Setup Assistenten von TestCollection TestManager CE

Der Setup Assistent wird TestCollection TestManager CE auf Ihrem Computer installieren. Klicken Sie auf "Weiter" um fortzufahren, oder auf "Abbrechen" um den Setup Assistent zu beenden.

Figure 1-2
Installation

Important information about the system requirements and how to deal with the program will appear in the following window. Read the text carefully and then confirm with ➡ **Next.**



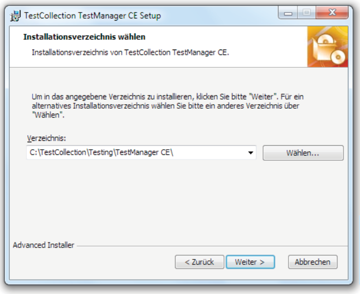
Liesmich

Bitte lesen Sie den folgenden Text aufmerksam

```
--------------------------------------------------

MCD Testsystem TestManager CE

TestManager CE Prüfprogramm | TestManager CE.exe

Hinweise zum Programm auf der TestManager CE CD

--------------------------------------------------
Stand dieser Dokumentation : 2009-10-12
Bezug auf Softwareversion  : V1.5.1
--------------------------------------------------
```

Figure 1-3
Information about the Program

The next step will be prompted to select the directory under which the program will be installed. Confirm or ➡ **change** the default path with Select and then confirm with ➡**Next.**



Installationsverzeichnis wählen

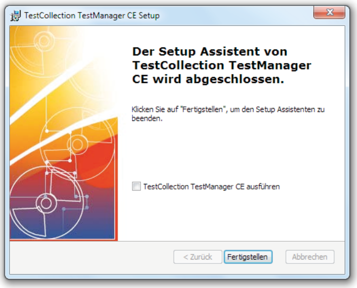
Installationsverzeichnis von TestCollection TestManager CE.

Um in das angegebene Verzeichnis zu installieren, klicken Sie bitte "Weiter". Für ein alternatives Installationsverzeichnis wählen Sie bitte ein anderes Verzeichnis über "Wählen".

Verzeichnis:
C:\TestCollection\Testing\TestManager CE\

Figure 1-4
Select Directory

To install the TestManager CE select ➡ **Install.** Once installation is completed, select ➡**Finish.**



Der Setup Assistent von TestCollection TestManager CE wird abgeschlossen.

Klicken Sie auf "Fertigstellen", um den Setup Assistenten zu beenden.

☐ TestCollection TestManager CE ausführen

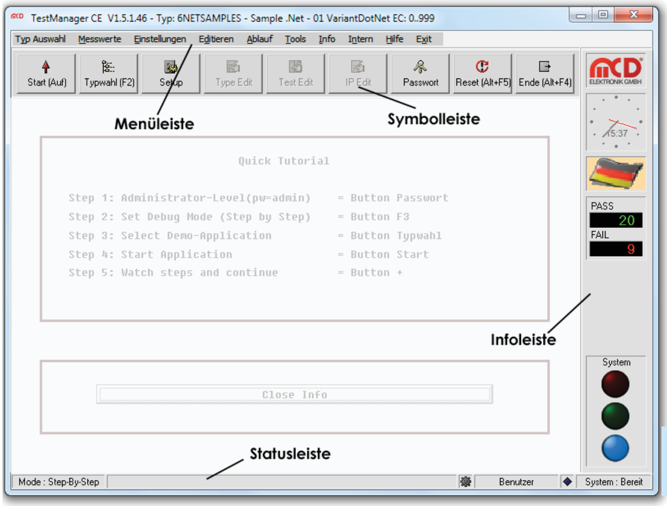
Figure 1-5
Completion of Setup

The MCD TestManager CE is now installed on your system and can be executed. A shortcut is also created on your desktop (TestCollection.TestManager CE), through which you can start the TestManager.

First application

Double click on the link of the TestManager CE to start. Depending on the operating system there may be a message that states that the program needs to be restarted. Should this occur, close the program and restart it. This program is a demo version, and will often prompt an info box that can be confirmed by ➡**OK**.
It also offers the ➡ **Order now** option to order a full version of TestManager.

The development environment



```
Quick Tutorial

Step 1: Administrator-Level(pw=admin)    = Button Passwort
Step 2: Set Debug Mode (Step by Step)    = Button F3
Step 3: Select Demo-Application          = Button Typwahl
Step 4: Start Application                = Button Start
Step 5: Watch steps and continue         = Button +
```

Figure 1-3
Program Window

Figure 1-3 shows the program window of the IDE. Here, you can access your settings, tools, functions and much more. You can also receive more information and examples to these settings, features, tools, etc.

The **main area** is used for the display of test results, when this option is activated. Alternatively, this area could be used to retrieve information during the inspection process. The **main menu** is integrated in the menu bar. This gains access to all functions, as long as they are enabled in the setup. The menu bar contains the following tabs (see Appendix - Important TMCE menus): Type selection, Values, Setup, Edit, Start, Tools, Info, Intern, Help and Exit.

The **toolbar** contains buttons for calling up important features without having to go through the menu. The number and layout of the buttons can be edited and customized to fit the needs of the user.
The toolbar contains the following default buttons: Start, Type Selection, Setup, Type Edit, Edit Test, Edit IP, Password, Reset, and End.

Information about the system is found in the **info bar**. You can also hide individual elements in this bar. It contains the status lights that reflect the system‘s current status. A clock and information about the currently selected language can also be found in this bar.

Tip:
In the general settings, the icon and info bar can be customized according to the preferences of its user.

The **status bar** contains information about which mode is currently selected, the license of the software, the user and the system.

☐ Exercise 1-1
Start an existing test process

Objective:
- Familiarization of test sequences
- Structure of test sequences
- Usage of TestManager

Step by step:

1. Log on as administrator
Navigate from the **toolbar** ➧ **Password**. Press the ➧ **F4** function key or select **Administrator** and type in ➧ **admin** as the password. Confirm the password by pressing Enter or the ➧ **OK** button.

Note:
Users in the TestManager can be divided into five password levels. Behind the various password levels, different permissions are stored. These can be edited by going to the **menu bar** ➧ **Intern** ➧ **Access Levels.**

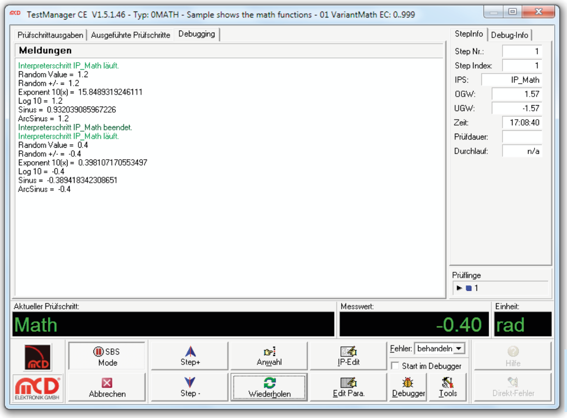
2. Selection of family type 0MATH:
Navigate from the ➧ **toolbar** ➧ **Typeselect** to open the type selection window. Open the tree structure of the **family type 0MATH** by double clicking on the family name or the sign ➧ +. Use the same approach to select

➡ **Code: 01 VariantMath,** the Family type, and then select sequence ➡ **1**. Confirm your choice with the ➡ **OK** button.

SBS and Automatic Mode:
During normal operation, the system is set to automatic mode. Here, the test is carried out precisely according to the specified test sequence. Through the individual outcomes of the test-steps, one result is examined so that it could rate the specimen. The SBS mode (Step-by-Step) gives the user the capabilities to process and edit the sequence step by step. The mode is provided for the devel oper, the test procedures or the Debugging. During a test step, information can be given and the source code can be edited.

3. Manually start the test sequences:
The test sequence can be started by ➡ **toolbar** ➡ **Start (up)** or alternatively via the 'arrow up' key on the keyboard. During single step mode (step-by-step, SBS) the system stops after every test sequence and allows the operator to select the next test step arbitrarily. Switching steps can take place either before the start of the test and during the test.

4. Repeat test step:
If the system is in SBS (Step-by-step) mode, the individual test steps can be repeated. This can be done via the ➡ **Repeat** button located at the bottom of the window.


TestManager CE V1.5.1.46 - Typ: 0MATH - Sample shows the math functions - 01 VariantMath EC: 0..999

Meldungen

```
Interpreterschritt IP_Math läuft.
Random Value =  1.2
Random +/- =  1.2
Exponent 10(x) =  15.8489319246111
Log 10 =  1.2
Sinus =  0.932039085967226
ArcSinus =  1.2
Interpreterschritt IP_Math beendet.
Interpreterschritt IP_Math läuft.
Random Value =  0.4
Random +/- =  -0.4
Exponent 10(x) =  0.398107170553497
Log 10 =  -0.4
Sinus =  -0.389418342308651
ArcSinus =  -0.4
```

Step Nr.:	1
Step Index:	1
IPS:	IP_Math
OGW:	1.57
UGW:	-1.57
Zeit:	17:08:40
Prüfdauer:	
Durchlauf:	n/a

Aktueller Prüfschritt: Math
Messwert: -0.40
Einheit: rad
Fehler: behandeln
Start im Debugger

Figure 1-4
Repeat Test Step

What is the procedure?
In this sequence, a random number is initially generated. **Figure 1-4** shows

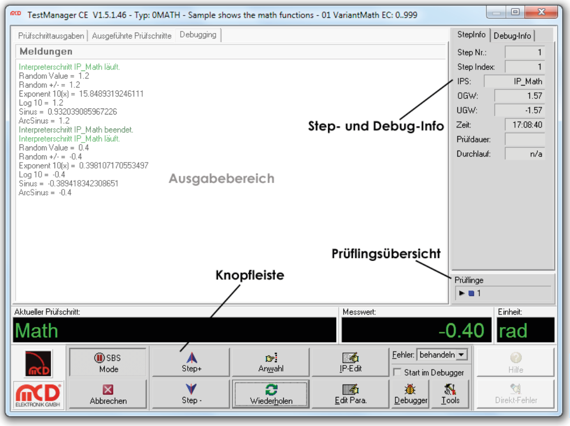
the step that was repeated three times. The first time, the random number generated was 0, the second time 1.2, and the third time 0.4. A randomly selected sign is also chosen for these random numbers. The random number is now used as an exponent to base 10 and the result is then calculated. Example from the picture: 10(to the power of) -0,4 = 0.398107170553497.
Now, in order to obtain the randomly generated number, the mathematical operation is reversed. The logarithm base 10 is calculated from the result. From this calculation, the prior generated random number of (-0.4) is retrieved. Now the sine of the random number is calculated (-0.38941 ...) and this operation is again reversed in order to calculate the arc sine of the result. Here, the randomly generated number retrieved should be displayed as the measured value in the measurement value window.

5. Selecting the next steps:
The next step can be initiated by selecting the ➡ **Step+** button.

The testing process is completed after only one step, since the test sequence only includes one step. After selecting the next steps with the **Step +** button, the test sequence is completed and the system returns to the main window.

Test Window

Once a trial starts, the normal program window switches to the test window.
The areas in the above right display the step-info and debug-info, the current contact position and the DUT overview being tested for the current contact position. If the DUT has only one contact position, the contact position indication will be hidden.



```
Interpreterschritt IP_Math läuft.
Random Value =  1.2
Random +/- =  1.2
Exponent 10(x) =  15.8489319246111
Log 10 =  1.2
Sinus =  0.932039085967226
ArcSinus =  1.2
Interpreterschritt IP_Math beendet.
Interpreterschritt IP_Math läuft.
Random Value =  0.4
Random +/- =  -0.4
Exponent 10(x) =  0.398107170553497
Log 10 =  -0.4
Sinus =  -0.389418342308651
ArcSinus =  -0.4
```

Step Nr.:	1
Step Index:	1
IPS:	IP_Math
OGW:	1.57
UGW:	-1.57
Zeit:	17:08:40
Prüfdauer:	
Durchlauf:	n/a

Prüflinge: 1

Aktueller Prüfschritt: Math
Messwert: -0.40
Einheit: rad
Fehler: behandeln

Figure 1-5
Test Window

The **output range** can be switched back and forth between test sequence output, the display of the executed test steps and the Debug display. SBS test steps can be controlled by the button panel, and depending on the password level, parameters of the steps can be edited as well. Test steps can be cancelled, forwarded (step +), switched back (step-) and repeated via the button panel. The sequence mode can be changed (from SBS to automatic and vice versa) via the Mode button. This can also be done during the course of testing. The interpreter editor (test step source code) and editor of the parameters (parameter data for test steps) can be called upon.
Additional tools and information can be customized with the Debugger and the Tools buttons. Process steps can be repeated, for example, verifying measurement results by performing test steps again. Through the error-detect setting, the user has the opportunity to influence the behavior of the test steps in case of failure
The S**tep- and Debug info-window** provides an overview of details to the steps, upper and lower limits, test time and the current test period.

2. Step by step for own testing process

After gaining initial experience with the test manager, the next question is:
How is a test sequence created?
This question will be answered by step by step illustrations with specific examples. The test sequence from Exercise 1-1 still has pretty little to do with measuring and testing technology, but is quite easy to understand because it concerns mathematical equations. This process merely scratches the surface of measurement technology and is initially set up to help you understand the build-up, the structure, and the elements involved in the testing process.

Family Type

The program allows testing of various DUT types. Each DUT type to test requires its own set of data: process of the tests, extent of tests, thresholds, test steps, etc. This set of data is known as the family type. Each family type consists of one or more variants. The family type is the umbrella term of the data records. Variants (variant types) with similar characteristics are subcategorized together under this 'umbrella'. One possible example would be a DUT that is available in different variants (variant types), for example, a car radio (12V) or a truck radio (24V). These can be managed under the same family type.

Exercise 2-1
Creating a New Family Type

To begin your testing process, first you have to create your own family type. In this exercise, you can derive the existing 0MATH family type. When you find 0MATH, there is a function that allows you to derivate or copy the existing family type. The system then creates a new family type, and gives the option of editing the name and the content included from the family of origin. That way, the 0MATH family type and its content remains unchanged and is preserved for further use.

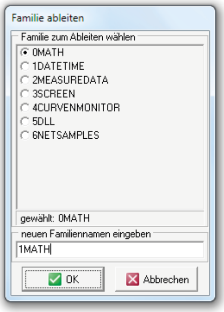
Step by step:

1. Log on as administrator
   Navigate from the ➡ **toolbar** ➡Password. Press the F4 function key or select Administrator and type in ➡admin as the password. Confirm the password by pressing Enter or the OK button

2. Open the **Edit Types and Variants** window via the ➡ **toolbar** ➡ **Type Edit**.

3. Select the family type **0MATH** in the window and select the button in the ➡ **toolbar** ➡ **Derivate family type from existing family.**

4. Assign a new name (example: 1MATH) and confirm this with the ➡**OK** button.


Familie ableiten

Familie zum Ableiten wählen
- [✓ 0MATH]
- 1DATETIME
- 2MEASUREDATA
- 3SCREEN
- 4CURVENMONITOR
- 5DLL
- 6NETSAMPLES

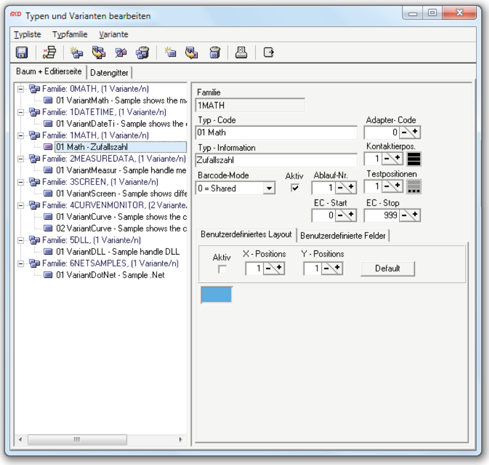
gewählt: 0MATH

neuen Familiennamen eingeben: 1MATH

Figure 2-1
Derive family type

Tip:
The name of the type family is used for creating files and directories. Here is where the test receives its title.

5. Change the **Type Code** (e.g.: 01 Math), since two variants with the same name are not allowed. The **Type Information** field can also be change (example: ran dom number), but this is not mandatory.


Typen und Varianten bearbeiten

- Familie: 0MATH, (1 Variante/n)
  - 01 VariantMath - Sample shows the m
- Familie: 1DATETIME, (1 Variante/n)
  - 01 VariantDateTi - Sample shows the
- Familie: 1MATH, (1 Variante/n)
  - 01 Math - Zufallszahl
- Familie: 2MEASUREDATA, (1 Variante/n)
  - 01 VariantMeasur - Sample handle me
- Familie: 3SCREEN, (1 Variante/n)
  - 01 VariantScreen - Sample shows diffe
- Familie: 4CURVENMONITOR, (2 Variante
  - 01 VariantCurve - Sample shows the c
  - 02 VariantCurve - Sample shows the c
- Familie: 5DLL, (1 Variante/n)
  - 01 VariantDLL - Sample handle DLL
- Familie: 6NETSAMPLES, (1 Variante/n)
  - 01 VariantDotNet - Sample .Net

Familie: 1MATH
Typ - Code: 01 Math
Adapter- Code: 0
Typ - Information: Zufallszahl
Kontaktierpos.: 1
Barcode-Mode: 0 = Shared
Aktiv: ✓
Ablauf-Nr.: 1
Testpositionen: 1
EC - Start: 0
EC - Stop: 999

Aktiv: ☐
X - Positions: 1
Y - Positions: 1

Bild 2-2
Type List

6. Save the Type List
   ➡ **toolbar** ➡ **Save Type List**

Family types of the currently used window **(Figure 2.2)** can also be created, renamed and deleted.
To create a new family type, select one of the given samples in the Edit Types and Variants window and from the ➡ **toolbar** ➡ **select create new type of family**. As a result, a new family type with no content is created.
Select the Rename family type button in the ➡**toolbar** in order to rename the already existent family. The family types listed in the left portion of the window can be deleted by selecting the delete whole family type with variants.
To save changes, go to toolbar ➡ **Type List** ➡ **Save Type List**.

Variant Type

A variant type is a clearly identifiable type of DUT. The data within a variant type determines the exact testing process that is supposed to be carried out by the TestManager system. In case there are two DUT's whose types of examination differ, there is an option to split them up and define them as two different variant types

Note:
You can manage similar DUT's from the given selection of variants. The minor differences can then be chosen and categorized under the filtered variants.

