Programming Manual

METTLER TOLEDO MultiRange

Weighing terminals ID20 / ID20 TouchScreen

Printed in Germany.

1. Contents

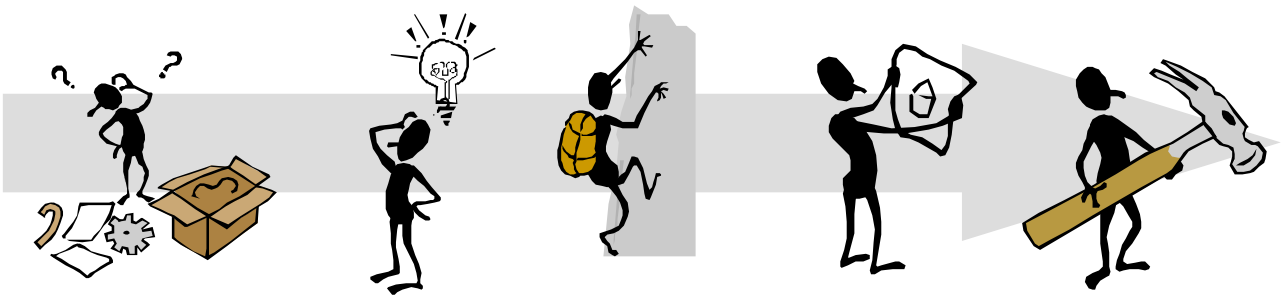
2. Introduction

2.1. This manual

The goal of this programming manual is to help you learn all necessary issues about weighing with the ID20 as fast as possible. It shows you how easy it is, to write applications for the ID20. Although it is a programming manual with very basic examples, this is not a handbook for programming newcomers.

When you start working with the ID20, we recommend to go through all the chapters step by step in the existing order. After that you ought to be fully aware of the weighing specific hard- and software and the ID20´s easy programming.

The chapters of this manual are built up in this order:



Introduction	Theory	Practice	Documentation	ID20-Tools
What's up ?	What to know about Weighing Guidelines, ID20 weighing specific Hard- and Software	How to program the ID20 with the different programming languages	All the programming commands to control the ID20	Useful applications, examples and ID20 – Tools

2.2. Operation fields

The ID20 represents the integration of an industrial standard PC architecture and a weight- and measures approved weighing terminal, prepared to be used in harsh industrial environment. It can be used to:

- replace or enhance today's existing applications where separate PCs and weighing-terminals were used
- for completely new weighing solutions or
- simply as an industrial PC in all kinds of applications.

Functions are made easier and more readable than in other products to give non-weighing specialists a good base for application writing.

ID20

ID20-IPC*

* The Industrial-PC-Version of the ID20 is the ID20-IPC. The ID20-IPC is not equipped with an integrated weighing capability (no Weighing Interface and no secondary display). Hence you can program the IPC like every other computer, but without the weighing specific command set, described in this manual.

2.3. Programmers support

METTLER TOLEDO' s goal is to provide software developers with the combination of the well known standard-PC platform and an easy access to weighing specific data.

Software designers are supported with a collection of very comfortable software functions. Using these functions, you can rapidly create professional weighing applications. To support as many programmers as possible, METTLER TOLEDO offers various kinds of different software languages for different operating systems.

These functions will help the programmer in practically every weighing-specific operation. For example, all calculations of gross-, net- and tare-values or the setting of permissible boundaries are taken over by the functions. The return code tells the programmer easily, if the call was successful or why the command has not been executed.

Additionally, most subjects in the context of approval issues are handled by the METTLER TOLEDO hard- and software. Only very few regulations have to be observed in order to write applications that fulfill the approval requirements (see guidelines on the following page).

3. Important Guidelines

3.1. Applications subjected to legal control

The ID20 terminal is approved for applications subject to legal control.
Due to the innovative conception of the terminal, legal requirements to be met by the application software are easy to fulfill. Nevertheless, it is important to observe subjects relevant to legal control when handling the software commands.

For legal verification purposes it has to be possible at anytime to reconstruct all weighing results printed or registered. This data has to be stored in the ID20 internal alibi file, according to the procedures in this document. To allow a correct reconstruction of complete data sets, weighing results have to be printed or registered together with date and time.

3.2. Alibi file

One of the big benefits of the ID20 is that you do not need a paper printer for the documentation of weighing results in applications subject to legal control. Most printers cannot be used in harsh, filthy or wet environment and in addition handling of paper is critical.

For this purpose, a special file, called MEMORY.MTA has been put on the ID20-harddisk. Weighing results that are printed or registered in accordance to legal verification have to be stored in this internal alibi memory. The file has a special compressed format, so it is not possible to read this file with a standard editor or tool. Every record is secured separately with a high-security and ID20-unique check-sum, so any manipulation will be detected.
The only possibility to verify the alibi file is the "SCALE" option in the scale driver program LIGHT.EXE (see chapter "Alibi File Authentication" on page 80). The editor performs a self-test when starting up, so manipulations are detected.

The alibi memory is physically represented by the 24MB file "MEMORY.MTA" on the harddisk. The user and the software developer are responsible for the correct use and state of this file. The size of 24 MB results from the approval authority guideline which demands, that weighing results have to be stored for at least 3 months:

It is possible to perform every 12 seconds – up to three months, 24 h around the clock - a new print into the alibi file, without overwriting the first entry! If the capacity of the alibi file is reached, the oldest entry will be overwritten.

Please note: Access to

- **the scale-driver LIGHT.EXE,**
- **the operation of the scale,**
- **the editor program LIGHT SCALE to verify the certification and**
- **the stored values in the alibi file**

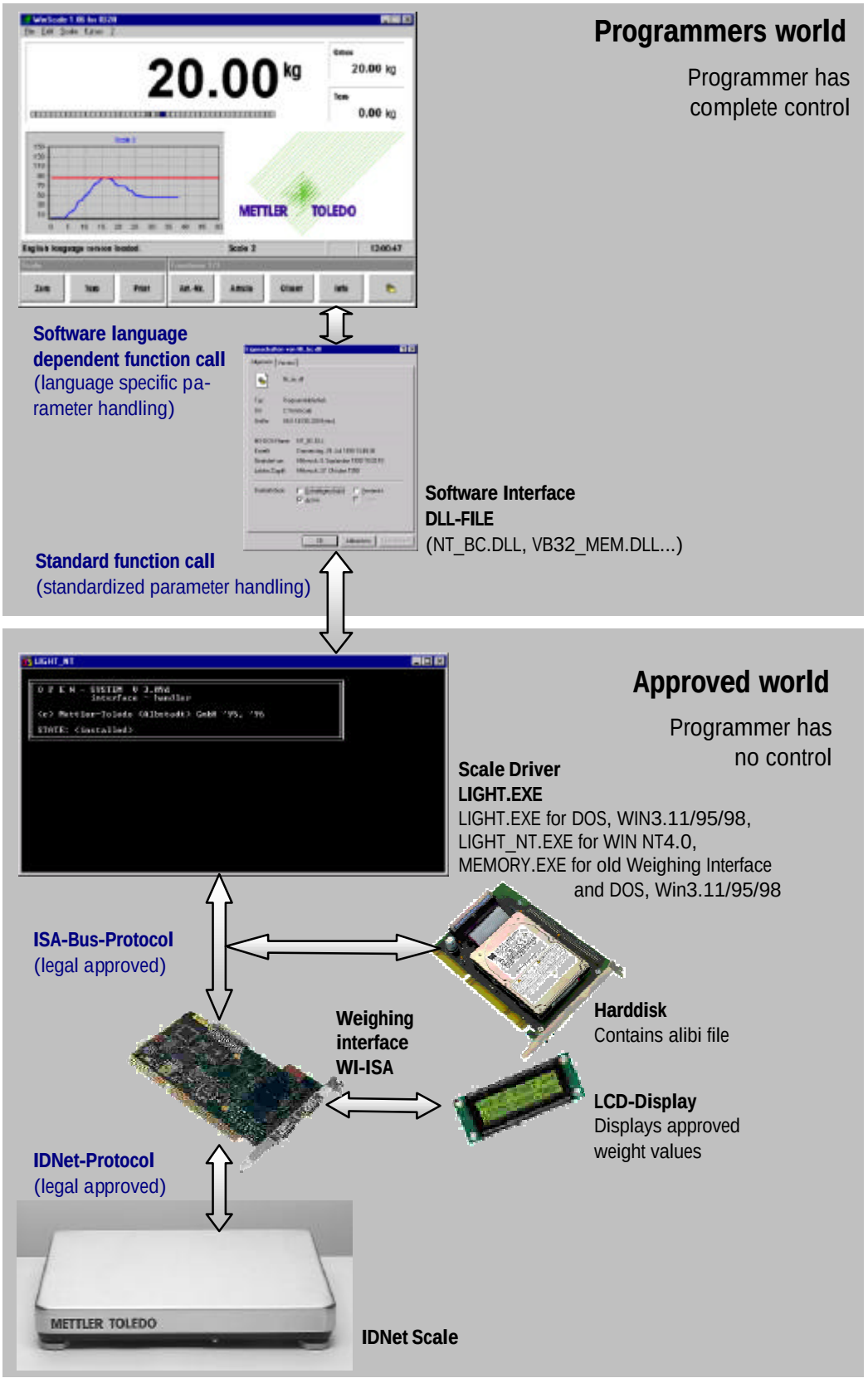
must be possible at any time !

The programmer for applications subject to legal control has to observe all regulations described in the chapter "Basic weighing commands".
It is not allowed to delete or modify the content of the file MEMORY.MTA !

4. ID20-Software architecture

4.1. Structure

The diagram shows the connected modules and the interfaces between the user application program and the weighing instrument:



4.1.1. Weighing interface software

The software on the weighing interface WI-ISA is responsible for the communication between the scale and the PC-based scale driver program. The software itself is located in an Electrically Erasable Programmable ROM (EEPROM). This technology makes it possible to download a new release direct from the ID20 harddisk into the weighing interface without opening the terminal.

In older Weighing Interface hardware versions the weighing interface software is located in an EPROM, so for software updates you have to open the ID20 to change the EPROM.

4.1.2. Scale driver program

Overview
The table shows an overview which kind of scale driver program has to be used, depending on the existing hardware and operating system:

	Old weighing interface 	New weighing interface WI-ISA 	
	MEMORY.EXE	**LIGHT.EXE**	**LIGHT_NT.EXE**
MS-DOS	X	X	-
Win 3.1/95/98	X	X	-
Win NT	-	-	X

For MS-DOS, Windows 3.11, Windows 95 and Windows 98:
When the operating system MS-Windows 3.11, 95 or 98 is used with the weighing interface WI-ISA, the scale driver program LIGHT.EXE has to run as a memory resident DOS-based background program.

The driver is responsible for the communication between the weighing interface software and the software interface (library) and the user application program respectively. The electrical communication between the new weighing interface and the CPU is done via a security protocol over the PC-ISA bus.

For the old weighing interface, the older scale driver program MEMORY.EXE has to be used ! MEMORY.EXE can not run under Windows NT.

For Windows NT:
When the operating system MS-Windows NT is used, the scale driver program has to run as a server task. Therefore, under Windows NT the program LIGHT_NT.EXE has to be used (call LIGHT_NT SERVER).

4.1.3. Software interface

The software library defines a standardized interface between the different programming languages and the scale driver program.

Internals:
All parameters to the scale driver have to be passed in the processor registers AX, BX, CX and DX. After that, a command-specific software interrupt has to be performed. AX contains the number of the function which has to be carried out. BX and DX (BX:DX) contain the pointer to an input string - if necessary. If integer values have to be handled, value 1 is in BX, value 2 in CX and value 3 is in DX. After the software interrupt, the return value can be found in the corresponding registers. During the software interrupt, the command blocks the application until the end. So, if e.g. a tare has to be carried out, the application stops until the tare function returns.

Note:
For the programmer in a high level language, these internals are not very important. All what's to do, is to link the correct software library (.LIB) into the project and to make sure that the correct dynamic link library (.DLL) is placed in the application program directory or in the windows system directory.

Programming hint:
Every result, coming from and going to the scale driver, are saved in a single static and space-saving buffer area. Therefore, the application programmer must save all results immediately in his application memory area. In other words, pointers to results should not be used because the memory content can change, so always copy results in variables: integers or fields for strings.

4.1.4. Application program

Applications can be programmed without any restriction on the ID20, which means it is possible to use all features of a modern PC like full graphics, full speed, multitasking (except multiple access to the weighing interface), internet connections, etc.

So it is possible to run e.g. a weighing program with a touch screen user interface in the foreground and an OLE or OPC connected application like MS-ACCESS or MS-EXCEL in the background as a data base.

Note:
Access to weighing data can be managed completely via the ID20-software commands, described in this manual. The only restriction concerns applications subjected to legal control.

If there is such a need for legal correct documentation, the programmer has to take care that each important weighing result is stored in the alibi file.

This has to be done by the call of two successive Weighing Interface commands: WI_WEIGHT() reads the actual weighing results and the proceeding WI_PRINT_VALUE() does the alibi print. Both commands do not need any parameters, so they are really easy to handle !

4.2. Driver integration for automatic start up

To avoid any problem when starting applications, it is necessary to install the specific scale driver for the concerned operating system in the correct way:

For MS-DOS, Windows 3.11, Windows 95 and Windows 98:
Under these DOS-based operating systems, LIGHT.EXE has to be placed and started in the autoexec.bat. It is not possible to start the driver program in a DOS-box under Windows, because the Windows application has no information about tasks in parallel running DOS-boxes.

Please note, that it is not allowed to run the LIGHT.EXE twice, because access to the weighing interface is only allowed by one scale driver program !

Please note that LIGHT.EXE is already installed correctly, if you receive a new ID20 with installed Windows 3.11, Windows 95 or Windows 98 from METTLER-TOLEDO!

For Windows NT:
LIGHT_NT.EXE has to be placed in the registry for automatic startup !
In new systems, this can be done by performing GENPORT.BAT, which is located in the root directory of the harddisk once.

Please note that LIGHT_NT.EXE is already correctly installed, if you receive a new ID20 with installed Windows NT from METTLER-TOLEDO!

4.3. Software download to weighing interface WI-ISA

If it is necessary to update a new weighing interface WI-ISA with a new firmware, this can be done by proceeding the following steps:

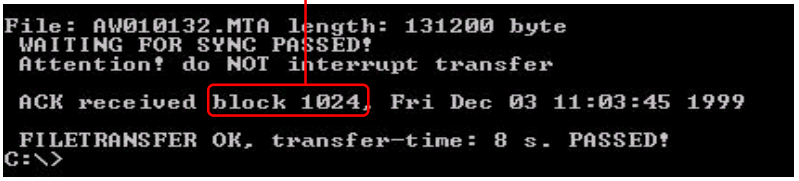
1. Go to MS-DOS or open a DOS-Box in Windows and select the root directory C:\ :

Note:
If your ID20 runs under Windows NT, please replace all `light` - calls in the following actions with `light_nt` - calls !

2. Close an eventually running scale driver program by entering: `C:\>light delete` or `C:\>light_nt delete`

3. Start the download of your new file AW010XXX (xxx is the Version No.)
In this example AW010132.MTA: `C:\>light download aw010132.mta_`

Now the screen informs about the download progress. The process is finished, after 1024 blocks are passed down into the weighing interface and the following success message appears:



During the download process, the message “Download active” is displayed on the approval LC - Display.

Note:
Never interrupt the download by switching power off or other manipulations! The boot loader in the weighing interface can be damaged, so the weighing interface has to be changed completely! If an error appears, exit from your DOS box, shut down windows and restart the ID20. Then, the boot loader mode will be active (see approval display). DOS will ask you for the down-load file, if windows starts up again, proceed another download.

4. Generate a new approval key. After a new software version was down-loaded, a new approval key has to be created. This key is used later to generate a secret authentication code for each record in the alibi file.
Note: The following call of light control will overwrite the alibi file MEMORY.MTA. So make sure, that you have saved the alibi file before !

