ACI
Laser-Components

Software Manual

Software for laser marking

Magic Mark

© ACI Laser GmbH

ACI Laser GmbH
Leubinger Str. 19
D-99610 Sömmerda

Fon +49 (3634) 3226-0
Fax +49 (3634) 3226-26

www.ACI-Laser.de
info@ACI-Laser.de

ACI
Laser-Components

Contents

6. Creating dialogue windows

6.1. User dialogue menu
6.2. Function Select
6.3. Function Group Box
6.4. Function Text
6.5. Function Text Box
6.6. Function Check Box
6.7. Function Option Button
6.8. Function Type 1 List Box
6.9. Function Type 2 Drop List Box
6.10. Function Type 3 Combo Box
6.11. Function Picture
6.12. Buttons
- OK Button
- Cancel Button
- Push Button

7. Special functions

7.1. Date/ time
7.2. Manipulation and handling of texts
7.3. Mathematical operations
7.4. Operators
7.5. Handling files
- Sequential files
- Files with direct access

7.6. Input Box
7.7. Message Box
7.8. Popup Menu
7.9. Function Dialogue

8. Laser and scanning control system

8.1. Marking parameters
8.2. Control system
8.3. Ma rking
- Rotation
- Vector
- Position
- Rectangle
- Barcode
- PDF 417 Code
- Data Matrix Code
- Circle
- Text
- Graphics

ACI
Laser-Components

1. Delivery

1.1. Specifications

Your software package contains three installation disks, which are called as follows:

Magic Mark Disk 1
Magic Mark Disk 2
Magic Mark Disk 3
Target.ini

Before working with Magic Mark you have to make a backup copy. For more detailed information about that please refer to your Windows Manual or to the Windows Online Help.

1.2. Software Manual

This software manual is part of the system. Please keep it carefully since it contains useful details about programming and the optimal operation of the laser.

In case the laser is sold, please hand over these instructions.

ACI reserves the right in accordance with technical process to update this manual at any time and without advance notice.
These instructions have been drawn up according to the latest technological developments.

Please read carefully through this manual.

1.3. Manufacturer

ACI Laser GmbH
Leubinger Str. 19
D- 99610 Sömmerda
Germany

Fon +49 (3634) 3226-0
Fax +49 (3634) 3226-26

www.ACI-Laser.de
info@ACI-Laser.de

1.4. Warranty

Manufacturer, programmer and author have carefully created the software as well as the corresponding manual. Nevertheless, any right to claim under guarantee regarding software and manual will be refused, in particular, if Magic Mark does not correspond to the requirements of the customer or in case of any occurring errors. Neither manufacturer, nor programmer or author are reliable for damages, destruction or consequential damages caused when using the software.

ACI
Laser-Components

2. Installation of Magic Mark

2.1. Requirements

To enable the operator to use Magic Mark successfully the following equippings are necessary:

- IBM-compatible Pentium PC
- operating system WIN 98 or WIN NT
- 64 MB RAM
- 100 MB free space on harddisk
- a serial interface
- a parallel interface
- monitor (recommended: 17 inch)
- keyboard, mouse

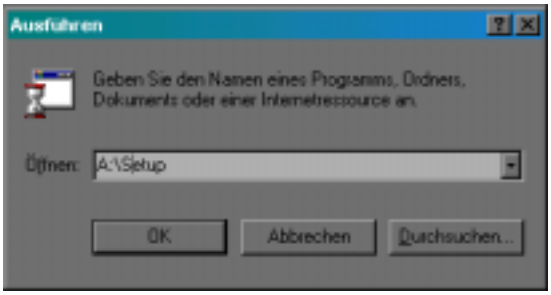
2.2. Installation under Windows

Magic Mark is installed under Windows on the harddisk.

Proceed as follows:

1. Switch on PC
2. Load Windows
3. Insert Magic Mark program disk 1 into floppy A or B.
4. Click the START button by using the mouse and choose „EXECUTE“. A WINDOWS box opens with a command line to be edited freely.
5. Enter the following into this command line depending on the floppy used:

   A:\Setup or B:\Setup
6. Confirm with OK.



Start menu of Windows 98

7. Now, follow further installing instructions.
8. Choose the program directory desired. As a standard it is defined as follows:

   C:\Programme\MagicMark
9. When installing MagicMark samples can be installed as well. In this case you choose:

   Installation complete

   If not, you choose:

   Installation without Samples
10. State a name of the laser program which should appear in the menu Programs. As a standard MagicMark appears.
11. As soon as the complete installation has been finished you get the message:

    Click Finish to complete Setup.

The installation may be interrupted at any time by striking ESC.

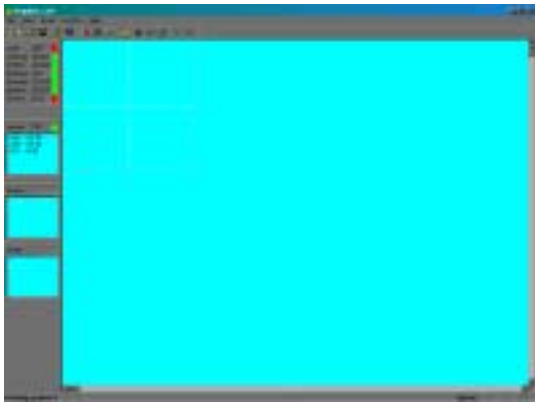
ACI
Laser-Components

3. Introduction in Magic Mark

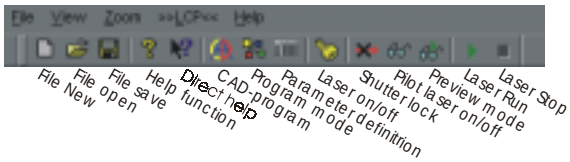
3.1. Software screen surface

Magic Mark has two functions. The one is to create marking programs, the other serves to supervise and control the laser. This manual gives detailed information about creating marking programs.

For more details about control and supervision of the laser please refer to the operating manual.



Software screen surface Magic Mark



Buttons Magic Mark

3.2. Structure of menu

Creating a new marking project

By using this button ya new marking project type *.lcp is created. A marking project consists of at least two components, the project itself and a macro type *.bas belonging to it.

Please take care that both project and basic macro are saved into the same directory.

Open marking project

By using this button you open an already existing marking project including the basic macro belonging to it.

Save marking project

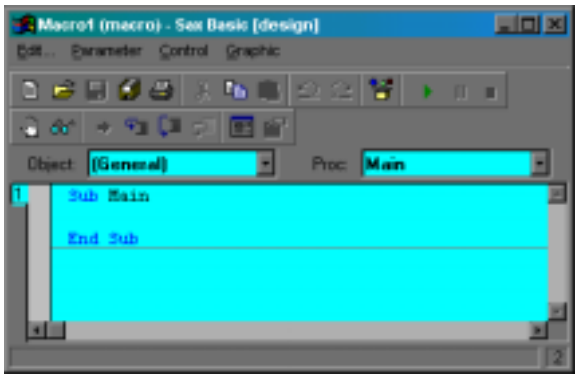
By using this button you save your marking project.

Help functions

By using this button you activate the online help

Call macro editor

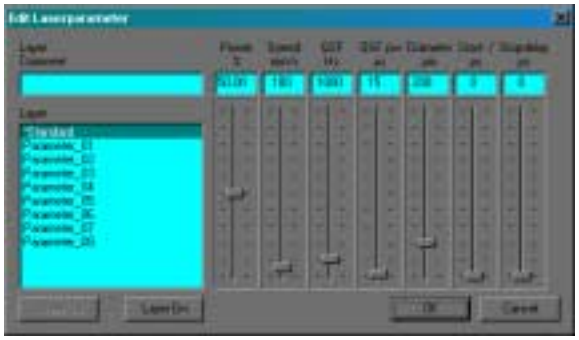
This button is used for creating and editing basic macros. You reach the programming mode. For more information please refer to chapter 4ff.



Basic macro editor

Definition of parameters

This menu enables to define different marking layers with regard to varying laser parameters.



Defintion laser parameters

Key switch laser ON/OFF

This button serves to swich on/off the laser by means of a defined password.

Locking Shutter

By using this button you lock the shutter. It is not possible to open it again.

Pilot laser ON/OFF

By using this button you activate/deactivate the pilot laser.

Preview mode

In the preview mode the marking jobs created can be shown on screen. For this, you have to activate the button and start the marking job.

Run marking job

Start your marking job, i.e. the preview by using this button.

Stop marking job

To stop a marking during the program run time you need to use this button.

4. Basic elements of programming

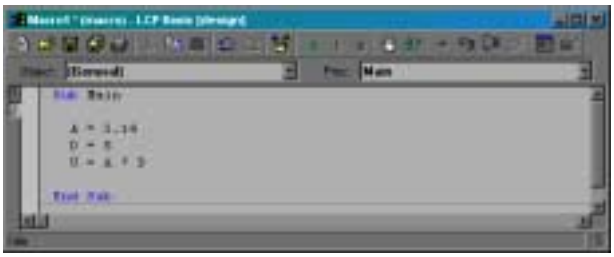
After having explained in detail the laser relevant functions of MagicMark the basic elements for programming under MagicMark will be described in the following chapters.
Core component of Magic Mark is Visual Basic as programming language enabling to create complexe program runs.

4.1. Basic elements

Program documentation

As this is necessary in any other programming language, MagicMark requires to have the program to be developed documented sufficiently. This demands some efforts when creating the program, but has a favourable effect later on in case of a possible trouble shooting or changings of the program. In addition, the program is made more transparent to third persons.

The documentation can be made as uncoded text, that means in colloquial speech. Tasks and features of variables should in particular be documented sufficiently.
Remarks and documentations can be introduced under Visual Basic with the following marks: „ ‘ “ .



Wrong: program without documentation

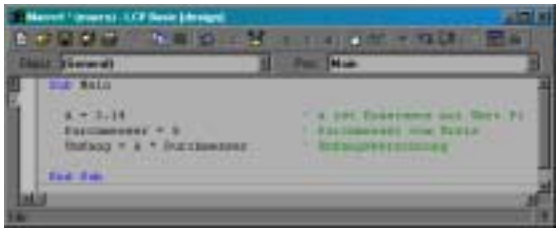
Right: program with documentation

Control elements

Control elements are necessary to show information on screen or to enable the user to enter inputs. Such control elements are for example buttons and list boxes.

Naming variables

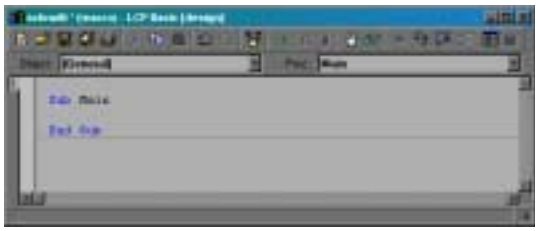
When naming variables for a further optimizing of the program code it is most important to choose a meaningful name indicating already the function of the variable by means of its name. Abbreviations are proved to be less useful as their meaning is forgotten too fast.



Appropriate naming of variables

4.2. The Visual Basic window

The following chapter explains in detail the different functional possibilities of the Visual Basic window.



The Visual Basic window

Visual Basic menu

The Visual Basic menu

The Visual Basic menu contains all important functions required to prepare the basic file.

First, let us look at the ListBox Object. This box indicates the active form or the active control element. However, the present event procedure of the control element to which belongs the program code is entered into the ListBox Proc.

Drop-down menu: File

This menu contains all functions required to handle files

 **Create new project**

By means of this symbol a new project can be created.

When creating a new project 4 types of modules are proposed:

- New Macro
- New Code Module
- New Object Module
- New Class Module

This manual basically explains how creating a new operable program (New Macro). The modules Code, Object und Class are object-orientated program modules which alone are not capable to operate.