Exercise 2-2
Management of Variants

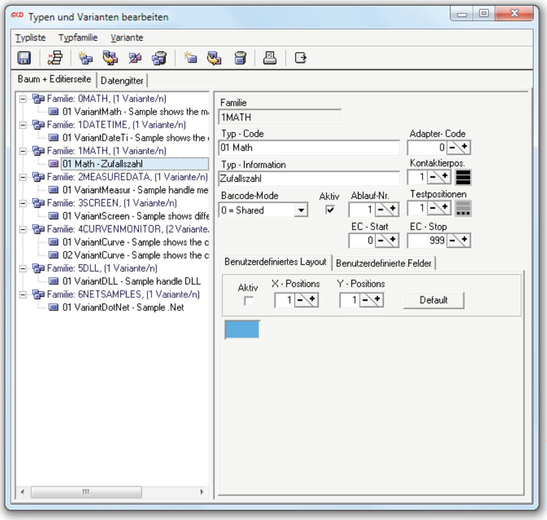
To establish your own sequence, create a new variant to the already derived family type (1MATH) from **Exercise 2-1.**

Step by step:

1. Log on as administrator
   Navigate from the ➡**toolbar** ➡**Password.** Press the F4 function key or select Administrator and type in ➡admin as the **password**. Confirm the password by pressing Enter or the OK button

2. Create variants:
   Variants can be created in the **Edit Types and Variants** window by going to the ➡**toolbar** ➡ **Type Edit**.

3. Select a family type in the left panel where a new variant type was created (from Exercise 2.1: 1MATH)➡ **toolbar** ➡ **Create New Variant**

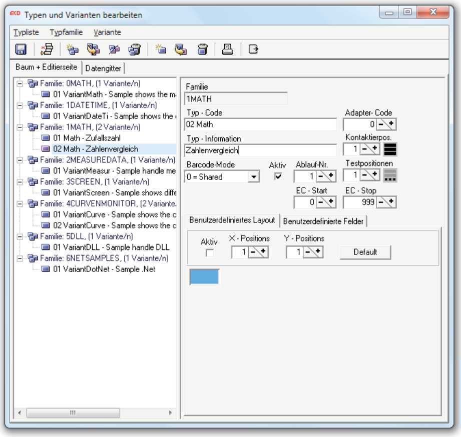

Typen und Varianten bearbeiten
Typliste
Typfamilie
Variante
Baum + Editierseite
Datengitter
Benutzerdefiniertes Layout
Benutzerdefinierte Felder
Default

Familie: 0MATH, (1 Variante/n)
01 VariantMath - Sample shows the m
Familie: 1DATETIME, (1 Variante/n)
01 VariantDateTi - Sample shows the
Familie: 1MATH, (1 Variante/n)
01 Math - Zufallszahl
Familie: 2MEASUREDATA, (1 Variante/n)
01 VariantMeasur - Sample handle me
Familie: 3SCREEN, (1 Variante/n)
01 VariantScreen - Sample shows diffe
Familie: 4CURVENMONITOR, (2 Variante
01 VariantCurve - Sample shows the c
02 VariantCurve - Sample shows the c
Familie: 5DLL, (1 Variante/n)
01 VariantDLL - Sample handle DLL
Familie: 6NETSAMPLES, (1 Variante/n)
01 VariantDotNet - Sample .Net

Familie: 1MATH
Typ - Code: 01 Math
Adapter- Code: 0
Typ - Information: Zufallszahl
Kontaktierpos.: 1
Barcode-Mode: 0 = Shared
Aktiv: ✓
Ablauf-Nr.: 1
Testpositionen: 1
EC - Start: 0
EC - Stop: 999
Aktiv: ☐
X - Positions: 1
Y - Positions: 1

Figure 2-3
Create Variants

4. ➡ **Edit Type code** (e.g.:02 Math), ➡ **Type Information** (e.g.: comparison) and activate the variant type with a checkmark by clicking on the empty box below ➡**Active.**


Typen und Varianten bearbeiten
Typliste
Typfamilie
Variante
Baum + Editierseite
Datengitter
Benutzerdefiniertes Layout
Benutzerdefinierte Felder
Default

Familie: 0MATH, (1 Variante/n)
01 VariantMath - Sample shows the m
Familie: 1DATETIME, (1 Variante/n)
01 VariantDateTi - Sample shows the
Familie: 1MATH, (2 Variante/n)
01 Math - Zufallszahl
02 Math - Zahlenvergleich
Familie: 2MEASUREDATA, (1 Variante/n)
01 VariantMeasur - Sample handle me
Familie: 3SCREEN, (1 Variante/n)
01 VariantScreen - Sample shows diffe
Familie: 4CURVENMONITOR, (2 Variante
01 VariantCurve - Sample shows the c
02 VariantCurve - Sample shows the c
Familie: 5DLL, (1 Variante/n)
01 VariantDLL - Sample handle DLL
Familie: 6NETSAMPLES, (1 Variante/n)
01 VariantDotNet - Sample .Net

Familie: 1MATH
Typ - Code: 02 Math
Adapter- Code: 0
Typ - Information: Zahlenvergleich
Kontaktierpos.: 1
Barcode-Mode: 0 = Shared
Aktiv: ✓
Ablauf-Nr.: 1
Testpositionen: 1
EC - Start: 0
EC - Stop: 999
Aktiv: ☐
X - Positions: 1
Y - Positions: 1

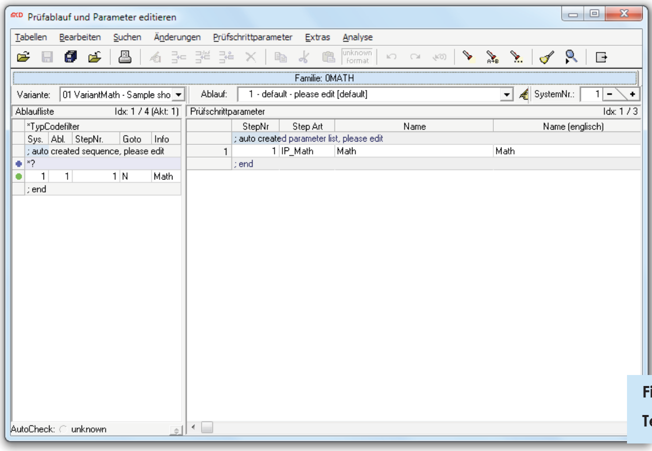
Figure 2-4
Activate Variants

5. Save type list to accept changes ➡ **toolbar** ➡ **Save Type List.**

Other functions that are available in variants are 'copy' and 'rename'. In order to copy a variant, select a given variant in the left window and choose the function ➡ **toolbar ➡ copy variant.** The same instructions apply when wanting to delete a variant. Select the variant in the left window and choose the function ➡ **toolbar ➡ delete variant.** Now a test sequence contains its own family type with two variants.

Test procedure

Tests run according to a set sequence. The test sequence editor in the main window can be selected from the ➡ **toolbar ➡ Test Edit** or by going through the ➡ **main menu ➡ Edit ➡ Test Sequence Edit**. When opening the test sequence editor, the data of the currently loaded family type will be displayed and made available for editing. If the data of another family needs to be edited, select a variant by clicking on the 'Type Select' button in the toolbar or select the family type in the test sequence editor.



Familie: 0MATH

Variante: 01 VariantMath - Sample sho
Ablauf: 1 - default - please edit [default]
SystemNr.: 1

Ablaufliste Idx: 1 / 4 (Akt: 1)

*TypCodefilter				
Sys.	Abl.	StepNr.	Goto	Info
; auto created sequence, please edit				
*?				
1	1	1	N	Math
; end				

AutoCheck: unknown

Prüfschrittparameter Idx: 1 / 3

	StepNr	Step Art	Name	Name (englisch)
	; auto created parameter list, please edit			
1	1	IP_Math	Math	Math
	; end			

Figure 2-5
Test Sequence Editor

The window of the test sequence editor consists of the menu bar, the toolbar, and the actual processing field. If several processes are loaded simultaneously, then the test sequence editor sets a tab for each sequence in between which they can be switched. The tab for each family consists of the activation bar with the selection fields for the variant, the process, and the system number, as well as the sequence list, the parameter inspector and test step list.

Important:
When editing the sequence list or the test steps, note that the changes will affect all variants in the family type, since all these variants share data. This is normally advantageous since the modification is relative to all the variants, and must only be made once. However, unintentional changes could cause another variant to be changed.

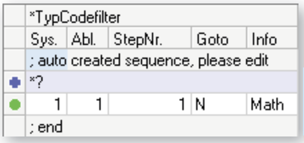
The sequence list and the test step list have an unlimited undo function. Edits can be done and undone up until the last save.

Sequence List

The sequence list determines which steps are executed in a test and in what order. More so, the sequence list can include more than one test sequence and test procedure of more than one variant type. The steps in the sequence list that are to be included in the actual test procedure depend on several filters. These filters are as follows: Type-code of the variant type, sequence number, and system number. A line of the sequence list can contain a comment, a Type code filter, a sequence step or an error. Comments are ignored by the program and could be used to describe the procedures to be used.
A type code filter activates or deactivates subsequent sequence steps. They serve as a criterion for selection, in other words, which variants are included in which sequence steps. Included type code filters begin with an (*) and are labeled with a plus sign in the first column. Excluded type code filters begin with a (^) and are marked with a minus sign.

Ablaufschritte

A test sequence consists of numerous sequence steps.



	*TypCodefilter				
	Sys.	Abl.	StepNr.	Goto	Info
	; auto created sequence, please edit				
✚	*?				
●	1	1	1	N	Math
	; end				

Figure 2-6
Sequence steps

The determined activity of the sequence step is specified in the first column. The **system number** is used for filtering and determines on which system the step should be executed. It must match the system number specified in the general setting. The system number „0“ is used for steps in all running systems.

The **sequence number** allows multiple testing procedures to be stored in one sequence list. The uploading of the sequence is determined by the sequence number stored in the sequence list. Included in the testing process are only those steps that have numbers identical to those in the type list or whose sequence number is 0.
The **test step number** links the sequence list to the actual test steps. If the test step is loaded into the test sequence, then the data of each test step is uploaded and the test step is carried out to its corresponding point.
The **jump type** determines if and when, after running a test step, the test sequence can exit and a jump can be executed.

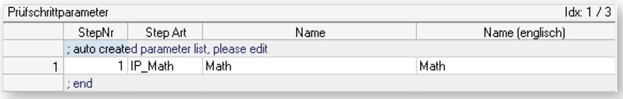
Sprungarten:

N	Never	Never jump, Details for target will be ignored
P	Pass	Jump is good
F	Fail	Jump is bad or invalid
A	Always	Always jump, result irrelevant
C	Calculated	Interpreter step calculates if a jump had occurred

A test step number is assigned to the jump target. The jump type and the jump target are separated by a hyphen and displayed in one column in the sequence list (example: ‚P - 1000‘).

Test steps

How a particular action is performed is defined in the test steps. The steps can be used multiple times in a sequence. A test step consists of several fields, and their test step number creates a link or a shortcut to the sequence list.


Prüfschrittparameter Idx: 1 / 3

	StepNr	Step Art	Name	Name (englisch)
	; auto created parameter list, please edit			
1	1	IP_Math	Math	Math
	; end			

Figure 2-7
Test steps

Fields of test step parameters:

StepNr:	Establishes a connection to the sequence list
StepArt:	Name of the interpreter step
Name/Surname: (German/Englisch)	Description of the test step
Unit:	Unit of measurement value
NCP:	Number of decimal places

Upper/ lower limit:	Boundaries in which the measured value moves
MDE:	Measurement data collection (on/off)
FRepeat:	Number of repetitions of the test step for errors
Frecover:	Debugging, error handling
FContinue:	Determines whether the case will continue in the error checking
Name parameter x/ Value parameter x:	Free usable pair of parameter name and value that is passed on to the interpreter step.

☐ Exercise 2-3
Creating a test sequence

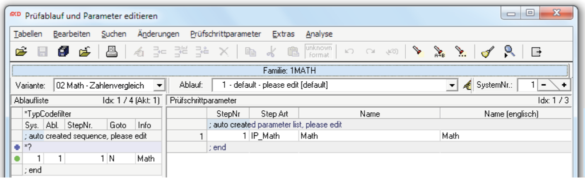
This exercise is to develop a testing procedure for the family type 1MATH (from Exercise 1-1 to 3). This should make it possible for the two variants 01 MATH and 02 MATH to obtain different sequences.

Step by step:

1. Log on as administrator
   Navigate from the toolbar> Password. Press the F4 function key or select Administrator and type in >admin as the password. Confirm the password by pressing Enter or the OK button

2. Navigate using the ➡ **toolbar** ➡ **Type select** ➡ and select the sequence of the family type 1MATH.

3. In order to adapt the sequence navigate using the ➡ **toolbar** ➡ **Test edit** ➡ Editor for test sequence. Add a new test step by selecting the sequence list then navigating to the ➡ **toolbar** ➡ **New test step.**



Familie: 1MATH

Variante: 02 Math - Zahlenvergleich   Ablauf: 1 - default - please edit [default]   SystemNr.: 1

Ablaufliste   Idx: 1 / 4 (Akt: 1)

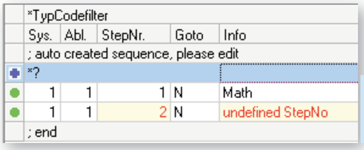
	Sys.	Abl.	StepNr.	Goto	Info
	*TypCodefilter				
	; auto created sequence, please edit				
✚	*?				
●	1	1	1	N	Math
	; end				

Prüfschrittparameter   Idx: 1 / 3

	StepNr	Step Art	Name	Name (englisch)
	; auto created parameter list, please edit			
1	1	IP_Math	Math	Math
	; end			

Figure 2-8
New test step

4. Once the new step appears, assign it as 2 ➡ in the StepNo column. Drag the test step in the sequence list and hold the left mouse button to place where desired.

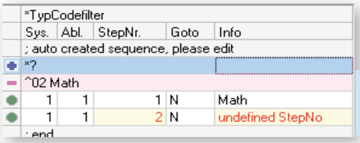


	*TypCodefilter				
	Sys.	Abl.	StepNr.	Goto	Info
	; auto created sequence, please edit				
✚	*?				
●	1	1	1	N	Math
●	1	1	2	N	undefined StepNo
	; end				

Figure 2-9
Step 2

The + above the first test step symbolizes an included variant filter; the * stands for all types. This row should always be listed at the top of the sequence list.

5. Now, through the help of the variant filter the sequence list has to be adjusted so that step number 1 is only called upon in the variant 01MATH and step number 2 is only called upon in variant 02MATH. To add new variant filters select the sequence list in the ➡ **toolbar** ➡ **New variant filter.** This filter should be assigned as (excluded) and is done by inserting ➡ **^02MATH** in the row beneath the + symbol.



	*TypCodefilter				
	Sys.	Abl.	StepNr.	Goto	Info
	; auto created sequence, please edit				
✚	*?				
▬	^02 Math				
●	1	1	1	N	Math
●	1	1	2	N	undefined StepNo
	; end				

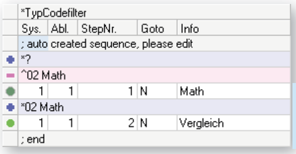
Figure 2-10
New variant filter

Now all the steps listed under the filter for variant 01Math are active.

6. Step number 2 should only be active for the variant type 02MATH. It is possible to use another excluded filter.

Tip:
For family types with several variants due to better readability
on to use inclusive (includierende) filter and the variations in the filter
to count for the following steps should be active.

Insert and then place the container on Variantenfi ➡ **symbol list** ➡ **New Varian tenfi** older. Edit the filter with ➡ * 02 Math



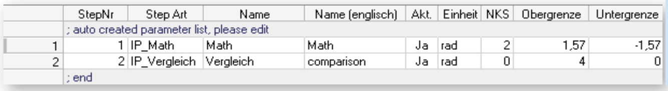
	*TypCodefilter				
	Sys.	Abl.	StepNr.	Goto	Info
	; auto created sequence, please edit				
✚	*?				
▬	^02 Math				
●	1	1	1	N	Math
✚	*02 Math				
●	1	1	2	N	Vergleich
	; end				

Figure 2-11
Variant filter placement

7. Now add a new test step to the test sequence list. Highlight this sequence list via the ➡**toolbar** and then choose ➡ **New test step.**

   Edit the fields of the new test step:

StepNr	➡ **2**
StepType	➡ **IP_Comparison**
Name	➡ **Vergleich**
Name	➡ **Comparison**
Act.	➡ **Yes**
Upper limit	➡ **4**
Lower limit	➡ **0**



	StepNr	Step Art	Name	Name (englisch)	Akt.	Einheit	NKS	Obergrenze	Untergrenze
	; auto created parameter list, please edit								
1	1	IP_Math	Math	Math	Ja	rad	2	1,57	-1,57
2	2	IP_Vergleich	Vergleich	comparison	Ja	rad	0	4	0
	; end								

Bild 2-12 Test parameters

8. Accept the changes by navigating to the ➡**toolbar**➡ **Save all.** Close editor for test sequence.

Interpreter

The interpreter (IP) executes interpreter steps (text files) that can be created and edited by the user. As mentioned in the beginning of the book, the Interpreter language is similar to PASCAL and BASIC. Included are procedures functions and libraries. The data types used are essentially real and string types.

How it works:

Procedure:
A group of related source codes is known as a procedure. This allows repetitive tasks to be edited in an effective, reusable way. This procedure can then be used repeatedly. Procedures do not deliver results or return value.

Functions:
A function is a programming concept similar to the procedure, which delivers returns. Existing functions can be used repeatedly.

Libraries:
A library is basically presents a collection of functions and procedures. If a library is included in a source code, the programmer has access to its functions and procedures.

Otherwise, the programmer is forced to edit the required functions / procedures himself.

Real data types:
The real data type is used for all numeric values (floating point numbers, whole numbers, logical and binary values).

String:
The string is used for strings.

Vector:
In the vector, a data type is implemented, the one-dimensional array of real numbers represents. An array is a composite of several objects, data type, to its various elements can be accessed.

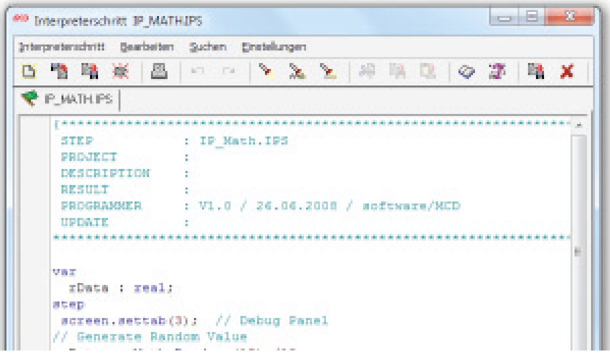
String vector:
Provides a one-dimensional array of strings

ComObject:
The ComObject is a „pseudo-data type" to define a ComObject (ActiveX Object).

NetObjects:
NetObjects is a pseudo-type data to integrate .NET classes in C #, VBB or Java.