   To make this key unique, the 7-digit serial number of the ID20 has to be entered with the control option of the Light driver:

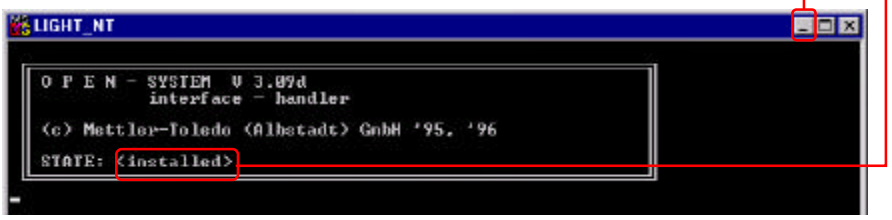
   ```
   C:\>light control xxxxxxx
   ```

   (replace xxxxxxx with ID20-Ser.Nr. !)

   After a short time the success message appears:

   ```
   <<< approval key now defined >>>
   C:\>
   ```

5. Ready ! Leave DOS-box by typing in `C:\>exit` to go back to Windows. Now shut down Windows and switch off the ID20. When switching on again, the little secondary LCD-Display shows the version of the new soft-ware for a few seconds.
   After booting the scale driver appears as shown, when installed correctly:
   (In Windows, click on the task symbol in the task bar to enlarge this window and click on to make it small again)



Note:
If no approval key was created with “LIGHT control”, you will retrieve this message. Then please go back to point 5 above.

5. How to program the ID20

5.1. Supported languages

METTLER TOLEDO supports software developers with the following programming languages / software interfaces for the different operating systems:

The mentioned terms are the corresponding ID20s directories:

Programming language	16 Bit systems Do not use for new projects		32 bit systems Recommended for new projects	
	MS-DOS	**MS-Win 3.11**	**MS-Win 98**	**MS-Win NT**
MS-Turbo Pascal	TP6_DOS	-	-	-
MS-C	MSC6_DOS	-	-	-
Borland C	BORL_C	-	-	-
MS-Visual Basic	-	WIN_VB	WIN95_VB	NT_VB
MS-Visual C	-	WIN_VC	WIN95_VC	NT_VC / -VC5
Borland C++ Builder	-	WIN_BC	WIN95_BC	NT_BC
Borland Delphi	-	DELPHI	DELPHI2	NT_DEL

Supported programming languages and where the libraries are located on the ID20.

Example:
You want to create an application with MS Visual Basic under Windows NT.
As a result, you will find all the necessary files in the directory :
After you have copied all these files in your own application directory, you can start the implementation !

Development and support for special operating systems like OS/2 and UNIX has been stopped approx. 1996. That means, that the software interfaces for these operating systems, which are still located on the ID20 harddisk, are not on the current state. Support or any modifications are not possible any more.

Note:
a) For new projects METTLER TOLEDO recommends the use of 32-bit software libraries due to better performance and optimum support !
b) Please do not use operating systems like OS/2 or UNIX for new projects !

5.2. How to build applications

Creating weighing specific applications for DOS or Windows is very easy with the use of the METTLER TOLEDO software interface.

All necessary files for each supported programming language can be found on the ID20 harddisk. You can get the directory names from the table on page 17.

In the following chapters, you'll find step-by-step instructions for program developers for two common programming languages:

The general processing should be similar if you are using other C- or Basic compilers / interpreters or if you are using Pascal / Delphi.

In case of problems, do not hesitate to contact your local METTLER TOLEDO dealer for support.

5.2.1. Example in Borland C++ Builder 4.0 (under Windows NT)

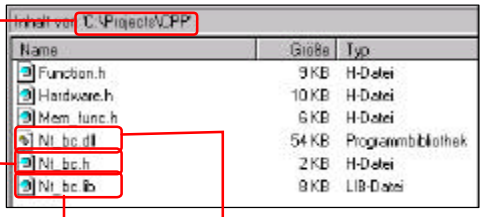
Preparing for work

First, we create a working directory for our new C++ project on the ID20 harddisk C:\PROJECTS\CPP.

In this directory, we have to copy the necessary METTLER TOLEDO Software-Interface files. To determine, where the files are located, a lookup in the software interface directory table on page 17 is helpful.

Due to the use of Borland C++ as programming language and Windows NT as operating system, the files located in the directory “C:\NT_BC“ are needed.

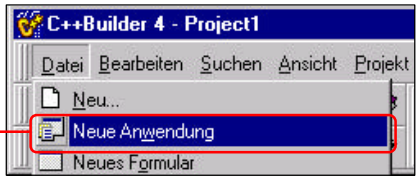
So the next preparation step is to copy these files into our project directory C:\PROJECTS\CPP:



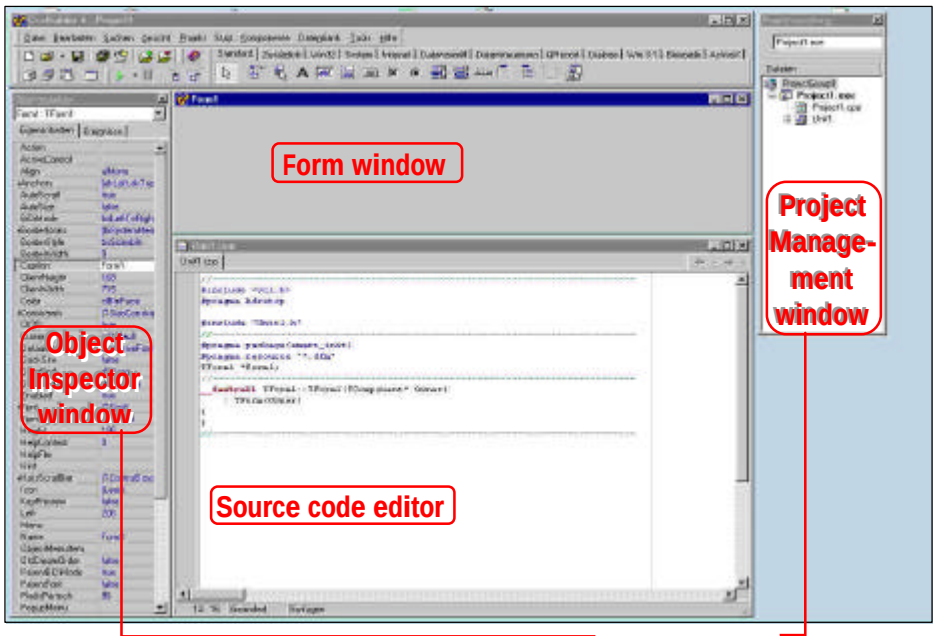
i The header file has to be included into our source code later. This file automatically includes the three other header files. The library file has to be linked into the new project, the dynamic link library will be loaded automatically during run time of the application.

Installing the Software Interface

Start the Borland C++ - Compiler. Select a new application by choosing FILE/NEW PROJECT...

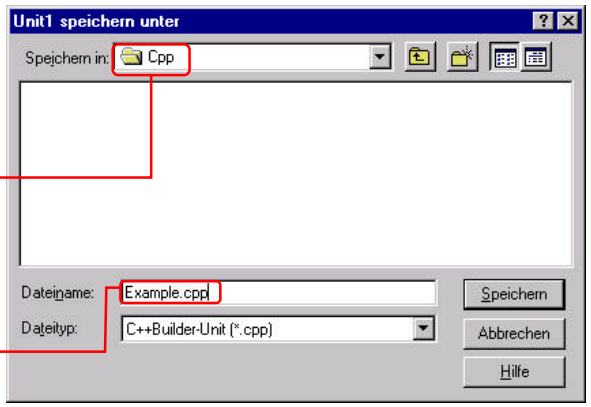


After arranging the windows, you will see a screen similar to this:



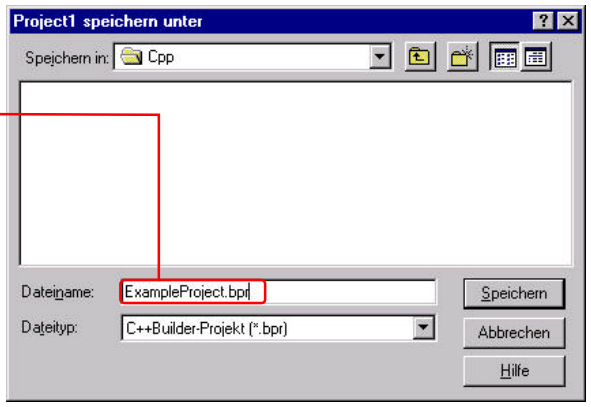
If not all windows are visible, open them with F11, Ctrl+Alt+F11, or use the window list editor by pressing Alt + O.

First of all, we will save the new project. Choose FILE/SAVE PROJECT...
As path, select the newly created project directory C:\PROJECTS\CPP:



The first window asks for a file name for the Unit file (our source code).
Please type in:

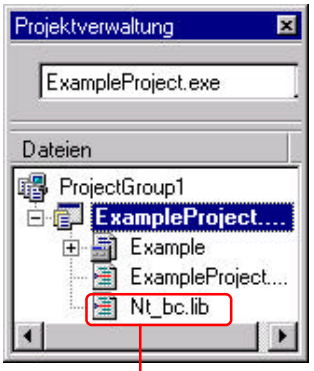
The second window asks for the project name, please type in:

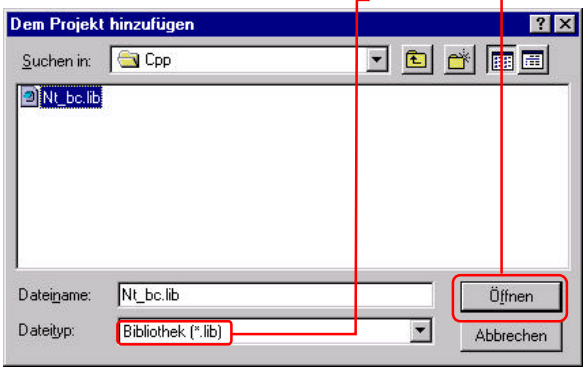


As the next step, we will link the METTLER TOLEDO library to our project.

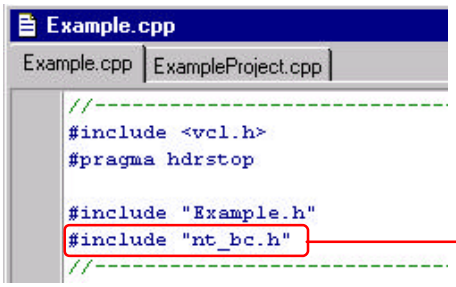
In the menu bar, choose PROJECT/ADD...

In the appearing file selector window choose “Library files” as file type. Now the library is displayed and can be added to our project with a click on :





The library is now included in our project.

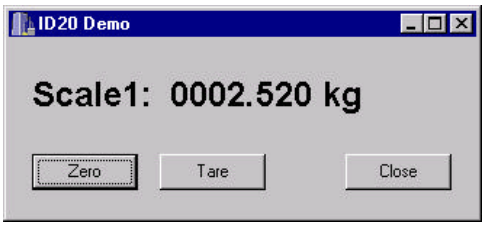


Before we can really start to implement our application code, we have do the last preparation step, which is to include the header file into our source unit. So, in the source code editor please type in:

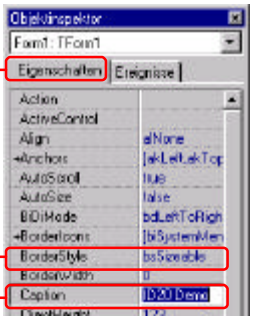
Ready ! Now we have constructed everything necessary to communicate with the METTLER TOLOEDO weighing interface. In the following steps, we will finally implement a small weighing application.

Programming the application in C++

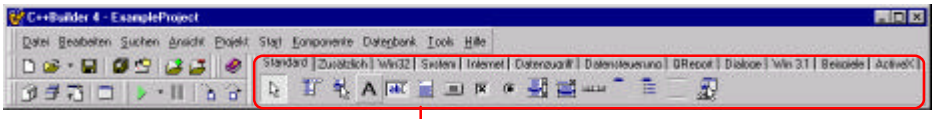
The goal is to create a little application, where we can see the connected scale and the actual weighing result. We need a possibility to tare the scale and set it to zero. Additionally, we want to have a button to close the application.



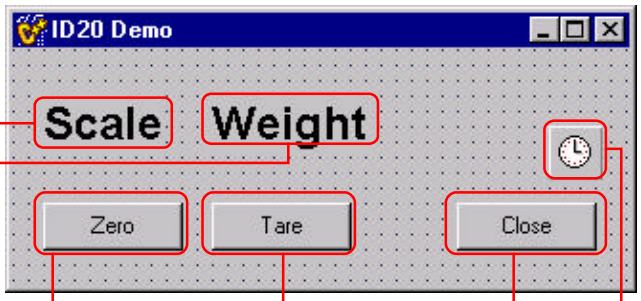
We select our form window FORM1 by clicking on it. In the object inspector, which now displays the properties, enter the settings as follows:



PROPERTY	SETTING
BORDER STYLE:	bsSizeable
CAPTION:	ID20 Demo
HEIGHT:	150
POSITION:	poDesktopCenter
WIDTH:	330



Now, out of the component palette, we add the following objects / components into the form and set their properties in the object inspector:



Label1

Property	Setting
Caption	Scale
Font	Arial, Bold, 18
Top	24
Left	16

Label2

Property	Setting
Caption	Weight
Font	Arial, Bold, 18
Top	24
Left	104

Button1

Property	Setting
Caption	Zero
Top	80
Left	16
Height	25
Width	73

Button2

Property	Setting
Caption	Tare
Top	80
Left	104
Height	25
Width	73

Button3

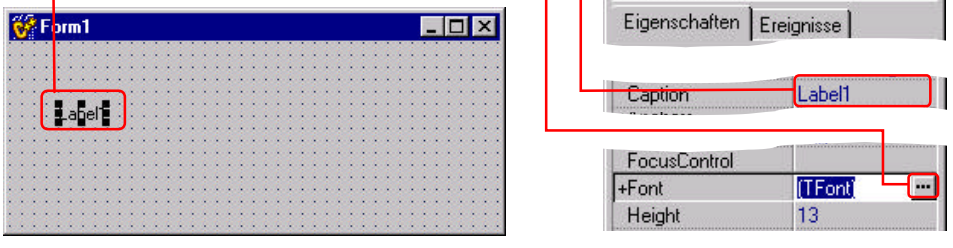
Property	Setting
Caption	Close
Top	80
Left	232
Height	25
Width	73

Timer1

Property	Setting
Enabled	True
Interval	250
Name	Timer1

Example:

To add the first label, click A in the component bar. The symbol changes its shape to A. Now, click on the desired place in the form area to place the new label. In the Object Inspector type in the caption and select the desired font with a click on:



Now, we will add own code to our application:

1. Double-click on the “Zero” button, and add this code to the OnClick handler:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        WI_ZERO();                              // set scale to zero
}
```

2. Double-click on the “Tare” button, and add this code to the OnClick handler:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        WI_SET_TARE("");                        // tare the scale
}
```

3. Double-click on the “Close” button, and add this code to the OnClick handler:

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    Close();
}
```

4. Because we want to display the updated weight value online, we have included the timer component. Double-click on the Timer symbol, and add this code to the OnTimer handler:

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    WI_WEIGHT();                                // read gross/net/tare
    Label2->Caption = WI__GET_NET();            // display net value
}
```

The function WI__GET_NET returns consistent weight values regarding the WI_WEIGHT call. WI_WEIGHT updates gross, net and tare weighing values. Please note the two underlines in WI__GET_NET !

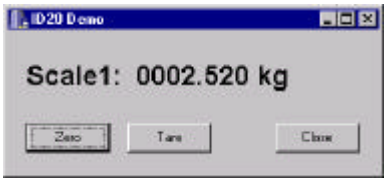
5. As last step, we want to determine at program start-up, which scale (scale number) is connected. Therefore, we can use the command WI_SCALE_INFO.

So activate Form1 with a click on it. In the Object Inspector, select the Events tab and double-click on the ONSHOW event. Add this code to the handler:

```
void __fastcall TForm1::FormShow(TObject *Sender)
{
    char cRet[5];
    strcpy(cRet, WI_SCALE_INFO());      // read scale(s) number
    // read scale numbers and actual scale string position:
    Label1->Caption = "Scale" + (String) cRet[cRet[0]-48] + ":";
}
```

The little calculation is necessary, because WI_SCALE_INFO returns the scale number in a string, where the first character is the position of the actual scale in the string. So, if e.g. “213” is returned, the scales with number 1 and 3 are connected (second and third character). The actual scale (first character), is on second position, that means in this case that the active scale has number 3.

