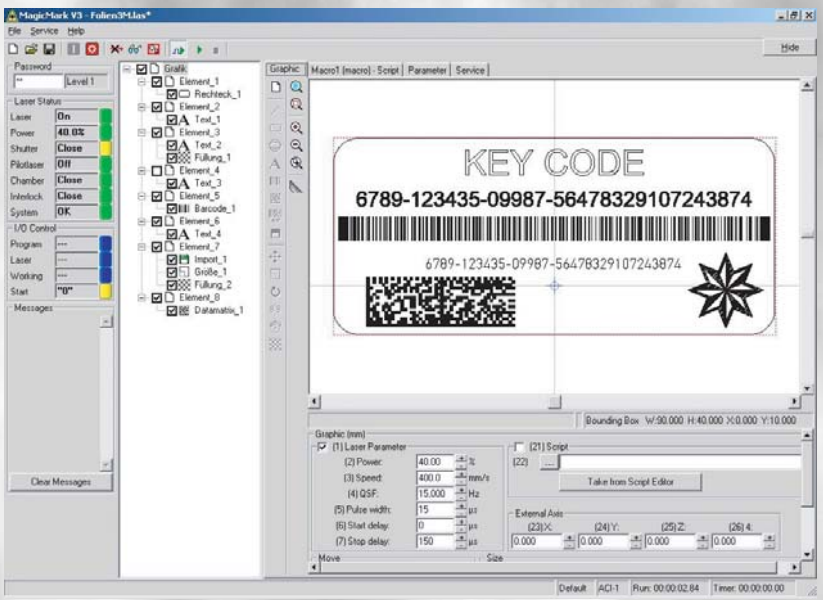
Manual
Laser Marking Software

Magic Mark

for Laser Marking Devices

DPL*Magic* Marker
DPL*Genesis* Marker
DPL*Nexus* Marker



ACI
Laser-Components

Copyright and Protection Rights

Manufacturer:	ACI Laser GmbH
	Österholzstraße 9
	D-99428 Nohra
Fon:	+49 3643 4152-0
Fax:	+49 3643 4152-77
Internet:	www.ACI-Laser.de
E-Mail:	info@ACI-Laser.de

Contents

1 Delivery

1.1 Scope of delivery

The software is supplied on CD.

Please make a backup copy before you work with this software. Instructions on how to do this are located in your Windows manual or in the Windows Online Help file.

1.2 Software manual

This software manual is part of the software. Please keep it in a safe place. It contains useful information regarding the program creation and the optimum operation of the connected laser.

Please submit this documentation together with the software if the software should be sold.

The vendor reserves the right to update this software manual at any time and without prior notification, in keeping with technical progress.

This manual was created in accordance with the currently valid technical status.

Please study the manual thoroughly.

1.3 Manufacturer

See the back of the cover sheet.

1.4 Warranty

The vendor, programmers and author have applied the greatest possible care in the creation of the software and the manual. Nevertheless, the vendor, programmers and author do not assume any guarantee for the software and the manual. In particular, the vendor, programmers and author offer no guarantee that this software corresponds to the demands of the customers, or that the software is entirely without errors. In no case can the vendor, programmers or author be made liable for any errors, destruction or resulting damage that can be derived from the use of the software.

2 Installation

2.1 System requirements

The following configuration is recommended for the successful application of this software:

- IBM-compatible Pentium 4 PC > 2 GHz,
- Windows 2000/XP operating system,
- 512 MB main memory,
- 100 MB free hard disk capacity,
- CD ROM drive,
- 2 free USB 2.0 interfaces (for the device and for the dongle),
- monitor (recommendation: 17 inch),
- keyboard, mouse.

2.2 Installation requirements

Windows XP

No particular requirements.
However, the latest Service Pack is recommended.

Windows 2000

The latest Service Pack is recommended.

2.3 Installation sequence

NOTE!
Consider absolutely the installation sequence described in the following!

1. Inserting the installation CD into the CD drive assembly.
2. Installation of the laser marking software from CD.
3. First putting of the delivereded dongle into an available USB port and installing the associated software component.
4. First connecting of the laser device with the computer and Installing the associated software component.

2.4 Software installation

NOTE!
Administrator rights are required to install this software!

To install the software, use the following procedure:

1. Start the computer and your operating system.
2. Insert the installation CD.
3. Should the installation not start automatically, initialise the file setup_343.exe in the main folder on the CD.
4. This software requires Microsoft .NET Framework. If this is not installed on your PC, then .NET Framework must be installed before the actual installation.
   An installation of the Microsoft Installer takes place automatically.
   In this case, follow the instructions displayed on the monitor.

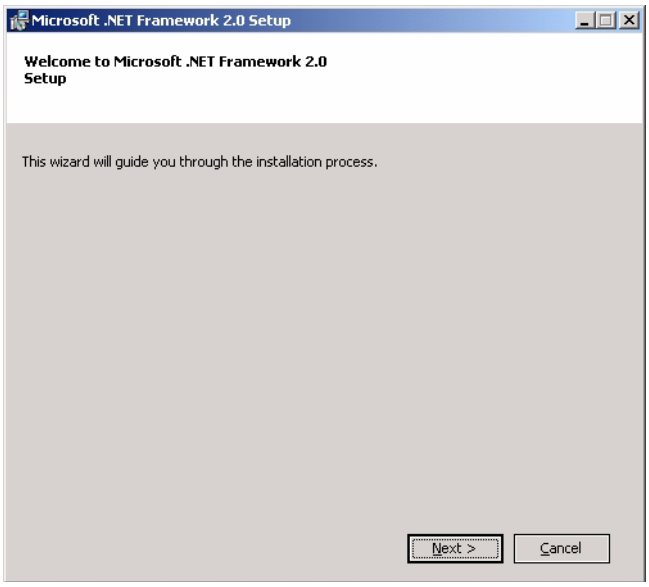


Fig. 1 Setup .NETFramework

5. This software also requires Microsoft Direct(X) runtime components. If this is not installed on your PC, then it must be installed before the actual installation.
   In this case, follow the instructions displayed on the monitor.

Fig. 2 Setup Direct(X) runtime components

6. Now the actual installation procedure for the software can take place.

Fig. 3 Setup Window

7. Click on the **Next** button.

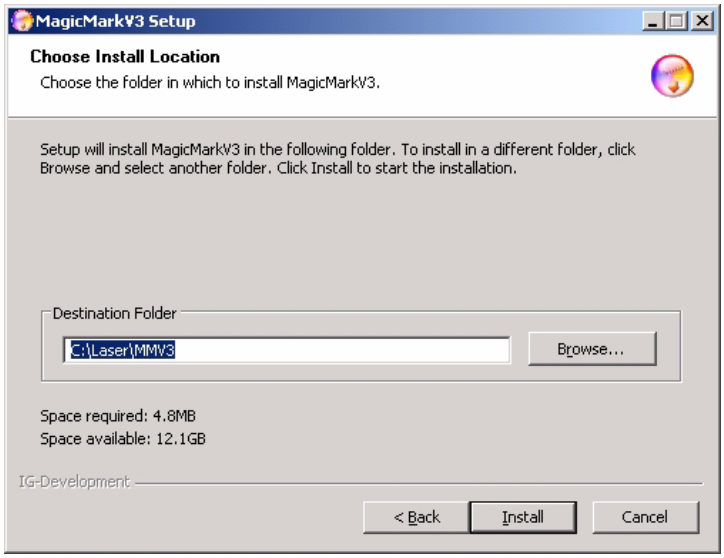


Fig. 4 Selection of the installation folder

8. Select the desired program folder with the help of the **Browse** button or use the default setting.
9. Click on **Next**.
10. Follow further instructions.
    The installation can be terminated at any time by pressing the **Esc** key or with the **Cancel** button.
11. Once the installation is complete, the message **Installation Complete** will appear.
12. Finish by clicking on the **Close** button.

The software will run now in Demo mode.

2.5 Dongle

1. Plug the delivered dongle into an available USB port.
   The software will only run in Demo mode without the dongle.
2. Because of the new hardware component an assistent to install the required software is started.
   Follow the installation instructions.