 **Open already existing project**

By clicking this symbol you open an already existing project. Choose the file requested in the directory.

 **Save**

The menu selection point -Save project- is used to save a created project. In case there is not yet a name stated for

the project this must be done before saving the project. If the project has already a name it can be saved without any confirmation.

If several projects are handled these can be saved altogether with -Save all-.

 **Print**

By clicking this symbol the presently opened job will be printed at a printer connected. The installation of the printer must be carried out under Windows.

Drop-down menu: Editing

 **Cut-out/copy**

This command cancels marked contents of the program. The contents cancelled remains in the interim memory.

 **Copy**

In opposite to the command cut-out marked parts of the program are kept. The contents is taken into the interim memory and remains available for subsequent treatment.

 **Insert**

This command effects that cut-out or copied parts are inserted at the current position.

 **Undo/redo**

The option Undo allows to cancel changings made after having saved the last time. In case that too many changings have been cancelled these can be reconstructed with the command Redo.

Drop-down menu: Object catalogue

 **Object catalogue**

By means of this object you reach the object catalogue with object libraries, class libraries, classes, methods, features, occurrences and constants which can be used in the code. Furthermore, the modules and procedures are shown which have been defined for the project.

Drop-down menu: Start / Stop

 **Start**

This command enables to start the current project under development environment.

 **Interruption**

Has a program been started under the development environment the programm will be interrupted at the current position by activating this command.

Stop

The current program stops and changes into the editing mode.

Drop-down menu: Debugging

The debugger as an auxiliary aid enables to check every single instruction during the execution of the program. It makes possible among others to check the contents of variables, the proper program run and serves to localize programming faults.

Step Into (single step)

This function enables to process a program step by step. If the program contains procedures it will branch into them. This procedures or functions are also processed step by step. Calling this function is possible by using F8.

Step Over (procedure step)

This function corresponds to function Step Into, however possibly included procedures and functions are executed.

Step Out (finish procedure)

This command is only available in the break mode. When activating this function the current procedure will be executed and stopped again at the next program line to be executed.

Command line indicator

If the program is processed in the debugging mode this symbols signalizes the current command line.

Breakpoint on / off

This command enables to insert a breakpoint at the current position, The program will then during the execution change into the break mode. If you use this function on a line on which a breakpoint has already been fixed, this will be cancelled.

Indicate value

This command used to indicate the current value is only available in the break mode.

Drop-down menu: User Dialogue

User Dialogue

This command enables to create user-specific programs. That means that user interfaces in Windows-format can be created of one's own. You will find a detailled description of this function in chapter 8.

5. Program development

This chapter compares different types of variables and explains terms like constants, loops, branches and functions.

5.1. Types of variables

Variables are parameters characterized by a name whose contents is changeable as already expressed in the name itself. Variables consist of two parts, a name and the value of the variable. The value can change during the program execution, the name of the variable on the other hand is firmly defined.

Variables are required to save and or to handle results of calculations or inputs from the program user.

Under Magic Mark – Visual Basic it is not necessarily required to declare variables. However we recommend to declare basically all variables to be processed. The declaration of a variable is meant to be the assignment of a type of variable (string, integer, etc.).

If you fail to declare the variables program faults may occur which are difficult to be discovered.

The following types of variables can be used:

(1) Byte
range: 0 up to 255 (1byte)
integer numbers acc. to ASCII – character set

(2) Boolean
fange: true/false (2bytes)
state true/false

(3) Integer
range: -32768 up to +32767 (2bytes)
integer variable without decimal place

(4) Long
range: -2.147.438.648 up to +2.147.483.647 (4bytes)
integer without decimal place with bigger range than integer

(5) Single
range negative: $-3{,}402823*10^{38}$ up to $-1{,}401298*10^{-45}$
range positive: $1{,}401298*10^{-45}$ up to $3{,}402823*10^{38}$
(4bytes)
floating point digit, for calculations with decimal places

(6) Double
Range negative: $-1{,}79769313486232*10^{308}$ up to $-4{,}94065645841247*10^{-324}$
range positive: $4{,}94065645841247*10^{-324}$ up to $1{,}79769313486232*10^{308}$
(8bytes)
floating point digit, , for calculations with decimal places

(7) Currency
range:

-922.337.203.685.477,5808 up to
922.337.203.685.477,5807
(8bytes)
mixture between integer and floating point digit for calculations up to the fourth decimal place.

(8) Date
range date:
01.01.100 till 31.12. 9999
range time:
00:00:00 till 23:59:59
(8bytes)
Used for indicating date and time

(9) String
range:
10bytes + 2bytes/character
string format, length of strings can be unlimited.

(10) Variant
Range of numerical values:
16bytes
range for strings:
22bytes + 2bytes/character
Default - type of variable, able to change into any type of variable. The variables are converted automatically. The use of this type of variable refers to variables, where the type may change during the program execution.

Declaration	Designation
Dim *digit* As Integer	declares *digit* as integer
Dim *character* As String	declares *character* as a string
Dim *value*	declares value as variable type Variant

Examples for declarations of variables

The declaration of the variables can also be effected by means of a shorthand.

Symbol	Type of variable
%	Integer
&	Long
!	Single
#	Double
@	Currency
$	String

Symbols for declaration of variables

It is advisable to use these symbols thus effecting a better layout of the program.

Declaration	Designation
Dim *digit%*	declares *digit* as integer
Dim *character$*	Declares *character as string*

Declaration of variables with symbols

The next two examples finally exemplify the importance of the declaration of variables:

```
Sub Main

    Dim Zahl!
    Dim Ergebnis!

    Zahl! = 2.5

    Ergebnis! = Zahl! * Zahl!

    Debug.Print "Ergebnis =" & Ergebnis!

End Sub
```

Ergebnis = 6.25

Program example 1

```
Sub Main

    Dim Zahl%
    Dim Ergebnis!

    Zahl% = 2.5

    Ergebnis! = Zahl% * Zahl%

    Debug.Print "Ergebnis =" & Ergebnis!

End Sub
```

Ergebnis = 4

Program example 2

Both of the programs lead to different results because of varied declarations of variables. In example 1, floating point digits are multiplied, whereas in the second example integer numbers are multiplied. By doing so it is rounded off up to x.50.

5.2. Constants

Constants, like variables, consist of two parts, the name and the value. However, neither name nor value are changeable during the execution of the program.
Constants may assume any type of variables.

(1) Const Pi! = 3.14

The value 3.14 is assigned to the constant Pi.

(2) Const Pi% = 3.14

In this case the value 3 is assigned to the constant Pi only.

5.3. Fields

Fields are lists of variables. Thus, one field contains several variables which can be one-dimensional, but also multidimensional.

If the size of a field remains constant during the execution of the program this is called an invariable field.

Under Magic Mark fields are used regularly when marking paletts with 5 parts for example.

```
Sub Main

    Dim Text(5) As String

    Text(0) = "Teil A"
    Text(1) = "Teil B"
    Text(2) = "Teil C"
    Text(3) = "Teil D"
    Text(4) = "Teil E"

    Debug.Print Text(0)
    Debug.Print Text(1)
    Debug.Print Text(2)
    Debug.Print Text(3)
    Debug.Print Text(4)

End Sub
```

Teil A
Teil B
Teil C
Teil D
Teil E

Program example 3

Example 3 shows an one-dimensional field, in which the variable „Text" has five elements of type String assigned.

The following chart shows once again the structure of the one-dimensional field described.

x-axis				
0	**1**	**2**	**3**	**4**
part A	part B	part C	part D	Part E

One-dimensional field - Text

If the field is now also extended into the y-axis you get a two-dimensional field. Thus, two values are necessary to declare a variable from a two-dimensional field. The following program example exemplifies the functioning of the two-dimensional field.

```
Sub Main

    Dim x, y As Integer
    Dim Feld(3,4) As Integer

    for x = 0 to 2
        for y = 0 to 3
                Feld(x,y) = x * y
                Debug.Print Feld(x,y)
        next y
    next x

End Sub
```

Program example 4

	x-axis			
y-axis	**Field(x,y)**	**0**	**1**	**2**
	0	0	0	0
	1	0	1	2
	2	0	2	4
	3	0	3	6

Two-dimensional field

Variant as an already mentionend type of variable has a special meaning to fields. This type can be used for all other types of variables shown in the following example:

```
Sub Main

    Dim Feld(2) As Variant

    Feld(0) = "Dies ist ein Text"
    Feld(1) = 100

    Feld(0) = Feld(0) & " und keine Zahl"
    Feld(1) = Feld(1) * 5

    Debug.Print Feld(0)
    Debug.Print Feld(1)

End Sub
```

Dies ist ein Text und keine Zahl
500

Program example 5

5.4. Loops

For-Next-Loop

The For-Next-Loop is meant to be a counter loop. Starting value, final value and optionally step size are defining this type of loop, the step size being positive, i.e. negative. Thus, it is possible to realize incrementing or decrementing loops. If there is no step size stated this will automatically be set to value 1.

If, for example, the loop should be quitted earlier because of a certain criterion, this is possible by means of the command Exit.

Syntax:

```
For Variable = Start To End [Step Schrittweite]
        ...
        [Exit]
        ...
Next Variable
```

The following program example shows the possibilities of this type of loop.

```
Sub Main

    Dim x!
    Dim Ergebnis!

    For x! = 0 To 1 Step 0.2
        Ergebnis! = x! * x!
        Debug.Print Ergebnis!
    Next x!

End Sub
```

0
0,04
0,16
0,36
0,64
1

Program example 6

While Loop

The While-loop is meant to be a „Boolsche“ loop. This kind of loop is repeated as long as the result is true and offers the possibility that the loop is never quitted. A practice-oriented application example is to wait for a laser starting signal initializing the marking process.

Syntax:

```
While Bedingung
        ...
Wend
```

In program example 7 a variable is reduced by 1 each. The loop is repeated as long as the if-condition is true.

```
Sub Main

    Dim i%

    i% = 5

    While i% > 2
        i% = i% - 1
        Debug.Print i%
    Wend

End Sub
```

4
3
2

Program example 7

Do Loop

The Do-loop is working similiarly to the While-loop. It is a „Boolsche“ loop, too. Main difference is that the Do-loop is passed at least one time, as the if-instruction comes not before the end.

Syntax:

```
Do
        ...
        [Exit]

Loop Until Bedingung
```

```
Sub Main

    Dim i%

    i% = 5

    Do
        i% = i% - 1
        Debug.Print i%

    Loop Until i%<2

End Sub
```

4
3
2
1

Program example 8

The so-called Endless loop is a special form of the Do-loop

Syntax:

```
Do
        ...
Loop
```

This kind of loop should be avoided as a controlled abnormal termination is not possible.

5.5. Branches

Branches are used to make case distinctions during the execution of the program. There are two main types of branches. In case of a Yes/No decision the If-Then-Else-Branch is used. In case you like to distinguish between several selection possibilities then using a Select-Case-Decision is recommended.

If-Then-Else

Syntax 1:

```
If Bedingung Then Anweisung
        ...
End If
```

Syntax 2:

```
If Bedingung Then Anweisung Else Anweisung
        ...
End If
```

Syntax 3:

```
If Bedingung Then
        ...
        Elseif Bedingung Then
        ...
        Else
        ...
End If
```

Example program:

```
Sub Main

Dim i!
i! = 5

While i! > -2
  If i! > 2 Then
     Debug.Print "i ist " & i! & "; also größer als 2!"
   Else
     Debug.Print "i ist " & i! & "; also kleiner als 2!"
  End If
  i! = i! - 1.5
Wend

End Sub
```

i ist 5; alo größer als 2!
i ist 3,5; alo größer als 2!
i ist 2; alo kleiner als 2!
i ist 0,5; alo kleiner als 2!
i ist -1; alo kleiner als 2!