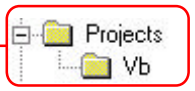
6. **Congratulations** - that’s all ! Don’t forget to save the project with FILE/SAVE ALL, and then start the compiler and linker with the F9 key. Enjoy your first self made ID20 C++ - project !



5.2.2. Example in Microsoft Visual Basic 4.0 (under Windows98)

Preparing for work

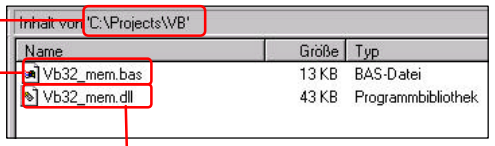
First, we create a working directory for our new Visual Basic project on the ID20 harddisk C:\PROJECTS\VB.



In this directory, we have to copy the necessary METTLER TOLEDO Software-Interface files. To determine, where the files are located, a lookup in the software interface directory table on page 17 is helpful.

Due to the use of MS Visual Basic as programming language and Windows 98 as operating system, the files located in the directory "C:\WIN95_VB" are needed.

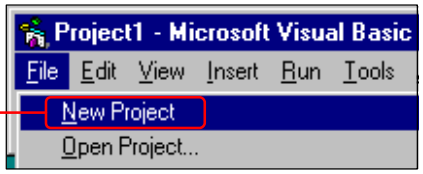
So the next preparation step is to copy these files into our project directory C:\PROJECTS\VB :



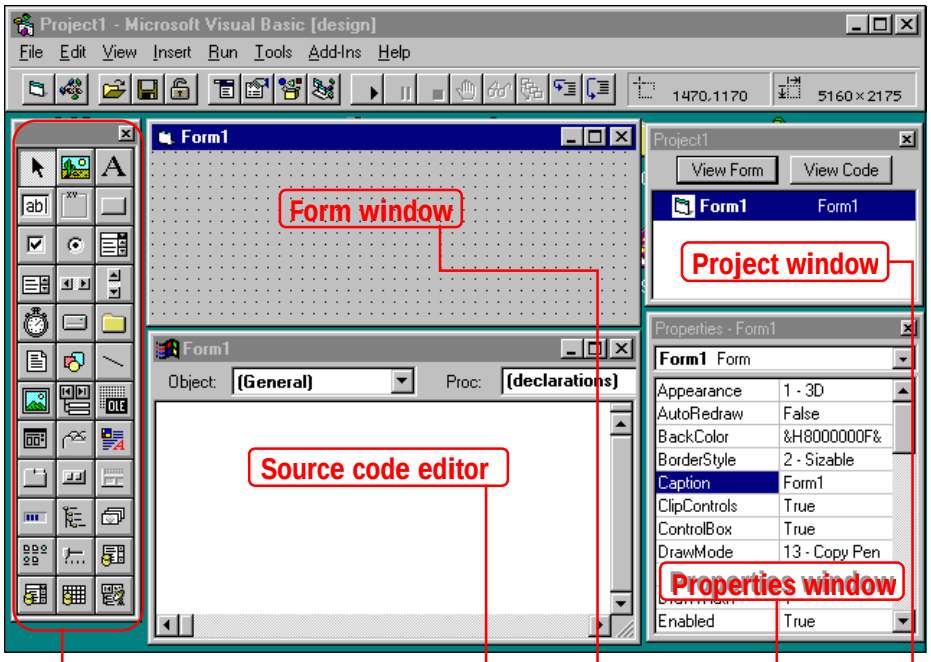
The .BAS file has to be linked into the new project. The next step is, to copy the dynamic link library into the Windows System directory C:\WINDOWS\SYSTEM . This DLL will be loaded later during run time of the application.

Installing the Software Interface

Start the MS - Visual Basic compiler. Select a new application by choosing FILE / NEW PROJECT :

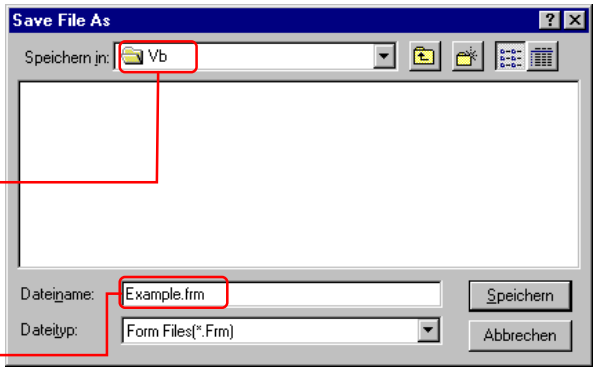


After arranging the windows, you will see a screen similar to this:



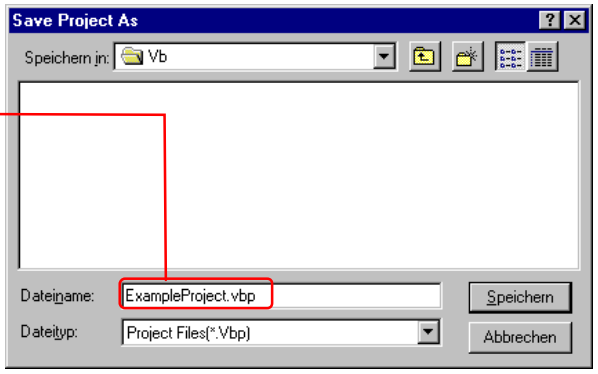
If not all windows are visible, open them with F7, Shift+F7, F4, & Ctrl+R.
If the toolbox isn't visible, select VIEW / TOOLBOX.

First of all, we will save the new project. Choose FILE / SAVE PROJECT.
As path, select the newly created project directory C:\PROJECTS\VB:



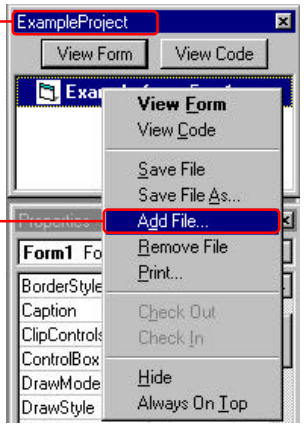
The first window asks for a file name for the Unit file (our source code).
Please type in:

The second window asks for the project name, please type in:

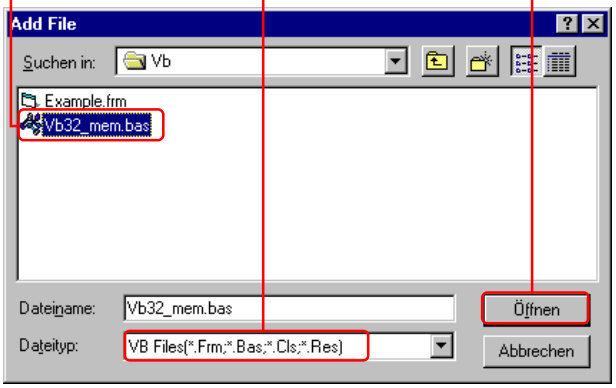


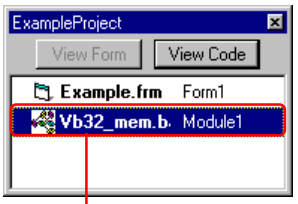
As the next step, we will include the METTLER TOLEDO software interface into our project:

In the properties window mark the first entry by a click on it. Now click with the *right* mouse button and choose:



In the appearing file selector window choose "VB files" as file type. Mark the file and add it to the project with a click on:





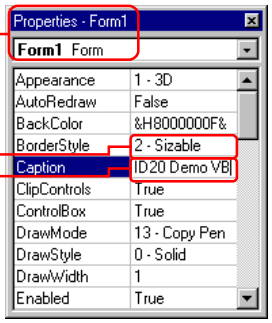
The Visual Basic Interface file is now included in our project.

Ready ! Now we have done everything necessary to communicate with the METTLER TOLOEDO weighing interface. In the following steps, we will finally implement a small weighing application.

Programming the application in VB

The goal is to create a little application, where we can see the connected scale and the actual weighing result. We need a possibility to tare the scale and set it to zero. Additionally, we want to have a button to close the application.

We select our form window FORM1 by clicking on it. In the object inspector, which now shows the properties, enter the settings as follows:



PROPERTY	SETTING
BORDER STYLE:	2 -Sizeable
CAPTION:	ID20 Demo VB
HEIGHT:	2000
WIDTH:	4400

Now, from the toolbox, we add the following objects / components into the form and set their properties in the properties window:

Label1

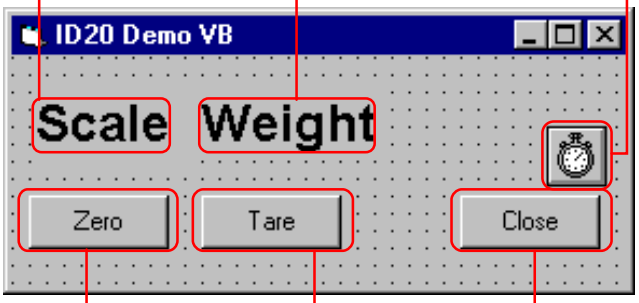
Property	Setting
AutoSize	True
Caption	Scale
Font	Arial,Bold,18
Top	24
Left	16

Label2

Property	Setting
AutoSize	True
Caption	Weight
Font	Arial,Bold,18
Top	24
Left	104

Timer1

Property	Setting
Enabled	True
Interval	250
Name	Timer1



Button1

Property	Setting
Caption	Zero
Top	80
Left	16
Height	25
Width	73

Button2

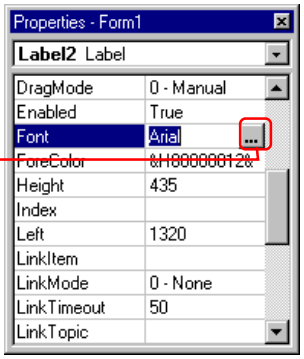
Property	Setting
Caption	Tare
Top	80
Left	104
Height	25
Width	73

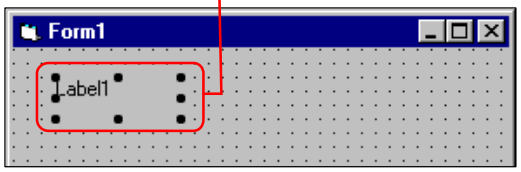
Button3

Property	Setting
Caption	Close
Top	80
Left	232
Height	25
Width	73

Example:

To add the first label, select A in the toolbox. Now, click on the desired place in the form area and hold the left mouse button down. Draw a rectangle to place the new label. In the properties window type in the caption and select the font with a click on :





Now, we will add own code to our application:

1. Double-click on the “Zero” button, and add this code to the Click handler:

```
Private Sub Command1_Click()
       WI_ZERO                ‘set scale to zero
End Sub
```

2. Double-click on the “Tare” button, and add this code to the Click handler:

```
Private Sub Command2_Click()
       WI_SET_TARE ("")       ‘tare the scale
End Sub
```

3. Double-click on the “Close” button, and add this code to the Click handler:

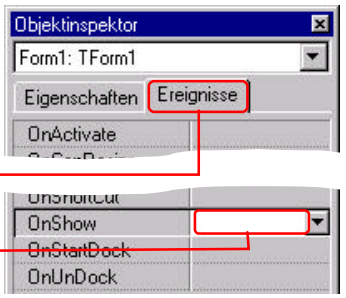
```
Private Sub Command3_Click()
       End                    'Close application
End Sub
```

4. Because we want to display the updated weight value online, we have included the timer component. Double-click on the Timer symbol, and add this code to the OnTimer handler:

```
Private Sub Timer1_Timer()
       WI_WEIGHT                      'read gross/net/tare
       Label2.Caption = WI__GET_NET  'display net value
End Sub
```

The function WI__GET_NET returns consistent weight values regarding the WI_WEIGHT call. WI_WEIGHT updates gross, net and tare weighing values. Please note the two underlines in WI__GET_NET !

5. As last action, we want to determine at program start-up, which scale (scale number) is connected. Therefore, we can use the command WI_SCALE_INFO.



So activate Form1 with a click on it. In the Object Inspector, select the Events tab and double-click on the ONSHOW event:

```
Private Sub Form_Load()
       cScaleInfo = WI_SCALE_INFO      'read scale(s) number
       cActive = Mid(cScaleInfo, 1, 1) 'acive scale position
       Label1.Caption = "Scale" & Mid(cScaleInfo, CInt(cActive) + 1, 1)
End Sub
```

The little calculation is necessary, because WI_SCALE_INFO returns the scale number in a string, where the first character is the position of the actual scale in the string. So, if e.g. “213” is returned, the scales with number 1 and 3 are connected (second and third character). The actual scale (first character), is on second position, that means in this case that the active scale has number 3.

6. **Congratulations** - that’s all ! Don’t forget to save the project with FILE/SAVE PROJECT, and then start the compiler and linker with the F5 key.
Enjoy your first self made ID20 VB - project !

5.3. Programming guidelines for Windows

If you develop a new program for MS-Windows, it is good style to consider the Microsoft Windows programming guidelines. You can find actual information in the internet: http://msdn.microsoft.com/winlogo/default.asp.

It is not only to "serve" Microsoft standards, but to serve your customers.
A user will "feel at home" in your new application, when he has the "Look & Feel" like in other standard Windows programs he knows, like MS-Word™ or MS-Excel™.

Win95 Logo
Your application should fulfill at least the specifications for the Win95-Logo.
If the program is certified to be conform, you can get the Win95-Logo from Microsoft. For this, please check the following points:

- The program files have to be in the PE-format (Portable Executable Format)
- The Microsoft conventions for user interfaces have to be observed
- The application should run under Win95 and WinNT if technical possible. Actually, this is not really feasible when you integrate weighing functions, because of the programming language - and operating system - dependant construction of the software interface.
- Provide the application with context menus for the right mouse button.
- Use the registry data base instead of ini-files (ini files are used only for very specific program data)
- Provide your symbols for data types and applications in the 16x16 and 32x32 format
- Use the standard navigation elements and standard dialog windows
- Use the standard system colors
- Support long file names
- Support Plug & Play

6. Software commands

6.1. General system commands

In this section you'll find all commands to get general information like weighing system availability and about software versions, installed on the ID20 system.

Command	Short description
1. MEMTRACE	Checks, if scale driver is already installed
2. SYS_VERSION	Version of scale driver software
3. WI_VERSION	Version of weighing interface software

6.1.1. MEMTRACE

Function	**Checks, if scale driver is already installed.** MEMTRACE returns ok, if the scale driver program runs properly. Use this function to decide if your application can start or not.	
Note	MEMTRACE is the most important command at all. If the scale driver program LIGHT.EXE or LIGHT_NT.EXE isn't running, you never get any weighing results !	
Syntax	**C** **VC++**	int MEMTRACE (void);
	BASIC **VBASIC**	Function MEMTRACE () As Integer
	PASCAL **DELPHI**	Function MEMTRACE: integer
Return	Integer	0: Scale driver is not running 1: Scale driver is running

Example

```
#include c__mem.h

void main ()
{
      int Ret;            // return value

      Ret = MEMTRACE();
      if (Ret)
      {
            printf("Driver loaded\n");
            //... start of application
      }
      else
      {
            printf("Driver not loaded\n");
            return;
      }
      //...
}
```

6.1.2. SYS_VERSION

Function

Version of scale driver software
Reads the software version of the scale driver program LIGHT.EXE or LIGHT_NT.EXE

Syntax

C **VC++**	char* SYS_VERSION (void);
BASIC **VBASIC**	Function SYS_VERSION () As String
PASCAL **DELPHI**	Function SYS_VERSION: string

Return

String — Max. length 8 bytes
Example: “3.09”

Example

```
#include c__mem.h

void main ()
{
      char Ret[9];        // return value

      strcpy (Ret, SYS_VERSION());
      printf(“Driver version: %s\n”, Ret);
      return;
}
```

6.1.3. WI_VERSION

Function

Version of weighing interface software
Reads the software version of the program in the Weighing Interface WI-ISA

Syntax

C **VC++**	char* WI_VERSION (void);
BASIC **VBASIC**	Function WI _VERSION () As String
PASCAL **DELPHI**	Function WI _VERSION: string

Return

String	Max. length 16 bytes Example: “AW01-0-0120 ” The last 3 digits show the version: 1.20

Example

```
#include c__mem.h

void main ()
{
      char Ret[17];       // return value

      strcpy (Ret, WI_VERSION());
      printf(“WI version: %s\n”, Ret);

      return;
}
```

6.2. Basic weighing commands

In this section you'll find commands to get and handle weighing results.