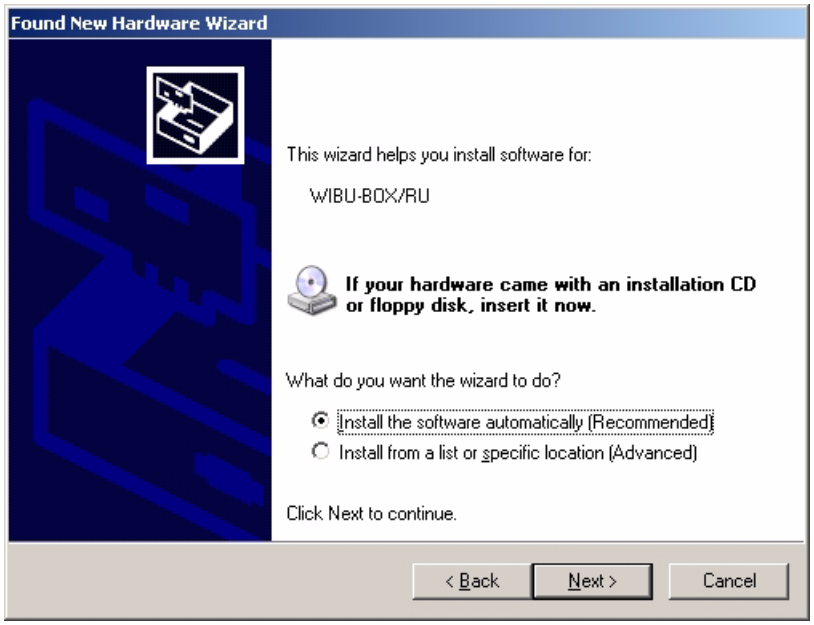


Fig. 5 Assistent to install the dongle software

2.6 Laser connection

1. Plug the connection cable of the laser device into an available USB port, connect the laser with the current supply and switch it on.
   The software will only run in Demo mode without the dongle.
2. Because of the new hardware component an assistent to install the required software is started.
   Follow the installation instructions.

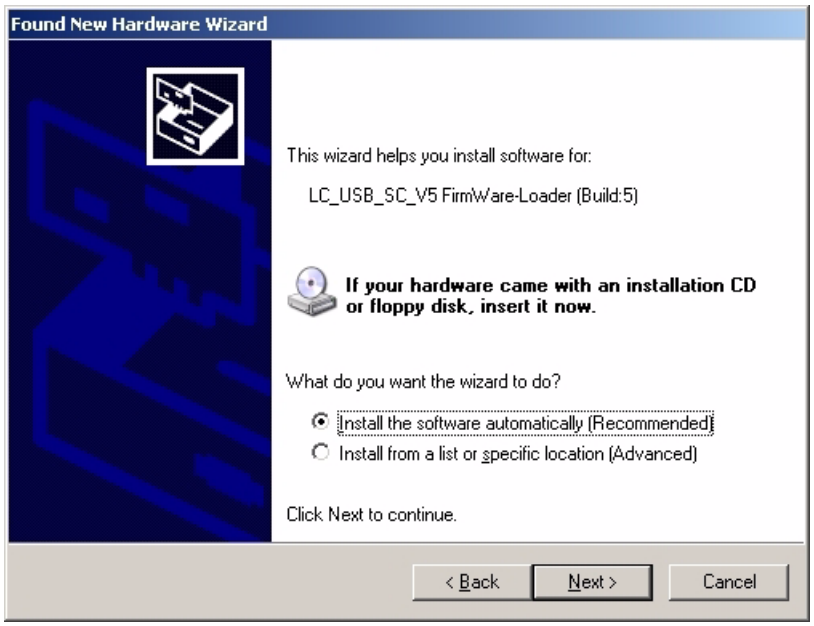


Fig. 6 Assistent to install the laser software

The installation is complete now.

2.7 Program start

The setup generates an entry under: **Start** → **Programs…**, use this entry to call up the program.

2.8 Specific parameters

NOTE!
The laser-specific parameters, which are enclosed with the laser, are imported via Service → Import Parameter.

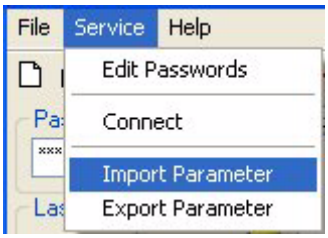


Fig. 7 Service menu

Using the **Import Parameter** option, the laser-specific parameters, which are enclosed with the laser, import these.
If you have changed the parameters, these can be saved with **Export Parameter**.

2.9 Exit program

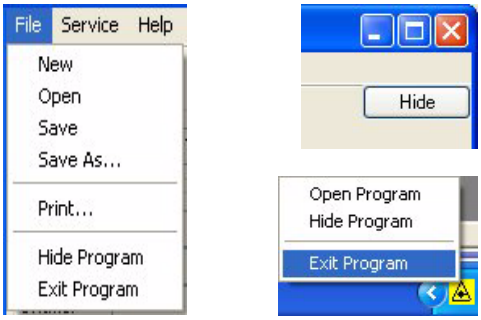


Fig. 8 File menu/Taskbar menu

End the program via: **File** → **Exit.**
You can also end the program using the **X** or by using the taskbar menu in the lower right.
If you merely want the program to run without a visible window, then select **Hide**, then you can only see the program as a task bar icon. In order to display the program window again, use the context menu (right mouse button) of the task bar icon and select **Open Program**.

2.10 Uninstalling

If necessary, uninstall the software via the operating system:
Start → **Settings** → **Control Panel** → **Software...**

2.11 Help

Fig. 9 Help menu

If the software manual is not at hand, the same information can be found by clicking on **Help**. You can also call up **Help** by pressing the F1 key. In this case **Help** is dependent on content, so that you, when it is possible, can be automatically guided to the right place.

2.12 Release

Fig. 10 Help menu

Information about the installed program version can be obtained by clicking on **Help** → **Info** after the program has been started.
The information window shows the version and build number of the program.

2.13 Language settings

The program adapts to the language that is set in the Control Panel. If no corresponding language version is on hand, the English version will be used.

3 Program description

This software is designed for the creation of marking programs as well as for the monitoring and control of a laser or a manufacturing cell, respectively.
The program was developed to make the integration of one or more lasers with a manufacturing cell possible.

The most important functions of the software are:

- The control of several lasers with only one PC:
  - The program is executed independently for each laser.
  - Each laser can exchange messages with other lasers.
  - The number of lasers is solely dependent on the power of the PC.
  - A license is required for each laser!
- Integration with the manufacturing cell:
  - High performance scripting language for controlling machine processes (handling systems)
  - Can be operated as a task bar symbol without a visible window.
  - Among other things, can communicate with other programs via a socket interface or file support.
  - Communication with external instruments via various interfaces.
  - Accesses databases.
  - Integration with host computer.
- Administration of all laser parameters:
  - Power output.
  - Speed.
  - Frequency.
  - Delays.
  - Interfaces.
- Graphic objects:
  - Precise input of parameters.
  - Fonts with single lines and all TrueType fonts.
  - Polar and multi line texts.
  - Various formats for numbers, date, time, etc.
  - 1D (normal) and 2D (data matrix) barcodes.
  - Basic objects (line, circle, rectangle).
  - Import of graphic images in various formats (DWG, DXF, HPGL, BMP, JPG and GIF).
  - Filling of all polygon-based objects.
  - Rotation of all objects.
  - Changes the size of all objects.
  - Moves all objects.

3.1 Program start

1. Start up the PC and the operating system.
2. Switch on the line voltage on the laser
3. Start the marking software via **Start** → **Programs** →**...**
4. The visual software interface is displayed.

The software generally starts with the standard **default** configuration. See also Current configuration on page 21.

Command line parameter

If you want to use a different configuration, then you must pass a command line parameter (/L:) followed by the name of the configuration to the software.
This usually takes place by using a shortcut.
So to use a configuration with the name **Laser1**, give: **/L:Laser1** as the parameter.

NOTE!
If two or more lasers are to be operated simultaneously, each laser (configuration) requires its own license!

Offline state

If the software cannot establish a connection to the laser or an existing connection is interrupted, the software shows this with the offline mode.

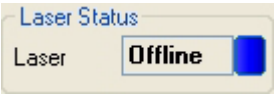


Fig. 11 Laser Status in the Offline mode

In this case the operating elements for controlling the laser are deactivated.