Program example 9

Select Case

A Select Case – branch is meant to be a case distinction enabling to select between several alternatives. The follwing example program exemplifies that, according to variable workpiece, a distinction is made so that various functions are called and thus different contents can be marked.

```
Sub Main

Dim Werkstück As Integer

Werkstück = 1     ' Auswahl Nr. 1

Select Case Werkstück
  Case 1
      Marking1  ' Funktionsaufruf Marking1
  Case 2
      Marking2  ' Funktionsaufruf Marking2
  Case 3
      Marking3  ' Funktionsaufruf Marking3
 End Select

End Sub
```

```
Sub Marking1
  DCS.Circle(9,0,0,10,0,360,False)
 'Kreis zeichnen
End Sub
```

```
Sub Marking2
  DCS.Rectangle(9,0,0,8,8,False)
'Rechteck zeichnen
End Sub
```

```
Sub Marking3
  DCS.Vector(0,0,10,10)       ' Vektor zeichnen
End Sub
```

Program example 10

The general syntax for this command is defined as follows:

Syntax:

```
Select Case Ausdruck
        Case 1
                ...
        case 2
                ...
        Case 3
                ...
        Case Else
                ...

End Select
```

5.6. Procedures and functions

To create the program code more clearly arranged it is advisable to summarize independent and finished parts of the program in sub-functions or procedures. Every single sub-function should be kept most clearly visible and short. Often, a written function or procedure can be divided again in small functions or procedures.

If programs are developed consistently according to that the programmer has in general no difficulties in coping with the code. Furthermore, errors are localized very fast and expansions of the program are easy to realize.

Procedures

Syntax:

```
Sub Prozedurname (Param1 As ... – ParamN As...)
        ...
End Sub
```

Let us look again at program example 10. In this program, several procedures have already been used.

In the main program the procedure Marking1 is called. The program branches at this position in the main program into the procedure Marking1 and executes this. In this case a circle is marked with the laser (refer to chapter 8). Naturally, variables can also be passed on at procedures. The following example exemplifies this:

```
Sub Main

Dim Radius%
Radius%=15

Marking1(Radius%)  ' Funktionsaufruf Marking1

End Sub
```

```
Sub Marking1(Radius%)
   DCS.Circle(9,0,0,Radius%,0,360,False)
 'Kreis zeichnen
End Sub
```

Program example 11

In this example the variable Radius is transferred when calling the procedure.

Functions

Functions are meant to be procedures returning a value to the main program.

```
Sub Main

Dim i%
Dim Ergebnis As Double

i% = 10

Ergebnis = Sqr (Summe(i%)) ' Wurzel aus Funktion
                                Summe
Debug.Print "Ergebnis = " & Ergebnis

End Sub
```

```
Function Summe(i%)
   Summe = i% + i%
   Debug.Print i%
End Function
```

10
Ergebnis = 4,47213595499958

Program example 12

Program 12 calculates the square root from the returned value of the function Sum. In function –Sum- the sum is calculated out of the variable i and itself.

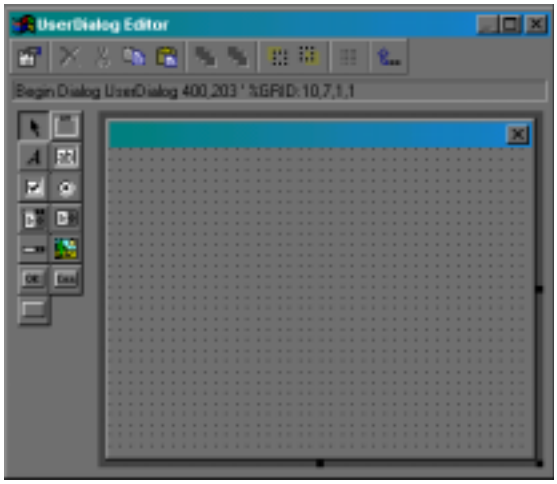
6. Creating dialogue windows

Components for the in- and output of texts and variables have been left out in the programs created so far. To create window screen surfaces the structure and inclusion of dialogue windows will be explained in the following.

6.1. User dialogue menu

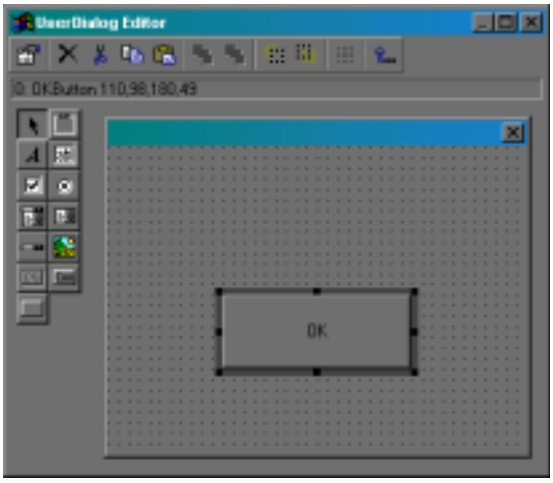
The user dialogue menu has already been mentioned in chapter 6.

Button: User Dialogue in the Visual Basic program window



User dialogue menu

First, a quite easy dialogue window will be programmed. For this, you have to create a new Basic Project. Now, click on button User Dialogue. On the left side you will find a series of control elements. Click on button OK. The mouse pointer is now changing into a coordinate axis. Now, move the mouse to the position of the window where the button should be inserted. Do leftclick and fix any size of the button. Release left-click as soon as the size appears right.



OK-button

If you wish to keep the window defined like that click button Confirm.

Button: Confirm

Then, the listing closes, you reach the program code again. You will notice that some lines have automatically been inserted into the program code defining your window.

```
Sub Main

  Begin Dialog UserDialog 400,203 ' %GRID:10,7,1,1
    OKButton 110,98,180,49
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg

End Sub
```

Program example 13

When executing the program your dialogue window defined opens waiting for confirmation per mouse-click on the OK-button.

To realize an operable dialogue window it is necessary to insert at least an OK-button, a PUSH-button or a CANCEL-button.

In the following, the upper menu will be explained in its most important points.

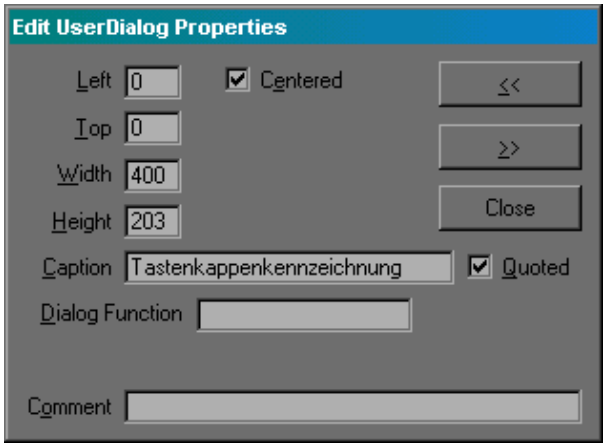
Menu

 Button: Edit Item Properties

By clicking this symbol the features of the dialogue window are defined. The window can be centred by activating the control box *Centered.* If the dialogue window should not be centred you have to switch off the control box and define the size of the window requested. The size of the window is depending on the resolution of your screen. In line *Caption* you can name the window. It is recommended to state a name according to the part to be marked. If the control box *Quoted* is activated the text entered there will then appear as string. If the control box is switched off the value entered under Caption becomes a variable which can be defined independently.

The field Dialogue Function supports complex processes, however details about that shall not be explained here.

Any comment can be entered into line *Comment*, but is only of use for the comprehension of the programmer.



Features of the dialogue window

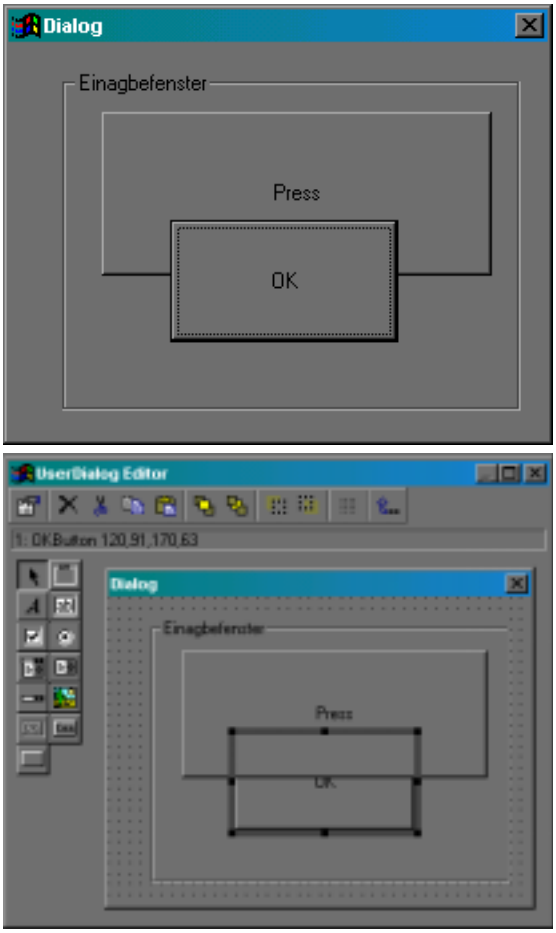
6.2. Function Select

The function Select serves to choose and mark control elements.

 Function Select

Let us look again at example 13. In this example the size of the OK-button should be modified. First, this button must be chosen as active element by clicking the function Select. When double-clicking the corresponding dialogue window opens enabling to edit independently.

```
Sub Main

   Begin Dialog UserDialog 400,203,"Dialog"
            OKButton 120,91,170,63
            PushButton 70,21,290,84,"Press"
            GroupBox 40,14,340,175,"Einagbefenster"
   End Dialog
   Dim dlg As UserDialog
   Dialog dlg

End Sub
```



Program example 14

6.3. Function GroupBox

 Button: Add GroupBox

This element serves to summarize optically the other control elements thus emphasizing in particular their belonging together. This box can be given a name consisting of a fixed text or a variable.

 Button: View of elements

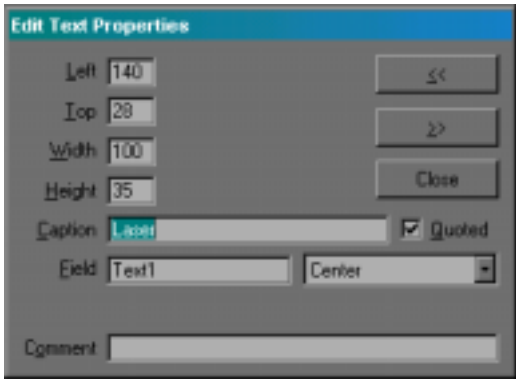
If several elements are used in one dialogue window the buttons View of elements serve to present the elements more clearly visible during the programming. If several elements are overlapping that element will be in the foreground during the program processing which is first in the dialogue programming.

The following program example will exemplify this context:

6.4. Function Text

This element serves to output in an easy manner texts in the User dialogue window enabling to visualize both fixed and variable text contents. These texts can be formatted (left justified, right justified, centred).