Interpreter Editor

The interpreter editor is accessed from the ➡ **toolbar** ➡ **IP Edit** ➡ or via the **menu bar** ➡ **Edit** ➡ **Interpreter steps edit**. The interpreter editor is used to enter the source code manually.



```
{*****************************************************************
 STEP            : IP_Math.IPS
 PROJECT         :
 DESCRIPTION     :
 RESULT          :
 PROGRAMMER      : V1.0 / 26.06.2008 / software/MCD
 UPDATE          :
******************************************************************

var
  rData : real;
step
 screen.settab(3);  // Debug Panel
// Generate Random Value
```

Figure 2-13
Interpreter Editor

nterpreter assistance

The interpreter assistance can be called upon via the function key ➡ **F1**. If you require help for a function or procedure of a specific command, you can select the term and press **CTRL + F1**. This key combination will jump directly to that command if it is contained in the Help folder.

Debugger

The test manager has a debugger for the interpreter, which allows the source code to work through instructions or blocks. The debugger also provides access to data (variables and parameters) and allows it to be edited.

☐ Exercise 2-4
Editing Interpreter Step

This exercise introduces the programming language and is therefore kept very simple. You are now only an exercise away from your first own test run. The only thing lacking is the editing of an interpreter step (IP-Step). In case you have questions about the source or the syntax during this exercise, you can consult the advice of the interpreter assistance.

The first own test sequence should generate a random number between 0 and 9. Furthermore, it should be examined whether this random number is larger or smaller than 4 and then a response will take place accordingly.

Der erste eigene Prüfablauf soll eine Zufallszahl zwischen 0 und 9 generieren. Weiter soll geprüft werden, ob diese Zufallszahl größer oder kleiner als 4 ist und entsprechend soll eine Ausgabe erfolgen.

Step by step:

1. Log on as administrator
   Navigate from the toolbar ➡**Password**. Press the F4 function key or select Administrator and type in ➡admin as the password. Confirm the password by pressing enter or the OK button.

2. Navigate using the ➡ **toolbar** ➡ **Type select** ➡ and select the sequence of the variant **02MATH.**

3. Open the interpreter editor via ➡ **toolbar** ➡ **IP Edit** and select in the interpreter editor window via the toolbar ➡ **a Blank Page document.**

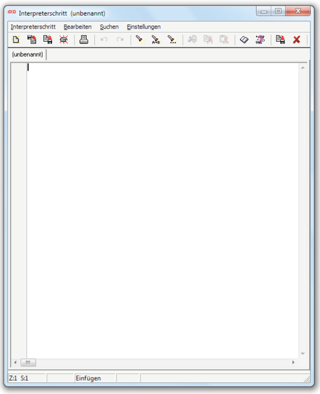


Figure 2-14
Edit IP

Save the step on the ➡ **menu bar** ➡ **Interpreter step** ➡ **Save as.** The name of the interpreter step must be the same as it was specified in the sequence schedule: ➡ **IP_Comparison**. Enter the name and ➡ **Save.**

4. Defining a variable:
Variables are defined before the actual step and fundamentally have the following structure:

Syntax: *IP_COMPARISON.IPS*

```
var
     //Variables are defined here

step
     //Actual source code is edited here
end. //The source code ends here
```

Realize the structure in your blank document and define a variable following the syntax:

A variable consists of an identifier and a type. The identifier defines the name of the variable.

Syntax:

```
Identifier  :  type;
```

In your case the **source code** would be entered as follows:

```
var
  Random Number  : Real;

step

end.
```

You have now defined a variable with the name ‘random number’. Through this name, you have access to the contents of the type ‚real‘ in the source code at anytime.

5. Generate a random number:
To generate a random number use the following function:

```
Math.random  (rRange: real) : real;
```

The name of the function is **‚Math.Random‘** and the value of type ‚real‘, which defines the range of random number, must be entered in brackets. The area should be defined from 0 to 9. The second indication of ‚real‘ gives the type the return value that the function supplies. This random number should be transfer red to the variable. Assignments will be implemented as follows:

For example:

```
Variable :=  value;
```

or in our case:

```
Variable := function ();
```

Source code: *IP_ COMPARISON.IPS*

```
var
  Random Number : Real;

step
  Random Number = Math.random (9);
end.
```

f you feel the need to include any additional information in the source code for traceability purposes, it can be realized through comments. Comments are prece ded by / / and cannot be translated by the compiler.

Source Code:

```
/ / This is a comment!
```

6. Random number comparison:
A comparison can be implemented by a so-called If-Then command (If-Then-Else). The ‚If' sends a logical command that the interpreter responds to. After ‚Then' a code is to be executed if the logical command delivers a ‚true' response, while 'Else' is the code for when the response is a 'fail'. It should be examined whether the random number is greater than, less than or equal to 4 and the result should then be sent to the debug window.

Compound instructions are formed together from a sequence of instructions and are then carried out in the order they were placed.
The partial information contained in the compound instructions is included by the words ‚begin' and ‚end'.

Source: *IP_ COMPARISON.IPS*

```
var
  Random Number : Real;

step
  Random Number = Math.random (9);

 If random number> 4 Then begin
     Debug.Show(0, 'random =' random number);
     Debug.Show(0, 'random number is greater than 4');
 end
 Else begin
     Debug.Show(0, 'random =' random number);
     Debug.Show(0, 'random number is less than or equal to 4');
  end;
end.
```

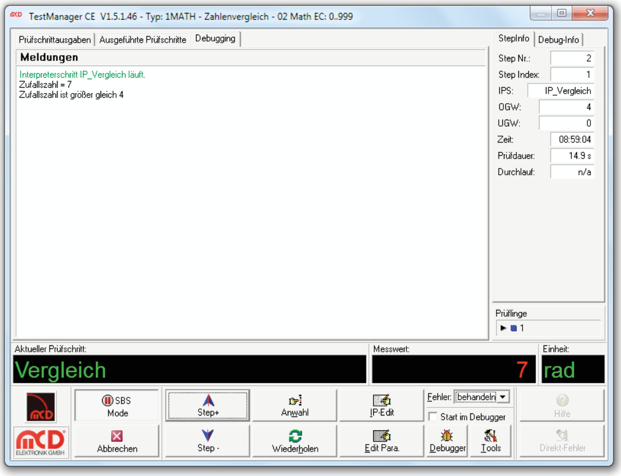
Tip:
If you want to know what the **'Debug.show'** function does and how it is entered, then select the function and press CTRL + F1 (for Help).

7. In this step, the random number in the test sequence window should be dis played. In addition, a short time delay should be inserted.

Source: *IP_ COMPARISON.IPS*

```
var
  Random Number : Real;

step
//**************Generating a random number******************
  Random Number := Math.Random (9);
//***************************************************************

//**************Query of size with issue:***************
  If random number > 4 Then begin
     Debug.Show(0,'random = ',random number);
     Debug.Show(0,'random number is greater than  4');
  end
  else begin
     Debug.Show(0,'random = ',random number);
     Debug.Show(0,'random number is greater than  4');
  end;
//***************************************************************

  SetValue(random number);      //Random number returned as a
  DateTime.Delay(2000);         //measurement value
                                //Time delay of 2 sec
  Repeat                        //This loop will continue re
                                  peatedly
  until StepContinue;           //until the
                                //StepContinue-event occurs
end.
```

8. Save and close the interpreter step and start the process.


TestManager CE V1.5.1.46 - Typ: 1MATH - Zahlenvergleich - 02 Math EC: 0..999

Meldungen

```
Interpreterschritt IP_Vergleich läuft.
Zufallszahl = 7
Zufallszahl ist größer gleich 4
```

Step Nr.:	2
Step Index:	1
IPS:	IP_Vergleich
OGW:	4
UGW:	0
Zeit:	08:59:04
Prüfdauer:	14.9 s
Durchlauf:	n/a

Aktueller Prüfschritt: Vergleich

Messwert: 7

Einheit: rad

Fehler: behandeln

Bild 2-15

Ablauf starten

Due of the limits, a 'Fail' is indicated by the test sequence because of the fact that a random number greater than 4 was generated.

3. Screen objects

Screen displays are initialized via the so-called screen objects. This category includes buttons, bitmaps, curves, dialogues, editing (input fields) frames, labels, meter, etc. .. Such objects can be initialized using the following syntax:

Screen.Objektname, example:

```
Screen.Dialog
```

These objects have procedures that enable them, for example, to activate, conceal, indicate, etc..

Example:

```
Screen.Dialog.Show
```

More information on the syntax can be found via the help of the interpreter.

Before using the screen objects, all objects must be hidden and reset to default value. If you want to present objects, you must first open and show a user-definable window. The following Syntax must be edited prior to use of screen objects:

```
Screen.ClrScr;  //Hide objects and reset to default values
Screen.Show;    //Open and view user definable window
```

☐ Exercise 3-1
Screen-Objekte

This exercise is aimed at dealing with screen objects. Creating a process in which two buttons are displayed on the test window, is the objective of this exercise. These are labeled as ‚Pass‘ and ‚Fail‘. With the help of these buttons, the test results may be affected as follows: When pressing the button reveals a pass, then the test results passed. When it reveals a ‚fail‘, the test results should be adjusted so that ‘fail’ applies.

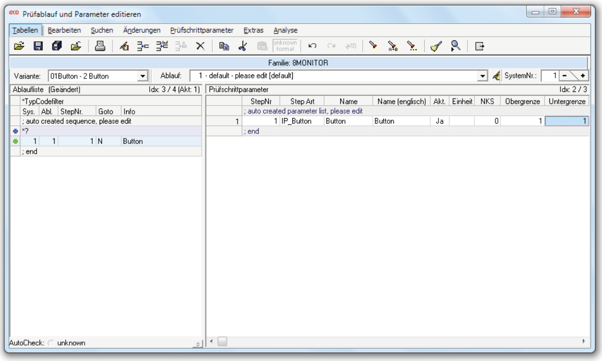
Step by step:

1. Log on as administrator
   Navigate from the ➡ **toolbar** ➡ **Password**. Press the F4 function key or select Administrator and type in >admin as the password. Confirm the password by pressing enter or the OK button.

2. Create a new family type with the name ➡ **8MONITOR** and a variant ➡ **01 Button**. Set it to 'active' and enter a name for the Type information (for example: 2 Button).

3. Select the variant of step 2 in the toolbar TypeSelect.

4. Edit a step in the process and test sequence parameter list. In order to be able to use a new interpreter step, type in a new name in the IP File column (example: IP_Button). Apply the upper and lower limits at a certain value, in order to influ ence the test results



Familie: 8MONITOR

Variante: 01Button - 2 Button   Ablauf: 1 - default - please edit [default]   SystemNr.: 1

Ablaufliste (Geändert)   Idx: 3 / 4 (Akt: 1)

Sys.	Abl.	StepNr.	Goto	Info
*TypCodefilter				
; auto created sequence, please edit				
*?				
1	1	1	N	Button
; end				

Prüfschrittparameter   Idx: 2 / 3

	StepNr	Step Art	Name	Name (englisch)	Akt.	Einheit	NKS	Obergrenze	Untergrenze
	; auto created parameter list, please edit								
1	1	IP_Button	Button	Button	Ja		0	1	1
	; end								

Figure 3-1
Screen objects

5. Edit interpreter step:
Edit your source code by defining a variable that can give you the value of the last pressed buttons and then initiate the monitor. Define yourself in a 'repeat until' loop of the two buttons and retrieve the information of which button was pressed. The loop can only then be abandoned when one of the buttons has been activated.
Depending on the button, customize the measured value of the process (‚Pass' or ‚Fail') in order to influence the result.

Source code: *IP_BUTTON.IPS*

```
var
  Button : Real;
step
//*****************Initialisierung Screen:**********************
  Screen.SetTab(1);
  Screen.ClrScr;
  Screen.Show;
//**************************************************************
```

➡ *weiter geht´s auf der nächsten Seite*

```
  //*********Loop to repeat with value adjustment:*******
 repeat
  Screen.Button.Show( 1, 100, 100, 'Pass', 100, 1, 24);
  Screen.Button.Show( 2, 300, 100, 'Fail', 100, 1, 24);

  Button := Screen.LastButton;

  If Button = 1 Then
  begin
     SetValue(1);
  end;

  If Button = 2 Then
  begin
     SetValue(100);
  end;

  GlobalVar.Set ('gButton', Button);

 until Button > 0;
//*************************************************************

 repeat

  DateTime.Delay(300);

 until StepContinue;

 Screen.ClrScr;
end.
```

The loop repeat...until StepContinue is allowed in the Step by Step – mode, so that the source code of the loop is repeated until the jump command to the next step (via the Step + button) takes place.

4. Global variables

Variables that are defined and initialized in a step are lost at the end of the step. Howe-ver, it is necessary in many cases that variables and their values of one or more steps be retained until the end of the sequence. Global variables have this property. Your life goes beyond the Step out and end up can be defined to the system end.
These global variables can be defined by the procedure ➡ **GlobalVar.Set** and retrieved by the function ➡ **GlobalVar.Get.** They must also have a clear identifier, like normal variables.
Variables, values and comments can be viewed and edited via the ➡ **menu bar** ➡ **Tools** ➡**Global Data**.

Converting data types

The test manager provides functions to convert data types. These are necessary (for example) so that return values from functions that are in a fixed data type can be con-verted. The **Val** function converts the text from strings into a real data type (a numerical value). It is also possible to convert a real value to a string via the **Str** function.

Programming made easy

It is recommended that when editing Interpreter steps, the variant for which the Interpreter step is edited, should be selected and carried out in the SBS-mode. If an IP-Step is not already created with the appropriate name, the interpreter editor will be opened by a pathway where the IP step is located in. The source code can be edited directly in this window. When the editor is closed the code is directly executed. In the case of an error, the programmer displays it directly in the editor. The error line in the interpreter editor is displayed in red and the status bar displays the error code description.
It is possible that the error is already in the source code, this is for example the case when using an undefined variable. The error can be solved instantly that way and the improved source code can be rerun. That way, the accuracy of the code and the behavior of the program can be reviewed. The source code can be edited and verified by programmer-friendly steps.

Note:
Another peculiarity of the interpreter is that some errors are first detected at run-time. The syntax in the editor is checked as 'OK'. In an actual start of the test step, the functions are in fact executed and type similarity is controlled.

During the exercises you will be given step by step functions and procedures that you have probably not been confronted by before. Should the syntax or the effects of its commands be unclear, see the Interpreter help for advice. This should be your standard operating procedure as it enables you to learn how to operate the TestManager and create interpreter steps independently.

☐ Exercise 4-1
Gambling

In this exercise, a new sequence (Gambling) is created. This will consist of 3 test steps with 4 user-entered digits (0-9) compared with 4 program-generated random numbers. The result is expressed as a percentage and will reflect the similarities of the numbers.

The names of the test steps are:
- Entry of Numbers
- Random
- Compare

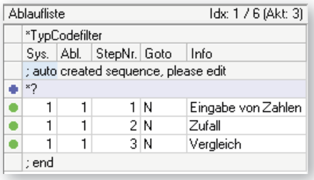
Mit den zugehörigen Interpreterschritten:
- IP_Entry
- IP_Random
- IP_Compare

Step by step:

1. Log on as administrator
   Navigate from the ➡ **toolbar** ➡ **Password.** Press the F4 function key or select Administrator and type in admin as the password. Confirm the password by pressing enter or the OK button.

2. Insert a new family type with the name ➡ **7GAMBLING** with the variant> 01 Random number, and set this to ➡ **active** and save.

3. Edit the test procedure with the above-mentioned three steps and define me aningful measurement limits. Note that the result should be displayed as a percentage.



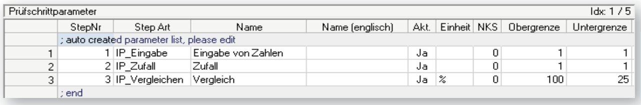
Ablaufliste Idx: 1 / 6 (Akt: 3)

*TypCodefilter

	Sys.	Abl.	StepNr.	Goto	Info
	; auto created sequence, please edit				
✚	*?				
●	1	1	1	N	Eingabe von Zahlen
●	1	1	2	N	Zufall
●	1	1	3	N	Vergleich
	; end				

Figure 4-1

Sequence List



Prüfschrittparameter Idx: 1 / 5

	StepNr	Step Art	Name	Name (englisch)	Akt.	Einheit	NKS	Obergrenze	Untergrenze
	; auto created parameter list, please edit								
1	1	IP_Eingabe	Eingabe von Zahlen		Ja		0	1	1
2	2	IP_Zufall	Zufall		Ja		0	1	1
3	3	IP_Vergleichen	Vergleich		Ja	%	0	100	25
	; end								

Figure 4-2

Test Step List

4. Choose the variant listed in step 2 via 'Typeselect' or for shortcut press (F2)

5. Editing the first interpreter step:
   Define the variables of the real-type, by allocating and assigning the numbers that the user entered. Another real-type variable will function as a counter and must therefore also be defined.

6. Assign an initial value to the counter (0 is offered) and initialize the screen.

7. Create an entry window on the test step display window and activate it.

8. Edit a 'repeat until' loop that, with the help of the counter, repeats itself so often until the value reaches an initial value of + 4. The purpose of this loop is that its content is repeated until the counter reaches its final value. Inside the loop, the four user-numbers are read and passed on to the variable.

9. By using the help of an If-Then instruction, ask whether or not a valid character was entered into the input field during the 'repeat until' loop. If this is the case, the display button can be confirmed and activated.

10. Define another If-Then assignment inside the loop in order to check if a button is activated. During this assignment, you can use the counter as a selector, with the help of a case instruction, in order to assign each number to the variables. Then, reactivate the input field and empty it. The button will be hidden again. It is important that you increment the counter.