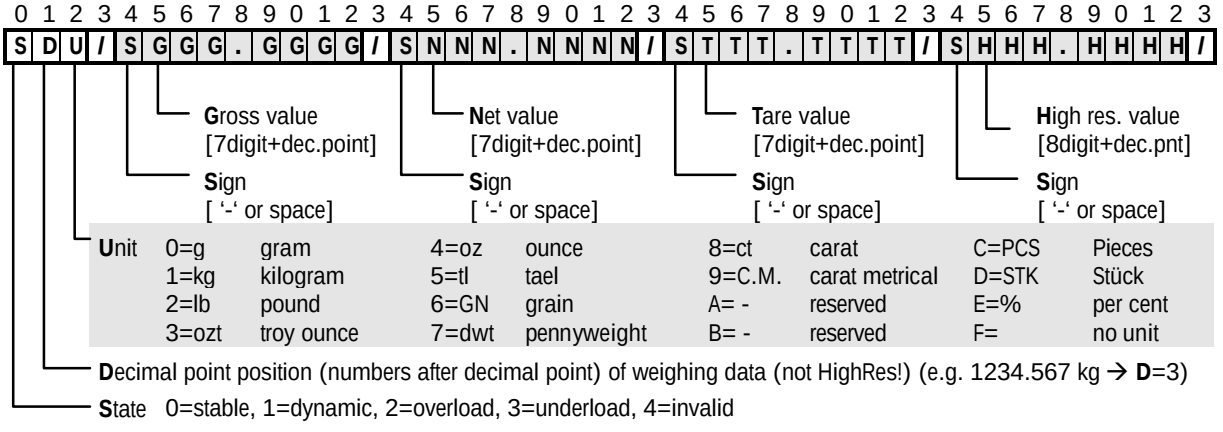
Command	Short description
1. WI_WEIGHT	Get complete weighing data set
2. WI_ZERO	Sets scale to zero
3. WI_SET_TARE	Tares the scale
4. WI_SCALE	Change actual active scale
5. WI_SCALE_INFO	Get info about connected scales
6. WI__GET_WEIGHT_STATE	Get the actual scale state
7. WI__GET_GROSS/NET/TARE	Get a single weight value (values are consistent)
8. WI_GROSS/NET/TARE	Get a single weight value (values are not consistent)
9. WI__GET_HIGHRES	Get high resolution value (cons.)
10. WI_HIGHRES	Get high resolution value (not co.)
11. WI_PRINT_VALUE	Generate a harddisk alibi print
12. WI__GET_TIME_STAMP	Get time stamp of last WI_WEIGHT
13. WI__GET_AUTHENTIFICATION	Get auth. code of last alibi print
14. WI_USER_DATA	Enters user data string for alibi file

6.2.1. WI_WEIGHT

Function	**Get complete weighing data set G/N/T/HR** Reads a complete weighing result data set with gross, net, tare and high-resolution value from the Weighing Interface WI-ISA.	
Note	After calling WI_WEIGHT, it is possible to get consistent single weight values by using the commands: WI__GET_GROSS, WI__GET_NET and WI__GET_TARE. Therefore you should not call WI_WEIGHT between one of these functions !	
Syntax	**C** **VC++**	char* WI_WEIGHT (void);
	BASIC **VBASIC**	Function WI _WEIGHT () As String
	PASCAL **DELPHI**	Function WI _WEIGHT: string
Return	String	Length: 44 bytes Example: "031/ 123.4567/ 000.0000/ 000.0000/123.45678/"
Example		

```
#include c__mem.h

void main ()
{
      char cRet[45];      // return value

      strcpy (cRet, WI_WEIGHT());
      printf("Result: %s\n", cRet);

      return;
}
```

Weighing data set structure:



6.2.2. WI_ZERO

Function

Sets scale to zero

Sets the Gross-, Net-, Tare- and High-Resolution value of the actual scale to zero if the scale is in the permissible zero setting range.

Info

The zero setting range of standard scales is ± 20% of full load range. That means, that a 600kg scale, which starts up with a max. load of 120kg, can be set to zero.

Syntax

C **VC++**	char* WI_ZERO (void);
BASIC **VBASIC**	Function WI _ ZERO () As String
PASCAL **DELPHI**	Function WI _ ZERO: string

Return

String Length: 2 bytes

"ZB"	Zero setting done correctly
"Z+"	Zero setting out of positive range
"Z-"	Zero setting out of negative range
"ZI"	Command cannot be executed

Example

```
#include c__mem.h

void main ()
{
      char cRet[3];       // return value

      strcpy (cRet, WI_ZERO());
      printf("Result: %s\n", cRet);

      return;
}
```

6.2.3. WI_SET_TARE

Function

Tares the scale
Tare with actual weight or with a user-specified value.
Option: erase the actual tare value.

Syntax

C **VC++**	char* WI_SET_TARE (char* val);
BASIC **VBASIC**	Function WI_SET_TARE (ByVal val As String) As String
PASCAL **DELPHI**	Function WI_SET_TARE (val: string) : string

Input

String — Tare the scale with value "val"

"" (empty)	Tare with actual weight
" " (space)	Erase actual tare value
" xx.yyy uuu"	Tare with value: xx.yyy and with unit: uuu

Note

If you tare manually, do not forget the space before the value, the space between value and unit and the unit after the value:
Wrong: WI_SET_TARE ("1.20");
Correct: WI_SET_TARE (" 1.20 kg");

Return

String — Length: 2 bytes

"TB"	Tare setting done correctly
"T+"	Tare setting out of positive range
"T-"	Tare setting out of negative range
"TI"	Command cannot be executed

Example

```
#include c__mem.h

void main ()
{
      char cRet[3];       // return value

      // tare with actual weight
      strcpy (cRet, WI_SET_TARE(""));
      printf("Result: %s\n", cRet);

      // erase actual tare value
      strcpy (cRet, WI_SET_TARE(" "));
      printf("Result: %s\n", cRet);

      // tare with 100 kg
      strcpy (cRet,
            WI_SET_TARE(" 100.00 kg"));
      printf("Result: %s\n", cRet);

      return;
}
```

6.2.4. WI_SCALE

Function	**Change active scale** Changes from the actual scale to the scale with the specified number.		
Syntax	**C** **VC++**	char* WI_SCALE (int num);	
	BASIC **VBASIC**	Function WI_SCALE (ByVal num As Integer) As String	
	PASCAL **DELPHI**	Function WI_SCALE (num: integer) : string	
Input	Integer	Scale number where to switch 1 Change to scale with number 1 2 Change to scale with number 2 3 Change to scale with number 3	
Return	String	Length: 2 bytes "SB" Scale switch executed correctly "SI" Scale not available	

Example

```
#include c__mem.h

void main ()
{
      char cRet[3];       // return value

      // select scale 2
      strcpy (cRet, WI_SCALE(2));
      printf("Result: %s\n", cRet);

      return;
}
```

6.2.5. WI_SCALE_INFO

Function	**Get info about connected scales** Returns actual active scale and all available scales.		
Syntax	**C** **VC++**	char* WI_SCALE_INFO (void);	
	BASIC **VBASIC**	Function WI_SCALE_INFO () As String	
	PASCAL **DELPHI**	Function WI_SCALE_INFO : string	
Return	String	Length: 4 bytes Format: “PXYZ” P Position of actual active scale (‘1’=X, ‘2’=Y, ‘3’=Z) X First connected scale number Y Second connected scale number Z Third connected scale number If no scale is connected, a zero and three spaces is returned (“0 “).	

Example

```
#include c__mem.h

void main ()
{
      char cRet[5];       // return value

      // Example:
      // Two scales are connected:
      // - first scale has No 2
      // - second scale has No 3
      // - scale 3 is actually active

      // Return: “223”
      // Means: Scales 2 and 3 are available
      // Second of this scales is active(No3)

      strcpy (cRet, WI_SCALE_INFO());
      printf(“Result: %s\n”, cRet);

      return;
}
```

6.2.6. WI__GET_WEIGHT_STATE

Function	**Get the actual scale state** Returns the state of the actual active scale.	
Note	The scale state is not the actual scale state at the moment when the command has been called ! The result corresponds to the scale state at the last call of WI_WEIGHT !	
Syntax	**C** **VC++**	char* WI__GET_WEIGHT_STATE (void);
	BASIC **VBASIC**	Function WI__GET_WEIGHT_STATE () As String
	PASCAL **DELPHI**	Function WI__GET_WEIGHT_STATE : string
Return	String	Length: 2 bytes 'D' Scale value was dynamic ' ' (Space) Scale value was stable 'I+' Scale value was overloaded 'I-' Scale value was underloaded 'II' Scale value was invalid

Example

```
#include c__mem.h

void main ()
{
      char cRet[3];       // return value

      strcpy (Ret, WI__GET_WEIGHT_STATE());
      printf("Result: %s\n", cRet);

      return;
}
```

6.2.7. WI__GET_GROSS, WI__GET_NET, WI__GET_TARE (consistent)

Function	**Get a single weight value (values are consistent)** Returns gross, net or tare value as a complete string, including sign and unit. The value is consistent to a previous call of WI_WEIGHT.		
Note	The weight value is not the actual value at the moment when the command has been called ! The result corresponds to the weight value at the last call of WI_WEIGHT, that means that the formula Net = Gross + Tare is always true. Please note the two underlines in WI__... !		
Syntax	**C** **VC++**	char* WI__GET_GROSS (void); char* WI__GET_NET (void); char* WI__GET_TARE (void);	
	BASIC **VBASIC**	Function WI__GET_GROSS () As String Function WI__GET_NET () As String Function WI__GET_TARE () As String	
	PASCAL **DELPHI**	Function WI__GET_GROSS : string Function WI__GET_NET : string Function WI__GET_TARE : string	
Return	String	Length: 13Bytes Format: “sxxx.yyyy zzz” (s=Sign, x.y=Value, z = Unit) Example: “-1234.000 kg ”	

Example

```
#include c__mem.h

void main ()
{
      char cRet[14];      // return value

      WI_WEIGHT();        // get all values

      strcpy (cRet, WI__GET_GROSS());
      printf(“Result: %s\n”, cRet);

      strcpy (cRet, WI__GET_NET());
      printf(“Result: %s\n”, cRet);

      strcpy (cRet, WI__GET_TARE());
      printf(“Result: %s\n”, cRet);

      return;
}
```

6.2.8. WI_GROSS, WI_NET, WI_TARE (not consistent)

Function	**Get a single weight value (values are not consistent)** Returns actual gross-, net- or tare value as a complete string, including sign and unit. The value is not consistent to a previous call of WI_WEIGHT !		
Note	The weight value is the actual value at the moment when the command has been called ! The formula Net = Gross + Tare is not true, if the scale value changes between the calls !		
Syntax	**C** **VC++**	char* WI_GROSS (void); char* WI_NET (void); char* WI_TARE (void);	
	BASIC **VBASIC**	Function WI_GROSS () As String Function WI_NET () As String Function WI_TARE () As String	
	PASCAL **DELPHI**	Function WI_GROSS : string Function WI_NET : string Function WI_TARE : string	
Return	String	Length: 13Bytes Format: “sxxx.yyyy zzz” (s=Sign, x.y=Value, z = Unit) Example: “-1234.000 kg ”	

Example

```
#include c__mem.h

void main ()
{
      char cRet[14];      // return value

      strcpy (cRet, WI_GROSS());
      printf(“Result: %s\n”, cRet);

      strcpy (cRet, WI_NET());
      printf(“Result: %s\n”, cRet);

      strcpy (cRet, WI_TARE());
      printf(“Result: %s\n”, cRet);

      return;
}
```

6.2.9. WI__GET_HIGHRES (consistent)

Function	**Read high resolution weight value (consistent)** Returns the high resolution weight value, including sign and unit. The value is consistent to a previous call of WI_WEIGHT.	
Note	The weight value is not the actual value at the moment when the command has been called ! The result corresponds to the weight value at the last call of WI_WEIGHT. Please note the two underlines in WI__... !	
Syntax	**C** **VC++**	char* WI__GET_HIGHRES (void);
	BASIC **VBASIC**	Function WI__GET_HIGHRES () As String
	PASCAL **DELPHI**	Function WI__GET_HIGHRES : string
Return	String	Length: 14 Bytes Format: “sxxx.yyyyy zzz” (s=Sign, x.y=Value, z = Unit) Example: “-123.45678 kg ”

Example

```
#include c__mem.h

void main ()
{
      char cRet[15];      // return value

      WI_WEIGHT();        // get all values

      strcpy (cRet, WI__GET_HIGHRES());
      printf(“Result: %s\n”, cRet);

      return;
}
```

6.2.10. WI_HIGHRES (not consistent)

Function	**Read high resolution weight value** Returns the high resolution weight value, including sign and unit. The value is not consistent to a previous call of WI_WEIGHT.		
Note	The weight value is the actual value at the moment when the command has been called !		
Syntax	**C** **VC++**	char* WI _HIGHRES (void);	
	BASIC **VBASIC**	Function WI_HIGHRES () As String	
	PASCAL **DELPHI**	Function WI_HIGHRES : string	
Return	String	Length: 14 Bytes Format: “sxxx.yyyyy zzz” (s=Sign, x.y=Value, z = Unit) Example: “-123.45678 kg ”	

Example

```
#include c__mem.h

void main ()
{
      char cRet[15];      // return value

      strcpy (cRet, WI_HIGHRES());
      printf(“Result: %s\n”, cRet);

      return;
}
```

6.2.11. WI_PRINT_VALUE

Function	**Generate a harddisk alibi print** Generates an alibi print on the harddisk.
Note	In applications subject to legal control this function replaces the need for an external alibi printer ! It is very important to observe following rule: 1. Get complete weighing data set with WI_WEIGHT 2. Check if value is stable: WI__GET_WEIGHT_STATE 3. If stable, generate alibi print with WI_PRINT_VALUE Please note, that WI_PRINT_VALUE only writes stable weighing results into the alibi file !
Info	If you want to use the same data which was written into the alibi file for special use in your application (e.g. for data bases, parallel documentation in a text file, printout on paper etc.), use the following commands after the call of WI_WEIGHT: 1. WI__GET_TIME_STAMP to get date and time 2. WI__GET_AUTHENTIFICATION to get key code 3. WI__GET_GROSS / NET / TARE for single results
Syntax	**C VC++** void WI_PRINT_VALUE (void);
	BASIC VBASIC Function WI_ PRINT_VALUE ()
	PASCAL DELPHI Function WI_ PRINT_VALUE

Example

```
#include c__mem.h

void main ()
{
      char cSta[3];       // stability
      char cTim[18];      // time stamp
      char cGro[14];      // gross value
      char cNet[14];      // net value
      char cTar[14];      // tare value
      int  iAut;          // auth. code
      WI_WEIGHT();        // read G/N/T
      strcpy (cSta, WI__GET_WEIGHT_STATE());

      // alibi print, if value was stable
      if (!strcmp (cSta, " "))
      {
            WI_PRINT_VALUE();
            iAut =
            WI__GET_AUTHENTIFICATION());
            strcpy (cTim,
            WI__GET_TIME_STAMP());

            strcpy (cGro, WI__GET_GROSS());
            strcpy (cNet, WI__GET_NET());
            strcpy (cTar, WI__GET_TARE());
      }
      return;
}
```

6.2.12. WI__GET_TIME_STAMP

Function	**Get time stamp of last WI_WEIGHT call** Reads time stamp (time and date) of the last data set returned by the call of WI_WEIGHT. Use this information for authentication of receipts, delivery notes etc.	
Info	Please note that the command is related to the moment of the last call of WI_WEIGHT – not to the moment, when you call WI__GET_TIME_STAMP ! ⇨ Please note the two underlines WI_ _ GET	
Syntax	**C** **VC++**	char* WI__GET_TIME_STAMP (void);
	BASIC **VBASIC**	Function WI__GET_TIME_STAMP () As String
	PASCAL **DELPHI**	Function WI__GET_TIME_STAMP: string
Return	String	Length: 17 bytes Format: “DD.MM.YY hh:mm:ss” DD Day hh hour MM Month mm minute YY Year ss second