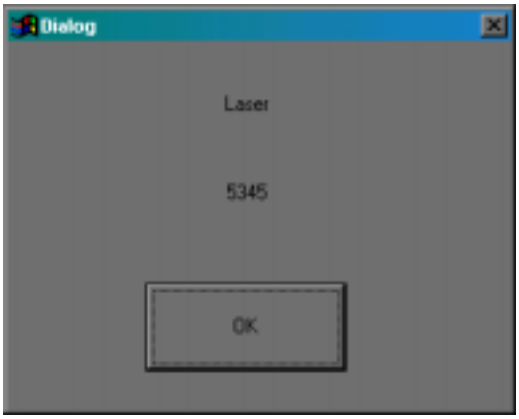
 Button Add Text



Text input

In the upper window you will realize that „Laser“ is directly described as string. If the control box –Quoted- is deactivated laser becomes a variable. In this case the contents of this variable is shown.

```
Sub Main

  Laser = 5345
  Begin Dialog UserDialog 400,203 ' %GRID:10,7,1,1
    OKButton 110,133,160,49
    Text 140,28,100,35,"Laser",.Text1,2
    Text 140,77,100,35,Laser,.Text2,2
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg

End Sub
```



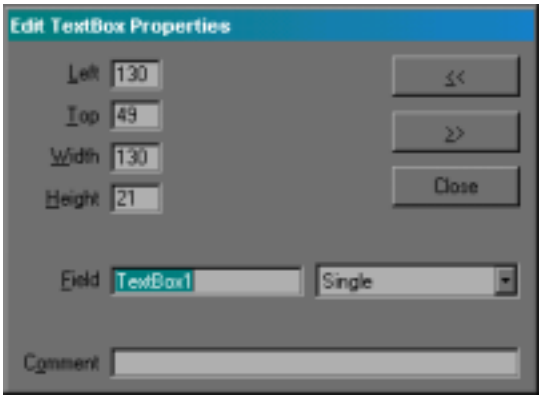
Program example 15

6.5. Function Text Box

This function offers a very easy possibility to input data and contents during the execution of the program.

 Button Add Text Box

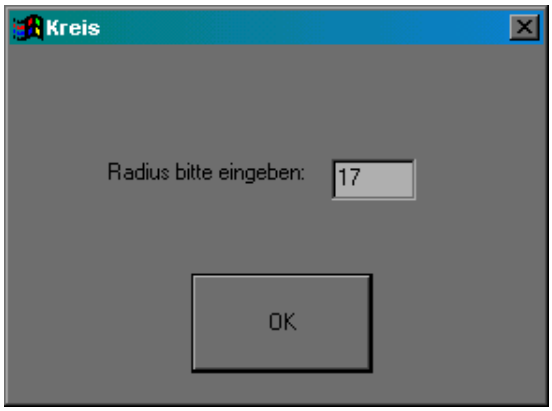
In program example 16 the ability of this function to operate will be described.



Definition text inputs

You will recognize that the basic structure of the definition is repeating except some slight variations in the different functions. In case of input the contents entered is written into a variable agreed on under Field. In the further program run it is possible to go back to this variable.

```
Sub Main

  Begin Dialog UserDialog 380,175,"Kreis"
    OKButton 130,112,130,49
    TextBox 230,56,60,21,.Variable
    Text 70,56,160,21,"Radius bitte eingeben:",.Text1
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
  Debug.Print Radius

  Radius  = dlg.Variable

  DCS.Circle (9,0,0,Radius,0,360,False)

End Sub
```



Program example 16

This example combines the functions Text output and Text input. The program realizes a circle with a radius of 17mm.

6.6. Function Check Box

The control box Check Box enables the user to activate, i.e. deactivate different options in one program run.

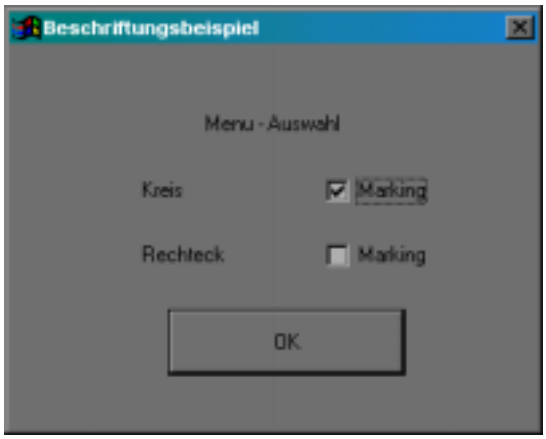
 Button Check Box

A CheckBox can adopt two kinds of conditions, true or false. If the check box is activated the condition changes into „True“. If the check box is not clicked on the condition „Wrong“ remains.

This function will be exemplified, too. Aim is to develop a program enabling the programmer to mark optionally a circle, a rectangle, both graphic elements or none of both.

```
Sub Main
    Auswahl
End Sub
```

```
Function Auswahl

  Begin Dialog UserDialog 400,203,"Beschriftung"
    Text 100,70,90,14,"Kreis",.Text1
    Text 100,105,90,14,"Rechteck",.Text2
    Text 130,35,140,14,"Menü - Auswahl",.Text3,2
    OKButton 120,140,180,35
    CheckBox 240,70,90,14,"Marking",.CheckBox1
    CheckBox 240,105,90,14,"Marking",.CheckBox2
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg

  If dlg.checkbox1 = 1 Then DCS.Circle (9,0,0,10,0,360,False)
  If dlg.checkbox2 = 1 Then DCS.Rectangle (9,0,0,10,10,False)

  End Function
```



Program example 17

6.7. Function Option Button

Like theCheckBox, the function Option Button is a switch which only adopts the conditions true or false. However, all control elements for this function of one group are independent from each other as they are combined by an ODER function. This means that only one option field of a group is able to adopt the condition true, the other elements remain as wrong.

Button: OptionButton

The example program belonging to this chapter corresponds to program 16, however it is only possible to choose alternatively a circle or a rectangle.

```
Sub Main
      Auswahl
End Sub
```

```
Function Auswahl

  Begin Dialog UserDialog 400,203,"Beschriftung"
    Text 130,35,140,14,"Menü - Auswahl",.Text3,2
    OKButton 120,140,180,35
    OptionGroup .Group1
      OptionButton 160,70,80,14,"Kreis",.OptionButton1
      OptionButton 160,91,100,14,"Rechteck",.OptionButton2
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg

  If dlg.Group1 = 0 Then DCS.Circle (9,0,0,10,0,360,False)
  If dlg.Group1 = 1 Then DCS.Rectangle (9,0,0,10,10,False)

End Function
```

Program example 18

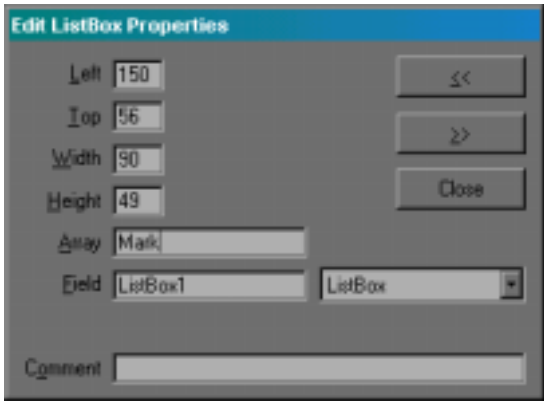
This way as many option buttons as you like can be summarized to one group.

6.8. Function Type 1 List Box

As already expressed in the name itself, the list box is used to indicate lists. If these lists contain more elements than can be displayed, scroll buttons are automatically inserted

enabling the user to scroll. However, there is no possibility to make inputs in this function. The definition of the Listbox is realized by an array which must be measured accordingly. The selection of the user is accessible by analysing the variable defined under Field.

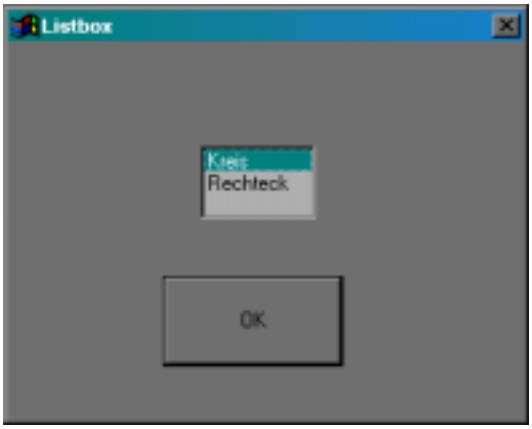
Button: ListBox



Definition of ListBox

This function will be exemplified by marking a rectangle and a circle.

```
Sub Main

  Dim Mark(2)As String

  Mark(0)= "Kreis"
  Mark(1)= "Rechteck"

  Begin Dialog UserDialog 400,203,"Listbox"
    OKButton 120,126,140,49
    ListBox 150,56,90,49,Mark(),.ListBox1
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg

  If dlg.ListBox1 = 0 Then DCS.Circle (9,0,0,10,0,360,False)
  If dlg.ListBox1 = 1 Then DCS.Rectangle (9,0,0,10,10,False)

End Sub
```



Program example 19

6.9. Function Type 2 Drop List Box

The DropListBox is a ListBox, too. However, the user has the possiblity to make an input during the execution of the program or alternatively to select between the options offered.

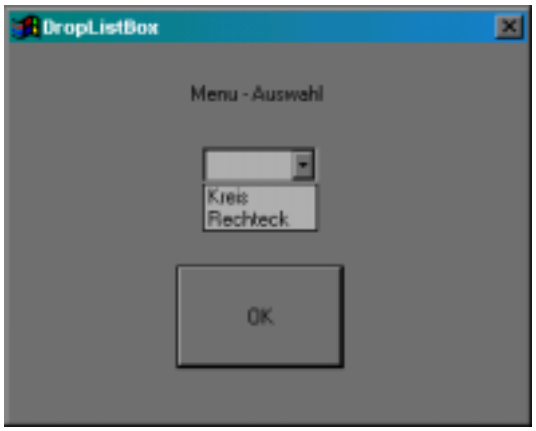
Button: DropListBox

The defintion of the accessible variable is realized by an array like it is the case for the ListBox.

Example:

```
Sub Main

  Dim Mark(2)As String

  Mark(0)= "Kreis"
  Mark(1)= "Rechteck"
    Begin Dialog UserDialog 400,203,"DropListBox"
    OKButton 130,119,130,56
    DropListBox 150,56,90,49,Mark(),.DropListBox1,1
    Text 140,21,140,14,"Menü - Auswahl",.Text1
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
```

```
  If dlg.DropListBox1 = "Kreis" Then DCS.Circle
(9,0,0,10,0,360, False)
  If dlg.DropListBox1 = "Rechteck" Then
DCS.Rectangle (9,0,0,10,10,False)

End Sub
```



Program example 20

6.10. Function Type 3 Combo Box

Regarding its function the ComboBox corresponds completey to the DropListBox, however the Combobox differs from the ListBox in its presentation. The Combobox keeps the selection window permanently opened and if necessary it can be scrolled in this window. However, the selection window of the DropListbox is not opened before it is activated.

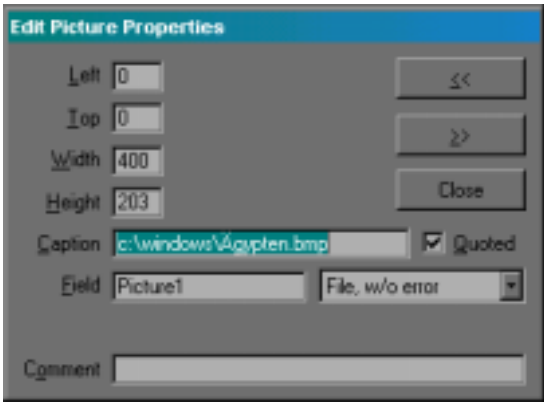
Button: ComboBox

6.11. Function Picture

The function Picture is used for designing backgrounds or for inserting company logos into the window. With the help of this function it is possible to include Bitmap files into the window.