11. Define the numbers entered as global variables and set the measurement value.

Source code: *IP_ENTRY.IPS*

```
var
  Number1      : Real;
  Number2      : Real;
  Number3      : Real;
  Number4      : Real;
  Counter      : Real;

step
//****************Screen initialize:**********************
  Screen.SetTab(1);       //focus on Prüfschrittausgabe
  Screen.ClrScr;
  Screen.Show;
//***************************************************************

//********create and activate an entry window:***********
  Screen.Edit.Setup (1, 1, '0..9', 0);    //input window
  Screen.Edit.Show ( 1, 100, 100, 50);
  Screen.Edit.Activate (1);
//***************************************************************

//******************Set counter to 0:********************
  Counter := 0;
//***************************************************************

 repeat      //repeat until counter is 4!

  Screen.Label.Show (1, 100, 30, ''Please enter a number:');

  If Screen.Edit.GetText( 1 ) <> '' Then //Query whether
                                          character
                                          was entered
  begin
     Screen.Button.Show( 1, 200, 100, 'Ok', 100, 1, 24);
     Screen.Button.Activate (1);
  end;

  If Screen.LastButton =1 Then  //If the button
                                  is clicked:
  begin

//*********Selection according to counter reading:***********
     Case counter Of
       0 : number1 := Val (Screen.Edit.GetText( 1 ),100);
       1 : number2 := Val (Screen.Edit.GetText( 1 ),100);
       2 : Number3 := Val (Screen.Edit.GetText( 1 ),100);
       3 : Number4 := Val (Screen.Edit.GetText( 1 ),100);
     end;
//***************************************************************

//************Increment counter and hide button:*********
     Screen.Edit.Activate (1);
     Screen.Edit.SetText (1, '');
     Screen.Button.Hide (1);
     Zaehler := Counter + 1;
```

➡ *continue to the next page*

```
//*****************************************************************
  end;

 until (counter = 4);

//**************Label and hide input field:***************
 Screen.Label.Hide (1);
 Screen.Edit.Hide(1);
//***************************************************************

//****************Define Global Variables:***************
 GlobalVar.Set ('gnumber1', number1);//Save figures in
                                       global variables
 GlobalVar.Set ('gnumber2', number2);
 GlobalVar.Set ('gnumber3', number3);
 GlobalVar.Set ('gnumber4', number4);
//***************************************************************
 SetValue(1);        //set measurement value

 repeat
 until StepContinue;

 Screen.ClrScr;      //Clear screen
end.
```

12. IP_Random edit:
    Generate 4 random numbers and store them in the Global Variables.

Source code: *IP_RANDOM.IPS*

```
var
  Random Number 1 : Real;
  Random Number 2 : Real;
  Random Number 3 : Real;
  Random Number 4 : Real;
step
//***************Zufallszahlen generieren:********************
  Random Number 1 := Math.Random (10);
  Random Number 2 := Math.Random (10);
  Random Number 3 := Math.Random (10);
  Random Number 4 := Math.Random (10);
//***************************************************************

//**************Globale Variablen setzen:*********************
  GlobalVar.Set ('gRandom Number 1', Random Number 1);
  GlobalVar.Set ('gRandom Number 2', Random Number 2);
  GlobalVar.Set ('gRandom Number 3', Random Number 3);
  GlobalVar.Set ('gRandom Number 4', Random Number 4);
//***************************************************************

  SetValue (1);
```

➡ *continue to the next page*

```
    repeat
    until StepContinue;
  end.
```

Tip:
Be careful when naming global and normal variables so that no misunderstandings occur and the syntax is understandable.

13. Edit IP_Comparison:
    With the help of If-Then instructions, check whether there is consensus between the inserted numbers and the random numbers. If necessary, adjust the result accordingly within the instructions. Enter the numbers for the user on the test step display window.

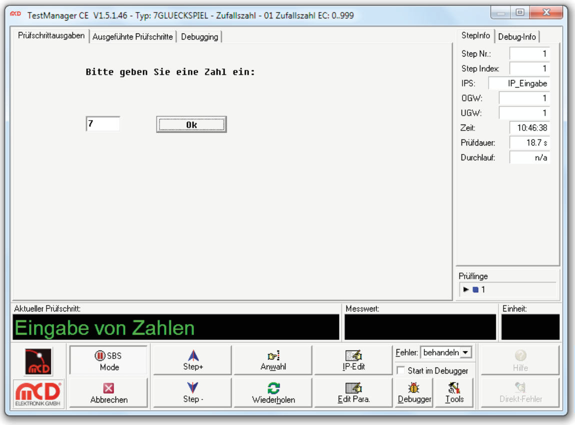
Source code: *IP_Comparison.IPS*

```
var
  Result : Real;

step
  Result :=  0;

//*********Compare figures and adjust results:*********
  If GlobalVar.Get ('gnumber1') = GlobalVar.Get
 ('gRandom Number1'1')
  Then
  begin
     Ergebnis := Ergebnis + 25;
  end;

  If GlobalVar.Get ('gZahl2') = GlobalVar.Get ('gZufallszahl2')
  Then
  begin
     Ergebnis := Ergebnis + 25;
  end;

  If GlobalVar.Get ('gZahl3') = GlobalVar.Get ('gZufallszahl3')
  Then
  begin
     Ergebnis := Ergebnis + 25;
  end;

  If GlobalVar.Get ('gZahl4') = GlobalVar.Get ('gZufallszahl4')
  Then
  begin
     Ergebnis := Ergebnis + 25;
  end;
//************************************************************
     Screen.SetTab (1);
     Screen.ClrScr;
     Screen.Show;
```

➡ *continue to the next page*

```
//****************Issue of numbers :*************************
     Screen.Label.Show (1, 100,  30, ''Your Numbers:');
     Screen.Label.Show (2, 100,  60, str (GlobalVar.Get
     ('gNumber')));
     Screen.Label.Show (3, 100,  90, str (GlobalVar.Get
     ('gNumber2')));
     Screen.Label.Show (4, 100, 120, str (GlobalVar.Get
     ('gNumber3')));
     Screen.Label.Show (5, 100, 150, str (GlobalVar.Get
     ('gNumber4')));

     Screen.Label.Show (6, 250, 30, 'random numbers':');
     Screen.Label.Show (7, 250, 60, str (GlobalVar.Get
     ('gRandom number'1')));
     Screen.Label.Show (8, 250, 90, str (GlobalVar.Get
     ('gRandom number2')));
     Screen.Label.Show (9, 250, 120, str (GlobalVar.Get
     ('gRandom number3')));
     Screen.Label.Show (10, 250, 150, str (GlobalVar.Get
     ('gRandom number4')));
//*************************************************************

     Screen.SetTab (1);
     SetValue (result);

     repeat
       DateTime.Delay (2000);
     until StepContinue;

     Screen.ClrScr;
end.
```

14. Save your edited data and restart the process.


TestManager CE V1.5.1.46 - Typ: 7GLUECKSPIEL - Zufallszahl - 01 Zufallszahl EC: 0..999

Bitte geben Sie eine Zahl ein:

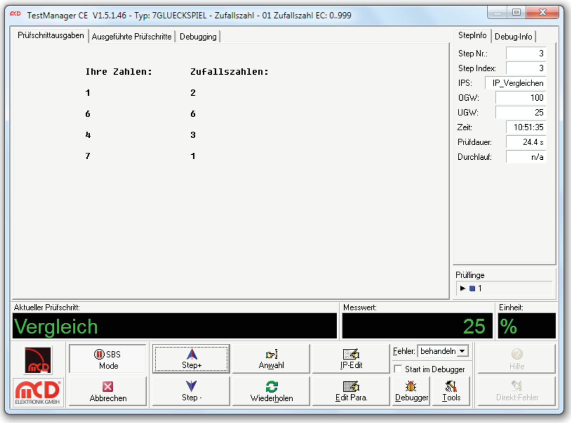
7 Ok

Step Nr.: 1
Step Index: 1
IPS: IP_Eingabe
OGW: 1
UGW: 1
Zeit: 10:46:38
Prüfdauer: 18.7 s
Durchlauf: n/a

Aktueller Prüfschritt: Eingabe von Zahlen

Fehler: behandeln

Figure 4-3
Enter numbers


TestManager CE V1.5.1.46 - Typ: 7GLUECKSPIEL - Zufallszahl - 01 Zufallszahl EC: 0..999

```
Ihre Zahlen:        Zufallszahlen:
1                   2
6                   6
4                   3
7                   1
```

Step Nr.: 3
Step Index: 3
IPS: IP_Vergleichen
OGW: 100
UGW: 25
Zeit: 10:51:35
Prüfdauer: 24.4 s
Durchlauf: n/a

Aktueller Prüfschritt: Vergleich
Messwert: 25
Einheit: %

Fehler: behandeln
Start im Debugger

Bild 4-4
Comparison

5. Tools

As mentioned in the beginning of the book, TestManager contains a variety of functions and tools. By navigating to the ➡ **menu bar** ➡**Tools** you can invoke the activated tools in the basic settings. These tools enable the user to obtain information and carry out settings and can also be used outside of the sequences.
Under the tools menu, you will find various monitors, such as CAN-, LIN- and IC-monitors and various ME-(Meilhaus) monitors. Signals are sent and received with these monitors. Setting possibilities of all kinds are also possible. These tools can also be called upon during sequences via the function ➡ **Action.Trigger**.

Exercise 5-1
Using Tools

This exercise deals with the application of tools. As an example, this exercise will illustrate how a curve monitor is used. Data for three curves should be passed on to the monitor. The mathematical functions sine, cosine and sine*cosine should be the result displayed on the curve monitor.

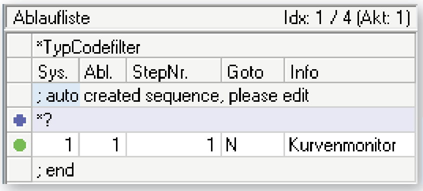
Step by step:

1. Log on as administrator
   Navigate from the ➡**toolbar** ➡ **Password**. Press the F4 function key or select Administrator and type in >admin as the password. Confirm the password by pressing enter or the OK button.

2. Insert a new family type with the name ➡ **9CURVEMONITOR** with the variant ➡ **01 SineCosine**, and set this variant to ➡ **active** and save.

3. Select the created family type via ➡ **Typeselect**.

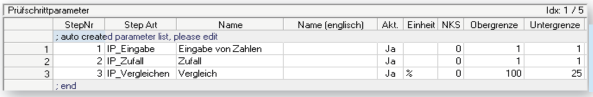
4. Test procedure:
   Edit a test sequence with a test step ➡ **IP_SINUS**, define upper and lower limits and set the step to active.



Ablaufliste   Idx: 1 / 4 (Akt: 1)

	*TypCodefilter				
	Sys.	Abl.	StepNr.	Goto	Info
	; auto created sequence, please edit				
✚	*?				
●	1	1	1	N	Kurvenmonitor
	; end				

Figure 5-1
Test Sequence List



Prüfschrittparameter Idx: 1 / 5

	StepNr	Step Art	Name	Name (englisch)	Akt.	Einheit	NKS	Obergrenze	Untergrenze
	; auto created parameter list, please edit								
1	1	IP_Eingabe	Eingabe von Zahlen		Ja		0	1	1
2	2	IP_Zufall	Zufall		Ja		0	1	1
3	3	IP_Vergleichen	Vergleich		Ja	%	0	100	25
	; end								

Bild 5-2

Test-Step Parameter

5. Interpreter step:
   Set up three variables of type **Vector**. These vectors serve as the function values of sine, cosine and sine*cosine. Furthermore, one variable, which should be of real-type, is needed for counting. This should enable the function values in single cells of the vector to be written (similar to the known 'data-type array').

6. Before the cells of the vectors are described, it should be ensured that all cells have no content. That way these vectors have a ‚length' of 0. This happens through the syntax:
   Example: Curve1: = [ ];

7. Create a loop that enables you to describe the cells of the vector individually. For this purpose, a For-To-Do loop is available, that enables the vectors to be described with a variety of values. Make sure that one cell of the vector is , Switched on' after each cycle.

   Source code: *IP_SINUS.IPS*

   ```
   For Counter := 1 To 1000 Step 1 Do begin
         Curve := Curve + [(Math.Sin( counter / 100 )) ];
   end;
   ```

   `[(Math.Sin (counter / 100))]` describes the current cell with the sine value from the counter / 100. The brackets [ ] represent the beginning and the end of the cell. With the syntax Curve1:= Curve1 + [ ]; the vector is overwritten by itself and an additional (new) cell [ ]. The length of the vector is thus increased by one element.

   Repeat this approach of steps 6 and 7 for the cosine and sine*cosine variants.

8. Settings of the curve monitor:
   The axes of the curve monitor can be named with the procedure ➡ **Curve.Scale. SetText.** The axis name for the Volt of the Y-axis and measuring points for the x-axis are available. You can insert the curve names into the curve monitor with the procedure ➡ **Curve.Name** and configure the monitor via ➡ **Curve.Setup.**

9. View of the curves:
The curves can be stated in the curve monitor with the help of the function ➡Action.Trigger. For this purpose, the ➡ **Action-Code** of the curve monitor (2020) is needed. Set the measurement value according to the value which you defined.

Quellcode: *IP_SINUS.IPS*

```
var
  Curve1   : Vector;
  Curve2   : Vector;
  Curve3   : Vector;
  Counter : Real;

step

//***************Fill vectors with values:********************
  Curve1 :=[];
  For Counter := 1 To 1000 Step 1 Do begin
     Curve1 := Curve1 + [(Math.Sin( counter  / 100 )) ];
  end;

  Curve2 :=[];
  For counter  := 1 To 1000 Step 1 Do begin
     Curve2 := Curve2 + [Math.Cos ( counter  / 100)];
  end;

  Curve3 :=[];
  For counter  := 1 To 1000 Step 1 Do begin
     Curve3 := Curve3 + ([Math.Sin ( counter  / 100)*Math.Cos
               (counter  / 100)]);
  end;
//*************************************************************

//***********Axis labels and graph names define:****************
  Curve.Scale.SetText ( 1, 1, 'volts' );
  Curve.Scale.SetText ( 1, 0, 'data points');

  Curve.Name( 1, 'curve 1', 'curve 2','curve 3');
//*************************************************************

//***************Edition of the curves:***********************
  Curve.Setup ( 1, 1000, curve1, curve2, curve3);
  Action.Trigger ( 2020);
//*************************************************************

  SetValue (1);

  repeat
  until StepContinue;

end.
```

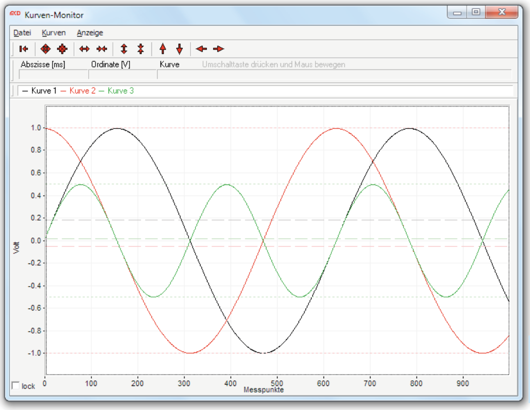


Bild 5-3

Output of Curve Monitor

6. Files

The TestManager provides the ability to create, access, and edit files. Different functions and procedures can be called upon, for example, the reading and writing in files. To activate these functions the following syntax must be used:

```
➡ File.(Function or procedure)
```

Important for the functions and procedures for files is to note whether or not the file should be opened or closed for execution. For example, a file must be opened before it is read or edited.

☐ Exercise 6-1
Reading and Writing Files

This exercise focuses on reading and writing files. This sequence should have a user-entered text placed into a text document (Ending.txt) and then read out again. The text should be displayed in the test step display window.

Step by step:

1. Log on as administrator Navigate from the ➡ **toolbar** ➡ **Password**. Press the F4 function key or select Administrator and type in >admin

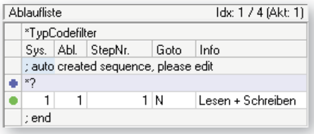
2. Create a new family type with a variant and set it to active.

Example:	Family Type Name:	11PRACTICE9
	Code Type:	01Txt file
	Information Type:	Read and Write

   Save the edited data.

3. Choose the variant of the new family type in step 2 via ➡ **Typeselect**.

4. Edit the sequence list and the test step parameter via ➡ **Test Edit**. One step is required for the sequence. Choose a name for the Step Type (example: IP_TXTFILE) and define the upper and lower limits, then save your data.



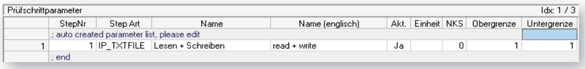
Ablaufliste Idx: 1 / 4 (Akt: 1)

*TypCodefilter

	Sys.	Abl.	StepNr.	Goto	Info
	; auto created sequence, please edit				
✚	*?				
●	1	1	1	N	Lesen + Schreiben
	; end				

Figure 6-1

Sequence List



Prüfschrittparameter Idx: 1 / 3

	StepNr	Step Art	Name	Name (englisch)	Akt.	Einheit	NKS	Obergrenze	Untergrenze
	; auto created parameter list, please edit								
1	1	IP_TXTFILE	Lesen + Schreiben	read + write	Ja		0	1	1
	; end								

Figure 6-2
Test Step Parameter

5. Start the sequence and edit the interpreter step. Use an input box, and for confirmation of the input, a button which allows the entered text in a variable to be passed. Save the text into a variable.

6. With the procedure ➡ **File.SetFileName** ( ) a file can be created. Name the file ➡ **‚Practice9.txt‘** and then open the file with the help of the procedure ➡ **File.Open** ( ).

Syntax:

```
File.SetFileName (1, 'Übung9.txt');
File.Open (1);
```

7. Write the text in the file with the help of your created variables.

Syntax:

```
File.Write (1, Variable);
```

8. In order to read the text, you have to position the file pointer back to the beginning of the file. The file pointer can be set at any location of the file via the procedure ➡ **File.Seek** ( ). The content of the file can be read via the function ➡ **File. Read** ( ) and can be passed directly to a variable. After you have read the file, you should close it again via the command ➡ **File.Close** ( ).

Syntax:

```
File.Seek (1,0);      //set file pointer to the beginning
     Content := File.Read (1, 'EOF',100);
     File.Close (1);
```

9. Enter the contents of the files in the test step display window and assign a lag period of approximately three seconds to allow the user to have time to register the output.

10. Delete the file with ➡ **File.Delete** and set the measured value within the boundaries you have defined.

Syntax:

```
        File.Delete (1);      //Delete the file!!!
```

The file should be deleted so that no implications arise when other applications and the input of shorter texts is applied

Tip:
Simply try it out, with and without deleting the file.

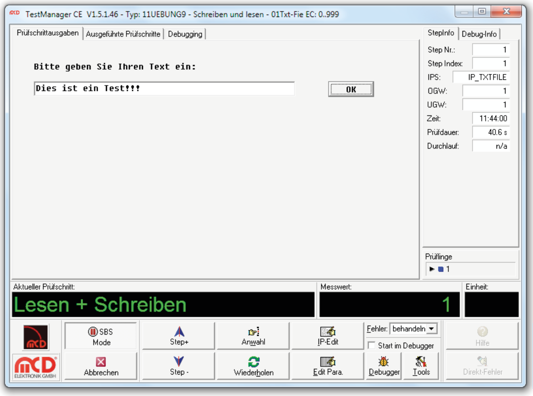
Source code: *IP_TXTFILE.IPS*

```
var
  Text     : String;
  Contents: String;

step
//**********************Screen initialize:*********************
  Screen.SetTab (1);
  Screen.ClrScr;
  Screen.Show;
//****************************************************************

//*****************Generate output and input field:*********
  Screen.Label.Show (1, 30, 30, ''Please enter your text:');
  Screen.Edit.Setup (1, 100,'' ,0);
  Screen.Edit.Show (1, 30, 60, 400);
  Screen.Edit.Activate (1);
//****************************************************************

//***********Repeat until button is pressed:*******************
  repeat
     If Screen.Edit.GetText( 1 ) <> ,' Then
     begin
       Screen.Button.Show (1, 480, 60, 'OK',70 ,1 , 24);
     end;
  until Screen.LastButton =1;
//****************************************************************

 //**************Write text in file:*****************************
  Text := Screen.Edit.GetText (1);

  File.SetFileName (1, 'Practice9.txt');
  File.Open (1);
  File.Write (1, text);
```

➡ *continue to the next page*

```
   File.Seek (1,0);      //set file pointer at the beginning
   Content := File.Read (1, 'EOF',100);
   File.Close (1);
//*************************************************************

//*****************Output:************************************
   Screen.Label.Show (2, 30, 200, 'Text aus File:');
   Screen.Label.Show (3, 30, 230, Inhalt);
//*************************************************************

   DateTime.Delay (3000);

   repeat
   until StepContinue;

   File.Delete (1);      //Delete the file!!!!
   SetValue (1);

end.
```

11. Save the interpreter step and restart the sequence in SBSModus. If you edited the deletion of the file after the ‚repeat until StepContinue‘ Loop, you can still see the text document after you have entered your text and confirmed. This requires you to navigate to Windows Explorer into the ➡ **Directory** where the TestManager was installed. The file is located in the folder ➡ **Data Type** ➡ **Type_** (**family type** name that you assigned). That way you can confirm if the letter was successfully transferred into the file.