Example

```
#include c__mem.h

void main ()
{
      char cSta[3];       // stability
      char cTim[18];      // time stamp
      int  iAut;          // auth. code

      WI_WEIGHT();        // read G/N/T
      strcpy (cSta,
      WI__GET_WEIGHT_STATE());

      // alibi print, if value was stable
      if (!strcmp (cSta, “ ”))
      {
            WI_PRINT_VALUE();
            iAut =
            WI__GET_AUTHENTIFICATION());

            strcpy (cTim,
            WI__GET_TIME_STAMP());
      }

      return;
}
```

6.2.13. WI__GET_AUTHENTIFICATION

Function	**Get authentication code of last alibi print** Reads authentication key of the last measuring value, saved with the last call of WI_WEIGHT. You can use this information for additional authentication of receipts, delivery notes etc.	
Syntax	**C** **VC++**	int WI__GET_AUTHENTIFICATION (void);
	BASIC **VBASIC**	Function WI__GET_ AUTHENTIFICATION () As Integer
	PASCAL **DELPHI**	Function WI__GET_ AUTHENTIFICATION : integer
Return	Integer	Length: 2 bytes Example –46804

Example

```
#include c__mem.h

void main ()
{
      char cSta[3];       // stability
      char cTim[18];      // time stamp
      int  iAut;          // auth. code

      WI_WEIGHT();        // read G/N/T
      strcpy (cSta,
      WI__GET_WEIGHT_STATE());

      // alibi print, if value was stable
      if (!strcmp (cSta, " "))
      {
            WI_PRINT_VALUE();

            iAut =
            WI__GET_AUTHENTIFICATION());

            strcpy (cTim,
            WI__GET_TIME_STAMP());
      }

      return;
}
```

6.2.14. WI_USER_DATA

Function	**Enters user data string for storing in alibi file** Additional user data can be written into the alibi file (MEMORY.MTA). This string is protected against manipulation.		
Note	The string will be saved in the alibi file not until when WI_PRINT_VALUE() is called !		
Syntax	**C** **VC++**	void WI_USER_DATA (char * val);	
	BASIC **VBASIC**	Function WI_USER_DATA (ByVal As String)	
	PASCAL **DELPHI**	Function WI_USER_DATA (val: string)	
Input	String	Length: max. 20 Bytes, incl. zero sign Example: “Terminal-No. 001”	

Example

```
#include c__mem.h

void main ()
{
      // write additional data in alibi file
      WI_USER_DATA(“Terminal-No. 001”));
      WI_PRINT_VALUE();   // make alibi print

      Return;
}
```

6.3. Special weighing commands

In this section you'll find commands for advanced weighing options .

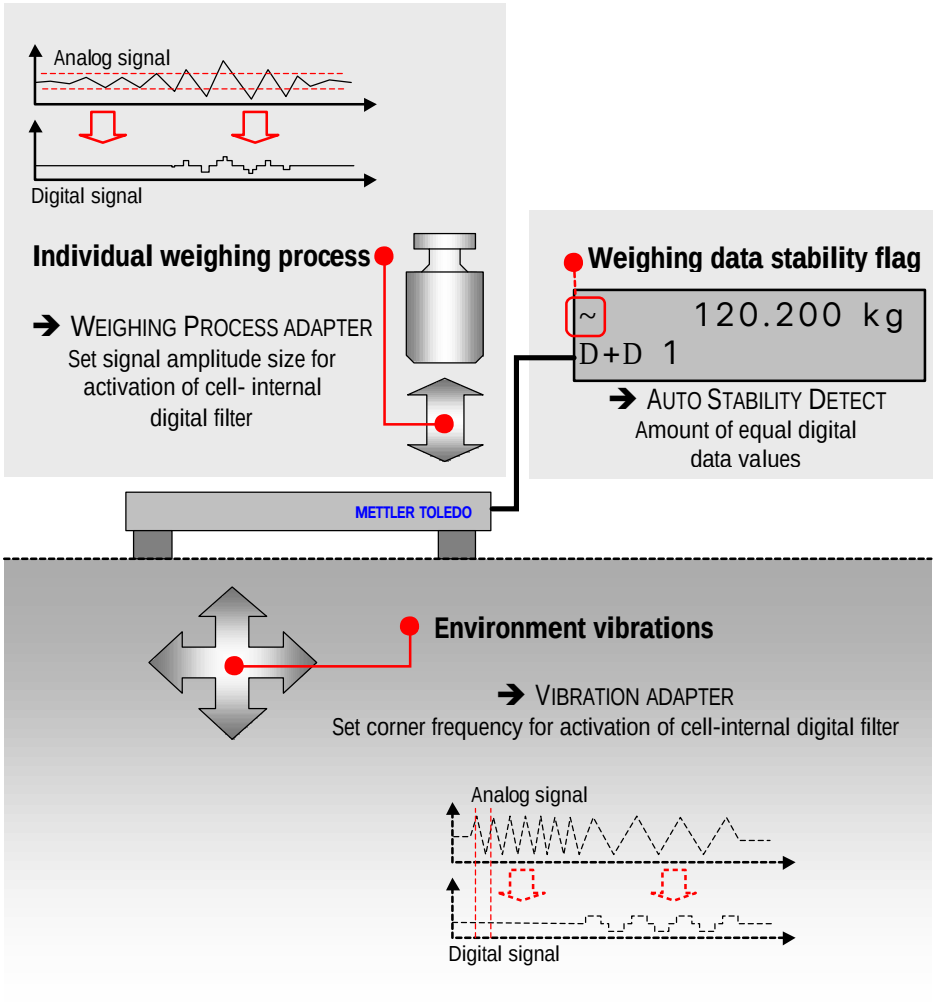
Command	Short description
1. WI_ADAPT_VIBRATION	Set environment vibration parms
2. WI_ADAPT_PROCESS	Set weighing process parameters
3. WI_ADAPT_STABILITY_DETECT	Set stability flag parameters
4. WI_SCALE_MODE	Set scale update rate
5. WI_IDENTBLOCK	Read Ident-block of actual scale
6. WI_AUTOTARE_ON	Switch auto tare function on
7. WI_AUTOTARE_OFF	Switch auto tare function off
8. WI_AUTOZERO_ON	Switch auto zero function on
9. WI_AUTOZERO_OFF	Switch auto zero function off
10. WI_RESTART_ON	Switch auto restart function on
11. WI_RESTART_OFF	Switch auto restart function off

6.3.1. Additional information to weighing filter commands:

To optimize your customer-specific application, concerning speed and resolution it is possible to use the WI_ADAPT - command set.

1.	WI_ADAPT_VIBRATION	Vibration adapter
2.	WI_ADAPT_PROCESS	Process adapter
3.	WI_ADAPT_STABILITY_DETECT	Automatic stability detection

Overview:



Understanding the WI_ADAPT command set

6.3.2. WI_ADAPT_VIBRATION

Function	**Set environment vibration parameters** Adapts load cell to customers environment vibration.	
Note	When using low values for the vibration adapter, the scale works fast and is very sensitive for external effects. When using high values for the vibration adapter, the scale works slow and is not sensitive for external effects.	
Info	You can read the possible parameters and the actual setting by using the command WI_IDENTBLOCK(“10”).	
Syntax	**C** **VC++**	char* WI_ADAPT_VIBRATION (int num);
	BASIC **VBASIC**	Function WI _ ADAPT_VIBRATION (ByVal num As Integer) As String
	PASCAL **DELPHI**	Function WI _ ADAPT_VIBRATION (num: integer) : string
Input	Integer	Value of vibration adapter (* = Standard) 1 Calm environment 2* Normal environment 3 Disturbed environment
Return	String	Length: 3 bytes “AVB” Setting done, OK “AVI” Setting was invalid, no change

Example

```
#include c__mem.h

void main ()
{
      char cRet[4];       // return value

      // calm, stable environment
      strcpy (cRet, WI_ADAPT_VIBRATION(1));
      printf(“Result: %s\n”, cRet);

      // high disturbed environment
      strcpy (cRet, WI_ADAPT_VIBRATION(3));
      printf(“Result: %s\n”, cRet);

      return;
}
```

6.3.3. WI_ADAPT_PROCESS

Function	**Set weighing process parameters** Adapts load cell to individual customers weighing goods, e.g. for oscillating objects.		
Info	You can read the possible parameters and the actual setting by using the command WI_IDENTBLOCK(“11”).		
Syntax	**C** **VC++**	char* WI_ADAPT_PROCESS (int num);	
	BASIC **VBASIC**	Function WI _ ADAPT_ PROCESS (ByVal num As Integer) As String	
	PASCAL **DELPHI**	Function WI _ ADAPT_ PROCESS (num: integer) : string	
Input	Integer	Value of process adapter (* = Standard) 1 Fine dosing (Fluids, fine bulk goods) 2* Universal weighing 3 Absolute weighing (Solids, animals)	
Return	String	Length: 3 bytes “APB” Setting done, OK “API” Setting was invalid, no change	

Example

```
#include c__mem.h

void main ()
{
      char cRet[4];       // return value

      // weighing oscillating goods
      strcpy (cRet, WI_ADAPT_PROCESS(1));
      printf(“Result: %s\n”, cRet);

      // weighing calm goods
      strcpy (cRet, WI_ADAPT_PROCESS(3));
      printf(“Result: %s\n”, cRet);

      return;
}
```

6.3.4. WI_ADAPT_STABILITY_DETECT

Function

Set stability flag parameters
Adjusts weighing speed and reproducibility of load cell.

Info

The ASD-function sets the stability flag, depending of the amount of equal digital weighing data, coming up from the load cell. The number of equal data can be set with the input parameter.
You can read the possible parameters and the actual setting by using the command WI_IDENTBLOCK("12").

Syntax

C **VC++**	char* WI_ADAPT_STABILITY_DETECT (int num);
BASIC **VBASIC**	Function WI _ ADAPT_ STABILITY_DETECT (ByVal num As Integer) As String
PASCAL **DELPHI**	Function WI _ ADAPT_ STABILITY_DETECT (num: integer) : string

Input

Integer — Value of vibration adapter (* = Standard)

	Speed	*Reproducibility*
0	Automatic Stability Detect disabled (ASD disabling is only possible in non legal applications !)	
1	Fast	Good
2*	⇑	⇓
3		
4	Slow	Very good

Return

String — Length: 3 bytes

"ASB" Setting done, OK
"ASI" Setting was invalid, no change

Example

```
#include c__mem.h

void main ()
{
      char cRet[4];       // return value

      // fast weighing
      strcpy (cRet,
            WI_ADAPT_STABILITY_DETECT(1));
      printf("Result: %s\n", cRet);

      // slow weighing
      strcpy (cRet,
            WI_ADAPT_STABILITY_DETECT(4));
      printf("Result: %s\n", cRet);

      return;
}
```

6.3.5. WI_SCALE_MODE

Function	**Set scale update rate** Sets scale data update rate.		
Info	This function may be used only with load cells, which support update rate setting (e.g. PikBrick). You can read the possible parameters and the actual setting by using the command WI_IDENTBLOCK(“14”).		
Syntax	**C** **VC++**	char* WI_SCALE_MODE (char* val);	
	BASIC **VBASIC**	Function WI_SCALE_ MODE (ByVal val As String) As String	
	PASCAL **DELPHI**	Function WI_SCALE_ MODE (val: string) : string	
Input	String	“5” + “xx” (“5”: fix, “xx”: update rate in updates/sec) Example: “515”: 15 updates/second	
Return	String	Length: 2 bytes “MB” Setting done, OK “MI” Setting was invalid, no change or command not supported.	

Example

```
#include c__mem.h

void main ()
{
      char cRet[3];       // return value
      char cUpd[26];      // actual updates

      strcpy(cUpd, WI_IDENTBLOCK(“14”));
      // returns e.g. “10 6 10 15 20”
      // means: actual update rate is 10/s
      // possible settings: 6, 10,15 and 20

      // now set 6 updates/s
      strcpy (cRet, WI_SCALE_MODE(“56”));
      printf(“Res: %s\n”, cRet);// ok

      // now set 20 updates/s
      strcpy (cRet, WI_SCALE_MODE(“520”));
      printf(“Res: %s\n”, cRet);// ok

      // now set 13 updates/s
      strcpy (cRet, WI_SCALE_MODE(“513”));
      printf(“Res: %s\n”, cRet);// failed

      return;
}
```

6.3.6. WI_IDENTBLOCK

Function

Read Ident block of actual scale
Returns basic load cell parameters.

Syntax

C **VC++**	char* WI_IDENTBLOCK (char* val);
BASIC **VBASIC**	Function WI_ IDENTBLOCK (ByVal val As String) As String
PASCAL **DELPHI**	Function WI_ IDENTBLOCK (val: string) : string

Input

String — Length: 3 bytes

“00”	Application of weighing platform
“01”	Country code
“02”	Language
“03”	First unit
“04”	Maximum load
“05”	Minimum load
“06”	Maximum tare
“07”	Maximum pre set tare
“08”	Proved resolution
“09”	Minimum reproducibility
“10”	Vibration adapter parameters
“11”	Weighing process adapter par.
“12”	Automatic stability control par.
“13”	Auto Zero state
“14”	Value update frequency
“15”	Software version of load cell
“16”	Load cell Identcode
“Ri”	Ranges and resolutions (i = 0..9)

Return

String — Maximum length: 25 bytes
Content depends of called function.

Example:

Input:	Return:
“04”	“60.000 kg”
“10”	”2123”
“15”	“IZ05-0-0222 ”

Example

```
#include c__mem.h

void main ()
{
      char cRet[26];      // return value

      // read first unit of load cell
      strcpy(cRet, WI_IDENTBLOCK(“03”));
      // returns e.g. “kg”
      printf(“Res: %s\n”, cRet);

      return;
}
```

6.3.7. WI_AUTOTARE_ON

Function	**Switch auto tare function on** Scale tares after weight is detected on the scale.	
Info	This command is processed by the Weighing-Interface, not the scale itself. If auto tare function is activated, the weighing-display shows a corresponding symbol  in the lower line:	
Syntax	**C** **VC++**	char* WI_AUTOTARE_ON (void);
	BASIC **VBASIC**	Function WI_ AUTOTARE _ON () As String
	PASCAL **DELPHI**	Function WI_ AUTOTARE _ON : string
Return	String	Length: 4 bytes “ATYB” Auto tare is activated “ATYI” Invalid, not executed

Example

```
#include c__mem.h

void main ()
{
      char cRet[5];       // return value

      // switch auto tare function on
      strcpy (cRet, WI_AUTOTARE_ON());
      printf(“Result: %s\n”, cRet);

      return;
}
```

6.3.8. WI_AUTOTARE_OFF

Function	**Switch auto tare function off** Scale does not tare automatically any more after weight is detected on the scale.		
Info	This command is processed by the Weighing-Interface, not the scale itself.		
Syntax	**C** **VC++**	Char* WI_AUTOTARE_OFF (void);	
	BASIC **VBASIC**	Function WI_ AUTOTARE _OFF () As String	
	PASCAL **DELPHI**	Function WI_ AUTOTARE _OFF : string	
Return	String	Length: 4 bytes “ATNB” Auto tare is deactivated “ATNI” Invalid, not executed	

Example

```
#include c__mem.h

void main ()
{
      char cRet[5];       // return value

      // switch auto tare function off
      strcpy (cRet, WI_AUTOTARE_OFF());
      printf(“Result: %s\n”, cRet);

      return;
}
```

6.3.9. WI_AUTOZERO_ON

Function	**Switch auto zero function on** Scale starts up automatically with zero. This eliminates drift effects of the load cell electronics.	
Info	Auto zero function can be switched on or off only with not approved scales. In approved scales, auto zero is always active ! The auto zero function only works if: 1. The scale displays zero 2. The zero point has to be corrected with max 0.5 e/s 3. The act. zero point is within ±2% of full load range	
Syntax	**C** **VC++**	char* WI_AUTOZERO_ON (void);
	BASIC **VBASIC**	Function WI_ AUTOZERO _ON () As String
	PASCAL **DELPHI**	Function WI_ AUTOZERO _ON : string
Return	String	Length: 4 bytes "AZYB" Auto zero is activated "AZYI" Invalid, not executed

Example

```
#include c__mem.h

void main ()
{
      char cRet[5];       // return value

      // switch auto zero function on
      strcpy (cRet, WI_AUTOZERO_ON());
      printf("Result: %s\n", cRet);

      return;
}
```

6.3.10. WI_AUTOZERO_OFF

Function	**Switch auto zero function off** Disables automatically zero setting feature at start up of non approved load cells.		
Info	Auto zero function can be switched on or off only with not approved scales. In approved scales, auto zero is always active !		
Syntax	**C** **VC++**	char* WI_AUTOZERO_OFF (void);	
	BASIC **VBASIC**	Function WI_ AUTOZERO _OFF () As String	
	PASCAL **DELPHI**	Function WI_ AUTOZERO _OFF : string	
Return	String	Length: 4 bytes "AZNB" Auto zero is deactivated "AZNI" Invalid, not executed	

Example

```
#include c__mem.h

void main ()
{
      char cRet[5];       // return value

      // switch auto zero function off
      strcpy (cRet, WI_AUTOZERO_OFF());
      printf("Result: %s\n", cRet);

      return;
}
```

6.3.11. WI_RESTART_ON

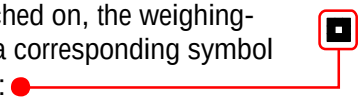
Function

Switch auto restart function on

Switch on the automatic function to restore zero point and tare value of a scale after a power off. The weight value after Power Up will be the same as before Power Down (if nothing has changed on the scale).