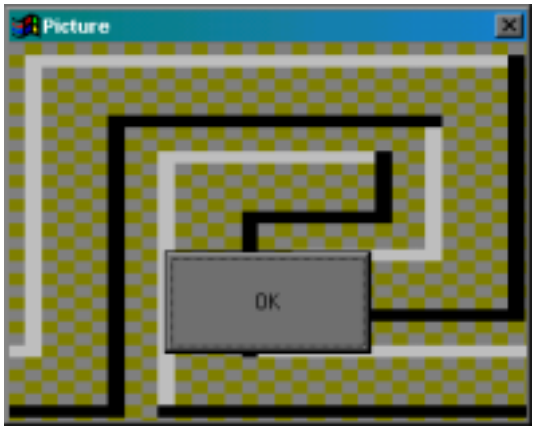
Button: Picture

In the following example, a standard Windows bitmap will be used as background of the window. For that purpose the complete directory of the file is indicated under Caption.



Definition of Bitmap file

```
Sub Main

  Begin Dialog UserDialog 400,203,"Picture"
    Picture
0,0,400,203,"c:\windows\Ägypten.bmp",0,.Picture1
    OKButton 120,112,160,56
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg

End Sub
```



Program example 21

6.12. Buttons

A button is continously required to confirm inputs effected or to start a process (marking). Under *MagicMark*-Visual Basic three different types of buttons are available, at least one of them must be there:

- **OK Button**
- **Cancel Button**
- **Push Button**

```
Sub Main

  Begin Dialog UserDialog 260,126,"Button"
    Text 60,28,140,28,"Please push a button"
    OKButton 10,91,60,21
    PushButton 100,91,60,21,"&Run"
    CancelButton 190,91,60,21
  End Dialog
    Dim dlg As UserDialog

    Debug.Print Dialog(dlg)

End Sub
```

Program example 22

The variable Dialogue(dlg) supplies according to the kind of using the button with the following results accessible in the further program run:

OK	OK-Button:	-1
Run	Push-Button:	1
Cancel	Cancel-Button:	0

7. Special functions

7.1. Date / time

The request to pursue the product's way from it's starting on makes it more and more necessary for automated markings to record the manufacturing date, i.e. time. The functions of *MagicMark*-Visual Basic described in the following chapter enable a lot of various kinds of marking date and time.

Date, Time, Now

The functions *Date*, *Time* and *Now* are used to determine the present times:

```
Sub Main

  DCS.TextTTF (9,-5,10,Date,Arial,2,2,2,False,3,False)
  DCS.TextTTF (9,-5,0,Time,Arial,2,2,2,False,3,False)
  DCS.TextTTF (9,-5,-10,Now,Arial,2,2,2,False,3,False)

  Debug.Print Date
  Debug.Print Time
  Debug.Print Now

End Sub
```

07.04.00
08:31:31
07.04.00 08:31:31

Program example 23

Program example 23 shows the results supplied by the functions used.

Hour, Minute, Second

Indicating the current hour, minute and second can be realized with the help of this function.

```
Sub Main

  Debug.Print Time

  DCS.TextTTF (9,-5,10,Hour(Time),Arial,5,2,2,False,3,False)
  DCS.TextTTF (9,-5,0,Minute(Time),Arial,5,2,2,False,3,False)
  DCS.TextTTF (9,-5,-10,Second(Time),Arial,5,2,2,False,3,False)


  Debug.Print Hour(Time)
  Debug.Print Minute(Time)
  Debug.Print Second(Time)

End Sub
```

09:27:44
9
27
44

Program example 24

Timer

This function determines the seconds run out since midnight. This is a real number, i.e. the time is determined with a precision of two decimal places.

```
Sub Main

  Debug.Print Timer

  DCS.TextTTF (9,-5,0,Timer,Arial,3,2,2,False,3,False)

End Sub
```

34566,86

Program example 25

Day, Month, Year

The functions to determine Day, Month and Year are analog to the functions Hour, Minute and Second.

```
Sub Main
```

```
Debug.Print Date

Debug.Print Day(Date)
Debug.Print Month(Date)
Debug.Print Year(Date)

End Sub
```

07.04.00
7
4
2000

Program example 26

MonthName, WeekDay, WeekDayName

Marking the name of the month, the number of the workday, i.e. the name of the workday is often used in the industrial marking process. The functions described in the following enable to realize these kinds of marking.

```
Sub Main

Debug.Print MonthName (Month(Date))
Debug.Print Weekday (Now)
Debug.Print WeekdayName (Weekday(Now))

End Sub
```

April
6
Freitag

Program example 27

However, for the function Weekday the following agreement has to be applied:

(1) - Sonntag
(2) - Montag
(3) - Dienstag
(4) - Mittwoch
(5) - Donnerstag
(6) - Freitag
(7) - Samstag

For a detailled description of these functions and their syntax please refer to the Online-help.

7.2. Manipulation and handling of texts

A selection of functions important for handling and manipulating texts are explained below. The functions described represent only a part of the complete extent of functions. Further functions, their syntax und functioning are available in the Online-help.

Len

Syntax: Len (String$)

Description: determines length of String$

Example:

```
Sub Main
   Debug.Print Len("Laser")
   Wert = Len("Laser")* 2
   Debug.Print Wert
End Sub
```

5
10

Program example 28

Left

Syntax: Left$ (String$,Len)

Description: creates string with length Len, starting on the left at string

Example:

```
Sub Main
   Debug.Print Left$("Laser",2)
End Sub
```

La

Program example 29

Mid

Syntax: Mid$ (String$,Index,[Len])

Description: creates string with length Len, starting at position Index.

Example:
```
Sub Main
  Debug.Print Mid$("Laser",3,2)
End Sub
```

se

Program example 30

Right

Syntax: Right$ (String$,Len)

Description: creates string with length Len, starting on the right at string.

Example:
```
Sub Main
   Debug.Print Right$("Laser",2)
End Sub
```

er

Program example 31

Str

Syntax: Str$ (Num)

Description: creates string out of numerical variable

Example:
```
Sub Main
  Debug.Print Str$(-5*8)
End Sub
```

-40

Program example 32

StrReverse

Syntax: StrReverse$ (String)

Description: creates string in reversed order of starting string.

Example:
```
Sub Main
  Debug.Print StrReverse$ (“Laser“)
End Sub
```

resaL

Program example 33

UCase

Syntax: UCase$ (String)

Description: creates string turning all small letters into capital letters.

Example:
```
Sub Main
  Debug.Print UCase$ (“Laser“)
End Sub
```

LASER

Program example 34

7.3. Mathematical Operations

A series of mathematical functions can be used for calculations within the execution of the program.

Sin, Cos, Tan, Atn

These functions supply the trigonometrical functions of numerical values.

Syntax:
```
Sin (Num)
Cos (Num)
Tan (Num)
Atn (Num)
```

Example:
```
Sub Main
   Debug.Print Sin(1)
   Debug.Print Cos(1)
   Debug.Print Tan(1)
   Debug.Print Atn(1)
End Sub
```

0,841470984807897
0,54030230586814
1,5574077246549
0,785398163397448

Program example 35

Exp, Log

These functions enable to calculate the exponential, i.e. the logarithm function.

Syntax:
```
Exp (Num)
Log (Num)
```

Example:
```
Sub Main
   Debug.Print Exp(1)
   Debug.Print Log(1)
End Sub
```

2,71828182845905
0

Program example 36

Sqr

This function supplies the square root of numerical values.

Syntax:
```
Sqr (Num)
```

Example:
```
Sub Main
   Debug.Print Sqr(81)
End Sub
```

9

Program example 37

Abs, Fix, Int, Round, Sgn

Syntax:
```
Abs (Num)
Fix (Num)
Int (Num)
Round (Num,[Stellen])
Sgn (Num)
```

Description:
Abs: forms the absolute value
Fix: forms an integer variable cutting off decimal places
Int: forms an integer variable carrying out a curve
Round: Rounds a numerical value to the number of decimal places given
Sgn: Returns a preceding sign value.

Example:

```
Sub Main
   Debug.Print Abs(-100)
   Debug.Print Fix(-100.2)
   Debug.Print Int(-100.8)
   Debug.Print Round(-100.88,1)
End Sub
```

100
-100
-101
-100,9

Program example 38

7.4. Operators

n → numerical value
s → string

- n1	preceding sign changing from n1.
n1 ^ n2	takes n1 as exponential (n2-times)
n1 * n2	multiplies n1 by n2.
n1 / n2	divides n1 by n2.
n1 \ n2	divides integer value of n1 by the integer value of n2.
n1 + n2	adds n1 and n2.
s1 + s2	connects s1 with s2.
n1 - n2	subtracts n2 from n1.
n1 & n2	connects n1 with n2.
n1 < n2	returns True, if n1 smaller than n2.
n1 <= n2	returns True, if n1 smaller or equal n2.
n1 > n2	returns True, if n1 bigger than n2.
n1 >= n2	returns True, if n1 bigger or equal n2.
n1 = n2	returns True, if n1 equal n2.
n1 <> n2	returns True, if n1 not equal n2.
s1 = s2	returns True, if s1 equal s2.
s1 <> s2	returns True, if s1 not equal s2.
n1 And n2	Bit-wise AND link of n1 with n2.
n1 Or n2	Bit-wise OR link of n1 with n2.

```
Sub Main
   N1 = 10
   N2 = 3
   S1$ = "asdfg"
   S2$ = "hjkl"
   Debug.Print -N1       '-10
   Debug.Print N1 ^ N2    ' 1000
   Debug.Print Not N1     '-11
   Debug.Print N1 * N2    ' 30
   Debug.Print N1 / N2    ' 3.3333333333333
   Debug.Print N1 \ N2    ' 3
   Debug.Print N1 + N2    ' 13
   Debug.Print S1$ + S2$  '"asdfghjkl"
   Debug.Print N1 - N2    ' 7
   Debug.Print N1 & N2    '"103"
   Debug.Print N1 < N2    'False
   Debug.Print N1 <= N2   'False
   Debug.Print N1 > N2    'True
   Debug.Print N1 >= N2   'True
   Debug.Print N1 = N2    'False
   Debug.Print N1<>N2   'True
   Debug.Print S1$ = S2$  'False
   Debug.Print S1$ <> S2$ 'True
   Debug.Print N1 And N2  ' 2
   Debug.Print N1 Or N2   ' 11
End Sub
```

Program example 39

7.5. Handling files

A file is a structured quantity of data which can for example be saved on the hard disk of your PC. To be able to have access to data records of a file this must be opened. Then, it is possible to have access to the file for making inputs and/or outputs.

This function enables to keep a record in files of marking processes, i.e. to use data from files for the laser marking without having to enter them manually.

MagicMark-Visual Basic is executing input and output processes by the help of data numbers. This number is assigned to a file or unit when it is opened by the command OPEN.

The physical file is described by its file description being a string in form of:

[Unit:] [directory] file name

The name of the unit determines which input/output unit is used. The track is informing *MagicMark* about the directory comprising the file requested. The file name determines which file must be searched on a certain unit.

Note: The file specification for data communication units differs from this (please refer to chapter 12.2. – Serial communication)

Sequential files

Sequential files can be created in a very easy way. Data entered into a sequential file are saved sequentially in the order of the handing over of the several data. When reading such a file this file must be worked off from the beginning on. This is an important disadvantage of sequential files.

The command OPEN opens, i.e. creates the file

Syntax: Open Dateiangabe For Modus As # Dateinummer [Len = Satzlänge]

Modes: Output – output
Append - extension
Input - read

Example:

Open c:\Temp\Mark.txt For Output As #1

After having opened such a file data records can be filed by the command PRINT.Please note that control characters are inserted between the various elements. If this is not done a line forms a complete string.