TestManager CE V1.5.1.46 - Typ: 11UEBUNG9 - Schreiben und lesen - 01Txt-Fie EC: 0..999

Bitte geben Sie Ihren Text ein:

Dies ist ein Test!!!

Step Nr.:	1
Step Index:	1
IPS:	IP_TXTFILE
OGW:	1
UGW:	1
Zeit:	11:44:00
Prüfdauer:	40.6 s
Durchlauf:	n/a

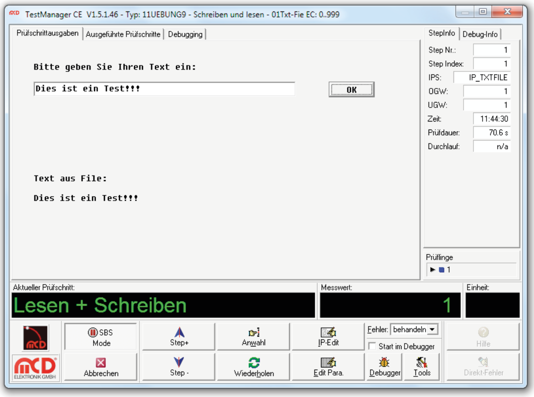
Aktueller Prüfschritt: Lesen + Schreiben

Messwert: 1

Einheit:

Fehler: behandeln

Figure 6-3
Read + Write


TestManager CE V1.5.1.46 - Typ: 11UEBUNG9 - Schreiben und lesen - 01Txt-Fie EC: 0..999

```
Bitte geben Sie Ihren Text ein:

Dies ist ein Test!!!

Text aus File:

Dies ist ein Test!!!
```

Step Nr.:	1
Step Index:	1
IPS:	IP_TXTFILE
OGW:	1
UGW:	1
Zeit:	11:44:30
Prüfdauer:	70.6 s
Durchlauf:	n/a

Aktueller Prüfschritt: Lesen + Schreiben
Messwert: 1
Einheit:
Fehler: behandeln

Figure 6-4
Read + Write

☐ **Exercise 6-2**
Writing, reading, renaming of files and introduction of the date-time functions.

This exercise shows another example in dealing with files. The task will be to create a sequence in which the user will be prompted to enter a user name. This username should be documented with the current date and time of entry.
In the second step, both the current user and previous user should be documented in a second text and saved with the current date and time.
In the third step, the text document with the previous user should be overwritten by the text documents with the current user. This can be realized the by renaming the files.

Step by step:

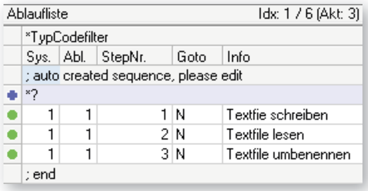
1. Log on as administrator
   Navigate from the toolbar Password. Press the F4 function key or select Administrator and type in admin

2. Create a new family type with a variant, set it to active, and save the edited data.

Example:	Family Type:	12PRACTICE10
	Code Type:	01File
	Information Type:	txt file

3. Choose the variant of the new family type in step 2 via ➡ **Typeselect**.

4. Create three steps in the sequence list and test step parameter list. Use three new step types, define names and limits for these steps, and save your data.

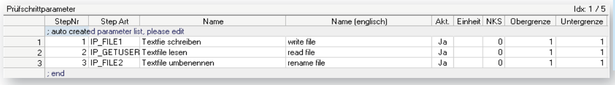


Ablaufliste — Idx: 1 / 6 (Akt: 3)

*TypCodefilter

	Sys.	Abl.	StepNr.	Goto	Info
	; auto created sequence, please edit				
	*?				
●	1	1	1	N	Textfie schreiben
●	1	1	2	N	Textfile lesen
●	1	1	3	N	Textfile umbenennen
	; end				

Figure 6-5
Sequence List



Prüfschrittparameter — Idx: 1 / 5

	StepNr	Step Art	Name	Name (englisch)	Akt.	Einheit	NKS	Obergrenze	Untergrenze
	; auto created parameter list, please edit								
1	1	IP_FILE1	Textfie schreiben	write file	Ja		0	1	1
2	2	IP_GETUSER	Textfile lesen	read file	Ja		0	1	1
3	3	IP_FILE2	Textfile umbenennen	rename file	Ja		0	1	1
	; end								

Figure 6-6
Test Step Parameters

5. Choose the variant of the new family type in step 3 via ➡ **Typeselect** and begin the sequence.

6. Edit the first interpreter step:
Insert variables of type Real in order to save the date (year, month and day) and time (hours, minutes and seconds). A variable of type String is also required in order to deposit the user‘s name to.

7. Create an input field in the test step display window and activate it. Call upon type label in this window to enter the username.

8. Place a button, with which an input can be confirmed, as soon as a character has been entered.

9. Save the user name in the applied variables.

10. Read the current system time with the function ➡ **DateTime.Time** ( ). With this function, the return value can be transferred directly to the specified variables.

Syntax:

```
DateTime.Time (hours, minutes, seconds);
```

In this case, hours, minutes and seconds are the variables that the current time will be assigned to.

11. 11. The function ➡ **DateTime.Date** ( ) delivers the current system date and transfers them to the variables (as in Step 9).

Syntax:

```
DateTime.Date ( year, month, day);
```

12. Now open the saved text document in which the current user should be stored.

13. With help of the procedure ➡ **File.WriteLn** ( ) you can write information in the text document line by line. With the help of this procedure write the user, the date and the time each in one line.

Syntax:

```
File.WriteLn (1, 'User:  ' + Anwender);
File.WriteLn (1, 'Date:  ' + Str (Jahr) + '-' + Str (Monat) +
'-'+Str (Tag));
File.WriteLn (1, 'Time:  '+Str(Stunden) + ':' + Str (Minuten) +
':' + Str (Sekunden));
```

14. Close the text document and set the measurement value.

Quellcode des ersten Interpreterschritts: *IP_FILES.IPS*

```
var
  User         : String;
  Hours        : Real;
  Minutes      : Real;
  Seconds      : Real;
  Year         : Real;
  Month        : Real;
  Day          : Real;

step
//**************Initialize screen:*****'''*******************
  Screen.SetTab (1);
  Screen.ClrScr;
  Screen.Show;
//**********************************************************

//***********Issue the user request:*************************
  Screen.Edit.Setup (1, 20,'' ,0);
  Screen.Edit.Show (1, 200, 100, 150);
  Screen.Edit.Activate (1);
//**********************************************************

//************Benutzeraufforderung ausgeben:*****************
  Screen.Label.Show (1, 20, 105, ''Enter user name:');
```

➡ continue to the next page

```
//***************************************************************

//************Repeat until button is pressed:***********
  repeat
     If Screen.Edit.GetText( 1 ) <> ,“ Then
     begin
       Screen.Button.Show (1, 380, 100, 'OK',70 ,1 , 24);
     end;
  until Screen.LastButton  = 1;
//***************************************************************

//**************Retrieve text, date and time:******************
  User:= Screen.Edit.GetText (1);

  DateTime.Time ( Hours, Minutes, Seconds);
  DateTime.Date ( Year, Month, Day);
//***************************************************************

//***************Write into File line by line****************
  File.SetFileName (1, 'User_Current.txt’);
  File.Open (1);

  File.WriteLn (1, 'User:  ' + User);
  File.WriteLn (1, 'Date:  ' + Str (Year) + '-' + Str (Month) +
  '-' + Str (Day));
  File.WriteLn (1, 'Time:  ' + Str (Hours) + ':' + Str
  (Minutes) + ':' + Str (Seconds));

  File.Close (1);

//***************************************************************

  SetValue (1);

  repeat
  until StepContinue;

end.
```

15. Edit the second interpreter step:
    Create three variables of type String, in order to be able to allocate the read user data.

16. Open the text document of the current user and enter the data into the variables.

Syntax:

```
Userdaten1 := File.ReadLn (1);
Userdaten2 := File.ReadLn (1);
Userdaten3 := File.ReadLn (1);
```

17. Close the document and enter the user data in the test step display window.

18. Open the document from the previous user, interpret the data line by line, and place it in the variables provided.

19. Close the document and submit the data. Enter the measurement value and a delay time of approximately 2-4 seconds to give the user time to read the given results.

20. Clear the screen.

Source code of the second Interpreter step: *IP_GETUSER.IPS*

```
var
  Userdata1 : String;
  Userdata2 : String;
  Userdata3 : String;

step
//******************File öffnen:********************************
  File.SetFileName (1, 'User_current.txt’);
  File.Open (1);
//*************************************************************

//****************Read line by line:***************************
  Userdata1 := File.ReadLn (1);
  Userdata2 := File.ReadLn (1);
  Userdata3 := File.ReadLn (1);
//*************************************************************

//******************Close file:*********************************
  File.Close (1);
//*************************************************************

//******************Initialize screen:*********************
  Screen.SetTab (1);
  Screen.ClrScr;
  Screen.Show;
//*************************************************************

//******************Output data:********************************
  Screen.Label.Show (4, 100, 40,  'Current users:');
  Screen.Label.Show (1, 100, 70,   Userdata1 );
  Screen.Label.Show (2, 100, 100,  Userdata2 );
  Screen.Label.Show (3, 100, 130,  Userdata3 );
//*************************************************************

//******************File Open:*********************************
  File.SetFileName (1, 'User_before.txt');
  File.Open (1);
//*************************************************************
```

➧ *continue to the next page*

```
//**************Read line by line:*************************
  Userdaten1 := File.ReadLn (1);
  Userdaten2 := File.ReadLn (1);
  Userdaten3 := File.ReadLn (1);
//***************************************************************

//*****************Close file:*****************************
  File.Close (1);
//***************************************************************

//*****************Output data:******************************
  Screen.Label.Show (5, 100, 180,  'previous user:');
  Screen.Label.Show (6, 100, 210,  Userdata1 );
  Screen.Label.Show (7, 100, 240,  Userdata2 );
  Screen.Label.Show (8, 100, 270,  Userdata3 );
//***************************************************************

  SetValue (1);

  DateTime.Delay (2000);

  repeat
  until StepContinue;
  Screen.ClrScr;

end.
```

21. Edit the third interpreter step:
You can rename files with the procedure ➡ **File.Move** ( ). Use the procedure and change the name of the document of the current user to that of the previ ous user. Through this, the current user becomes the previous user at the end of the sequence. When re-running the sequence, the file of the current user is deleted and recreated. That way, the current user and the previous user is stored.

Syntax:

```
File.Move ('C:\...\TypeData\TYPE_12PRACTICE10\User_current.
txt', 'C:\...\User_before.txt', True);
```

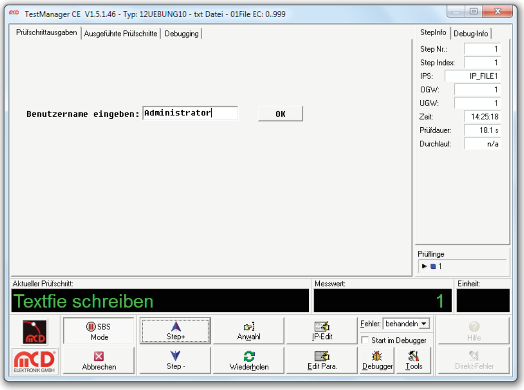
The procedure requires the specification of paths to rename the files.

22. Enter the measuring value and a delay time of about one second.

Source code for the third interpreter step: *IP_FILE2.IPS*

```
var

step

//*****************Rename File:***************************
  File.Move ('C:\TestCollection\Testing\TestManager CE\TypeD-
ata\TYPE_12PRACTICE10\User_current.txt', 'C:\TestCollection\
Testing\TestManager CE\TypeData\TYPE_12PRACTICE10\User_before.
txt', True);
//************************************************************
  SetValue (1);

  DateTime.Delay (1000);

  repeat
  until StepContinue;

end.
```

23. Save your interpreter steps and start the sequence.


TestManager CE V1.5.1.46 - Typ: 12UEBUNG10 - txt Datei - 01File EC: 0..999

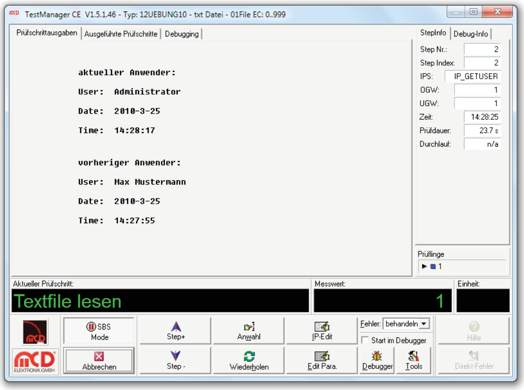
Benutzername eingeben: Administrator

Step Nr.: 1
Step Index: 1
IPS: IP_FILE1
OGW: 1
UGW: 1
Zeit: 14:25:18
Prüfdauer: 18.1 s
Durchlauf: n/a

Aktueller Prüfschritt: Textfie schreiben
Messwert: 1
Einheit:

Fehler: behandeln
Start im Debugger

Figure 6-7
Starting a Sequence- Step 1


TestManager CE V1.5.1.46 - Typ: 12UEBUNG10 - txt Datei - 01File EC: 0..999

```
aktueller Anwender:

User:  Administrator

Date:  2010-3-25

Time:  14:28:17

vorheriger Anwender:

User:  Max Mustermann

Date:  2010-3-25

Time:  14:27:55
```

Step Nr.:	2
Step Index:	2
IPS:	IP_GETUSER
OGW:	1
UGW:	1
Zeit:	14:28:25
Prüfdauer:	23.7 s
Durchlauf:	n/a

Prüflinge: 1

Aktueller Prüfschritt: Textfile lesen

Messwert: 1

Einheit:

Fehler: behandeln

Figure 6-8
Starting a Sequence - Step 2

INI-File

The TestManager also supports access and the editing of INI files. These files are initialization files that are, for example, used for storing program settings. INI files can be interpreted, interpreted section by section, described, deleted, and deleted by sections by the TestManager. Functions and procedures are available for this purpose.

An INI file consists of the components section, the elements and the values. These so-called sections represent an umbrella term under which items can be stored with their values.

Example INI file structure:

```
[Section1]
Item1 = Value1
Item2 = Value2

[Sektion2]
Item1 = Value1
Item2 = Value2
```

Item1 in Section1 and Item 2 in Section 2, and their values can coincidentally be identical but are to be treated fundamentally different.

☐ Exercise 6-3
Create read and write INI files

This exercise deals with INI files and should give you a better understanding of how to deal with them. The task at hand is to edit a sequence, which consists of two steps. In the first step, an INI file will be opened or created. In this file, three randomly generated numbers (length, width and height) should be written in the 'dimensions' section. From these generated numbers, the volume should be calculated and stored in the 'calculated values' section. The reading and the output data from the INI file should be realized in the second step.

Step by step:

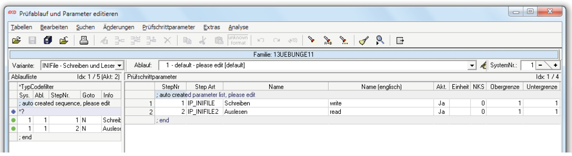
1. Log on as administrator
   Navigate from the toolbar> Password. Press the F4 function key or select Administrator and type in >admin

2. Create a new family type with a variant, set it to active, and save the edited data.

Example:	Family Type:	13PRACTICE11
	Code Type:	INIFile
	Information Type:	Writing and Reading

3. Choose the variant of the new family type in step 2 via Typeselect.

4. Add a new step to both the sequence list and the test step parameters and include two new step types. Defined the limits, activate these steps and save the edited data.



Prüfablauf und Parameter editieren

Familie: 13UEBUNGE11

Variante: INIFile - Schreiben und Leser   Ablauf: 1 - default - please edit [default]   SystemNr.: 1

Ablaufliste   Idx: 1 / 5 (Akt: 2)

	Sys.	Abl.	StepNr.	Goto	Info
	*TypCodefilter				
	; auto created sequence, please edit				
	*?				
●	1	1	1	N	Schreib
●	1	1	2	N	Auslese
	; end				

Prüfschrittparameter   Idx: 1 / 4

	StepNr	Step Art	Name	Name (englisch)	Akt.	Einheit	NKS	Obergrenze	Untergrenze
	; auto created parameter list, please edit								
1	1	IP_INIFILE	Schreiben	write	Ja		0	1	1
2	2	IP_INIFILE2	Auslesen	read	Ja		0	1	1
	; end								

Figure 6-9

Test Sequence Editing

5. Select the variable created via Typeselect and start the sequence.

6. Edit the first interpreter step:
   Create two variables of type String to store the section name and four type real variables in order to save the values.

7. Open or create an INI file using the procedure ➡ **INIFile.SetFileName (Path + Name.INI)**. It is available to choose a path that leads to the desktop of the computer you are using or in the TestManager installation directory.

Syntax:

```
INIFile.SetFileName ('C:\...\TypeData\TYPE_13PRACTICE11\PRAC-
TICE11.INI');
```

Tip:
Please note that the path you use can differ from the one above.

8. Generate three random numbers with ➡ **Math.Random** ( ) function and assign it to the intended variables.

9. Write section in the INI file:
   Assign the variables for the section to the name ➡ **Dimensions**. With help of the procedure ➡ INIFile.WriteReal (section, item, value), you can write a real value for an item in the section.

Syntax:

```
Section1 := 'Dimensions';
INIFile.WriteReal ( Section1,  '1: length', value1);
```

10. Calculate the volume and pass the value to the variable provided here by you.

11. Assign the name of the second section of the proposed variables and write the volume of this section in the INI file.