Info

If restart is switched on, the weighing-display shows a corresponding symbol in the lower line:



Syntax

C VC++	char* WI_RESTART_ON (void);
BASIC VBASIC	Function WI_ RESTART _ON () As String
PASCAL DELPHI	Function WI_ RESTART _ON : string

Return

String	Length: 4 bytes
	"RSYB" Auto restart function enabled
	"RSYI" Invalid, not executed

Example

```
#include c__mem.h


void main ()
{
      char cRet[5];       // return value

      // switch auto restart function on
      strcpy (cRet, WI_RESTART_ON());
      printf("Result: %s\n", cRet);

      return;
}
```

6.3.12. WI_RESTART_OFF

Function	**Switch auto restart function off** Switch off the automatic function to restore zero point and tare value of a scale after a power off. The weight value is set to zero after Power Up.		
Syntax	**C** **VC++**	char* WI_RESTART_OFF (void);	
	BASIC **VBASIC**	Function WI_ RESTART _OFF () As String	
	PASCAL **DELPHI**	Function WI_ RESTART _OFF : string	
Return	String	Length: 4 bytes “RSNB” Auto restart function disabled “RSNI” Invalid, not executed	

Example

```
#include c__mem.h

void main ()
{
      char cRet[5];       // return value

      // switch auto restart function off
      strcpy (cRet, WI_RESTART_OFF());
      printf(“Result: %s\n”, cRet);

      return;
}
```

6.4. System commands

In this section you'll find all commands concerning functions of the ID20 system itself.

Command	Short description
1. SYS_WI	Sets or reads special system modes
2. WI_BEEP	Generates a beep
3. WI_KEYBOARD_ON	Enable ID20 foil keyboard
4. WI_KEYBOARD_OFF	Disable ID20 foil keyboard
5. WI_SERVICE_AUTARK	Starts scale service mode

6.4.1. SYS_WI

Function

Sets or reads special system modes

Sets special parameters in the weighing interface Wi-ISA

Syntax

C **VC++**	char* SYS_WI (char func);
BASIC **VBASIC**	Function SYS_WI (ByVal func As String) As String
PASCAL **DELPHI**	Function SYS_WI (func : string) : string

Input

String

WI-Function and corresponding switch
Format: “FFS”

FF	S	Function
“00”	“0”	Keyboard repeat function off
“00”	“1”	Keyboard repeat function on
“01”	“0”	Returns ID20 serial number.
“07”	“0”	Disable BIU
“07”	“1”	Enable BIU
“!!”	“0”	Enable Quick weight Mode
“!!”	“1”	Disable Quick weight Mode

Return

String

Length depends of function, max. 13 bytes.
Format: “XFFS”
(“X”: Fix, “FF”: Function, “S”: Switch)
“XI” if addressed function was invalid
Example (successful returns)

FF	S	Return string:
“00”	“0”	“X000”
“01”	“0“	“X01 1234567”
“07”	“0”	“X070”
“!!”	“1”	“X!!1”

Example

```
#include c__mem.h

void main ()
{
      char cRet[14];      // return value
      char cSer[14];      // serial number

      // enable BIU
      strcpy (cRet, SYS_WI(“071”));
      printf(“Result: %s\n”, cRet);

      // disable BIU
      strcpy (cRet, SYS_WI(“070”));
      printf(“Result: %s\n”, cRet);

      // read ID20 serial number
      strcpy (cSer, SYS_WI(“010”));
      printf(“Serial number: %s\n”, cSer);

      return;
}
```

6.4.2. WI_BEEP

Function	**Generates a beep** Generates an acoustic signal (beep) with specified length on the weighing interface board.		
Syntax	**C** **VC++**	char* WI_BEEP (int iLen);	
	BASIC **VBASIC**	Function WI_BEEP (ByVal iLen As Integer) As String	
	PASCAL **DELPHI**	Function WI_BEEP (iLen: integer) : string	
Input	Integer	Signal length in steps of 10ms. Shortest time 10ms, longest time 2,55 s 1 Beep 10 ms 2 Beep 20 ms ... ... 255 Beep 2550 ms	
Return	String	Length: 2 bytes “BB” Command executed correctly “BI” Maximum time step exceeded	
Example			

```
#include c__mem.h

void main ()
{
      char cRet[3];       // return value

      // beep 100ms
      strcpy (cRet, WI_BEEP(10));
      printf(“Result: %s\n”, cRet);

      return;
}
```

6.4.3. WI_KEYBOARD_ON

Function	**Enable ID20 foil keyboard** Switches the ID20 foil keyboard on.		
Syntax	**C** **VC++**	char* WI_KEYBOARD_ON (void);	
	BASIC **VBASIC**	Function WI_ KEYBOARD_ON (ByVal cRes As String)	
	PASCAL **DELPHI**	Function WI_ KEYBOARD_ON : string	
Return	String	Length: 3 bytes "KYB" Keyboard function enabled "KYI" Invalid, not executed	
Example			

```
#include c__mem.h

void main ()
{
      char cRet[4];       // return value

      // switch keyboard on
      strcpy (cRet, WI_KEYBOARD_ON());
      printf("Result: %s\n", cRet);

      return;
}
```

6.4.4. WI_KEYBOARD_OFF

Function	**Disable ID20 foil keyboard** Switches the ID20 foil keyboard off		
Syntax	**C** **VC++**	char* WI_KEYBOARD_OFF (void);	
	BASIC **VBASIC**	Function WI_ KEYBOARD_OFF (ByVal cRes As String)	
	PASCAL **DELPHI**	Function WI_ KEYBOARD_OFF : string	
Return	String	Length: 3 bytes "KNB" Keyboard function disabled "KNI" Invalid, not executed	

Example

```
#include c__mem.h

void main ()
{
      char cRet[4];       // return value

      // switch keyboard off
      strcpy (cRet, WI_KEYBOARD_OFF());
      printf("Result: %s\n", cRet);

      return;
}
```

6.4.5. WI_SERVICE_AUTARK

Function	**Starts the scale service mode** After the start of the scale service mode, the weighing interface takes control over all actions until the user ends the service mode, that means that the command does not return until the user leaves the service mode with F11.	
Note	Neither the operating system nor the application has any control during the time in the service mode !	
Syntax	**C** **VC++**	void WI_SERVICE_AUTARK (void);
	BASIC **VBASIC**	Function WI_ SERVICE_AUTARK ()
	PASCAL **DELPHI**	Function WI_ SERVICE_AUTARK

Example

```
#include c__mem.h

void main ()
{
      // start the service mode

      WI_SERVICE_AUTARK());

      // now service mode is finished

      return;
}
```

6.5. Parallel I/O control

In this section you'll find all commands concerning the parallel I/O possibilities of the ID20 system, like the Option194* and the Binary Interface Unit* (BIU).

Command	Short description
1. OPT94_VERSION	**Read parallel I/O device info**
2. OPT94_WRITE	**Set outputs of parallel I/O device**
3. OPT94_READ	**Reads inputs of parallel I/O device**
4. SYS_PORT_OUT	**Set single output of parallel I/O device**
5. SYS_PORT_IN	**Reads single input of parallel I/O device**

* Additional accessory, optional

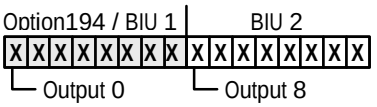
6.5.1. OPT94_VERSION

Function	**Read parallel I/O device info** Returns information about connected parallel I/O (Option194 and Binary Interface Unit BIU).		
Note	It isn't allowed to mix Option194 and BIU's in a system !		
Syntax	**C** **VC++**	char* OPT94_VERSION (int iNum);	
	BASIC **VBASIC**	Function OPT94_VERSION (ByVal iNum As Integer) : As String	
	PASCAL **DELPHI**	Function OPT94_VERSION (iNum : integer) : string	
Input	Integer	iNum: always 0 (zero) !	
Return	String	Information about number of available parallel I/O outputs / inputs. Length: 12 bytes Format: "IKdd-0-0vvvv" (d = device number, v = WI-Version) IK15...: 1 x Option194-ISA or 1 x BIU IK16...: 2 to 8 BIU's Example: "IK16-0-0132": 2 or more BIU's, WI-Version: 132	

Example

```
#include c__mem.h

void main ()
{
      char cRet[13];      // return value

      strcpy (cRet, OPT94_VERSION(0));
      printf("Result: %s\n", cRet);

      return;
}
```

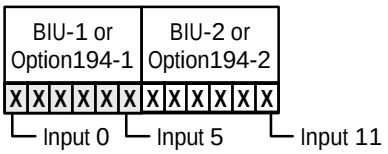
6.5.2. OPT94_WRITE

Function	**Set outputs of parallel I/O device simultaneous** Switches all outputs of an option194 (Opt194-ISA with 8 outputs) or a Binary Interface Unit (one BIU with 8 outputs, two BIU's with 16 outputs) simultaneously to logical on or off.		
Note	It isn't allowed to mix Option194 and BIU's in a system !		
Info	The command is fully compatible between the low-power switching option 194 and the power switching Binary Interface Unit (also called BIU or Relay-Box). The input parameter iNum exists because of historical reasons. Always set it to 0 !		
Syntax	**C** **VC++**	void OPT94_WRITE (int iNum, char* cOut);	
	BASIC **VBASIC**	Function OPT94_WRITE (ByVal iNum As Integer, ByVal cOut As String)	
	PASCAL **DELPHI**	Function OPT94_WRITE (iNum : integer; cOut : string)	
Input	Integer, String	First parameter: always 0 (zero) Sec. parameter: 16 bytes output bit pattern Format: "XXXXXXXXXXXXXXXX"  Example: OPT94_WRITE (0,"1010101010101010");	

Example

```
#include c__mem.h

void main ()
{
      // reset all 8 outputs of option 194
      OPT94_WRITE (0, "0000000000000000");

      // set all 8 outputs of option 194
      OPT94_WRITE (0, "1111111100000000");

      // set output 1 of option 194
      OPT94_WRITE (0, "1000000000000000");

      return;
}
```

6.5.3. OPT94_READ

Function	**Reads inputs of parallel I/O device simultaneous** Reads the logical states of all inputs of an option194 (Opt194-ISA with 6 inputs) or a Binary Interface Unit (one BIU with 6 of 8 inputs, two BIU's with 12 of 16 inputs) simultaneously.		
Note	It isn't allowed to mix Option194 and BIU's in a system !		
Info	The command is fully compatible between the low-power switching option 194 and the power switching Binary Interface Unit (also called BIU or Relay Box). The input parameter iNum exists because of historical reasons. Always set it to 0 !		
Syntax	**C** **VC++**	char* OPT94_READ (int iNum);	
	BASIC **VBASIC**	Function OPT94_ READ (ByVal iNum As Integer) As String	
	PASCAL **DELPHI**	Function OPT94_ READ (iNum : integer) : string	
Input	Integer	Parameter always 0 (zero)	
Return	String	State of the parallel I/O inputs Length: 12 bytes Format: "XXXXXXXXXXXX"  	

Example

```
#include c__mem.h

void main ()
{
      char cInp[13];      // input state

      // if option 194 is installed
      // read the 6 inputs of option 194
      strcpy (cInp, OPT94_READ (0);

      // if 1 BIU is installed
      // read the first 6 inputs of BIU
      strcpy (cInp, OPT94_READ (0);

      // if 2 BIUs are installed
      // read the 2 x 6 inputs of the BIUs
      strcpy (cInp, OPT94_READ (0);

      return;
}
```

6.5.4. SYS_PORT_OUT

Function

Set single output of parallel I/O device
Switches a single output of an option194 (Opt194-ISA with 8 outputs) or alternatively a Binary Interface Unit (maximum 8 BIU's, each 8 outputs) to logical on or off.

Note

It isn't allowed to mix Option194 and BIU's in a system !

Info

If you want to control multiple outputs simultaneously, you can use the command OPT94_WRITE, described on page 68.
If you want to use the BIU, first enable the BIU with the SYS_WI command on page 61 for one time.

Syntax

C **VC++**	int SYS_PORT_OUT (int iPort, int iState);
BASIC **VBASIC**	Function SYS_PORT_OUT (ByVal iPort As Integer, ByVal iState As Integer) : As integer
PASCAL **DELPHI**	Function SYS_PORT_OUT (iPort : integer; iState : integer) : integer

Input

Integer1, Integer2

Integer 1: Output port address
Integer 2: 1=on, 0=off

Port address	*Physical port*
4	Option194, output 0
5	Option194, output 1
...	...
11	Option194, output 7
12	BIU 1, output 0
13	BIU 1, output 1
...	...
19	BIU 1, output 7
20	BIU 2, output 0
...	...
75	BIU 8, output 7

Example: Switch on output 0 of first BIU:
SYS_PORT_OUT (12,1);

Return

Integer

0	Output port set successfully
-1	Selected port not available

Example

```
#include c__mem.h
void main ()
{
      int iState;         // return value

      SYS_WI("071");      // Enable BIU
      // switch on output 2 of BIU 5
      iState = SYS_PORT_OUT(46,1);
      printf("Result: %i\n", ciState);

      return;
}
```

6.5.5. SYS_PORT_IN

Function

Read single output of parallel I/O device
Reads a single input of an option194 (Opt194-ISA with 8 outputs) or alternatively a Binary Interface Unit (maximum 8 BIU's, each 8 outputs).

Note

It isn't allowed to mix Option194 and BIU's in a system !

Info

If you want to read multiple inputs simultaneously, you can use the command OPT94_READ, described on page 69 (only up to 16 inputs).
If you want to use the BIU, first enable the BIU with the SYS_WI command on page 61 for one time.