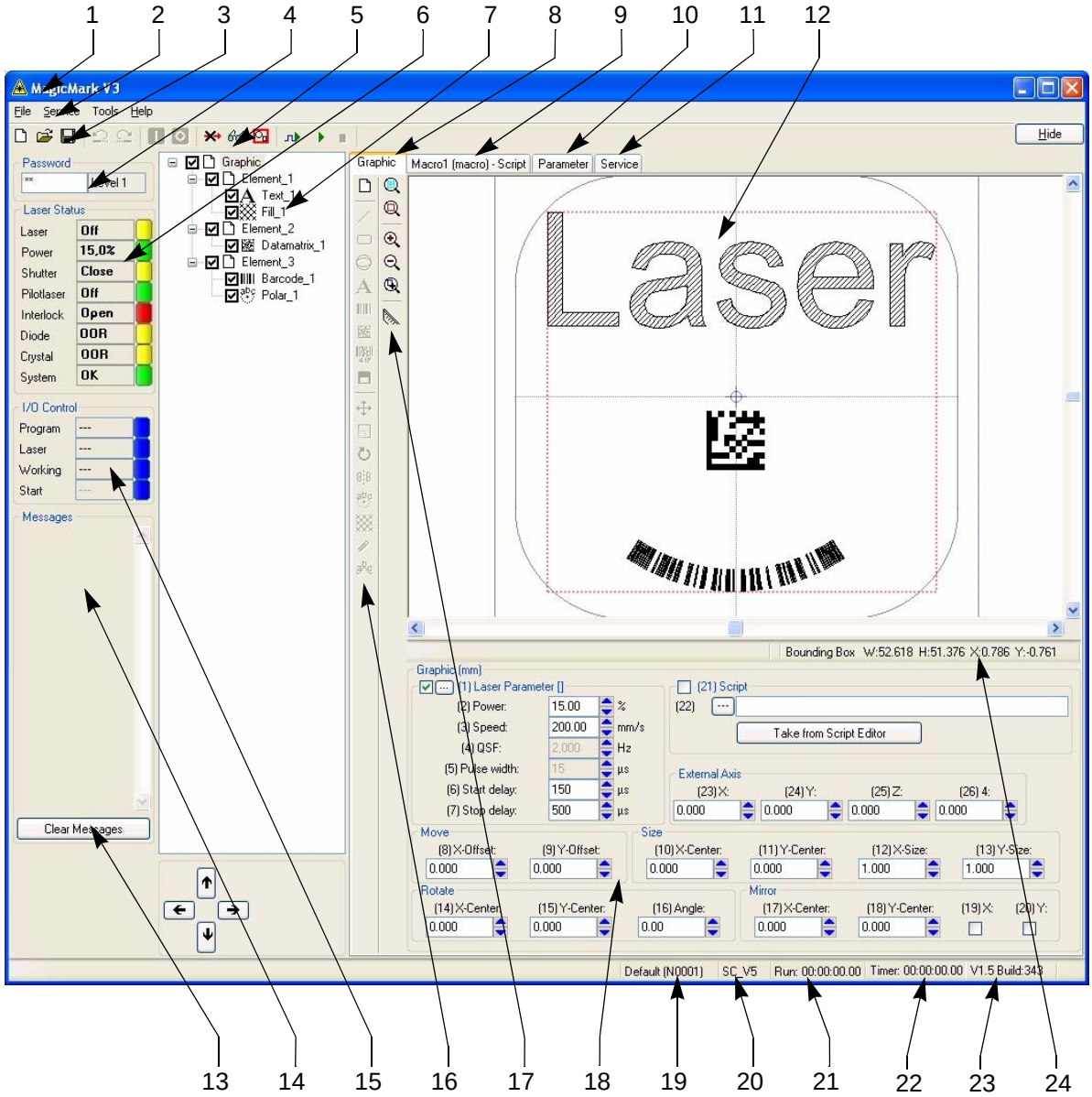
Should this have happened unintentionally, check or correct the interface settings in the **parameter** area, see Parameter area on page 55.

Connect

Fig. 12 Service menu

With the **Connect** command, the software establishes a connection to the laser.

3.2 Software user interface



1	Title bar	13	Button to clear messages
2	Menu bar	14	Message window
3	File toolbar	15	I/O control display
4	Password group	16	Graphic toolbar
5	Laser toolbar	17	Zoom toolbar
6	Laser status group	18	Parameter area graphic
7	Element window	19	Designation of the laser/number of the device
8	Graphic area	20	Current configuration
9	Script area	21	Run time timer
10	Parameter area	22	User timer
11	Service area	23	Version/build number of the software
12	Graphic window	24	Status line graphic window

3.3 Entry elements

Text box

Fig. 13 Text box

The text box is primarily for entering texts, such as the font type, as is shown here.

Multi line text box

Fig. 14 Multi line text box

The multi line text box can accept several lines of text.
It has two scroll bars that are automatically activated when the lines cannot be directly shown.

Numerical input box

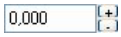


Fig. 15 Numerical input box

Normally numerical values are entered here.

The numerical values can be changed with the **+** and **-** keys.
If you activate the **Shift** key in addition, then the change takes place in larger increments (coarser).

On the other hand, if the **Ctrl** key is activated, then the changes are smaller (more precise).

If the mouse pointer is located within the entry field, then the numerical values can be changed with the arrow keys **up** and **down** as well as with the **Page up** and **Page down** keys.
The **Shift** and **Ctrl** keys also work the same way in this case.

If you are using a wheel mouse, then this can also be used to change the values.
The **Shift** and **Ctrl** keys also work in this case.

List box

Fig. 16 List box

No values or texts are entered in this field, but a list with possible values from which you make a selection is displayed after clicking on the arrow key.

Check box

Fig. 17 Activation field

If the check mark is visible, then the relevant function is activated.
A click of the mouse changes the status (deactivated).

3.4 Access rights

Enter password

Fig. 18 Password group

The access rights are awarded in 4 levels via password entry.

Service:	Access to all functions including the service area.
Level 1:	Access to all functions. Default setting: **L1**.
Level 2:	Only the execution of laser marking. Default setting: **L2**.
No password:	Only cancelling of marking.

1. Write the corresponding password into the left entry window.
2. The correct entry is confirmed by display of the corresponding level in the right window.
3. The allocated functions are enabled.

Change passwords

Fig. 19 Service menu

Click on **Edit Passwords**.
The authorised passwords for the respective level are displayed for editing.

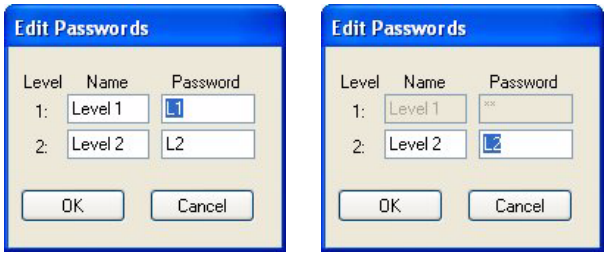


Fig. 20 Dialog window for changing the password for levels 1 and 2, with the passwords following the first installation

Enter the desired new passwords and the names to be displayed and confirm with **OK**.

3.5 File management

Fig. 21 File toolbar

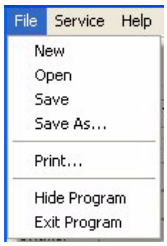


Fig. 22 File menu

New file

Click on the **New** icon or on **New** in the file menu.
A new file is created in the element window (tree structure). See also Graphic parameter on page 39.

Open file

Click on the **Open** icon or on **Open** in the file menu.
The dialog box **Open File** is displayed.

Fig. 23 Open file dialog

Here the file to be opened is selected.

Save File

Click on the **Save** icon or on **Save** in the file menu.
If the file has never been saved, then the **Save File** dialog box opens.

Fig. 24 Save file dialog

Here the file is given the desired name.

If the file has been saved at least once before, then it will be saved under its own name without the **Save file** dialog being shown.

If the file should be saved under a different name, then select **Save As …** in the file menu.

Print file

To print a file, select the menu option **Print…** in the file menu.
The Print dialog is displayed

Fig. 25 Print file dialog

Here you select the printer to be used and start the print output.
What is just being displayed in the graphic window is output on the printer.
At this time the selection is adjusted to the print medium, whereby the output from the printer depicts a somewhat larger selection.

3.6 Laser control

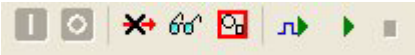


Fig. 26 Laser toolbar

Switching the laser on and off

1. Click on the icon Laser **ON** or **OFF** in order to switch the laser on or off, respectively.
   Activating the laser **ON** button is only possible with a level 2 password or higher!

NOTE!
For security purposes, the password many only be known to authorised persons. By no means should it be written down on the monitor, PC, keyboard, laser, etc.

The laser is activated.
It is possible that a warm-up period is required, depending on the ambient temperature, before the full operating readiness of the system is reached. When the operating readiness is attained, the laser status display switches from **Off** to **On** and the colour changes from red to green. The laser is operative. **OK** is displayed in green in System.

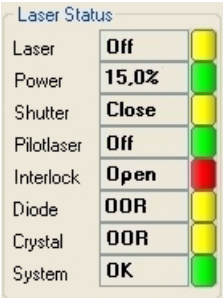


Fig. 27 Laser status display

2. To switch the laser off click on the laser **OFF** icon.
   The laser is deactivated.

NOTE!
In this case it means an electronic switch off. The system is not disconnected from the line voltage.

Further points in the Laser Status Display:

Power:	Shows laser power output in %.
Shutter:	Shows shutter is open or closed.
Pilot laser:	Shows pilot laser is on or off.
Interlock:	Shows if the interlock is open or closed.
Diode:	Shows the temperature state of the diode. OOR Temperature out of range. OK Temperature within the limit.
Crystal:	Shows the temperature state of the crystal. OOR Temperature out of range. OK Temperature within the limit.
System:	Shows the system state.