Example:

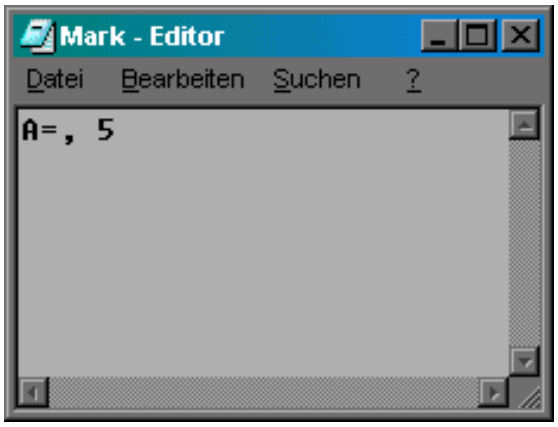
Print #1,"A=";“,“;A

By means of this command String A= as well as the variable A is taken out into file Mark.txt.

Then, the file must be closed again duly.

Close #1

```
Sub Main

  A%=5

  Open "c:\temp\Mark.txt" For Output As #1
  Print #1,"A=";",";A%
  Close #1

End Sub
```



Program example 40

If a further data record should be coupled to a consisting file this is made by the help of the function APPEND.

```
Sub Main

   B%=10
   Open "c:\temp\Mark.txt" For Append As #1
  Print #1,"B=";",";B%
  Close #1

End Sub
```

Now, the file Mark.txt is consisting of two lines with two elements each.

Program example 41

The following example makes clear how this data can be read out of the file and then for example used for the marking.

Doing this is possible by means of the commands Input and Line Input. Input enables to read out the elements of each line, whereas Line Input reads out the complete line as string.

```
Sub Main

  Dim Text$
  Dim Wert%
  Dim Zeile$
  Open "c:\temp\Mark.txt" For Input As #1
    While Not EOF(1)
        Input #1,Text$,Wert
        Line Input #1,Zeile$
        Debug.Print Text$;Wert%
        Debug.Print Zeile$
    Wend
  Close #1

End Sub
```

A= 5
B=, 10

Program example 42

Files with direct access

Directly accessible files allow to have access to selected data records. This is realized by the help of a data record number.

```
Sub Main
   Dim V As Variant
   V="Text"
   Open "C:\temp\Beispiel.txt" For Random As #1 Len =15
   Put #1,1, V
   Close #1
End Sub
```

Program example 43a

```
Sub Main
   Dim V As Variant
   Open "c:\temp\Beispiel.txt" For Random  As #1 Len=15
   Get #1,1, V
   Debug.Print V
   Close #1
End Sub
```

Text

Program example 43b

In example 43a, a file with direct access is created. The length of a string is max. 15 characters. A data record is entered into the file by using the command Put.

Put StreamNum, [RecordNum], var

By the help of the command Get (program example 43b) the data saved like this can be read out again.

Get StreamNum, [RecordNum], var

The record number is describing the position within the file.

7.6. Input Box

The Input box offers, additionally to the User-Dialogue-Editor, a simplified possibility to enter data.

Syntax:

InputBox[$](Prompt$[, Title$][, Default$][, XPos, YPos])

Prompt$: Dialogue appearing in the input window and asking the user to act.

Title$: Title of the Input Box

Default$: Here, the user can be given special instructions

Xpos: X-position, at which the Input Box becomes visible on monitor.

Ypos: Y-position at which the Input Box becomes visible on monitor.

Program example 44 shows the functioning of the Input Box.

```
Sub Main
  L$ = InputBox("Bitte Namen eingeben:" ,"Eingabefenster"
               ,"Michael Müller")
  Debug.Print L$
  DCS.TextTTF(9,0,0,L$,Arial,5,1,2,0,3)
End Sub
```

Program example 44

7.7. Message Box

The Message Box is the counterpart to the Input Box. This box enables in a simple way how to present data and variables on screen.

Syntax:

MsgBox(Message$[, Type][, Title$])

Message$: This string is shown on screen as information for the user

Type: The type contains the buttons shown

Button	Value	Effect
vbOkOnly	0	OK
vbOkCancel	1	OK and cancel
vbAbortRetryIgnore	2	Abort, retry, ignore
vbYesNoCancel	3	Yes, no, cancel buttons
vbYesNo	4	Yes, no
vbRetryCancel	5	Retry, cancel

Icon	Value	Effect
	0	No Symbol
vbCritical	16	Stop Symbol
vbQuestion	32	Question Symbol
vbExclamation	48	Attention Symbol
vbInformation	64	Information Symbol

Title$: This string describes the title of the Message Box.

The program examples 45 examplify different applications of the Message Box.

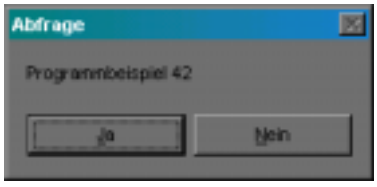
```
Sub Main
  'Buttons
  MsgBox("Programmbeispiel 45",0,"Test")
  MsgBox("Programmbeispiel 45",1,"Test")
  MsgBox("Programmbeispiel 45",2,"Test")
  MsgBox("Programmbeispiel 45",3,"Test")
  MsgBox("Programmbeispiel 45",4,"Test")
  MsgBox("Programmbeispiel 45",5,"Test")
  'Icons
  MsgBox("Programmbeispiel 45",16,"Test")
  MsgBox("Programmbeispiel 45",32,"Test")
  MsgBox("Programmbeispiel 45",48,"Test")
  MsgBox("Programmbeispiel 45",64,"Test")
End Sub
```

Program example 45a

In the first part of the program the Message Boxes supply with different buttons. The selection made by the user can be queried.

```
Sub Main
  If MsgBox("Programmbeispiel 45",4,"Abfrage")= vbYes
Then
    MsgBox("Es wurde Ja gedrückt!",16,"Information")
      Else
      MsgBox("Es wurde Nein gedrückt!",32,"Information")
  End If
End Sub
```

Program example 45b



7.8. Pop Up Menu

The function PopUpMenu enables to create a menu by which different actions can be realized.

Syntax: ShowPopupMenü (StrArray$( )
[, PopupStyle][, XPos, YPos])

In StrArray$ all possibilities of selecting are fixed in an one-dimensional field. The PopupStyle describes the direction of the menu in connection with the X and Y coordinate. The following values are allowed for the PopupStyle:

PopupStyle	Value	Effect
vbPopupLeftTopAlign	0	Basic setting
vbPopupUseLeftButton	1	Selection only possible with left-click
vbPopupUseRightButton	2	Selection possible with left or right click
vbPopupRightAlign	4	Menu is in the right corner of the x-position
vbPopupCenterAlign	8	Menu is centred around the x-position
vbPopupVCenterAlign	16	Menu is centred around the y-position
vbPopupBottomAlign	32	Menu button is at the y-position

Program example 46 shows how using this function

```
Sub Main
  Dim Items(0 To 2) As String
  Items(0) = "Kreis &K"
  Items(1) = "Rechteck &R"
  Items(2) = "Vector   &V"

  Auswahl = ShowPopupMenü (Items)

  If Auswahl = 0 Then Kreis    ' Popup Menü mit Auswahl Kreis
  If Auswahl = 1 Then Rechteck ' Popup Menü mit Auswahl Rechteck
  If Auswahl = 2 Then Vector   ' Popup Menü mit Auswahl Vector
 End Sub

Function Kreis
  DCS.Circle(9,0,0,10,0,360,False)
End Function

Function Rechteck
  DCS.Rectangle(9,0,0,10,10,False)
End Function

Function Vector
  DCS.Vector(0,0,20,20)
End Function
```

Program example 46

7.9. Function Dialogue

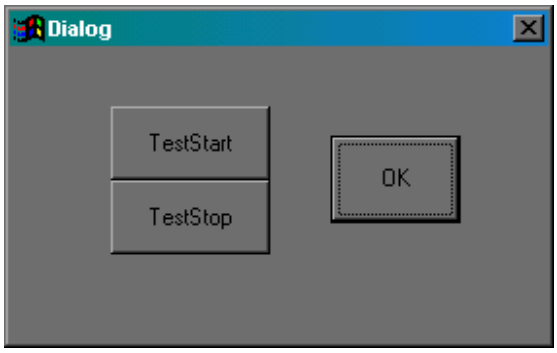
The function Dialogue in the User dialogue menu can among others be used to keep a dialogue window permanently opened during the execution of a program. The following example explains how this is working.

```
Sub Main
  Begin Dialog UserDialog 370,140,"Dialog",.DialogFuncStatus
    OKButton 220,42,90,42
    PushButton 70,28,110,35,"TestStart",.TestStart
    PushButton 70,63,110,35,"TestStop",.TestStop

  End Dialog
  Dim dlg As UserDialog
  Dialog dlg

End Sub

Function DialogFuncStatus%(DlgItem$, Action%, SuppValue%)
  Static TestRun As Boolean

  Select Case Action%
  Case 1

  Case 2

    If DlgItem$ = "TestStart" Then
      TestRun = True
      DialogFuncStatus% = True
    End If
    If DlgItem$ = "TestStop" Then
      TestRun = False
      DialogFuncStatus% = True
    End If
```

```
  Case 3

  Case 4

  Case 5

    If TestRun = True  Then
      DOIT
    End If
    DialogFuncStatus% = True
  End Select
End Function
```

```
Sub DOIT
  DCS.Speed(200)
  DCS.Circle(9,0,0,5,0,360,False)
End Sub
```

When calling the User Dialogue the function *DialogFuncStatus* will be worked off. A Select-Case-Branch is analysing the actions of the user.



Program example 47

The routine DOIT is repeated until it is stopped by the user, i.e. the program is finished by the OK-button.

8. Laser and scanning control system

The control instructions described in this chapter serve for the direct laser and scanning control system. This commands can be divided into four sections:

- Marking paramters
- Control system
- marking
- supervision

8.1. Marking parameters

Marking parameters are such characteristics which define the marginal conditions for the marking jobs. These are among others the power, frequency and the spot velocity.

This again can be divided into two groups. The first group is directly affecting the reaction of the material, whereas the second group is influencing the geometry of the marking image.

Material settings

Among these parameters are:

- power
- frequency
- speed

For setting these parameters the following commands are used:

♦ Setting of laser power:

DCS.Power(Power)
Sets laser power in per cent
Type of variable: long

♦ Setting of laser power with delay:

DCS.PowerWait(Power,Wait_ms)
Sets laser power in per cent and waits for delay time
Type of variable: long

♦ Setting of frequency:

DCS.QSF(Frequenz)
Sets Q-switch frequency in Hz
Type of variable: long

♦ Setting of pulse width:

DCS.QSF_PW(QSF_PW)
Sets pulse width in µs
Type of variable: long

By the help of the below mentioned functions

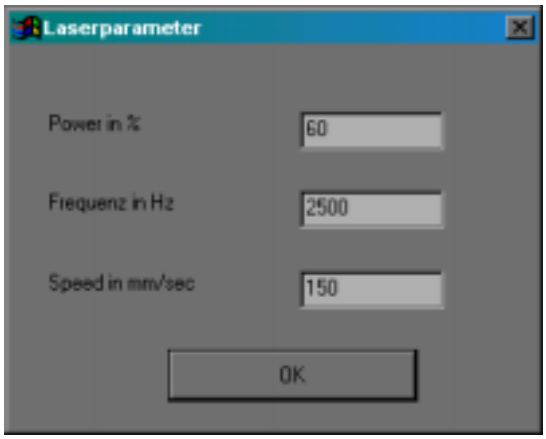
DCS.GetPower()
DCS.GetQSF()
DCS.GetSpeed()
DCS.GetQSF_PW()

it is possible to read out these laser parameters again. The return value is one of the type of variables *long.*

The program example 48 shows how using these commands.

```
Sub Main
   LaserParameter(PWR,QSF,SPEED)

   DCS.Power(PWR)                          'Setzen der
         Laserparameter
   DCS.QSF(QSF)
   DCS.Speed(SPEED)

   Leistung = DCS.GetPower()    ' Auslesen der Parameter
   Frequenz = DCS.GetQSF()
   Geschwindigkeit = DCS.GetSpeed()

   Debug.Print Leistung;Frequenz;Geschwindigkeit

End Sub
```

```
Function    LaserParameter(PWR,QSF,SPEED)
Begin  Dialog  UserDialog  400,203,"Laserparameter“
   Text 30,35,140,21,"Power in %",.Text1
   Text 30,77,140,21,"Frequenz in Hz",.Text2
   Text 30,119,140,21,"Speed in mm/sec",.Text3
   OKButton 120,161,190,28
   TextBox 220,35,110,21,.Power
   TextBox 220,77,110,21,.Frequenz
   TextBox 220,119,110,21,.Geschwindigkeit
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg

  PWR=dlg.Power
  QSF=dlg.Frequenz
  Speed=dlg.Geschwindigkeit

End Function
```



Program example 48

Setting the pulse width is very important. Standard setting for this is 6µsec. This value results from considering the maximum Q-switch frequency of 75KHz. This frequency corresponds to a cycle duration of 13,33µsec. The proportion of the pulse signal for controlling the Q-switch should not be smaller than 1:1. Therefore, 7,33µsec is the loading time for the laser crystal, the rest is for the emission of the pulse. As the length of the actual laser pulse is in the range of 20.80 nsec. a shortening of the pulse width is possible. Changing the pulse width is very important for marking by tempering. Thus, after the emission of the laser pulse it is possible to aftertreat the penetration point some µsec. long by cw-irradiation.