12. Set the measurement value within the limits defined by you.

Source code: *IP_INIFILE.IPS*

```
var
  Section1    : String;
  Section2    : String;
  Value1      : Real;
  Value2      : Real;
  Value3      : Real;
  Volume      : Real;

step
```

➡ *continue to the next page*

```
//******************Open or create IniFile:********************
  INIFile.SetFileName ('C:\...\TypeData\TYPE_13PRACTICE11\
  Practice11.INI');
//**************************************************************

//*****************Generate random values:*******************
  Value1 := Math.Random (100);
  Value2 := Math.Random (100);
  Value3 := Math.Random (100);
//**************************************************************

//*************Section1 in INIFile schreiben:******************
  Section1 := 'Dimensions';

  INIFile.WriteReal ( Section1, '1: length', value1);
  INIFile.WriteReal ( Section1, '2: width', value2);
  INIFile.WriteReal ( Section1, '3: Height', value3);
//**************************************************************

//***************Calculated volume:*****************************
  Volume := value1 * value2 * value3;
//**************************************************************

//*************Write volume in Section2:******************
  Section2 := ''calculated values';

  INIFile.WriteReal ( Section2,  '1: volume', volume);
//**************************************************************
  repeat
  until StepContinue;

  SetValue (1);
end.
```

13. Edit the second interpreter step:
    Define six variables of type String; two for the allocation of sections and four for the item names.

14. Initialize the screen for the output of screen objects.

15. Open the INI file with the procedure➡ **INIFile.SetFileName**.

16. Read sections:
    Read the sections about the procedure ➡ **INIFile.Load** ( ) and set the internal pointer with the order ➡ **INIFile.First** on the first read-in section.
    Now, the section name can be passed directly to the intended variable with the function ➡ **INIFile.Name**.

Syntax:

```
INIFile.Load ('');
INIFile.First;          //set pointer to the first section
Section1 := INIFile.Name;
```

To read the second section, you must have the internal pointer set on the next element. This is done by ➡ **INIFile.Next**. With the ➡ **INIFile.Name** command, the section name can be passed to the variable.

Syntax:

```
INIFile.Next;           //set pointer to the next section
Section2 := INIFile.Name;
```

17. Items to read from the INI file:
Using the just-mentioned command ➡ **INIFile.Load** (Section) the contents of a section can be read. Should items now be interpreted from this section, then the process will be similar to the interpretation of the other sections.
The internal pointer must be set on the first element of the Section. For this purpose the procedure ➡ **INIFile.First** is used again. You can now use the function ➡ **INIFile.Name** to pass the item name to a variable. The value of the item can be read with ➡ **INIFile.Value**. Should this be passed directly to a real variable, then you must convert from string to a real using ➡ Val ( ). You can then move the internal pointer to the next element with ➡ **INIFile.Next**.

Syntax:

```
INIFile.Load (Section1);
INIFile.First;
Variable1  := INIFile.Name;
Value1     := Val (INIFile.Value, 0);
INIFile.Next;
```

18. Proceed as done in the16th step to interpret the item from the second section.

Syntax:

```
INIFile.Load (Section2);
INIFile.First;
Compute-name := INIFile.Name;
Calculated value := Val (INIFile.Value, 0);
```

19. Enter the variables using the screen functions in the test step display window and insert the measured value within your defined boundaries.

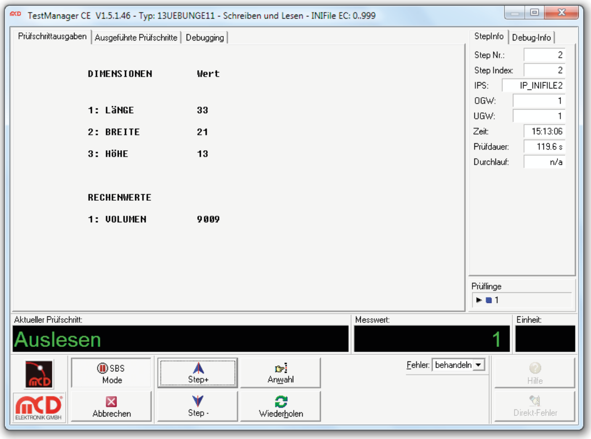
Quellcode: *IP_INIFILE2.IPS*

```
var
  Section1             : String;
  Section2             : String;
  Variable1            : String;
  Variable2            : String;
  Variable3            : String;
  Value1               : Real;
  Value2               : Real;
  Value3               : Real;
  Calculated value     : Real;
  Compute-name         : String;

step
//****************Initialize screen:**********************
  Screen.SetTab (1);       //focus on Test step results
  Screen.ClrScr;
  Screen.Show;
//***************************************************************

//**************Open or create the IniFiles:***********
  INIFile.SetFileName ('C:\TestCollection\Testing\TestManager
  CE\TypeData\TYPE_13PRACTICE11\PRACTICE11.INI');
//***************************************************************

//***************Section names are read out:************
  INIFile.Load ('');
  INIFile.First;       //set pointer on the first section
  Section1 := INIFile.Name;     //read name of the entry,
                                hat the pointer points to
  INIFile.Next;        //set pointer on the next section
  Section2 := INIFile.Name;
//***************************************************************

//**********Retrieve items and values from Section1 :*********
  INIFile.Load (Section1);
  INIFile.First;
  Variable1 := INIFile.Name;
  Value1     := Val (INIFile.Value, 0);
  INIFile.Next;
  Variable2 := INIFile.Name;
  Value2    := Val (INIFile.Value, 0);
  INIFile.Next;
  Variable3 := INIFile.Name;
  Value3    := Val (INIFile.Value, 0);
//***************************************************************

//***********Retrieve items and values from Section2 :*********
  INIFile.Load (Section2);
```

➡ *continue to the next page*

```
    INIFile.First;
    Compute-name      := INIFile.Name;
    Calculated value := Val (INIFile.Value, 0);
  //***************************************************************

  //****************Ausgabe:**********************************
    Screen.Label.Show ( 1, 100, 30, Section1);
    Screen.Label.Show ( 2, 250, 30, 'value');
    Screen.Label.Show ( 3, 100, 80, Variable1);
    Screen.Label.Show ( 4, 250, 80, Str (value1));
    Screen.Label.Show ( 5, 100, 110, Variable2);
    Screen.Label.Show ( 6, 250, 110, Str (value2));
    Screen.Label.Show ( 7, 100, 140, Variable3);
    Screen.Label.Show ( 8, 250, 140, Str (value3));
    Screen.Label.Show ( 9, 100, 200, Section2);
    Screen.Label.Show (10, 100, 230, Rcomputer name);
    Screen.Label.Show (11, 250, 230, Str (calculated value));
  //***************************************************************

    repeat
    until StepContinue;

    SetValue (1);
  end.
```

20. Save your interpreter steps and restart the process.



```
DIMENSIONEN        Wert

1: LäNGE           33
2: BREITE          21
3: HöHE            13

RECHENWERTE
1: VOLUMEN         9009
```

Step Nr.: 2
Step Index: 2
IPS: IP_INIFILE2
OGW: 1
UGW: 1
Zeit: 15:13:06
Prüfdauer: 119.6 s
Durchlauf: n/a

Aktueller Prüfschritt: Auslesen
Messwert: 1
Einheit:
Fehler: behandeln

Figure 6-10
Sequence start

Contents of INI files:

```
[DIMENSIONEN]
1: LENGTH=33
2: WIDTH=21
3: HEIGHT=13
[Calculated value]
1: VOLUME=9009
```

7. Hardware Access

A license is required to operate the program with access to all interfaces. To gain access to hardware with the demo version, a short-term license can be requested.
A full version of TestManager can be obtained by requesting a license and the receipt of a key.

License Request

Navigate via the ➡ **menu bar** ➡ **Intern** ➡ **License** and ➡ **License Request** to reach the window where the licenses can be requested.

Serial interface

The module for the serial port is used to access all types of RS232 ports regardless of what cards or hardware are realized in the PC. If the COM Port in the system administration of Windows appears, it can be used by the program. The module provides additional ports for additional modules (eg LIN) and adopts the administration of the port parameters.

Through the **Basic settings** (➡**Toolbar** ➡ **Setup** ➡ **Serial Ports**), a serial port can be created and configured.

☐ **Exercise 7-1**
Using Serial Interfaces

Please note that you can only perform this exercise if your computer has the required COM port. For the implementation of the exercise, a plug, that you should be able to make yourself, is necessary. This plug is a 9-pin Sub-D connector and needs to be bridged according to the following circuit diagram:

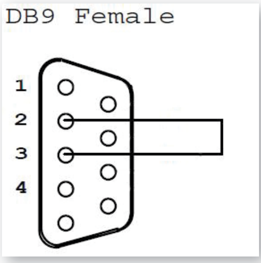


Figure 7-1
Schematic 9-pin Sub-D socket

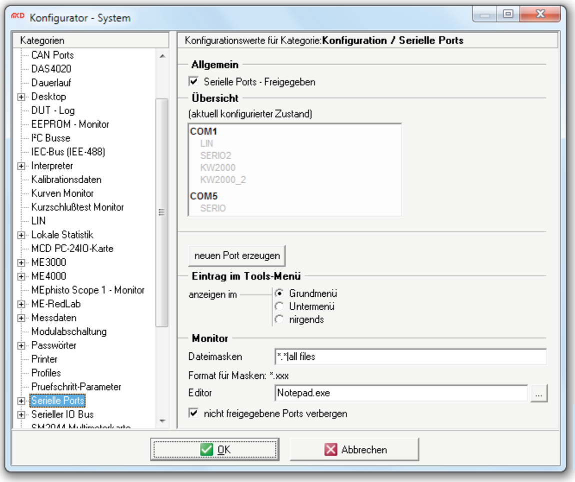
Using Serial Port

This exercise aims at the usage of the serial interface. The task at hand is to request a short-term license, create a new serial port in the basic settings and configure it. During the sequence, a string should be sent and received via the serial interface. The inspection of the received strings should yield that the sent string corresponds to the received string. If this is not the case, then the result of the process should be adjusted accordingly.

Step by step:

1. Log on as administrator
   Navigate from the ➡**toolbar** ➡ Password. Press the F4 function key or select Administrator and type in >admin

2. Create a serial port:
   Navigate via the ➡**toolbar** ➡ **Setup** in the basic settings. Select ➡ **Serial Ports** from the panel on the left side of the window. Create a **new port** with the help of the button



Konfigurator - System

Kategorien: CAN Ports, DAS4020, Dauerlauf, Desktop, DUT - Log, EEPROM - Monitor, I²C Busse, IEC-Bus (IEE-488), Interpreter, Kalibrationsdaten, Kurven Monitor, Kurzschlußtest Monitor, LIN, Lokale Statistik, MCD PC-24IO-Karte, ME3000, ME4000, MEphisto Scope 1 - Monitor, ME-RedLab, Messdaten, Modulabschaltung, Passwörter, Printer, Profiles, Pruefschritt-Parameter, Serielle Ports, Serieller IO Bus

Konfigurationswerte für Kategorie:**Konfiguration / Serielle Ports**

Allgemein
✓ Serielle Ports - Freigegeben

Übersicht
(aktuell konfigurierter Zustand)

```
COM1
  LIN
  SERIO2
  KW2000
  KW2000_2
COM5
  SERIO
```

neuen Port erzeugen

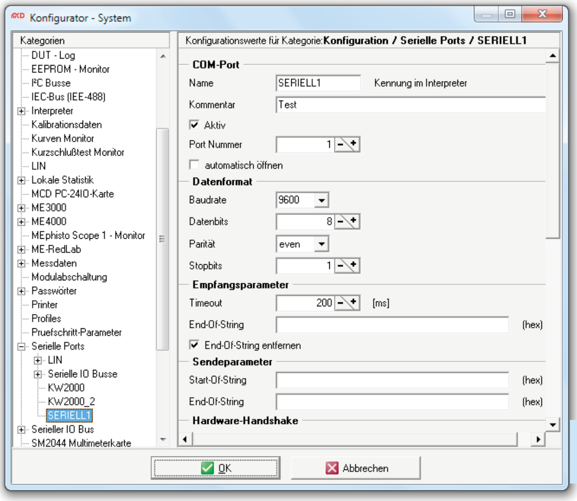
Eintrag im Tools-Menü
anzeigen im: [✓ Grundmenü] Untermenü nirgends

Monitor
Dateimasken: *.*|all files
Format für Masken: *.xxx
Editor: Notepad.exe
✓ nicht freigegebene Ports verbergen

OK Abbrechen

Figure 7-2
Create Serial Port 1

3. Enter a ➡**Name** (example: SERIAL1) for the COM port and optionally add a comment. Set the port to ➡ **Active** and set the ➡ **Baudrate** to ➡ **9600**. Confirm your changes with the> OK button.



Kategorien
- DUT - Log
- EEPROM - Monitor
- I²C Busse
- IEC-Bus (IEE-488)
- Interpreter
- Kalibrationsdaten
- Kurven Monitor
- Kurzschlußtest Monitor
- LIN
- Lokale Statistik
- MCD PC-2410-Karte
- ME3000
- ME4000
- MEphisto Scope 1 - Monitor
- ME-RedLab
- Messdaten
- Modulabschaltung
- Passwörter
- Printer
- Profiles
- Pruefschritt-Parameter
- Serielle Ports
  - LIN
  - Serielle IO Busse
  - KW2000
  - KW2000_2
  - SERIELL1
- Serieller IO Bus
- SM2044 Multimeterkarte

Konfigurationswerte für Kategorie: **Konfiguration / Serielle Ports / SERIELL1**

COM-Port
- Name: SERIELL1 — Kennung im Interpreter
- Kommentar: Test
- [✓] Aktiv
- Port Nummer: 1
- [ ] automatisch öffnen

Datenformat
- Baudrate: 9600
- Datenbits: 8
- Parität: even
- Stopbits: 1

Empfangsparameter
- Timeout: 200 [ms]
- End-Of-String: (hex)
- [✓] End-Of-String entfernen

Sendeparameter
- Start-Of-String: (hex)
- End-Of-String: (hex)

Hardware-Handshake

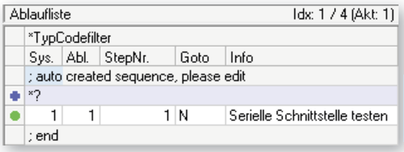
Figure 7-3
Create Serial Port 2

4. Insert a new family type with a variant, set this to 'active' and save the data.

Example:	Family Type:	14SERIAL
	Code Type:	Serial
	Information Type:	Test

5. Select the variable created via Typeselect and start the sequence

6. In the test step parameters, add a new name for the step type and define the limits. Set the step to 'active' and save.

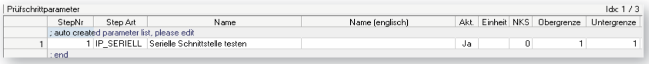


Ablaufliste — Idx: 1 / 4 (Akt: 1)

*TypCodefilter

	Sys.	Abl.	StepNr.	Goto	Info
	; auto created sequence, please edit				
✚	*?				
●	1	1	1	N	Serielle Schnittstelle testen
	; end				

Figure 7-4
Sequence List



Prüfschrittparameter — Idx: 1 / 3

	StepNr	Step Art	Name	Name (englisch)	Akt.	Einheit	NKS	Obergrenze	Untergrenze
	; auto created parameter list, please edit								
1	1	IP_SERIELL	Serielle Schnittstelle testen		Ja		0	1	1
	; end								

Figure 7-5
Test Step Parameters

7. Request a short-term license on the ➡ **menu bar** ➡ **Intern** ➡ **License**.

8. Start the sequence in SBS mode and edit the interpreter step.

9. Edit interpreter step:
   Add two variables of type string, in order to store the sent and received data.

10. Using the help of the screen commands, initialize the monitor to display data in the test step display window.

11. Assign the variable to send a string of your choice.

12. With the help of the function ➡ **RS232.Exists** (‚Name of the interface') ask whether the applied interface exists and enter the result on the test step display window.

    Syntax:

```
If RS232.Exists ('SERIAL1') = 1 Then
begin
  Screen.Label.Show (3, 100, 30, ''interface exists!');
end;
```

13. Open the applied interface with the command ➡**RS232.Open** ( ).
    Using ➡ **RS232.IsOpen** ( ) ask whether the port has been opened and state the result as well.

    Syntax:

```
RS232.Open ('SERIAL1');

If RS232.IsOpen ('SERIAL1') = 1 Then
begin
  Screen.Label.Show (4, 100, 60, 'Port is open!');
end;
```

14. Send the string stored in the variable using the command ➡ **RS232.Send** ( ) via the applied serial interface.

    Syntax:

```
RS232.Send ('SERIAL1', send);
```

15. With the help of function ➡ **RS232.Read** ( ), send the data received via the inter face to the second variable.

    Syntax:

```
Reception := RS232.Read ('SERIAL1');
```

16. State the sent and the received string and insert a lag time of about one second.

17. Compare if the received string corresponds to the transmitted string and adjust the measurement value accordingly. The sent string must correspond to the received string due to the wiring of the socket.

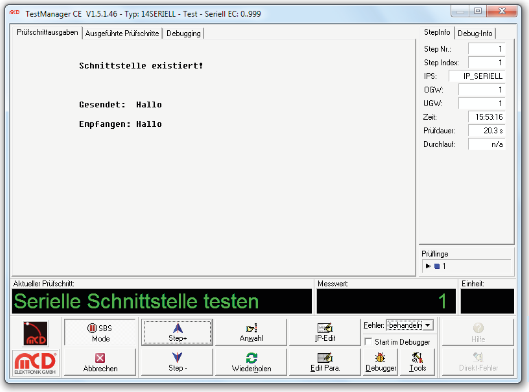
Source code: *IP_SERIELL.IPS*

```
var
  Reception    : String;
  Send         : String;
Step

  Screen.SetTab (1);       //focus on test step display
  Screen.ClrScr;
  Screen.Show;

//****************Assign string:******************************
  Send := 'Hello';
//************************************************************

//*********Queries whether interface is configured:*************
  If RS232.Exists ('SERIAL1') = 1 Then
  begin
    Screen.Label.Show (3, 100, 30, ''interface is
  end;                    configured!');
//************************************************************


//****************Open interface :************************
  RS232.Open ('SERIELL1');

  If RS232.IsOpen ('SERIAL1') = 1 Then
  begin
     Screen.Label.Show (4, 100, 60, 'Port is open'!');
  end;
//************************************************************

//****************Send:************************************
  RS232.Send ('SERIAL1', send);
//************************************************************

//****************Receive:**********************************
  Reception := RS232.Read ('SERIAL1');
//************************************************************
//****************Ausgabe:*************************************
  Screen.Label.Show (1, 100,  90, 'Sent:   ' +  send);
  Screen.Label.Show (2, 100, 120, 'Received: ' + reception);
  DateTime.Delay (1000);
//************************************************************
  repeat
  until StepContinue;
```

➡ *continue to the next page*

```
//****************Messwert setzen:*****************************
  If received = Send Then
  begin
      SetValue (1);
  end
  Else begin
      SetValue (0);
  end;
//***************************************************************

end.
```

18. Save your interpreter step and start the sequence.


TestManager CE V1.5.1.46 - Typ: 14SERIELL - Test - Seriell EC: 0..999

```
Schnittstelle existiert!

Gesendet:  Hallo
Empfangen: Hallo
```

Step Nr.:	1
Step Index:	1
IPS:	IP_SERIELL
OGW:	1
UGW:	1
Zeit:	15:53:16
Prüfdauer:	20.3 s
Durchlauf:	n/a

Aktueller Prüfschritt: Serielle Schnittstelle testen
Messwert: 1
Einheit:
Fehler: behandeln
Start im Debugger

Bild 7-6
Serial Interface Test

Tip:
Should the result of the step fail repeatedly, then it could be that you are operating a non-licensed system. Check if you have access to a license and if necessary request a short-term license.