Lock shutter

Click on the **ShutterLock** icon to close and lock the shutter.

Switching the pilot laser on and off

Click on the **Pilot laser** icon to switch the pilot laser on or off, respectively.

Showing the bounding box

Click on the **Bounding box** icon to switch the output of the bounding box by the laser (pilot laser) on or off.

Start marking

Click on the icon **Run** to start the marking process.

NOTE!
If a script is accessible, then this will start.
Otherwise the graphic image will be output directly.

Stop marking

Click on the **Stop** icon to stop the marking process.

Enable external start

If the icon **ExternStart** is activated, the laser/the marking program can be started by means of an external signal.

I/O control

Fig. 28 I/O control

The state of the output and input signals (ExternStart) is displayed in the I/O control window.

Message window

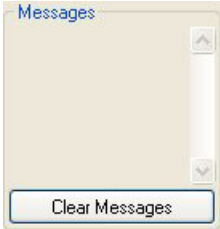


Fig. 29 Messages

All of the software messages are displayed in the message window. These can be messages regarding the condition of the laser or also problems when opening the configuration files, etc.
Clicking on the **Clear Messages** button clears the displayed messages.

3.7 Graphic area

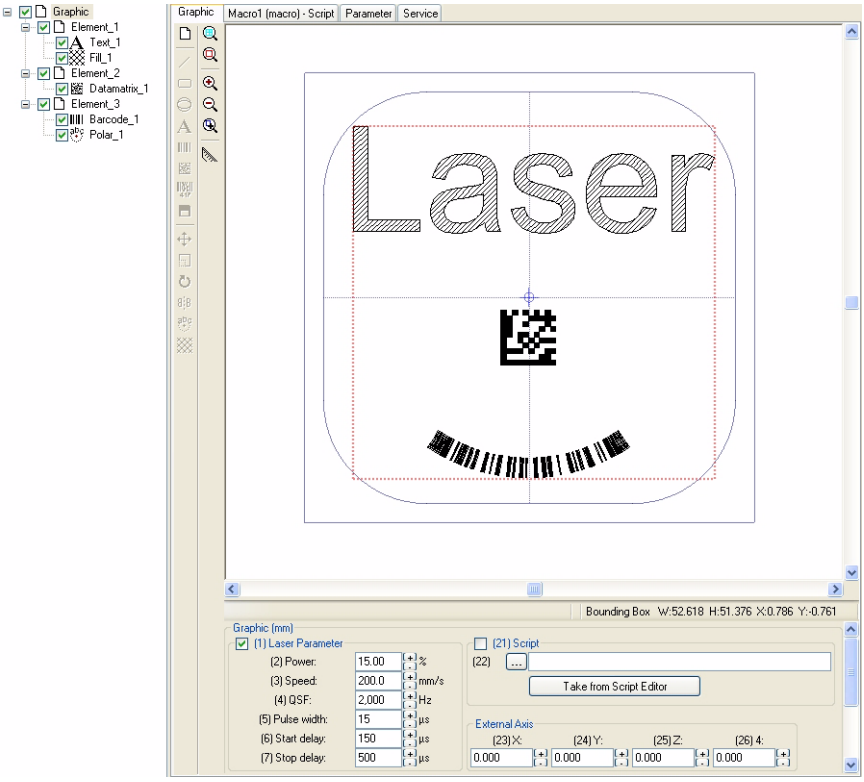


Fig. 30 Graphic area

The graphic area is for creating graphic elements.
The image is displayed in the graphic window and is continuously updated.

Element window

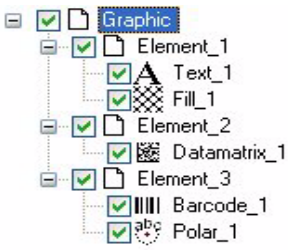


Fig. 31 Element window

The elements, objects and modifiers are clearly arranged as shown in the element window.

Status line graphic window

Fig. 32 Status line graphic window

There is a status line below the graphic window.

Fig. 33 Display result of measurement

Here is where the result of a completed measurement is shown:

L:	length of the measured segment.
X,Y:	length along the X or Y axis, respectively.

Fig. 34 Current pointer position

This area shows the current pointer position when the pointer is over the graphic window.

Fig. 35 Area bounding box

The area bounding box shows the measurements of the bounding box surrounding the graphic:

B:	width of the box.
H:	height of the box.
X,Y:	midpoint coordinates of the box.

Parameter area - graphic

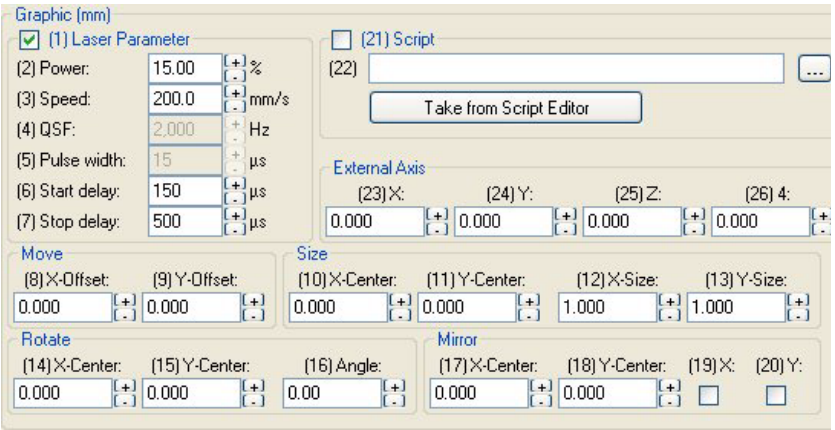


Fig. 36 Parameter area - graphic

Different parameter input masks are displayed, depending on the selected branch in the element window, in the parameter area.

3.7.1 Element window

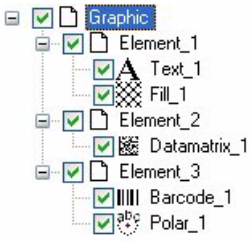


Fig. 37 Element window

The element window is for the clear, structured management of the graphic **Elements**, **Objects** and **Modifiers**.

Object designates:	**Rectangle**, **Barcode**, **Text**, ...
Modifiers are:	**Rotation**, **Filling**, **Mirroring**, ...
Elements	combine **Objects** and **Modifiers** in one unit.

NOTE!
Elements provide their own local coordinate system for the objects and modifiers contained in them. These are displayed by means of a small crosshair in the graphic window.

NOTE!
Element branches can contain either other element branches or objects and modifiers, but not both at the same time!

Coordinate systems

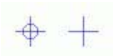


Fig. 38 Display coordinate system

With these symbols the coordinate system 0-point of the **graphic element** or the normal **elements** is shown in the graphic window.
The higher order **graphic element** is constantly displayed, the 0-point of the other **elements** are only shown if an element or an object contained therein is selected.
The higher order **graphic element** with its coordinate system represents the reference point for all lower order **elements**.
An **element** that contains **objects** represents the reference system of just these **objects**.

Elements and objects

Use **elements** to combine **objects** or **objects** and **modifiers** that form a unit, respectively

Example:
In order to generate a filled text, first create an element followed by a text object and fill modifier.
The **element** now represents the complete text.
Now the **element** can be moved to any position in the marking field.

Enable element

By adding or removing this check mark, the corresponding branch of the tree structure is activated or deactivated, respectively.
A deactivated branch is not shown and also not output.
It is only possible to output a deactivated branch using script control.

PopUp menu

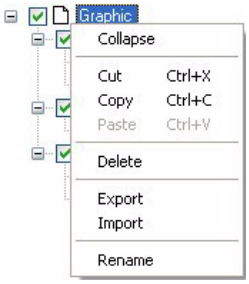


Fig. 39 Element window PopUp menu

A popup menu opens by right-clicking on the mouse in the element window.

The top menu item is **Collapse** or **Extend**, depending on whether or not the branch of the tree structure is extended or collapsed.
By activating this menu item, the depiction of the tree structure can be influenced.

Cut, **Copy** and **Paste** enables you to remove a branch of the tree structure or to paste it in a new place.

Delete deletes the selected branch permanently.

Export saves a branch to a file for later use.

Import imports a branch or an entire program to the selected position.

If you want to give a branch a new name, then use **Rename**.

Drag & Drop

The so-called Drag and Drop function is also supported.
You can drag branches to new positions or copy them in.
Procedure:

1. Keep the left mouse key pressed above the branch that you want to move or copy.
2. Keeping the left mouse key pressed, drag the branch to the desired position.
3. To move it, release the left mouse key.
   To copy activate the **Ctrl** key before releasing the left mouse key. You will then see a small + sign at the mouse pointer.