8.2. Control system

Switching on/ switching off routines of the laser described in chapter 4 can just as direct be initialized with commands. This way states of the laser can automatically be changed during the execution of programs.

Thus, the shutter for example can be opened and closed during a program run, or the laser can be moved into the StandBy mode.

Additionally, an automatic marking is made possible for the use within automatic production lines by the help of a digital in-/output.

- Shutter:

DCS.LaserShutter(State)
Opens/closes shutter
States: True=open / False=closed
Type of variable: bool

♦ Pilot-Laser:

DCS.LaserPilot(State)
Activates/deactivates pilot laser
States: True=on / False=off
Type of variable: bool

♦ Start of marking:

DCS.Input(0)
Reads input no. 0
States: True=high (24V) False=low (0V)
Type of variable: bool

♦ End of marking:

DCS.Output(0,State)
Sets output no. 0 to high (24V-State=True) low (0V-State=False)
Type of variable: void

```
Sub Main
  DCS.LaserON()
  DCS.LaserPilot(True)
  DCS.LaserShutter(True)

  DCS.Circle(9,0,0,3,0,360,False)

  DCS.LaserShutter(False)
  DCS.LaserStandBy()
End Sub
```

Program example 49

The program 49 activates the laser, switches on the pilot laser, opens the shutter. Then, the laser draws a circle by means of the parameters last used. After that, the laser is moved into state StandBy.

8.3. Marking

The commands mentioned below serve for the direct scanning control system, i.e. the actual flow deviation. Thus, using a series of commands enables to process easy graphic basic elements, texts, barcodes as well as vector graphics.

Rotation

Each graphic element treated in the following can be rotated in a certain angle around a defined point.

DCS. Rotation(A,B,C)

A – rotation coordinate X [mm]
B – rotation coordinate Y [mm]
C – rotating angle [°]

Vector

This function enables to draw a line from coordinate A to coordinate B.

DCS. Vector(A,B,C,D)

A – starting coordinate X [mm]
B – starting coordinate Y [mm]
C – ending coordinate X [mm]
D – ending coordinate Y [mm]

```
Sub Main
  For i = 0 To 350 Step 10
    DCS.Rotation(0,0,i)
    DCS.Vector(8,8,10,10)
  Next i
End Sub
```

Program example 50

Example 50 shows an easy example for marking a scale in a range of 10°

steps. For this, a loop is passed with a step of 10 (rotating angle).

Position

This command guides the galvanometer mirrors to the defined coordinate X,Y.

DCS. Pos(A,B)

A – position X [mm]
B – position Y [mm]

Rectangle

This function enables to draw a rectancle. Three variations are possible:

- Simple rectangle
- Filled rectangle
- Rectangle with rounded corners

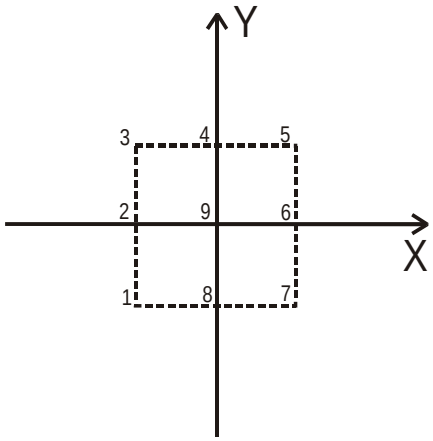
DCS.Rectangle(A,B,C,D,E,G)
DCS.Rectangle_Filled(A,B,C,D,E,G)
DCS.Rectangle_Rounded(A,B,C,D,E,F,G)

A – basic reference point
B – basic coordinate X [mm]
C – basic coordinate Y [mm]
D – width [mm]
E – height [mm]
F – radius [mm]
G – filling [True-filled / False-not filled)
The parameter basic reference point allows the alignment of the rectangle into X and Y direction.

For this, the following convention is valid:

BaseRef	X	Y
1	Left-justified	Lower edge aligned
2	Left-justified	Centred
3	Left-justified	Upper edge aligned
4	Centred	Upper edge aligned
5	Right-justified	Upper edge aligned
6	Right-justified	Centred
7	Right-justified	Lower edge aligned
8	Centred	Upper edge aligned
9	Centred	Centred

Alignment by basic reference point



Basic reference point

Circle

This function enables to draw a circle.

DCS.Circle(A,B,C,D,E,F,G)

A – basic reference point
B – central point X [mm]
C – central point Y [mm]
D – radius [mm]
E – starting angle [°]
F – ending angle [°]
G – filling [True-filled / False- not filled)

```
Sub Main
  DCS.Rectangle(9,0,0,10,10,False)
  DCS.Rectangle_Filled(9,0,0,10,10,False)
  DCS.Rectangle_Rounded(9,0,0,10,10,3,False)
  DCS.Circle(9,0,0,5,0,360,False)
End Sub
```

Program example 51

Texts

By the help of this function any texts can be created. This text can be influenced in form and design by a multitude of parameters.

DCS.TextTTF(A,B,C,D,E,F,G,H,I,J,K)

A – basic reference point
B – basic coordinate X [mm]
C – basic coordinate Y [mm]
D – text contents as string or variable
E – font name
F – height of text [mm]
G – distance between characters
H – font width [400-800]
I – normal [0] / in italics [1]
J – quality [0-low / 1-medium / 2-high]
K – filling [True-filled / False-not filled)

Parameter D defines the contents of the text to be marked. If that text is written in inverted commas this string is actually marked. The position is fixed analog to the graphic elements treated so far by the parameters A, B and C. Should a variable be marked instead of a string the name of the variable is given in parameter D. However, make sure that the variable is declared. The height of the font is stated in parameter F and the distance between characters can be changed in parameter G.

It is valid:

0.5 distance reduced by factor 2
1 normal distance
2 distance increased by factor 2

Naturally, any other values are admissible.

The font width is fixed in parameter H. It is valid:

400 normal font
800 fat font

Any other valence is allowed. However, please note that certain font types do not admit any fat font.

Parameter I determines, whether the font is marked in normal printing style or in italics.

0 normal printing style
1 in italics

Each character of a TrueType Font consists of a finite number of vectors. As higher this number is, as cleaner the font will be marked. If the number of vectors is reduced the font will become more and more „cornered“, in exchange the scanning speed clearly increases. Therefore, according to application this parameter must be set correspondingly.

0 bad quality
high scanning speed
1 average quality
average scanning speed
2 high quality
slow scanning speed


low resolution *high resolution*

```
Sub Main
  Dim Hello As Single
  Hello = 110

  DCS.TextTTF(9,0,0,"Hello",“Arial“,10,1,400,False,1,False)
  DCS.TextTTF(9,0,10,Hello!,“Arial“,10,1,400,False,1,false)
End Sub
```

Program example 52

Vector graphics (HPGL)

Complex graphics, as for example expensive type plates, are created in a graphic program by using the HPGL-interface. That means, that every graphic program disposing of an export filter according to HPGL is in a position to create graphics which can be marked by the help of Magic*Mark*, as for example Corel Draw.

To integrate HPGL files into Magic*Mark* the following command is used:

DCS.ImportHPGL(A,B,C,D,E)

A – basic reference point
B – basic coordinate X [mm]
C – basic coordinate Y [mm]
D – HPGL file
E – pen

The functioning of the commands mentioned above will be exemplified in the following example. For this, a file has been created in a graphic program, for example Corel Draw, named Logo,plt. The extension *.plt points out that this is a HPGL file.

Logo.plt

```
Sub Main
    DCS.ImportHPGL(9,0,0,“C:\Logo.plt“,255)
End Sub
```

Program example 53

Parameter E, the pen number, is very important. By assigning a pen to a certain level in the graphic program only selected levels for example can be marked. This enables the user to change the laser parameters from level to level. If you think for example of creating labels by means of laser radiation the whole label can be created in one file, but text and frame to be cut out are on different levels. Thus, different laser parameters can be assigned to these levels. If the parameter E is set to value 255, all levels are marked.

Barcode

Magic*Mark* offers the possibility to mark all current types of barcodes. Like it is done for the other marking functions the barcode to be marked is defined by parameters.

DCS.Barcode(A,B,C,D,E,F,G,H,I,J)

A – type of barcode
B – width of module
C – data
D – data appendix
E – basic reference point
F – basic coordinate X [mm]
G – basic coordinate Y [mm
H – height barcode 1 (MainHigh)

I – height barcode 2 (AddOnHigh)
J – height barcode 3 (LongHigh)
K – Inversion [0-normal / 1-inverted]

A – type of barcode

The types of barcodes available are listed in the appendix of this manual.

B – Width of module

The width of the module determines the relation between bar and gap in the barcode. Standard value for this is 1. Increasing the value effects that the ratio is moving to bigger gaps. When reducing the value the bars move respectively closer together.

C – Data

This parameter contains the data to be transformed into a barcode. For that, it is important to observe respectively appendix B, particularly when selecting the characters and the number of places to be coded. When using strings please note that these must be put in inverted commas, otherwise the string supposed will be interpreted as variable.

D – Data - appendix

For barcodes with an appendix (bar-code nos. 21-28) the data for the appendix are filed in this parameter. Further details are according point C – data.

E,F,G – positioning

According to the commands treated so far the positioning here is also effected analog.

H, I, J – height barcode

The following three parameters H, I, J define the height of the barcode. The convention as mentioned below is valid:

Heights of barcodes

For simple barcodes (Nos. 1-16) only the indication of the parameter H (MainHigh) is decisive, LongHigh and AddOnHigh get 0 assigned as value.

K – inversion

Parameter K serves for marking barcodes both on light and dark materials by indicating, whether the barcode has to be marked normally or inverted. It is valid:

- Marking of light materials with dark reaction: K=0
- Marking of dark materials with light reaction: K=1

Program example 54 exemplifies the use of this function.

```
Sub Main
  DCS.Barcode(16,0.5,1234,0,9,0,-15,5,0,0,0)      ' Code
128C
  DCS.Barcode(21,0.5,1234567,12,9,0,0,5,3,7,0) ' EAN-8+2
  DCS.Barcode(11,0.5,"Hello","",9,0,15,5,0,0,0)    ' EAN-
128B
End Sub
```

Program example 54

After having marked a barcode, one should check regularly if this is readable.

Data Matrix Code (ECC 200)

The DataMatrix Code is a two-dimensional code by the help of which a multitude of information can be accommodated in a very confined space. The Data Matrix Code the most common today is the ECC 200.