8. Measured data

Each test step generates a set of data after execution. Generated are the real measured value, the result of the step, and also a set of metadata. This, for example, refers to the time of testing, the duration of the step, and the obtained data that is available in that type of test step, e.g. the number of the test step, the unit, etc.
The standard way to store this data is in DBF format (dBase). This requires an installed BDE (Borland Database Engine). The measurement data can be stored in standard format or in a freely definable spreadsheet. To avoid large amounts of data a built-in data maintenance is available.
The path, in which the data are stored in, is configurable. In type-related storage, a sub-directory is created for each family type. All types of data are stored together by default.
There is also the possibility to transfer the measurement data to a DLL.

☐ Exercise 8-1
Dealing with measurement data

This exercise will demonstrate the use of measurement data. It is intended to summarize the measurement data in DBF format and realize the local storage.

Step by step:

1. Log on as administrator
   Navigate from the toolbar Password. Press the F4 function key or select Administrator and type in admin.

2. Activate the data logging in DBF format by navigating to the basic settings ➡ **toolbar** ➡ **Setup**. In the left panel of the basic settings, select ➡ **Measured Data** and open the content by selecting the ➡ **+ box**.

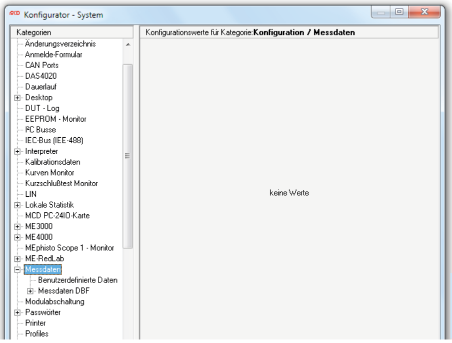
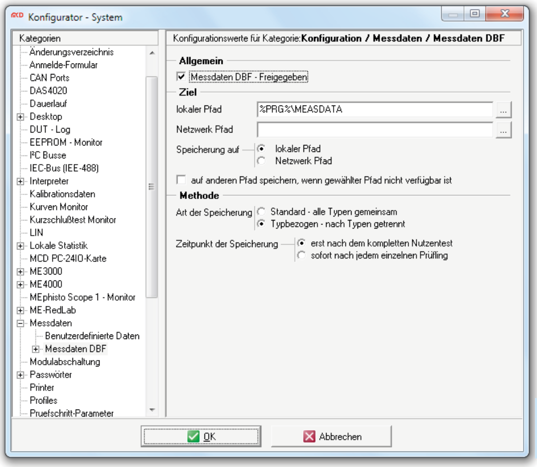


Figure 8-1
Data configuration

3. Select ➡ **Measure Data DBF** and in the right field under General ➡ checkmark the box titled ‘Measure data DBF-Enable’.



Konfigurationswerte für Kategorie:**Konfiguration / Messdaten / Messdaten DBF**

Allgemein

[✓] Messdaten DBF - Freigegeben

Ziel

lokaler Pfad: %PRG%\MEASDATA

Netzwerk Pfad:

Speicherung auf: [✓ lokaler Pfad] Netzwerk Pfad

[ ] auf anderen Pfad speichern, wenn gewählter Pfad nicht verfügbar ist

Methode

Art der Speicherung: Standard - alle Typen gemeinsam [✓ Typbezogen - nach Typen getrennt]

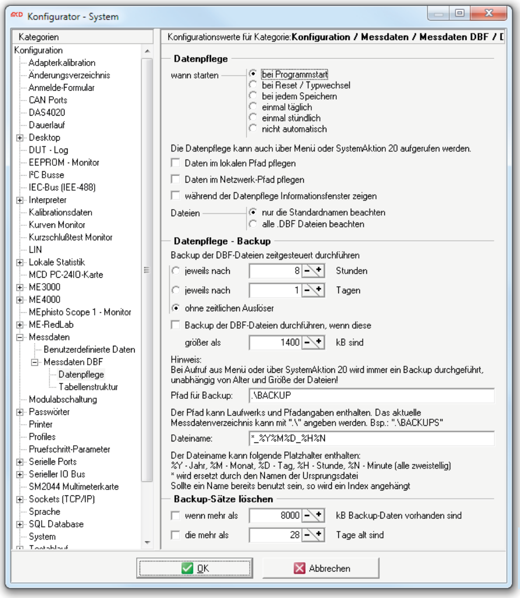
Zeitpunkt der Speicherung: [✓ erst nach dem kompletten Nutzentest] sofort nach jedem einzelnen Prüfling

Figure 8-2

DFB data release

The measured values are thus stored in a DBF file under the installation directory folder MEASDATA. Alternatively, a network path can be specified. The data is separated by type and stored locally only after the complete functionality test has taken place.

4. Open the menu to the setting ‘Data maintenance’ (subpart of measured data DBF in the left panel of the window). Set it to ‘Start when program starts’ under data maintenance.



Konfigurationswerte für Kategorie:**Konfiguration / Messdaten / Messdaten DBF / D**

Datenpflege

wann starten: [✓ bei Programmstart] bei Reset / Typwechsel; bei jedem Speichern; einmal täglich; einmal stündlich; nicht automatisch

Die Datenpflege kann auch über Menü oder SystemAktion 20 aufgerufen werden.

☐ Daten im lokalen Pfad pflegen

☐ Daten im Netzwerk-Pfad pflegen

☐ während der Datenpflege Informationsfenster zeigen

Dateien: [✓ nur die Standardnamen beachten] alle .DBF Dateien beachten

Datenpflege - Backup

Backup der DBF-Dateien zeitgesteuert durchführen

jeweils nach 8 Stunden

jeweils nach 1 Tagen

[✓ ohne zeitlichen Auslöser]

☐ Backup der DBF-Dateien durchführen, wenn diese größer als 1400 kB sind

Hinweis:
Bei Aufruf aus Menü oder über SystemAktion 20 wird immer ein Backup durchgeführt, unabhängig von Alter und Größe der Dateien!

Pfad für Backup: .\BACKUP

Der Pfad kann Laufwerks und Pfadangaben enthalten. Das aktuelle Messdatenverzeichnis kann mit ".\" angeben werden. Bsp.: ".\BACKUPS"

Dateiname: *_%Y%M%D_%H%N

Der Dateiname kann folgende Platzhalter enthalten:
%Y - Jahr, %M - Monat, %D - Tag, %H - Stunde, %N - Minute (alle zweistellig)
* wird ersetzt durch den Namen der Ursprungsdatei
Sollte ein Name bereits benutzt sein, so wird ein Index angehängt

Backup-Sätze löschen

☐ wenn mehr als 8000 kB Backup-Daten vorhanden sind

☐ die mehr als 28 Tage alt sind

Figure 8-3
Setting Data Maintenance

5. Save the settings by clicking on the ➡ **OK** button.

6. Choose the sequence of family type ➡ **2MEASUREDATA** through ➡ Typeselect and start the sequence in **Automatic mode**.

7. Via Windows Explorer, navigate through the ➡**Installation Directory** of the TestManager to the folder ➡ **MEASDATA**. There you will find a folder ➡ **TYPE_2MEASUREDATA** that includes the DBF Files ➡ **FAILDATA** ➡ **MEASDATA** ➡ and➡ **PASSFAIL**.

8. You can view the contents of DBF files using Microsoft Excel.

9. Reports

Measurements and results can be viewed, saved, and printed. To customize the printouts according to the wanted user preference, the TestManager is equipped with a powerful report generator. With its help, the printouts can be designed freely. It can refer back to draft and edit them as well. Measurement values can be retrieved, managed, and printed via the ➡ **menu bar** ➡ **Values** ➡ **Measured Values**. Through the use of reports, common ‚templates‘ can be created so that you do not have to write a new report for every sequence.

Exercise 9-1
Creating Your Own Reports

The objective of this exercise is to create your own report, in which the measured values for the already-existing 2MEASUREDATA sequence can be printed and stored. This way, the design will be customized individually.

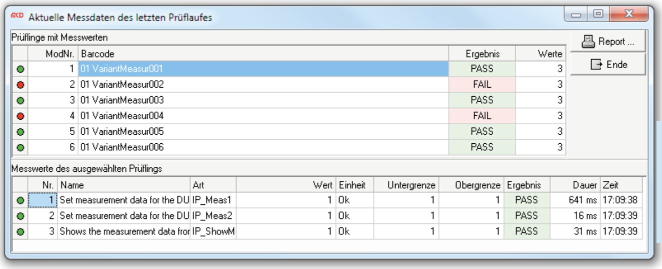
Step by step:

1. Überprüfen Sie Ihren Passwortlevel, sollten Sie nicht als Administrator beim System angemeldet sein, ändern Sie dies.

2. Request a short-term license, in case your software is not licensed.

3. Choose via ➡ **Typeselect** the sequence of family type ➡ **2MEASUREDATA** and start the sequence in automatic mode.

4. After completion of the sequence, navigate to the ➡ **menu bar** ➡ **Values** ➡ **Measured Values** to indicate the current measured values of the last test process..



Prüflinge mit Messwerten

	ModNr.	Barcode	Ergebnis	Werte
🟢	1	01 VariantMeasur001	PASS	3
🔴	2	01 VariantMeasur002	FAIL	3
🟢	3	01 VariantMeasur003	PASS	3
🔴	4	01 VariantMeasur004	FAIL	3
🟢	5	01 VariantMeasur005	PASS	3
🟢	6	01 VariantMeasur006	PASS	3

Messwerte des ausgewählten Prüflings

	Nr.	Name	Art	Wert	Einheit	Untergrenze	Obergrenze	Ergebnis	Dauer	Zeit
🟢	1	Set measurement data for the DU	IP_Meas1	1	Ok	1	1	PASS	641 ms	17:09:38
🟢	2	Set measurement data for the DU	IP_Meas2	1	Ok	1	1	PASS	16 ms	17:09:39
🟢	3	Shows the measurement data fror	IP_ShowM	1	Ok	1	1	PASS	31 ms	17:09:39

Figure 9-1 Measu-rement Values

5. Activate the button ➡ **Report** to open the administration window.

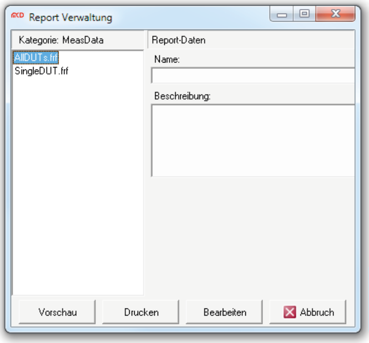


Figure 9-2
Report Management

6. Select the existing report AllDUTs.frf and confirm your choice with the button ➡ **Edit** to customize the report to your own preferences.

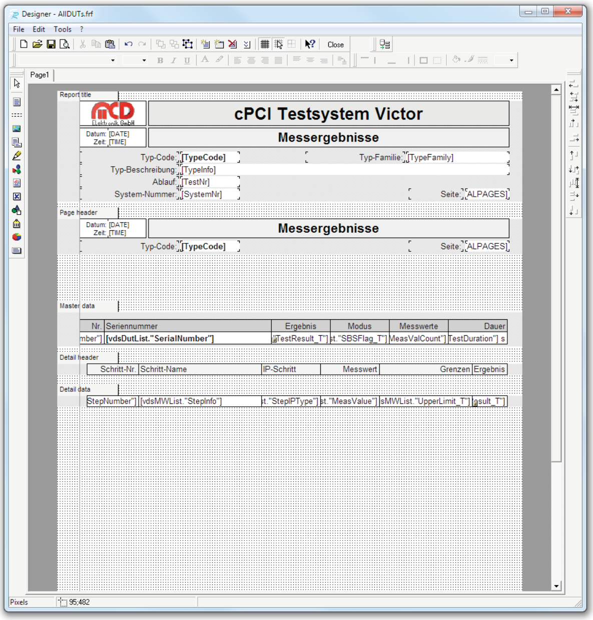
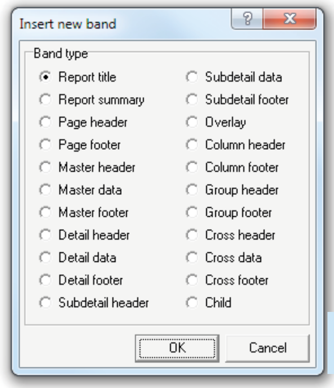


Figure 9-3
Designer Example

7. To edit an empty report, please navigate to ➡ **toolbar** ➡ **New report**

8. Now create your own title report, by going to ➡ **toolbar** ➡ **Insert band**



Insert new band

Band type

[✓ Report title]	Subdetail data
Report summary	Subdetail footer
Page header	Overlay
Page footer	Column header
Master header	Column footer
Master data	Group header
Master footer	Group footer
Detail header	Cross header
Detail data	Cross data
Detail footer	Cross footer
Subdetail header	Child

Figure 9-4
Creating a Report Title

9. Select ➧ **Report title** from the window and confirm with the button ➧ **OK**.

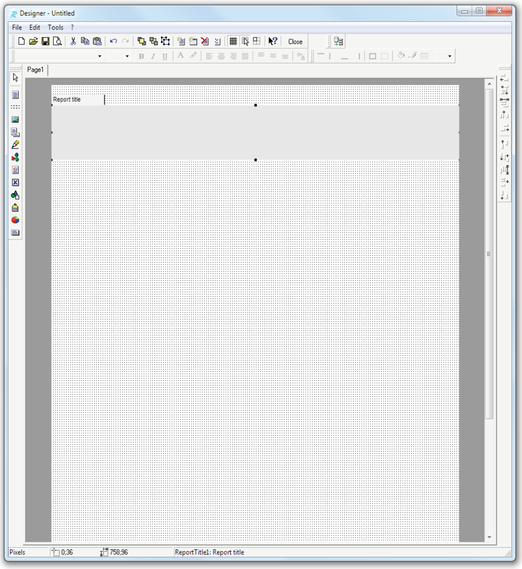
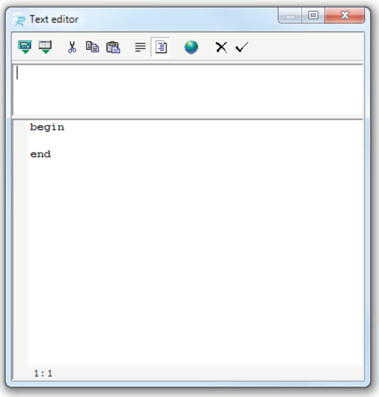


Figure 9-5
Select Report Title

10. Select a place in the report, where you want to insert your title report.

11. Via the ➧ **toolbar** ➧ **Insert rectangle objects,** rectangular fields can be inserting in the report. Place one of these fields in your report title.

12. A text editor window can be opened to allow you to enter a text in the box or have the system transfer variables over.



```
begin

end
```

Figure 9-6
TextEditor

13. In the upper box, type a suitable name for your report, and confirm your entry. Drag it to the desired location and edit the desired font size.

14. Depending on your preference, you can set up a color, frame, or something similar to your field via the ➡ **toolbar.**

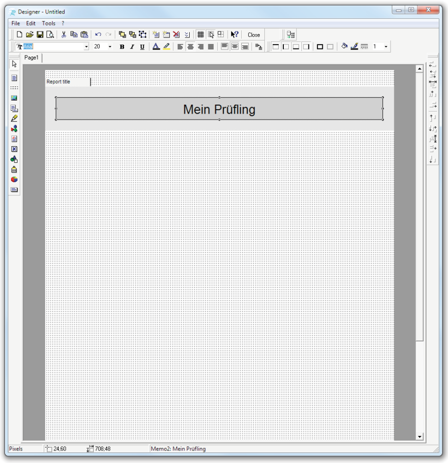


Figure 9-7
Designer Example

15. To display variables like the current date on the report, place another text box. Navigate via ➡ **toolbar** of the text editor window ➡ **to the insert data field in left field**

➡ **vdsDutList** and ➡ **TestTime_T** in the right.

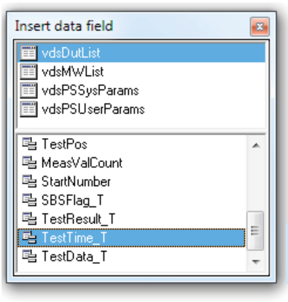


Figure 9-8
Designer Example

This approach can retrieve any of the variables of the system and must not edit each sequence by hand.

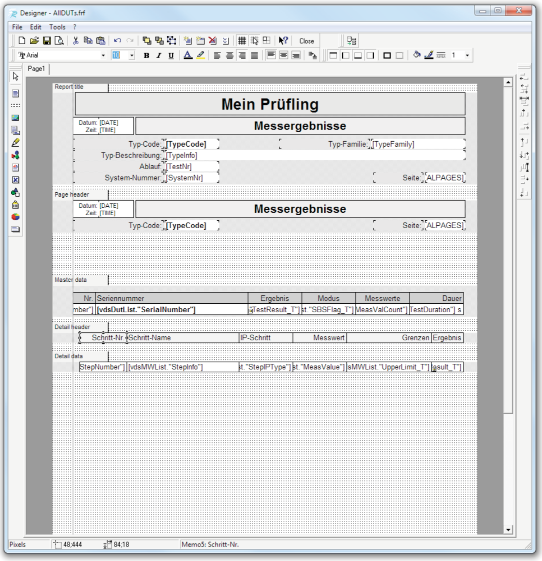
16. This way, you can add more variables such as Family Type, Code Type, test time, etc. to your report title.

    Several variables can be used in one field; this can serve as an advantage for page numbers:

    [PAGE#] of [TOTAL PAGES]

    This way, the current page number of the total number of pages can be displayed.

17. Insert a new band and for the type select ➡ **Page header.** Use this command to add a header to your report. Right-clicking on the header can set if this is to be used on the first page. In this case it is not necessary. Create a header according to your preferences.