Target positions that are not possible or not permitted are shown as a prohibiting sign instead of the mouse pointer. If the left mouse key is released on these occasions, the process is terminated.

3.7.2 Graphic toolbar

Fig. 40 Graphic toolbar (shown here in a horizontal position)

New element

Click on this icon to create a new element in the tree structure of the element window. See also Element parameter on page 40.

Line

Click on this icon to insert the object **line** into the tree structure of the element window. See also Line parameter on page 41.

Rectangle

Click on this icon to insert the object **rectangle** into the tree structure of the element window. See also Rectangle parameter on page 41.

Ellipse

Click on this icon to insert the object **ellipse** into the tree structure of the element window. See also Ellipse parameter on page 42.

Text

Click on this icon to insert the object **text** into the tree structure of the element window. See also Text parameter on page 42.

Barcode

Click on this icon to insert the object **barcode** in the tree structure of the element window. See also Barcode parameter on page 43.

Datamatrix

Click on this icon to insert the object **datamatrix** in the tree structure of the element window. See also Datamatrix parameter on page 44.

PDF417

Click on this icon to insert the object **PDF417** in the tree structure of the element window. See also PDF417 parameter on page 45.

NOTE!
You need a special license to use the PDF417 code!

Import

Click on this icon to insert the object **import** in the tree structure of the element window.
Import enables you to import image files that have been created by external programs. See also Import parameter on page 46.

Move

Click on this icon to insert the **Move** modifier in the tree structure of the element window.
All objects that are located in the same element in front of this modifier can be moved by the indicated value. See also Move parameter on page 47.

Size

Click on this icon to insert the **Size** modifier in the tree structure of the element window.
All objects that are located in the same element in front of this modifier can be changed in size. See also Size parameter on page 47.

Rotation

Click on this icon to insert the **Rotation** modifier in the tree structure of the element window.
All objects that are located in the same element in front of this modifier can be rotated around a defined point. See also Rotation parameter on page 48.

Mirror

Click on this icon to insert the **Mirror** modifier in the tree structure of the element window.
All objects that are located in the same element in front of this modifier can be mirrored on the X- and/or Y-axis of a virtual coordinate system. See also Mirror parameter on page 48.

Polar

Click on this icon to insert the **polar marking** in the tree structure of the element window.
All objects that are located in the same element in front of this modifier can be aligned to a virtual circle. See also Polar parameter on page 48.

Fill

Click on this icon to insert the **Fill** modifier in the tree structure of the element window.
All objects that are located in the same element in front of this modifier can be filled. See also Fill parameter on page 49.

NOTE!
Fillings can only be applied to closed polygons (continuous lines).
If fillings are applied to other structures, there may be unexpected results.

Wobble

Click on this icon to insert the **Wobble** modifier in the tree structure of the element window.
For all objects that are located in the same element in front of this modifier the lines can be changed in his width and structure.
See also Wobble parameter on page 49.

Pfad

Click on this icon to insert the **Path** modifier in the tree structure of the element window.
All objects that are located in the same element in front of this modifier the lines can be aligned at a wavy path.
See also Path parameter on page 49.

3.7.3 Zoom toolbar

Fig. 41 Zoom toolbar

Zoom to marking area

A click on this icon results in the graphic window showing the entire marking area.

Zoom to bounding box

A click on this icon results in the graphic window showing the content of the bounding box in a frame-filling way.

Zoom plus

A click on this icon increases the zoom factor.

Zoom minus

A click on this icon decreases the zoom factor.

Zoom window

After activating this icon a rectangle can be drawn with the mouse that depicts the new zoom area.
The icon remains active until you complete the action or you click on it again.
The first click with the left mouse button defines the first corner of the zoom rectangle, the second click defines the concluding second corner.

NOTE!
If you are using a wheel mouse, you can also zoom using the wheel.
The values in the entry fields can also be changed in this manner.
Also use the Shift and Ctrl keys with the entry fields to influence the changes.

Measurement

After activating this symbol you can draw a line in the graphic window with the mouse.
The icon remains active until you complete the action or click on the icon again.
The first click with the left mouse button defines the beginning of the line, the second click defines the end of the line.
The length of the line, as well as the lengths along the X- and Y-axes are shown in the status line of the graphic window.

3.7.4 Graphic window

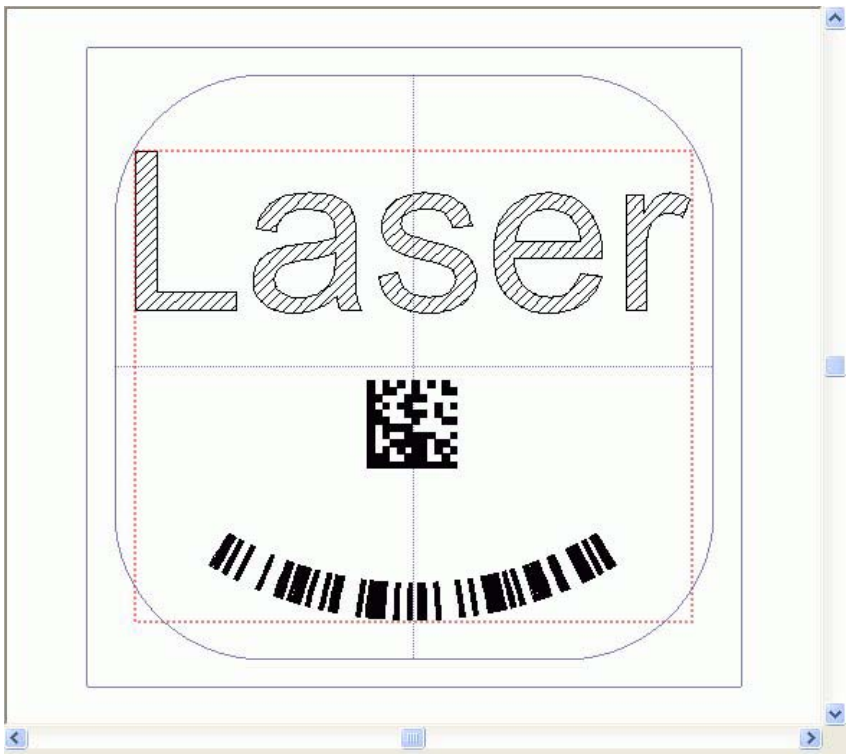


Fig. 42 Graphic window

Apart from the graphic elements in the Graphic window, you can also see the following:

1. A pale blue rectangle with rounded corners, which contains a crosshair.
   This is the recommended marking area.
   If you go outside of this area with your marking, then you will have to expect reductions in quality, such as diminished output, etc.
2. A pale blue rectangle that surrounds the first one.
   This the maximum possible physical marking area, that the deflecting/scanning mirror can reach. Marking beyond this boundary is not possible!
3. A dotted red square.
   This is exactly big enough to just encompass the graphic elements, therefore the designation "bounding box".

Moving the view

You can move the view in the graphic window by left-clicking the mouse anywhere in the window and then moving the mouse, still holding down the button. The view should follow the mouse movement until you release the mouse button.

NOTE!
When the same procedure is applied with the right mouse button, the element or object currently selected in the element window can be moved.

3.7.5 Graphic parameter area

3.7.5.1 Graphic parameter

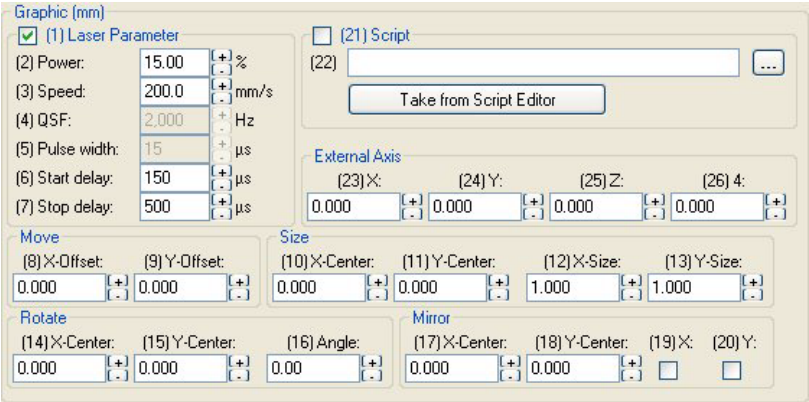


Fig. 43 Graphic parameter

This parameter group is part of the root branch of the tree structure in the element window and is therefore contained in every graphic. All parameters apply universally, i.e. to the entire image.

Laser parameter

If this group is active, it determines the settings for the laser. If it is inactive, then the default and test values from the parameter area are applied.
See Default and test values on page 57.

Move

With this the entire graphic can be moved in the X- or Y-directions.

Size

This group permits a minor size adjustment.