Syntax

C **VC++**	int SYS_PORT_IN (int iPort);
BASIC **VBASIC**	Function SYS_PORT_IN (ByVal iPort As Integer) : As integer
PASCAL **DELPHI**	Function SYS_PORT_IN (iPort : integer) : integer

Input

Integer — Input port address

Port address	*Physical port*
4	Option194, input 0
5	Option194, input 1
...	...
9	Option194, input 5
10-11	not available
12	BIU 1, input 0
13	BIU 1, input 1
...	...
20	BIU 2, input 0
...	...
75	BIU 8, input 7

Example: read input 0 of first BIU:
SYS_PORT_IN (12);

Return

Integer — Input state

0	Low
1	High
-1	Selected port not available

Example

```
#include c__mem.h
void main ()
{
      int iState;         // return value
      SYS_WI("071");      // Enable BIU
      // read input 0 of BIU 4
      iState = SYS_PORT_IN(36);
      printf("Result: %i\n", ciState);
      return;
}
```

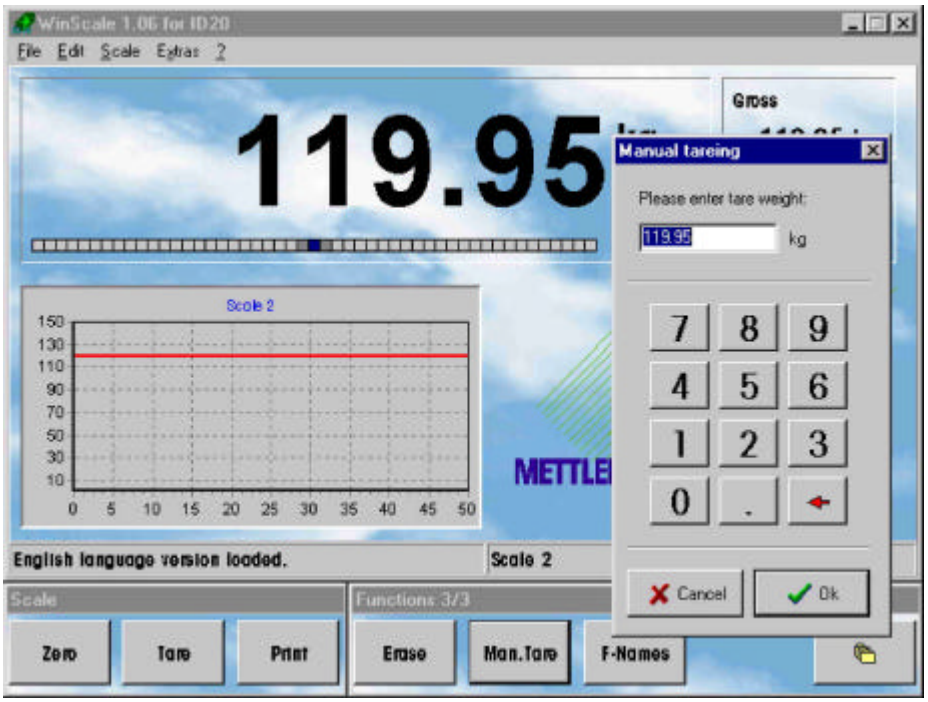
7. Basic control applications

7.1. WinScale application for MS-Windows 95/98/NT

WinScale is designed as a complete solution for all the different people, who are working for and with the ID20.

It supports
- our customers as an Easy-To-Use basic weighing program
- our service technicians as a complete system diagnostic tool
- our sales personal as an demonstration program for sales purposes.

Last, but not least, also application programmers can get an idea about the possibilities, when they create applications for the ID20.



WinScale for ID20

By the way:
WinScale is installed free of charge on every new ID20. It supports the most common languages: English, German, French, Spanish, Italian, Dutch and Russian. WinScale can be copied free and used without charges !

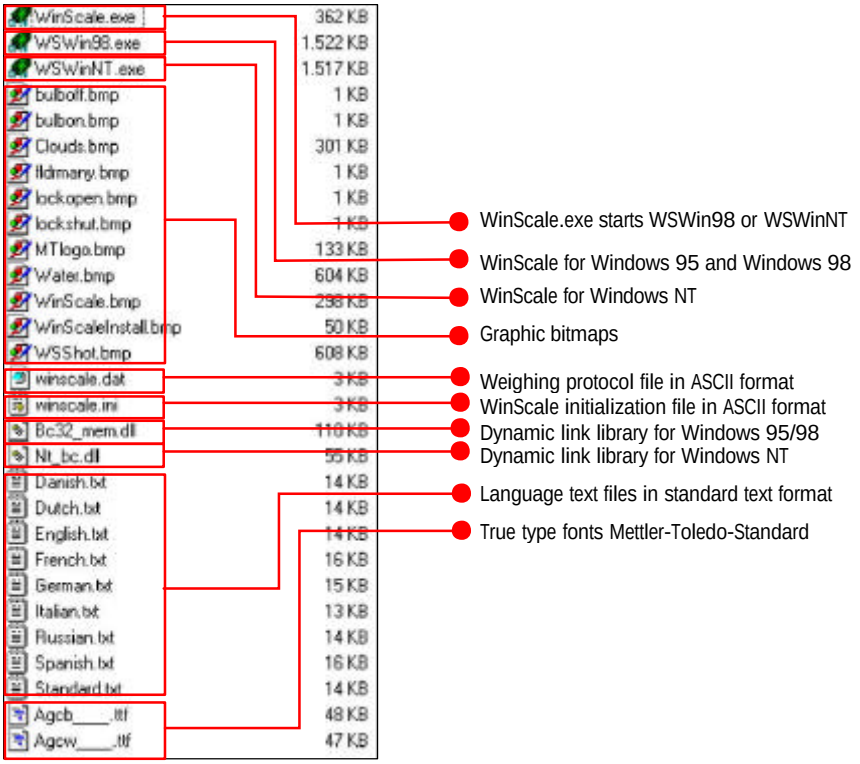
7.1.1. Structure

WinScale has a very open structure. That means, all texts, most of the graphics, colors as well as a lot of parameters can be changed very easily:

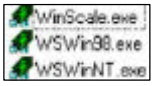
Because all this data can be manipulated in standard text files with any text editor (or the integrated editor in WinScale) – without changing or recompiling any source code - it is very easy to translate WinScale in special languages or modify WinScale texts in customers standard expressions.

It is also very easy to modify the user surface, e.g. by including the customers logo and company colors so that the customer can work with WinScale in his Corporate Identity (CI) Look-And-Feel.

All files relating to WinScale are included in the ID20-directory C:\WinScale:



File structure of WinScale



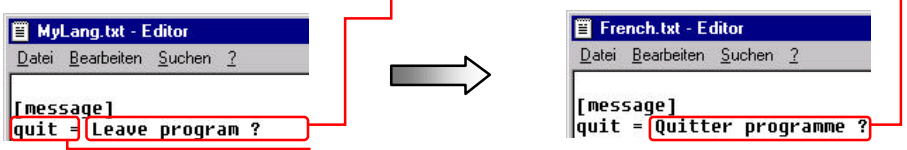
When WinScale.exe is started, it decides -depending on the running operating system- which WinScale version has to be started. There are two different compiled versions of the same software, because of the need of different software interfaces for Windows95/98 and WindowsNT.

7.1.2. Translating or editing texts in WinScale

All texts, which are used in WinScale, are editable in standard ASCII files. They can be edited with every standard editor (e.g. the integrated WinScale editor or Notepad.exe). That means, translations in other languages or the exchange of terms are very simple. The language files are loaded “online” in WinScale.

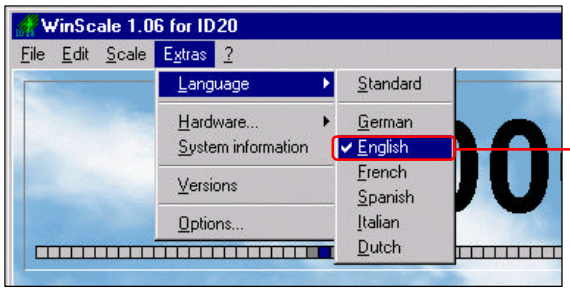
If a translation for a special language has to be created, the file “MyLang.txt” (my language) should be used. If you do so, this new language can be reached over the menu point “Language->MyLanguage”. Normally, “MyLang.txt” is only a simple copy of the English text file “English.txt”.

Translate simply by exchanging the right-side expressions of every line. The left expression is used by WinScale to identify the text. The example shows you, how to translate an English term into a French one



The terms on the left side are used to identify the text, so never change them !

It is possible to test the modifications “online” if you run an editor and WinScale parallel. If you save the edited text file in your editor, reload the corresponding language in WinScale to see the modifications immediately.

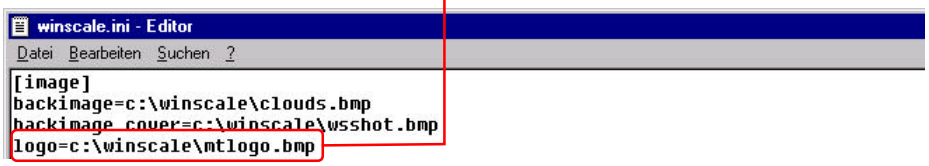


This method is very helpful to check the sense of the new text and if the text length still fits in WinScale.

7.1.3. INI-File of WinScale

winscale.ini

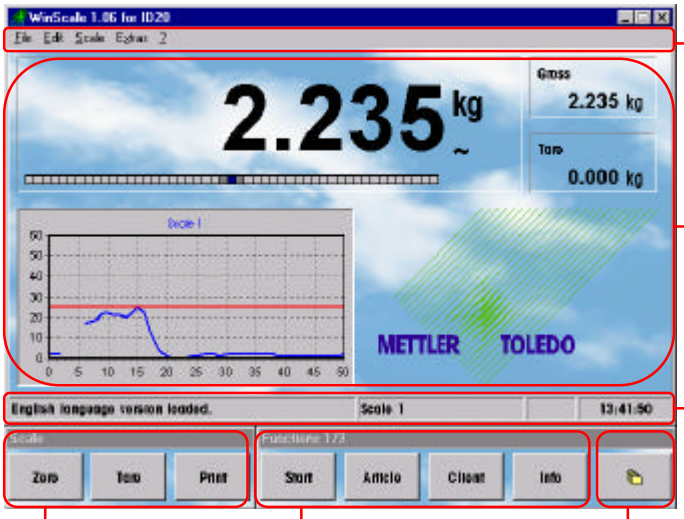
The basic settings of WinScale are located in the WINSCALE.INI file. Normally it isn’t necessary to do changes manually, because all important settings can be reached comfortable in WinScale using the menu point EXTRAS/OPTIONS. The exception is the exchange of bitmaps. If you e.g. want to exchange the company logo, the path to the new bitmap has to be changed in the block [image], point LOGO.



The initialization file of WinScale must be handled with care.
Incorrect changes can lead to program mistakes or other serious errors !
The INI file also contains the system passwords for Mastermode and Options.

7.1.4. User weighing program

The user surface of WinScale consists of these basic regions:



Menu bar, for special use only (Service, Initialization, Edit)

Information field, Weight values and weight graph

Info bar, Information about last step, actual scale, actual time

Basic scale buttons, always visible

Function keys, dependent of selected level

Function key level

A standard user can do his work completely with the use of the 8 function keys of the foil keypad. There is no need to use specialties in the menu bar when processing normally.

WinScale can be also used as an emergency program, while the customer application is not finished or actually not working. If weighing results are printed in the alibi file or on a printer with F3, they are also saved in a standard ASCII file, called WINSCALE.DAT. This makes it possible to get all weighing specific data later back into data bases or in other programs as MS-Excel etc.

7.1.5. Service functionality

With the integrated service functionality of WinScale (only reachable over the menu bar), it is possible to check hard- and software of the ID20.
With the menu point “Scale” you can reach the MASTERMODE and the SERVICEMODE.

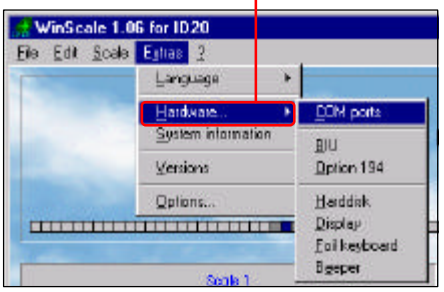
The Mastermode informs about scale parameters and allows the service technician to change them, like vibration adapter, weighing process adapter, automatic stability control, auto-tare and restart – option. The access to the Mastermode can be prevented by a password, to enable in EXTRAS/OPTIONS/SECURITY.

The Servicemode is only accessible for METTLER TOLEDO service technicians with a special password. Here it is possible to change weighing relevant scale data. Any change, which is saved, increments the scale identcode, that means, that the system is not longer approved any more !

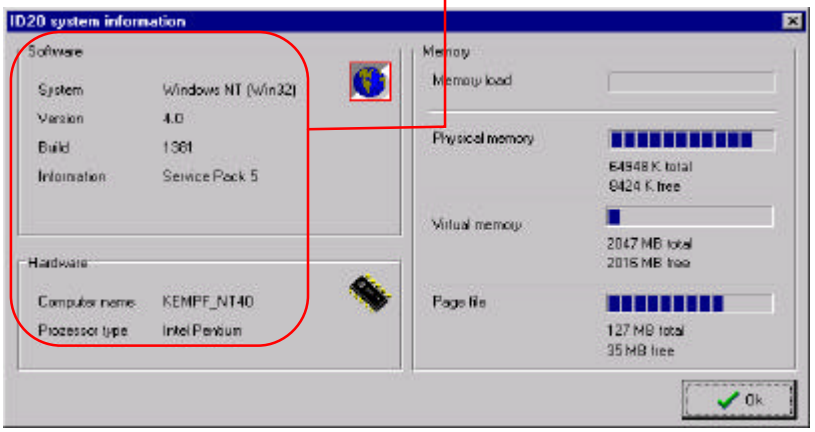
Access to Master- and Servicemode should only be done by METTLER TOLEDO service technicians ! Incorrect settings can lead to wrong results and malfunctions!

With the menu point “Extras” you can reach the hardware test and setting functionality. It is possible to test:

- Serial interface ports
- Parallel interface ports (Binary Interface Unit BIU and Option194)
- Harddisk
- TFT-Display
- Foil keyboard
- Beeper



With the menu point EXTRAS / SYSTEM INFORMATION, you can get an overview over your computers system configuration, regarding the ID20 as a

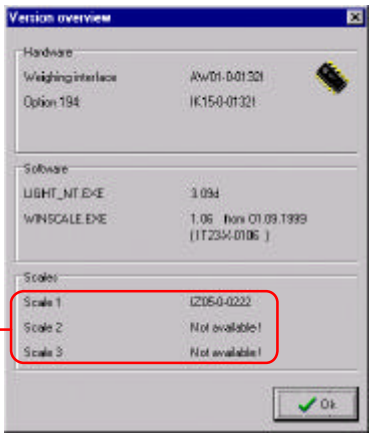


standard industrial PC. This can be helpful, when integrating the ID20 into customers networks or locating system overload errors.

The last point in the Extras is the version info. Here you can get a fast overview, which METTLER TOLEDO specific hard- and software is installed.

This is important, if you want to check, if an update was successful or not.

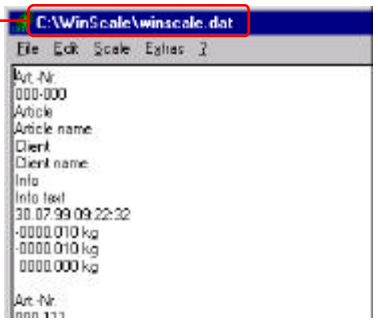
The version info also informs about scale software versions ,connected to the ID20.



7.1.6. Integrated text editor

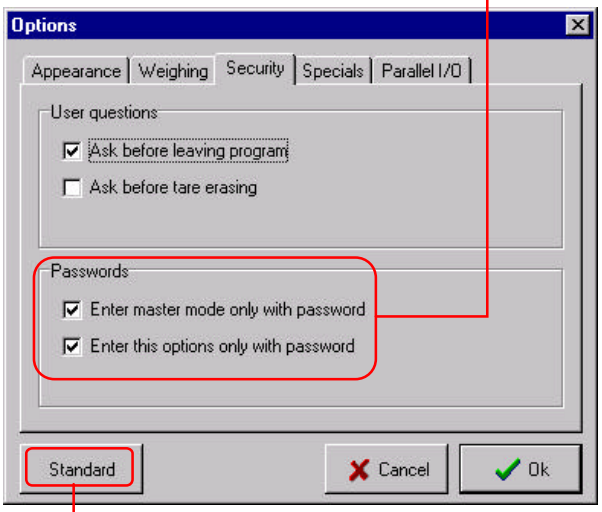
The integrated standard text editor makes it possible to edit the weighing result data file or to edit or translate any language file without having access to a Windows editor.

Open the editor with FILE/OPEN and close it again with FILE/CLOSE.