Mein Prüfling

Datum: [DATE]
Zeit: [TIME]

Messergebnisse

Typ-Code: [TypeCode] Typ-Familie: [TypeFamily]
Typ-Beschreibung: [TypeInfo]
Ablauf: [TestNr]
System-Nummer: [SystemNr] Seite: [ALPAGES]

Datum: [DATE]
Zeit: [TIME]

Messergebnisse

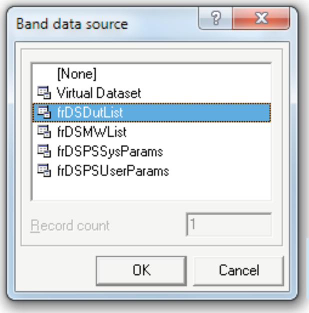
Typ-Code: [TypeCode] Seite: [ALPAGES]

Nr.	Seriennummer	Ergebnis	Modus	Messwerte	Dauer
mber"]	[vdsDutList."SerialNumber"]	TestResult_T"]	st."SBSFlag_T"]	MeasValCount"]	TestDuration"] s

Schritt-Nr.	Schritt-Name	IP-Schritt	Messwert	Grenzen	Ergebnis
StepNumber"]	[vdsMWList."StepInfo"]	st."StepIPType"]	st."MeasValue"]	sMWList."UpperLimit_T"]	sult_T"]

Figure 9-9
Example

18. Create another band from type ➡ **Master data**. Set a source code correlating to the data. To do this, double-click on the band:



[None]
Virtual Dataset
frDSDutList
frDSMWList
frDSPSSysParams
frDSPSUserParams

Figure 9-10
New Band

19. Choose ➡ **frDSDutList** and confirm with ➡ **OK**.

20. Realize the displayed module number, serial number, result, mode, duration and measurement values from the table of information. The used data can be found in the division ➡ **vdsDutList**.

Module number	➧ **[vdsDutList.“Module number“]**
Serial number	➧ **[vdsDutList.“SerialNumber“]**
Result	➧ **[vdsDutList.“TestResult_T“]**
Mode	➧ **[vdsDutList.“SBSFlag_T“]**
Measurement Values	➧ **[vdsDutList.“MeasValCoun“]**
Duration	➧ **[vdsDutList.“TestDuration“] s**

➧ *Master data example, see Figure 9-9*

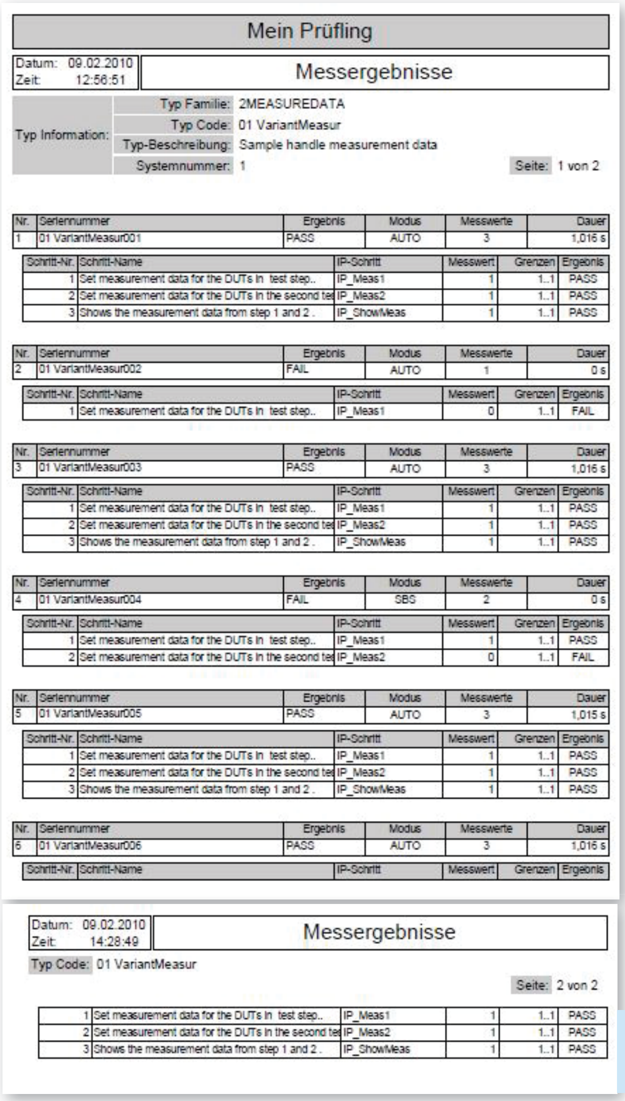
21. Create a new band from type ➧ **Detail header** and one from type ➧ **Detailed data**. These two bands are closely linked and serve the display of the measured values of the individual interpreter steps. The names are stored in the detail header while the variables (values) are stored in the detail data.

    Detail Header:
    ➧ **Step No.**
    ➧ **Step Name**
    ➧ **IP Step**
    ➧ **Measurement Values**
    ➧ **Limits**
    ➧ **Results**

    The detailed data section will fall back on ➧ **frDSMWList** and give out the variables (values) to the detail header.

    ➧ *Detail header + data example, see Figure 9-9*

22. You can vary the distances of the bands with the help of the gray areas. To see your report, go to ➧ **toolbar** ➧ **Preview report** and adjust the layout to your preferences and check whether the variables are displayed correctly.



Mein Prüfling

Datum: 09.02.2010
Zeit: 12:56:51

Messergebnisse

Typ Information:	Typ Familie:	2MEASUREDATA
	Typ Code:	01 VariantMeasur
	Typ-Beschreibung:	Sample handle measurement data
	Systemnummer:	1

Seite: 1 von 2

Nr.	Seriennummer	Ergebnis	Modus	Messwerte	Dauer
1	01 VariantMeasur001	PASS	AUTO	3	1,016 s

Schritt-Nr.	Schritt-Name	IP-Schritt	Messwert	Grenzen	Ergebnis
1	Set measurement data for the DUTs in test step..	IP_Meas1	1	1..1	PASS
2	Set measurement data for the DUTs in the second te	IP_Meas2	1	1..1	PASS
3	Shows the measurement data from step 1 and 2 .	IP_ShowMeas	1	1..1	PASS

Nr.	Seriennummer	Ergebnis	Modus	Messwerte	Dauer
2	01 VariantMeasur002	FAIL	AUTO	1	0 s

Schritt-Nr.	Schritt-Name	IP-Schritt	Messwert	Grenzen	Ergebnis
1	Set measurement data for the DUTs in test step..	IP_Meas1	0	1..1	FAIL

Nr.	Seriennummer	Ergebnis	Modus	Messwerte	Dauer
3	01 VariantMeasur003	PASS	AUTO	3	1,016 s

Schritt-Nr.	Schritt-Name	IP-Schritt	Messwert	Grenzen	Ergebnis
1	Set measurement data for the DUTs in test step..	IP_Meas1	1	1..1	PASS
2	Set measurement data for the DUTs in the second te	IP_Meas2	1	1..1	PASS
3	Shows the measurement data from step 1 and 2 .	IP_ShowMeas	1	1..1	PASS

Nr.	Seriennummer	Ergebnis	Modus	Messwerte	Dauer
4	01 VariantMeasur004	FAIL	SBS	2	0 s

Schritt-Nr.	Schritt-Name	IP-Schritt	Messwert	Grenzen	Ergebnis
1	Set measurement data for the DUTs in test step..	IP_Meas1	1	1..1	PASS
2	Set measurement data for the DUTs in the second te	IP_Meas2	0	1..1	FAIL

Nr.	Seriennummer	Ergebnis	Modus	Messwerte	Dauer
5	01 VariantMeasur005	PASS	AUTO	3	1,015 s

Schritt-Nr.	Schritt-Name	IP-Schritt	Messwert	Grenzen	Ergebnis
1	Set measurement data for the DUTs in test step..	IP_Meas1	1	1..1	PASS
2	Set measurement data for the DUTs in the second te	IP_Meas2	1	1..1	PASS
3	Shows the measurement data from step 1 and 2 .	IP_ShowMeas	1	1..1	PASS

Nr.	Seriennummer	Ergebnis	Modus	Messwerte	Dauer
6	01 VariantMeasur006	PASS	AUTO	3	1,016 s

Datum: 09.02.2010
Zeit: 14:28:49

Messergebnisse

Typ Code: 01 VariantMeasur

Seite: 2 von 2

Schritt-Nr.	Schritt-Name	IP-Schritt	Messwert	Grenzen	Ergebnis
1	Set measurement data for the DUTs in test step..	IP_Meas1	1	1..1	PASS
2	Set measurement data for the DUTs in the second te	IP_Meas2	1	1..1	PASS
3	Shows the measurement data from step 1 and 2 .	IP_ShowMeas	1	1..1	PASS

Figure 9-11 Measurement Results

23. Save your report via the ➡ **menubar** ➡ **File** ➡ **Save as** under a new name.

10. DLL

The acronym DLL stands for Dynamic Link Library. A DLL can include a program code, data and resources in any combination. The portable executable file format is usually used for this purpose. A DLL serves the purpose of reducing the required storage space on the hard disk or main memory. But other application fields are also possible. An example of using a DLL:

The program code, which is used by multiple applications, can be summarized in a DLL and stored on the hard disk. The advantage is that the program code only has to be read once into memory and not from every single application that wants to access it.

☐ Exercise 10-1
Dealing with DLLs

This exercise should shed some light on how to deal with DLLs. The Kernel32.dll, which is found in ➡ **Windows directory** under subdirectory ➡ **System32** should be used. This DLL contains a function to generate a beep via the speaker inside your computer. The Kernel32.dll needs to be included in your sequence and the beep function to be introduced to the system. Using this function, implement a sound pattern.

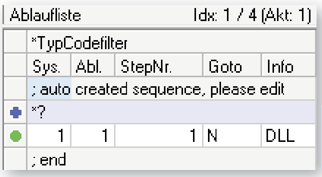
Step by step:

1. Log on as administrator
   Navigate from the toolbar Password. Press the F4 function key or select Administrator and type in admin.

2. Create a new family type with a variant, set this to 'active' and save the data.

Example:	Family Type:	15DLL
	Code Type:	01DLL
	Information Type:	Kernel32

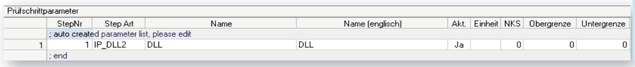
3. Edit the sequence list and the test step parameter. All you need is one test step. Enter a new name for the 'step type' in order to generate one new interpreter step.



Ablaufliste	Idx: 1 / 4 (Akt: 1)

	*TypCodefilter				
	Sys.	Abl.	StepNr.	Goto	Info
	; auto created sequence, please edit				
✚	*?				
●	1	1	1	N	DLL
	; end				

Figure 10-1
Sequence List



Prüfschrittparameter

	StepNr	Step Art	Name	Name (englisch)	Akt.	Einheit	NKS	Obergrenze	Untergrenze
	; auto created parameter list, please edit								
1	1	IP_DLL2	DLL	DLL	Ja		0	0	0
	; end								

Figure 10-2
Test step parameters

4. Select the variant from step 3 via ➡ **Typeselect** and start the sequence.

5. Edit the interpreter step:
The beep function to be used from the **Kernel32.dll** delivers a real value after execution. In order to assign this value, you should first create a variable of type real.

6. If a DLL is integrated into an interpreter step, the DLL should be made known to the system and is assigned with a so-called alias. Via this alias, the DLL can be called upon in the interpreter. With the procedure ➡ **DLL.OPEN** (alias and path in which the DLL is stored in) the DLL is opened and managed by the internal alias.

Syntax:

```
DLL.Open ('Kernel32', 'C:\Windows\System32\Kernel32.dll');
```

The DLL can now be addressed by the name Kernel32.

7. The procedure ➡ **DLL.REGISTER()** enables functions and procedures to be found in a DLL and verifies the sequence step by applying a name using an alias.

Syntax:

```
DLL.REGISTER ('Kernel32', 'Beep', 'S:I,I:L');
```

The beep function contained in the DLL Kernel32 is introduced to the system. In most cases, when calling upon functions or procedures from a DLL, variables must be transferred. This must also be specified when registering the beep com mand since it depends on the types of data to be transferred, in this case by specifying ➡ **'S:I,I:L'**. Through this information, the call format of the function / procedure is defined:

The first letter specifies the call type, here three types are differentiated.
➡ **P** for a Pascal call, ➡ **C** for a C-Call and ➡ **S** for a standard call. The call type must correspond to the routine's call type in the DLL.

The definition of the call type is followed by a colon (:). The codes for each para meter are separated by a comma and can be obtained from a list in the interpreter help. The return value is separated from the parameters by a colon, which also follows the indication of the type.

Thus, for the syntax:

```
DLL.REGISTER ('Kernel32', 'Beep', 'S:I,I:L');
```

following explanation:

The beep function from the DLL Kernel32 is introduced to the system. Calling upon the function is done via a standard call. Two parameters of type integer are transferred, and the beep function returns a Boolean type value (this corresponds to a logic value). The integer frequency (pitch) in Hz and tone duration in mil liseconds is then sent to the function.

8. The system is now aware of the DLL of the Kernel32 and beep function in 'pitch'. The beep command can now be called and the return value of the variables assigned. The invocation of a command (function or procedure) is done with the function ➡ **DLL.Call** ().
Through this task, the previously registered function can be called upon from the also previously opened DLL. In this function, the parameters defined in the registration are transferred.

Syntax:

```
Value := DLL.Call ('Kernel32', 'Beep', 3500, 50);
```

The beep function is called upon from the Kernel32 DLL, to run with the frequency 3500 Hz and duration of 50ms. The return value is transferred to the variable value.

By repeatedly calling upon the beep command, sound patterns can be created with altered frequencies and durations.

Syntax:

```
    Value := DLL.Call ('Kernel32', 'Beep', 3500, 50);
    Value := DLL.Call ('Kernel32', 'Beep', 4500, 50);
    Value := DLL.Call ('Kernel32', 'Beep', 5500, 50);
    …
```

9. At the end of your interpreter step, close the DLL using the procedure ➡ **DLL.CLOSE().** When the command is executed, all registrations are deleted. This means that when a new instruction is to be carried out after the DLL. CLOSE command, then a new registration is required for the DLL to be opened and instructed.

Syntax:

```
  DLL.CLOSE ('Kernel32');
```

The DLL with the internal alias Kernel32 is closed and the registration of the beep command is deleted.

Source code: *IP_DLL.IPS*

```
  var
    Value : Real;

  step
  //**************Open the DLL:********************************
    DLL.Open ('Kernel32', 'C:\Windows\System32\Kernel32.dll');
  //***************************************************************

  //**************Register the Beep command:*****************
    DLL.REGISTER ('Kernel32', 'Beep', 'S:I,I:L');
  //***************************************************************

  //**************Access the Beep command :*********************
    Value := DLL.Call ('Kernel32', 'Beep', 3500, 50);
    Value := DLL.Call ('Kernel32', 'Beep', 4500, 50);
    Value := DLL.Call ('Kernel32', 'Beep', 5500, 50);
    Value := DLL.Call ('Kernel32', 'Beep', 6500, 50);
    Value := DLL.Call ('Kernel32', 'Beep', 5500, 50);
    Value := DLL.Call ('Kernel32', 'Beep', 4500, 50);
    Value := DLL.Call ('Kernel32', 'Beep', 3500, 50);
  //***************************************************************

  //*********Close the DLL and set the values:**********
    DLL.CLOSE ('Kernel32');
    SetValue (0);
  //***************************************************************

  end.
```

10. Save your interpreter step and restart the sequence. Now, the sound of the generated sequence of notes (pitch) should be heard through the internal speaker. Please note that after a certain frequency level, sounds will no longer be transmitted from the speakers. The frequency range of human hearing is about 20 to 20,000 Hz.

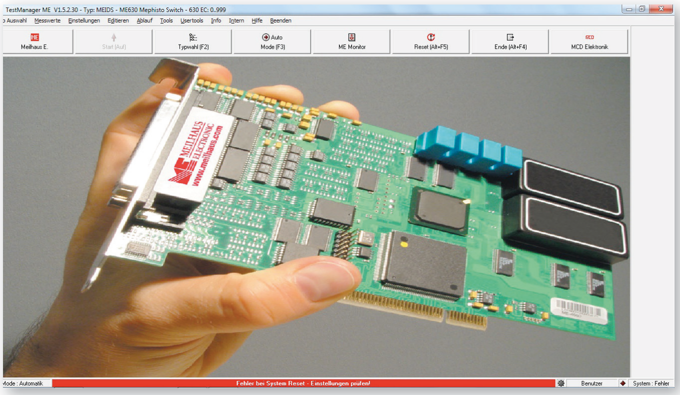
11. TestManager ME

The TestManager CE provides the perfect integration of the ME-iDS driver concept made by Measurement Computing Corporation. This software allows the user to the simple and rapid entry into the Meilhaus ME-iDS system allows drivers.

To use this part of the book eff ective, can you please upload the MCD TestManager ME www.mcd-elektronik.de/deutsch/meil.html down under. On this side there is the free trial version available.

Install the TestManager ME on your PC. To do this, follow the description to install the TestManager CE at the beginning of the book (see Chapter 1 introduction).
Reboot the system, as used by the CE version, with double the shortcut on your desktop.

The development environment TestManager ME


TestManager ME V1.5.2.30 - Typ: MEIDS - ME630 Mephisto Switch - 630 EC: 0..999

Mode : Automatik

Fehler bei System Reset - Einstellungen prüfen!

Benutzer

System : Fehler

Figure 11-1 Program window

The program window of TestManager ME is similar to the structure of the TestManager CE window. Major differences lie in the structure of the menu and the toolbar this new menu items or buttons are created. In the menu bar user tools anchored the point again, while the toolbar with a Meilhaus E. -, ME one monitor and one MCD-button has been fitted. Operation and menu navigation are identical in both versions, therefore, at this received will not be elaborated on this (see appendix - Differences in the menu of TestManager ME).

For more information about the company‘s products please contact Measurement Computing Corporation over the button ➡ **Meilhaus E.**, In the case of an existing internet connection directs to the homepage. Get information about products, drivers and software, references to books and literature, etc.

The button ➡ **MCD Elektronik** guides you to the MCD website, which you have already downloaded the trial version.

The ME is about the Monitor tool ➡ME button on the toolbar to monitor reach.

The special TestManager ME version was created only for ease of instruction and is not further developed. The card is fully support the MEILHAUS In TestManager CE contained and can be used directly.

Appendix

System requirements

- Windows 7®, Vista®, XP®, 2000® oder NT®
- Program directory on local drive
- Write-in access in the program directory
- Approx. 25 MB free hard disk
- 32 MB RAM
- VGA screen (640x480 Pixels)
- Pentium ® PC or compatible
- Installed hardware dependant on equipment
- Optional: installed network
- Optional: installed Borland Database Engine (BDE)

These values are minimum specifications; the computer can of course be better equipped. The program itself sets no files outside of its program directory. One exception is files in paths which are specifically stated (e.g. for measured value storage on the network).
The program can also be started on regular (office) PC's, if the access to the unavailable hardware in the program settings is switched off.
The program supports Windows XP style when enabled in the operating system and when the Help is activated on the desktop page of the default setting.

Useful Web Links for TestManager Users

www.mcd-elektronik.com	MCD Elektronics website. Information about TestManager CE, downloading the demo version, as well as the tutorial.
www.meilhaus.com	Distributor of cards and associated measurement technology products.