Rotation

The graphic can be rotated around a point.

Mirror

The graphic can be mirrored on the axes of a virtual coordinate system.

Script file

By specifying a script file and setting the activating check mark, the graphic can be linked with a script file (*.bas).

Clicking on the button opens a file dialog for easy selection of the file.
With **Take from Script Editor** the currently active script program can be adopted.
If the graphic file (*las) is loaded and no other script is loaded, then the script indicated here will be loaded into the script editor.

External axis

The setting values here are passed to a script program for processing by means of a Call-Back function.
See LC_ExternAxis(...) on page 109.

3.7.5.2 Element parameter

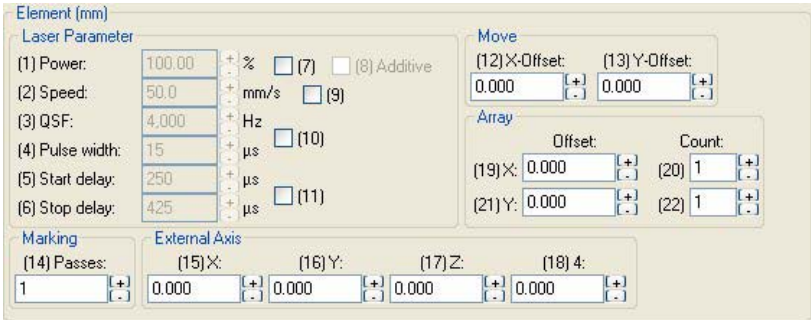


Fig. 44 Element parameters

The parameter area of the element branch is available at the beginning of each group of graphic objects and their modifiers.

Laser parameter

Here, too, you have the possibility of assigning the laser new parameters.
The individual activation boxes permit you to determine exactly, which parameters you want to change.

External axis

Here you can specify the position of up to four external axes as well. These values are sent to a script program for processing.
See LC_ExternAxis(...) on page 109.

Move

The graphic that belongs to this element can be moved in X- and Y-directions.

NOTE!
The move can also be executed by clicking and dragging it in the graphic window with the right mouse button!

Passes

This value determines the number of marking passes.
Normally 1, however if you wish to engrave the material more deeply, then the value can be increased.

Array

With this you can define an array of your marking.
You can define the distance (Offset) and count independently for the X and Y direction.

3.7.5.3 Line parameter

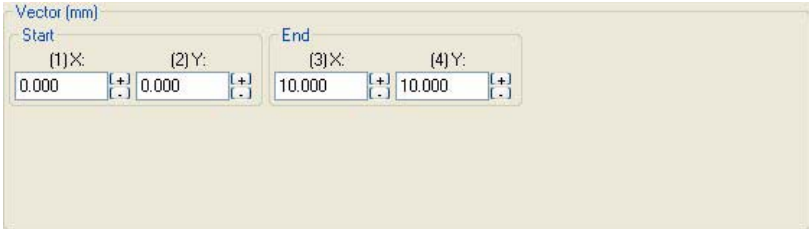


Fig. 45 Line parameter

Because a line has a beginning and an end, these values can be entered separately.

NOTE!
By clicking and dragging in the graphic window with the right mouse button, you can change both values simultaneously and in doing so move the line.

3.7.5.4 Rectangle parameter

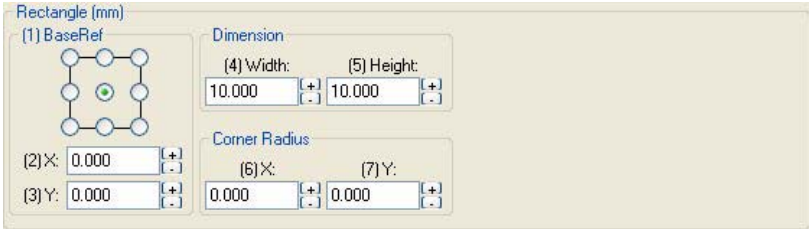


Fig. 46 Rectangle parameter

BaseRef

The **BaseRef** group enables you to establish which point of the object is to be fixed to which coordinates. See BaseRef on page 131.

NOTE!
The BaseRef can be changed by clicking and dragging in the graphic window the right mouse button.

Dimension

Here is where you enter the width and height of your rectangle.

Corner radius

To create a rectangle with rounded corners, enter the desired radii here.

NOTE!
Both text fields must contain a radius > 0 to generate an acceptable rectangle.
Both values are usually the same.

3.7.5.5 Ellipse parameter

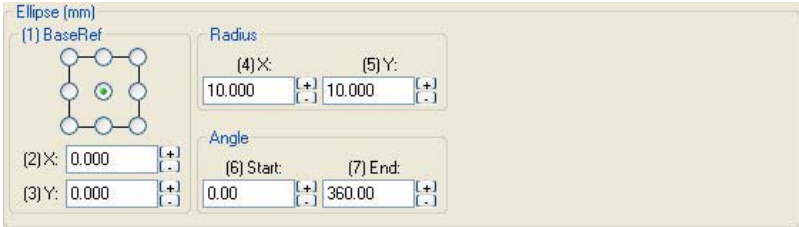


Fig. 47 Ellipse parameter

BaseRef

The **BaseRef** group enables you to establish which point of the object is to be fixed to which coordinates. See BaseRef on page 131.

NOTE!
The BaseRef can be changed by clicking and dragging the right mouse button in the graphic window.

Radius X, Y

Stating the two radii results in the ellipse.
A circle results from equal radii.

Angle

If you want an ellipse/circle segment, then you determine the segment with the beginning or end angle.

3.7.5.6 Text parameter

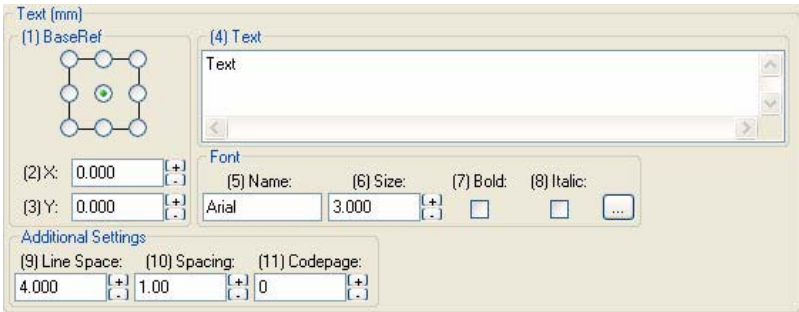


Fig. 48 Text parameter

BaseRef

The **BaseRef** group enables you to establish which point of the object is to be fixed to which coordinates. See BaseRef on page 131.

NOTE!
The BaseRef can be changed by clicking and dragging the right mouse button in the graphic window.

Text

The text to be output is entered in this entry field.
A multi line text is also possible.
For special data, such as date or time, see Format specifications on page 132.

Font

In the **Font** group the font to be used is determined.

Press the key to open the font selection dialog.

Additional settings

Line space: Determines the line spacing in multi line texts.

Spacing: Is a factor with which the character spacing can be changed.

Codepage: Permits the specification of a codepage for special cases, e.g. Asian characters.

3.7.5.7 Barcode parameter

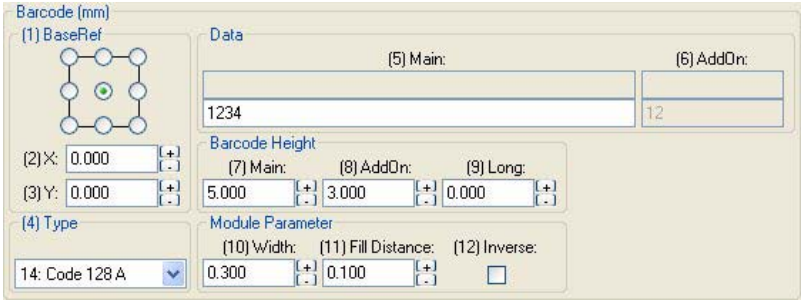


Fig. 49 Barcode parameter

BaseRef

The **BaseRef** group enables you to establish which point of the object is to be fixed to which coordinates. See BaseRef on page 131.

NOTE!
The BaseRef can be changed by clicking and dragging the right mouse button in the graphic window.

Type

Selection of the barcode type to be applied.

Data

The barcode content is entered in these entry fields.
The text box **AddOn** is only used for barcodes with additional data.
For special data, such as date or time, see Format specifications on page 132.
For information on the abilities of the individual barcodes, see Barcode specification on page 134.

Barcode height

For the definition of the barcode heights see Barcode specification on page 134.

Module parameter

By **Module Width**, the width of the narrowest barcode bar is meant.

With **Fill Distance** you determine the distance at which the laser draws the lines necessary for filling.