7.1.7. Options

The are a lot of options, to set WinScale individually to customers needs. For example, you can prevent the access to the master mode or to this complete option menu:



With a click to the standard button, all option points are set to the most common settings and also the colors of WinScale are set to factory setting.

In the option menu, register tab WEIGHING, you can also give the scales an individual, more useful name. This name will be displayed in the info bar, so that the user can identify the actual chosen scale much more easily.

The first field informs the user about the last event, e.g. the result of a tare function and also displays error messages. The last field shows the actual time.

7.2. Scale application for MS-DOS

7.2.1. SCALE.EXE

The application program SCALE.EXE is a simple weighing program, which runs under MS-DOS or in a DOS-Box in Windows95 or Windows98.
Please note that the program does not run in a DOS-Box under WindowsNT.

For WindowsNT, and also the other Windows versions, use the Windows-based program WinScale. WinScale is a replacement for SCALE.EXE and SERVICE.EXE.

SCALE.EXE can be started with a simple click on [1] and [↵] on the foil keyboard. This starts the batch file "1.BAT". Depending on the used weighing interface, the scale driver program MEMORY.EXE or LIGHT.EXE will be also automatically loaded if necessary.

7.2.2. Features

Besides the basic weighing functions, SCALE.EXE has some useful features, so that it can be used as an emergency weighing program.

Identification keys
There are 4 identification keys, which can be labeled by the user. If the user presses an ID key, a free text can be entered.

Weighing data export
When weighing results are printed to the alibi file, this data as well as the actual texts from the identification keys are printed simultaneously into a standard text file (C:\BORL_C\SCALE.DAT). This makes it possible to take over the results later in an evaluation program or into the normal application program. Additionally, it is also possible to printout the weighing data on a GA46 printer.

Serial ports configuration
The serial ports can be configured in four configurations:
- Serial port not used
- GA46: if a GA46 printer is attached, for printouts
- Barcode: for text entries into the identification keys with a barcode reader
- Command/Response: Remote control / communication with the ID20

7.3. Service application for MS-DOS

7.3.1. SERVICE.EXE

The application program SERVICE.EXE is a simple weighing program, which runs under MS-DOS or in a DOS-Box in Windows95 or Windows98.
Please note that the program does not run in a DOS-Box under WindowsNT.

For WindowsNT, and also the other Windows versions, use the Windows-based program WinScale. WinScale is a replacement for SCALE.EXE and SERVICE.EXE.

SERVICE.EXE can be started with a simple click on [2] and [↵] on the foil keyboard. This starts the batch file "2.BAT". Depending on the used weighing interface, the scale driver program MEMORY.EXE or LIGHT.EXE will be also automatically loaded if necessary.

7.3.2. Features

Information
SERVICE.EXE gives information about the installed soft- and hardware versions. Please note that the scale driver program is displayed always as MEMORY.EXE, even when the LIGHT.EXE is running with a new Weighing Interface.

Mastermode, Servicemode
In the Mastermode, the settings of the vibration adapter, the weighing process adapter and the automatic stability control can be manipulated. The password-protected Servicemode enables service technicians to calibrate the load cell etc. Please note, that changes in the Servicemode increase the scale's Ident-Code, so the system will loose it's legal validation !

Hardware test
This option allows the test of basic harddisk function, display pixel control, MF-II keyboard test, foil keyboard test, test of serial ports (with loopback connector) and the test of the parallel I/Os: Option 194 and the BIU (Binary Interface Unit).

7.4. Alibi file authentication

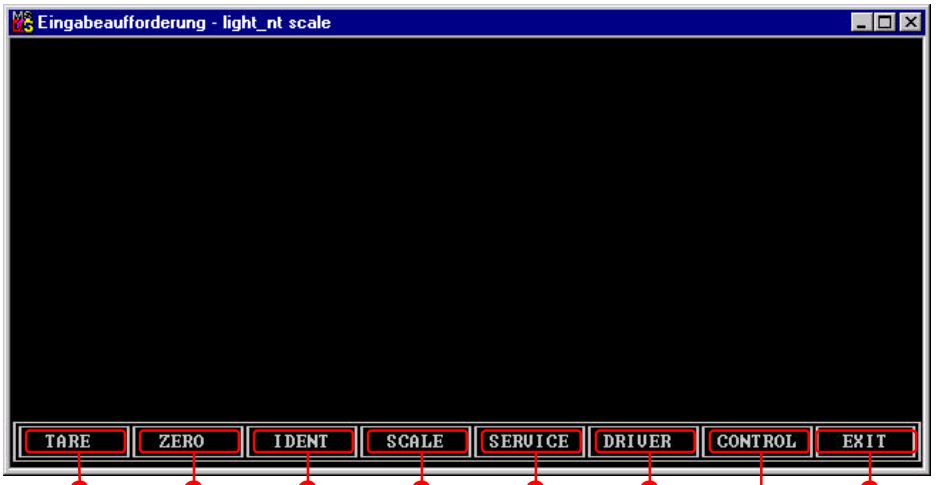
The only way to authenticate the alibi file for purposes subject to legal control is the integrated scale program / editor as part of the scale driver LIGHT.EXE respectively LIGHT_NT.EXE.

To start the DOS-based program, please open a DOS-Box and proceed as follows:

If Windows95 or Windows98 is running, please enter:

If WindowsNT is running, please enter:

The following menu appears on your screen. Select the desired function of the scale program via the function keys F1 to F8:



Tare scale

Zero scale

Scale Ident-code

Choose scale

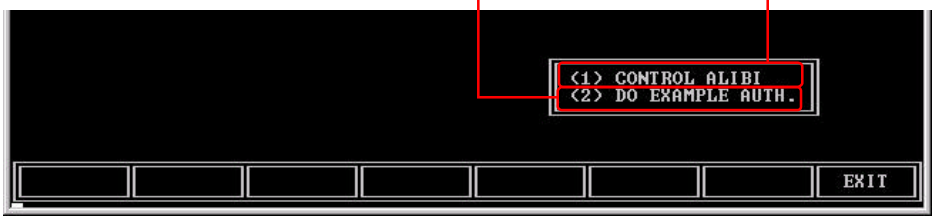
Start Service-mode

Verify legal programs

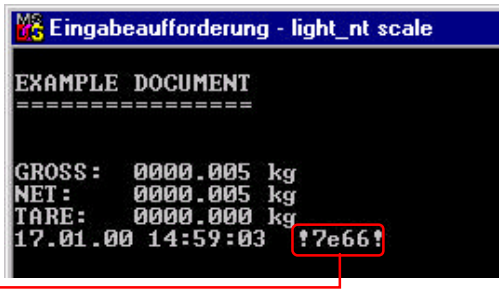
Leave program

To validate the alibi file, choose “CONTROL” by pressing F7.

In the next screen, choose between controlling the alibi file by pressing 1 or making an example authentication print by pressing 2 now:

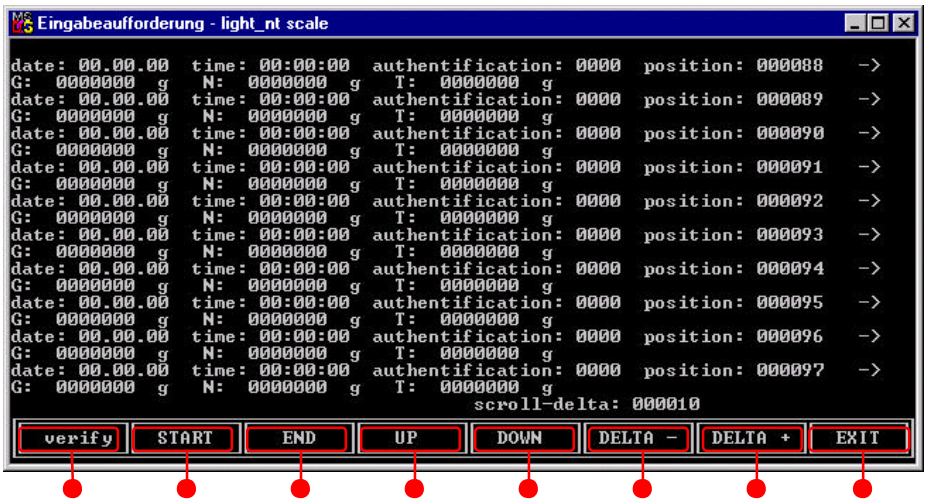


If an example print is made, the gross/net/tare value and the date / time stamp and the individual authentication key is printed in the alibi file on the harddisk.

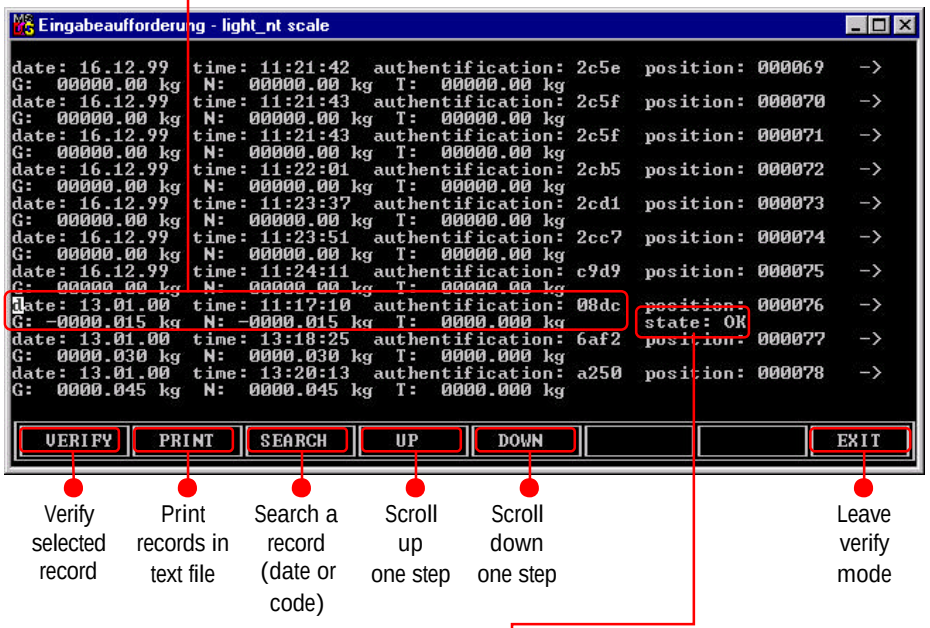


F8 returns to the previous menu.

To control the alibi file entries, press [1]. The screen with 10 new data sets appears similar as follows:



With scroll up/down now step to the page until the desired data set to verify is shown. With press on F1, the cursor jumps into the data field. Now select the desired record with the cursor up/down keys or F4/F5.



Another push on F1 now checks the integrity of the data record. If the data set is correct, the state is displayed as “OK”. If the record was manipulated, the state is “FALSE”. If the data set is unused, the state is displayed as “FREE”.

For documentation purposes, it is possible to realize a printout of chosen data sets with F2 into a standard text file. The file can be found in C:\MEMORY.PRT.

This special METTLER TOLEDO editor is legal approved to check the integrity of the compressed data sets in the ID20 alibi file. The encryption is realized with a special, secret algorithm to control the correctness of weight, date and time data corresponding to the authentication code. Every manipulation will be detected.

Notes

Notes

Mettler-Toledo (Albstadt) GmbH
D-72423 Albstadt, Germany
Tel. +49 7431 140, Fax +49 7431 14373
Internet: http://www.mt.com

AT	**Mettler-Toledo Ges.m.b.H., A-1100 Wien** Tel. (01) 604 19 80, Fax (01) 604 28 80
AU	**Mettler-Toledo Ltd., Port Melbourne, Victoria 3207** Tel. (03) 9646 4551, Fax (03) 9645 3935
BE	**n.v. Mettler-Toledo s.a., B-1651 Lot** Tél. (02) 334 02 11, Fax (02) 378 16 65
BR	**Mettler-Toledo Indústria e Comércio Ltda.** São Paulo, CEP 06465-130 Tel. (11) 421 5737, Fax (11) 725 1962
CA	**Mettler-Toledo Inc., Ontario L7R3Y8,** Tel. (905) 681 7011, Fax (905) 681 1481
CH	**Mettler-Toledo (Schweiz) AG,** CH-8606 Greifensee Tel. (01) 944 45 45, Fax (01) 944 45 10
CN	**Mettler-Toledo Changzhou Scale Ltd.** Changzhou City, Jiangsu 213001 Tel. (519) 664 20 40, Fax (519) 664 19 91
CZ	**Mettler-Toledo, spol, s.r.o., CZ-12000 Praha 2** Tel. (2) 22 51 69 52, Fax (2) 22 51 81 92
DE	**Mettler-Toledo GmbH, D-35353 Giessen** Tel. (0641) 50 70, Fax (0641) 52 951
DK	**Mettler-Toledo A/S, DK-2600 Glostrup** Tel. (43) 27 08 00, Fax (43) 27 08 28
ES	**Mettler-Toledo S.A.E., E-08038 Barcelona** Tel. (03) 223 22 22, Fax (03) 223 02 71
FR	**Mettler-Toledo s.a., F-78222 Viroflay** Tél. (01) 309 717 17, Fax (01) 309 716 16
HK	**Mettler-Toledo (HK) Ltd., Kowloon HK,** Tel. (852) 2744 1221, Fax (852) 2744 6878
HR	**Mettler-Toledo, d.o.o., CR-10010 Zagreb** Tel. (1) 660 2189, Fax (1) 660 3009
HU	**Mettler-Toledo Kft, H-1173 Budapest** Tel. (1) 257 9889, Fax (1) 257 7030
IN	**Mettler-Toledo India Pvt Ltd, Mumbai 400 072** Tel. (22) 857 08 08, Fax (22) 857 50 71
IT	**Mettler-Toledo S.p.A., I-20026 Novate Milanese** Tel. (02) 333 321, Fax (02) 356 29 73
JP	**Mettler-Toledo K.K., Shiromi, J-Osaka 540** Tel. (6) 949 5901, Fax (6) 949 5945
KR	**Mettler-Toledo (Korea) Ltd., Seoul (135-090)** Tel. (82) 2 518 20 04, Fax (82) 2 518 08 13
MY	**Mettler-Toledo (M) Sdn.Bhd., 47301 Petaling Jaya** Tel. (603) 703 2773, Fax (603) 703 8773
MX	**Mettler-Toledo S.A. de C.V., Mexico CP 06430** Tel. (5) 547 5700, Fax (5) 541 2228
NL	**Mettler-Toledo B.V., NL-4000 HA Tiel** Tel. (0344) 638 363, Fax (0344) 638 390
NO	**Mettler-Toledo A/S, N-1008 Oslo** Tel. (22) 30 44 90, Fax (22) 32 70 02
PL	**Mettler-Toledo, Sp. z o.o., PL-02-929 Warszawa** Tel. (22) 651 92 32, Fax (22) 42 20 01
RU	**Mettler-Toledo AG, 10 1000 Moskau** Tel. (095) 921 68 12, Fax (095) 921 63 53
SE	**Mettler-Toledo AB, S-12008 Stockholm** Tel. (08) 702 50 00, Fax (08) 642 45 62
SEA	**Mettler-Toledo (SEA), Sdn.Bhd., 47301 Petaling Jaya** Tel. (603) 704 1773, Fax (603) 703 1772
SG	**Mettler-Toledo (S) Pte. Ltd., Singapore 139959** Tel. (65) 890 0011, Fax (65) 890 0012
SK	**Mettler-Toledo, service s.r.o., SK-83103 Bratislava** Tel. (7) 525 2170, Fax (7) 525 2173
SI	**Mettler-Toledo, d.o.o., SI-1236 Trzin** Tel. (016) 162 18 01, Fax (061) 162 17 89
TH	**Mettler-Toledo (Thailand), Bangkok 10310** Tel. (662) 719 6480-7, Fax (662) 719 6479
TW	**Mettler-Toledo Pac Rim AG, Taipei** Tel. (886) 2 2579 5955, Fax (886) 2 2579 5977
UK	**Mettler-Toledo Ltd., Leicester, LE4 1AW** Tel. (0116) 235 0888, Fax (0116) 236 5500
US	**Mettler-Toledo, Inc., Columbus, Ohio 43240** Tel. (614) 438 4511, Fax (614) 438 4900

00506141G