Example: Data Matrix Code

For calling the Data Matrix function the following command is used:
DCS.DataMatrix(A,B,C,D,E,F,G,H,I,J)

A – module width
B – data
C – lines
D – columns
E – Data Matrix type
F – stil
G – format
H – border
I – basic reference point
J – basic coordinate X [mm]
K – basic coordinate Y [mm

A- Module width

The module width determines the proportion of dimensions in the Data Matrix Code. Basic value for the module width is value 1. This means that each square of the code is 1mm x 1mm. Thus, a code with 12x12 pixels has an extension of 12mm x 12mm. The resolution of the laser beam is naturally many times higher. The module width can be reduced accordingly in dependance of the material to be marked and the features of the scanner.

B- data

The code ECC 200 can process both alpha-numerical data and strings. If a string is processed this has to be put in inverted commas.

C,D- lines, columns

The number of squares in vertical and horizontal direction are defined in the parameters C and D. For the Data Matrix Code is valid:
Symbol forms: max. 24 x square
max. 6 x rectangular
Thus results a min./max. size of symbol:

Min./Max: 10x10...144x144 (lines)
8x18 ...16x48 (lines)

Please note also that only even-numbered stagings are allowed.

Example: 14 x 14
38 x 38
8 x 18

Most important is value 0 for the parameters C and D. If these parameters are set to 0, then automatically the least possible number of cells is used.

E- Data Matrix type

The ECC 200 Code, as already mentioned, is the form of the Data Matrix Code the most frequently used. To complete the picture, further possible types able to be defined by parameter E are stated below.

Data Matrix Code	
Code	**Parameter E**
ECC 000	0
ECC 010	1
ECC 040	2
ECC 150	3
ECC 060	4
ECC 070	5
ECC 080	6
ECC090	7
ECC 100	8
ECC 110	9
ECC 120	10
ECC 130	11
ECC 140	12
ECC 200	**26**

F- Style

The function Style enables to give a mirrored view of the code.

0: normal
1: mirrored

G- format

Parameter G defines the format of the code, but is only effective for the codes ECC 000 up to ECC 140. When using the standard code ECC 200 this parameter has no influence on the result.

Parameter G can assume values from 1 to 6:

1: for numerical data
2: capital letters
3: capital letter, numbers and punctuation (“.“ ; “,“)
4: capital letters and numbers
5: for the first 128 ASCII - characters
6: 8 Bit / user-defined

H- boarder

This parameter defines the width of the boarder.

0: Standard

```
Sub Main
  DCS.DataMatrix(1,"ACI",0,0,26,0,0,0,9,0,0) 'ECC200
End Sub
```

Program example 55

Filling routine

For filling texts a filling routine can be used. It is possible to influence both the filling algorithm (cross hatching / parallel line / contour filling) and the distance between lines.

DCS.Fill(A,B,C,D,E)

A – filling mode
B – distance between lines
C – angle
D – 2nd distance between lines
E – 2nd angel

A – filling mode

Optionally, the following modes are available:

0 – no filling
1 – parallel with a line
2 – parallel with two lines
3 – contour filling

5 – like 1, but without outline
6 – like 2, but without outline

B – distance between lines

The distance between the filling lines is directly stated in millimetres and is refering to the filling with only one line (for example filling mode A1)

C – angle

Parameter C defines the angle of the first filling line.

D – 2nd distance between lines

This is defined analog to parameter B, however for the 2nd filling line

E – 2nd angle

This is defined analog to parameter C, however for the 2nd filling line

The parameters D and E are not relevant for the filling modes 1 and 5. In this cases the parameters D and E should be set to 0.

For the filling mode 3 only the distance between lines is relevant.

A filling routine set remains active as long as another filling command is activated.

9. Communication

9.1. Start marking and end marking

According to chapter 10.2. the following routines can be used for the start/end commands:

- Start marking:

DCS.Input(0)
Reads output no. 0
States: True=high (24V) False=low (0V)
Type of variable: bool

- End marking:

DCS.Output(0,State)
Sets output to no. 0 High (24V-State=True) Low (0V-State=False)
Type of variable: void

For more detailled information about the hardware-technical connection of the laser to an automatic production line please refer to your operating manual.

The following example program serves for exemplifying the above mentioned commands:

```
Sub Main
  Begin Dialog UserDialog 370,140,"Dialog",.DialogFuncStatus
    PushButton 30,42,140,42,"JobStart",.JobStart
    PushButton
200,42,140,42,"Programmabbruch",.Programmabbruch
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
End Sub
```

```
Function DialogFuncStatus%(DlgItem$, Action%, SuppValue%)
  Static TestRun As Boolean

  Select Case Action%
  Case 1

  Case 2

    If DlgItem$ = "JobStart" Then
      TestRun = True
      DialogFuncStatus% = True
    End If

  Case 3

  Case 4

  Case 5

    If TestRun = True  Then
      DOIT
    End If
   DialogFuncStatus% = True
  End Select
End Function
```

```
Sub DOIT
  DCS.Speed(20000)
  DCS.Output(0,False)
  Do
    Start=DCS.Input(0)
  Loop Until Start = True
  DCS.Circle(9,0,0,5,0,360,False)
DCS.TestMarkingComplete(True)
  DCS.Output(0,True)
End Sub
```

Program example 56

After having used the JobStart button the program remains in a loop which is only left under the condition:

-signal start marking set -

The output end marking serves for indicating that the marking process is finished.

10. Error treatment

When executing the program it may be that errors occur regularly. Therefore, make sure when developing programs that the programs tolerate possible errors. Such fault-tolerant programs do not diminish the program run, but generally go on working where the error does not influene any more. Usually, this is the module which has caused the faulty module.

To be able to develop programs which are resistant against possible errors the command *On Error* can be used. This command enables to inform the program how to behave in case of an error. In this context the commands *Goto* and *Resume Next* are used.

Goto

Syntax: On Error Goto Zielmarke
...
Zielmarke:
...

If any error occurs in the program part after having used the command *On Error Goto Zielmarke* the program jumps to the aligning mark. Here, the user can for example be informed about the error occurred.

```
Sub Main
 Berechnung
End Sub
```

```
Function Berechnung
  On Error GoTo Fehler
  a!=InputBox ("Eingabe Zahl")
  t!=InputBox ("Eingabe Teiler")
  Ergebnis = a!/t!
  Debug.Print Ergebnis
  Exit Function
  Fehler:
  MsgBox("Es ist ein Fehler aufgetreten", 48, "Fehler")
End Function
```

Program example 57

The program example recognizes input errors possibly made by the user.
Among these are for example:

- division by 0
- input strings
- no input

Resume Next

Syntax: On Error Resume Next
...

This command effects that a program line in which an error occurs is skipped. By doing this errors are avoided, but may consequently have wrong results of calculations.

```
Sub Main
 Berechnung
End Sub
```

```
Function Berechnung
  On Error Resume Next
  a!=InputBox ("Eingabe Zahl")
  t!=InputBox ("Eingabe Teiler")
  Ergebnis = a!/t!
  Debug.Print a!;t!;Ergebnis
  Exit Function
End Function
```

Program example 58

ACI
Laser-Components

Appendix A – Laser commands

Marking parameters

long Power(Power)
long PowerWait(Power, Wait_ms)
long GetPower()
long QSF(QSF)
long GetQSF()
long QSF_PW(QSF_PW)
long GetQSF_PW()
long Speed(Speed)
long GetSpeed()
long Diameter(Diameter)
long GetDiameter()

double PinAngleX(Angle)
double GetPinAngleX()
double PinAngleY(Angle)
double GetPinAngleY()
BOOL PinX(Pin)
BOOL GetPinX()
BOOL PinY(Pin)
BOOL GetPinY()

long StartStopDelay(StartDelay, StopDelay)
long SetFirstPulse(Pos, Value)
void SetBreakAngleDelay(BreakAngle, BreakDelay)
BOOL SetMirrorX(MirrorX)
BOOL SetMirrorY(MirrorY)
long SetRotation(Rotation)

long SetJumpDelay(Pos, Value)
long SetFirstPulseONDelay(FPONusDelay)
double SetShrinkX(Value);
double GetShrinkX();
double SetShrinkY(Value);
double GetShrinkY();
long SetOffsetX(Value);
long GetOffsetX();
long SetOffsetY(Value);
long GetOffsetY();
BOOL TestMarkingComplete();

ACI
Laser-Components

Control system

long LaserON()
long LaserOFF()
long LaserStandby()
long GetLaserState() 0=OFF 1=Standby 2=ON
bool LaserShutter(State)
bool LaserPilot(State)

BOOL Input(Port)
void Output(Port, State)

Marking

void Rotation(RotPointX, RotPointY, RotDegree);
void Vector(X1, Y1, X2, Y2)
void Pos(X, Y)
void Rectangle(BaseRef, BasePointX, BasePointY, Width, Height, Fill)
void Rectangle_Filled(BaseRef, BasePointX, BasePointY, Width, Height, Fill)
BOOL Barcode(BarcodeType, ModuleWidth, MainMessage, AddOnMessage, BaseRef, BaseX, BaseY, MainHigh, AddOnHigh, LongHigh, Inverse)
BOOL DataMatrix(ModuleWidth, Message, rows, cols, ECCtype, style, format, border, BaseRef, BaseX, BaseY);
void Circle(BaseRef, MidPointX, MidPointY, Radius, AlphaStart, AlphaEnd, Fill)
void Rectangle_Rounded(BaseRef, BasePointX, BasePointY, Width, Height, Radius, Fill)
BOOL TextTTF(BaseRef, BasePointX, BasePointY, Text, FontName, TextHeight, SpaceAdjust, FontWeight, FontItalic, Quality, Fill)
short HPGLOpen(FileName)
void HPGLClose(Handle)
BOOL ImportHPGL(BaseRef, BasePointX, BasePointY, FileName, PenFlag)
void HPGL(Handle, BaseRef, BasePointX, BasePointY, PenFlag)
BOOL Fill(Mode, Size1, Angle1, Size2, Angle2)

Mode: 0 == no fill
1 == parallel 1 line
2 == parallel 2 line
3 == contour
4...7 as 0...3 without object contour

BOOL Polar(PointX, PointY, Radius)
BOOL Size(PointX, PointY, SizeX, SizeY)

ACI
Laser-Components

Appendix B – Types of barcodes

Types of barcodes

Parameter A	Type of barcoder	Numbers	Letters	Length
1	2 of 5	Yes	No	Variable
2	Interleaved 2 of 5	Yes	No	Variable
3	Code 39	Yes	Capital letters	Variable
4	Code 93	Yes	Capital letters	Variable
5	Codabar	Yes	No	Variable
6	EAN-8	Yes	No	7
7	EAN-13	Yes	No	12
8	UPC-A	Yes	No	11
9	UPC-E	Yes	No	10
10	EAN-128 A	Yes	Capital letters	Variable
11	EAN-128 B	Yes	Yes	Variable
12	EAN-128 C	Yes	No	Even-numbered
13	POSTNET(1.20)	Yes	No	Variable
14	Code-128 A	Yes	Capital letters	Variable
15	Code-128 B	Yes	Yes	Variable
16	Code-128 C	Yes	No	Even-numbered
21	EAN-8+2	Yes/Yes	No/No	7/2
22	EAN-8+5	Yes/Yes	No/No	7/5
23	EAN-13+2	Yes/Yes	No/No	12/2
24	EAN-13+5	Yes/Yes	No/No	12/5
25	UPC-A+2	Yes/Yes	No/No	11/2
26	UPC-A+5	Yes/Yes	No/No	10/5
27	UPC-E+2	Yes/Yes	No/No	11/2
28	UPC-E+5	Yes/Yes	No/No	10/5
31	PDF417	Yes	Yes	Variable