CAUTION!
The fill spacing might also be slightly changed by the program in order to comply with the required module width.
If this is not desired on critical materials, it can be prevented by setting the module width to a whole number multiple of the fill distance.

Inverse is selected if the laser generates light instead of dark lines. In this case the spaces in the barcode are then lasered.

3.7.5.8 Datamatrix parameter

Fig. 50 Datamatrix parameter

BaseRef

The **BaseRef** group enables you to establish which point of the object is to be fixed to which coordinates. See BaseRef on page 131.

NOTE!
The BaseRef can be changed by clicking and dragging the right mouse button in the graphic window.

Data

The barcode content is entered in this text box.
For special data, like date or time, see Format specifications on page 132.

Module parameter

Choose the required version of the datamatrix barcode under **Type**.

Size fixes the size of an individual point of the datamatrix code. The total size results from the number of points that comprises the datamatrix code, multiplied by the stated **Size**.

With **Fill Distance** you determine the distance at which the laser draws the lines necessary for filling.

CAUTION!
The fill spacing might also be slightly changed by the program in order to comply with the required module width.
If this is not desired on critical materials, it can be prevented by setting the module width to a whole number multiple of the fill distance.

3.7.5.9 PDF417 parameter

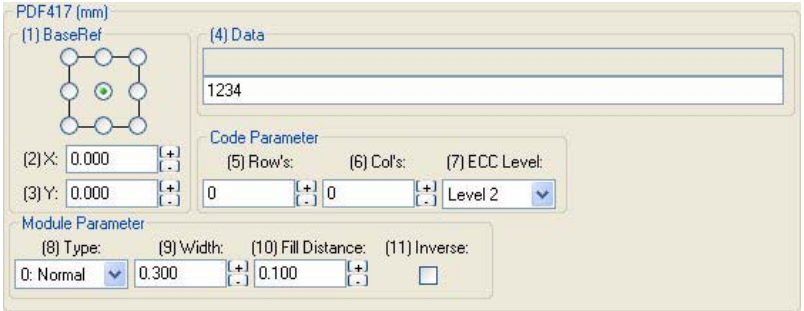


Fig. 51 PDF417 parameter

CAUTION!
A special license is required to use the PDF417 code!

BaseRef

The **BaseRef** group enables you to establish which point of the object is to be fixed to which coordinates. See BaseRef on page 131.

NOTE!
The BaseRef can be changed by clicking and dragging the right mouse button in the graphic window.

Data

The barcode content is entered in this text field.
For special data, like date or time see Format specifications on page 132.

Code parameter

By specifying **Rows** and/or **Columns**, the number of rows or columns, respectively, can be predetermined.

ECC determines the degree of error tolerance.

Module parameter

A few different versions of PDF417 barcodes are available under **Type**.

By **Module Width**, the width of the narrowest barcode bar.

With **Fill Distance** you determine the distance at which the laser draws the lines necessary for filling.

CAUTION!
The fill spacing might also be slightly changed by the program in order to comply with the required module width.
If this is not desired on critical materials, it can be prevented by setting the module width to a whole number multiple of the fill distance.

Inverse is selected if the laser generates light instead of dark lines. In this case the spaces in the barcode are then lasered.

3.7.5.10 Import parameter

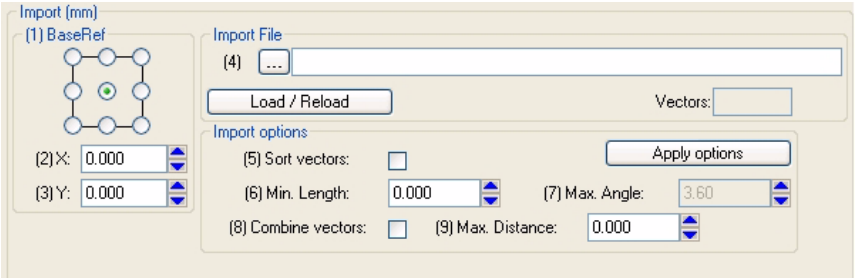


Fig. 52 Import parameter

BaseRef

The **BaseRef** group enables you to establish which point of the object is to be fixed to which coordinates. See BaseRef on page 131.

NOTE!
The BaseRef can be changed by clicking and dragging the right mouse button in the graphic window.

Import file

In this text field the files to be imported are entered.

Activating the ... key opens an import dialog.

Fig. 53 Import dialog

Select the file to be imported and click on **Open**.

File types

DWG: AutoCAD Format.
DXF: AutoCAD Format.
HPGL: HPGL Format.
BMP: Bitmap Format.
JPG: JPG-Bitmap Format.
GIF: CompuServe Bitmap.

Load/Reload

Clicking this button reloads the graphic image once more, in case it has been altered in the meantime.

Vectors

For inforrmation about the file to import the number of existing vectors is shown here.

Import options

Here you can defin, if vector files should be optimised for the laser output and how to do this.

Sort. vectors: Reducing the number of vectors.
You can set the parameters **min. length** of a vector and **max. angle** of vectors border on each other.

Combine vectors: Combining of vectors to realise compact areas.
You can set the parameter **max. distance** between two vector points.

Apply options

Clicking this button the choosed import parameters are applied.

3.7.5.11 Move parameter

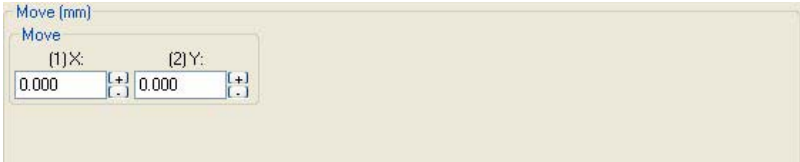


Fig. 54 Move parameter

Specifying the X- and Y-values moves the graphic by the indicated distance.

3.7.5.12 Size parameter

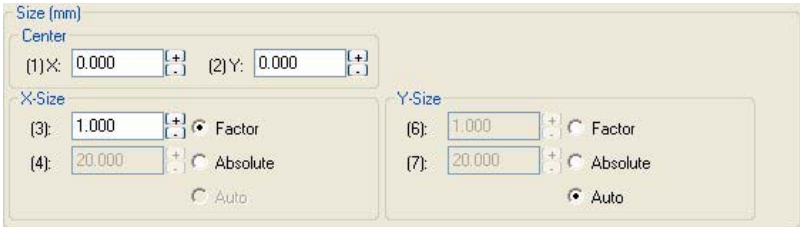


Fig. 55 Size parameter

It is possible to change the size of a graphic (X-size, Y-size) around a fixed point (X-centre Y-centre) with the **Size** modifier.
It is possible to give an relative factor or an absolute size value.

3.7.5.13 Rotation parameter

Fig. 56 Rotation parameter

You can rotate a graphic around a fixed point X-centre Y-centre by a certain angle.

3.7.5.14 Mirror parameter

Fig. 57 Mirror parameter

By specifying a centre point (X-centre, Y-centre) of a virtual coordinate system, you can mirror the graphic at the axes of this coordinate system.

3.7.5.15 Polar parameter

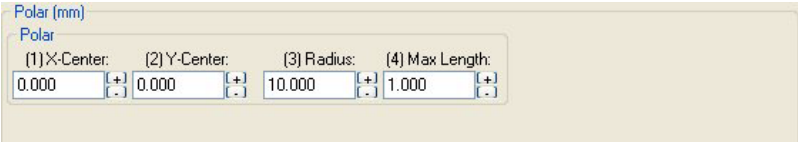


Fig. 58 Polar parameter

The graphic is aligned to the circumference of a virtual circle, which you have specified with (X-centre, Y-centre and radius).

Max Length defines the maximum length of a straight-running line section.
1 mm is normally a good value.
In special cases it may be necessary to change this value.

3.7.5.16 Fill parameter

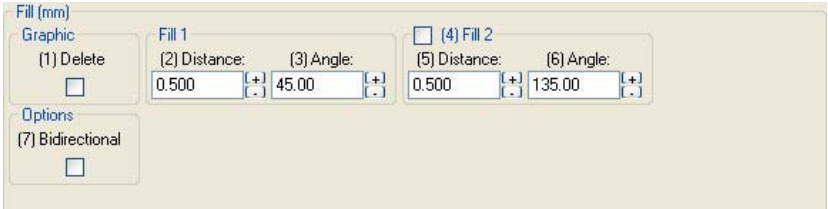


Fig. 59 Polar parameter

A graphic can be filled using the **Fill** modifier.

To execute this, the software looks for closed polygon figures (continuous lines) and then draws parallel lines in them at the specified **distance** and **angles**.

There are two sets of line **distances** and **angles** at your disposal.

If you activate the box **Delete Graphic**, then only the fill lines remain, the original graphic is deleted.
By activating **Bi-directional,** then every second line is marked in reversed direction.

NOTE!
Fillings can only be applied on closed polygons (continuous lines).
If fillings are applied to other structures, the results can be unexpected.

3.7.5.17 Wobble parameter

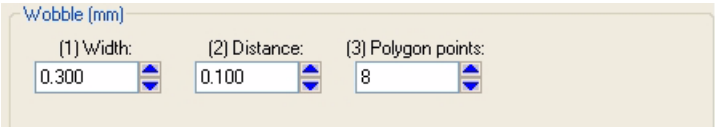


Fig. 60 Wobble parameter

With the modifier **Wobble** a line can shown with lined up polygones. You can set the **width** and the **distance** of the polygones and the number of **polygon points**. So the width and the structure of the line are defined.

3.7.5.18 Path parameter

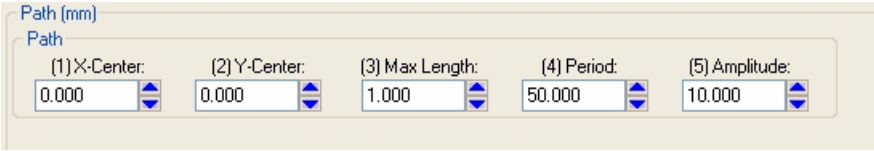


Fig. 61 Path parameter

A graphic or text element can lined up on a wavy path using the **Path** modifier.
For the wave style you can set the values **X-Center**, **Y-Center**, **Max. Length**, **Period** and **Amplitude**.

3.8 Script area

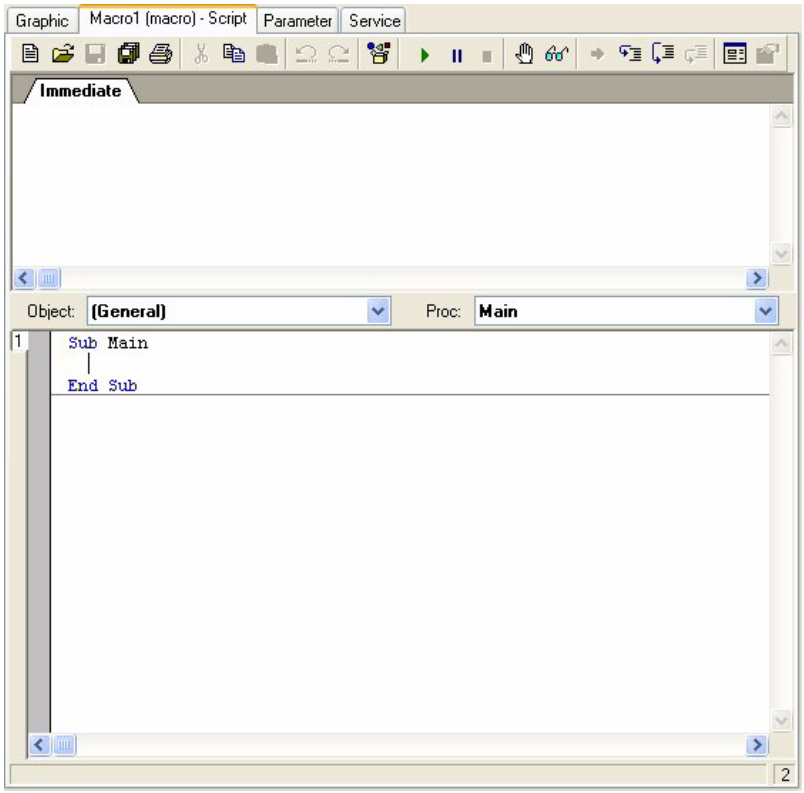


Fig. 62 Script area

In the script area you can do all the things that go beyond the simple output of graphic objects.

Beginning with the creation of a simple process over the recurring output of a drawing as a reaction to a control entry, up to the control of a complete system with conveyor belt, various interfaces and possibly several lasers.

Information that applies to the script program window can be found under Programming on page 62.

Details regarding the programming language used is in Programming on page 62

And the special language extensions of this software are under Laser specific script extensions on page 107.

3.8.1 Script samples

3.8.1.1 Serial numbers

You can find this example in the installation directory inside the folder **Samples\SerialNumber**.

Graphic part

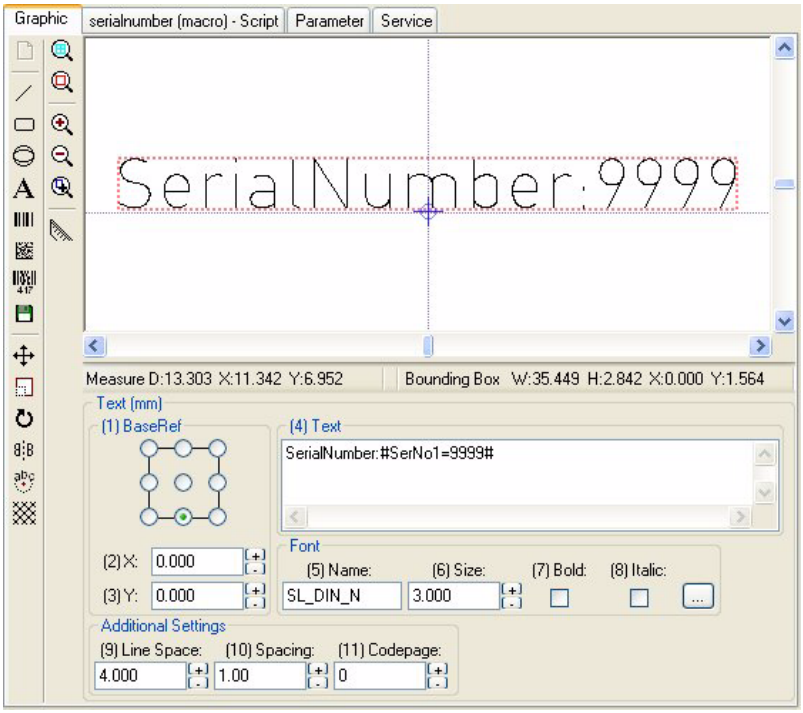


Fig. 63 Graphic example serial numbers

The text in the multi line text field (SerialNumber:#SerNo1=9999#) is comprised of two parts:

1. A fixed text (SerialNumber:) and
2. a format specification (#SerNo1=9999#).

For further information about format commands see also Format specifications on page 132 and Special format specifications on page 133.

The part in front of the **=** (SerNo1) is for identification purposes and the part behind the **=** (9999) is the default value.

As the script isn't running, or doesn't even yet exist at the time the graphic view is being calculated, the default value is displayed.

The script program now has the task of pulling a serial number from a file and to make it available on demand at runtime.
Finally the serial number is increased by one and written back into the file.

Script part

```
'Version 1.0
Option Explicit

Dim SerNo1 As Long
```

```
Sub Main
   SerNo1 = CLng(EX.ReadXmlFormat("SerNo1", "1", "C:\SerNumber.xml"))
   LC.Mark("", True)
   SerNo1 = SerNo1 + 1
   EX.WriteXmlFormat("SerNo1", CStr(SerNo1), "C:\SerNumber.xml")
End Sub
```

```
Public Function LC_Formatter(sFormat As String, sString As String,
sDefault As String) As String
   If(sString = "SerNo1") Then LC_Formatter = Format(SerNo1, "000000")
End Function
```

Sample 1 Serial numbers

Main program

The first line is a comment and gives information about the version of this script program.
The second line, if at hand, forces the programmer to also declare all variables used in the program (Dim...), this procedure can only be recommended!

In the next line a variable (SerNo1) is declared, because it will take up the actual serial number later.

The actual program consists of two parts:

1. The main program **Main** and
2. the CallBack function LC_Formatter... See also LC_Formatter(...) on page 109.

Main is called up (started) when the program is started and carries out the following actions:

1. Reading out the serial number from an XML file.
2. Starts the actual marking (LCMark("", True)).
3. After the marking is completed the serial number is increased by one and finally,
4. written back into the XML file.

CallBack function

The CallBack function (LC_Formatter...) is called up automatically during the marking and works as follows:

In the header (`LC_Formatter(sFormat As String, sString As String, sDefault As String) As String`) three values are passed:

1. In **sFormat** is the entire format string (#SerNo1=9999#).
2. In **sString** is the identification (SerNo1).
3. and in **sDefault** is the default value (9999).

The CallBack function now checks whether or not the identification is the one that is determined for that serial number (`If(sString = "SerNo1") Then`).
If yes, then the CallBack function passes the formatted serial number as a string (`LC_Formatter = Format(SerNo1, "0000")`).

In addition, further test outputs (Debug.Print...) were added to the example program that is on your computer, to enable you to get an overview more easily.

3.8.1.2 Excel content

You will find this example in the **Samples\Excel** folder in the installation directory.

NOTE!
Microsoft Excel must be installed on your computer in order to execute this example program!

Graphic part

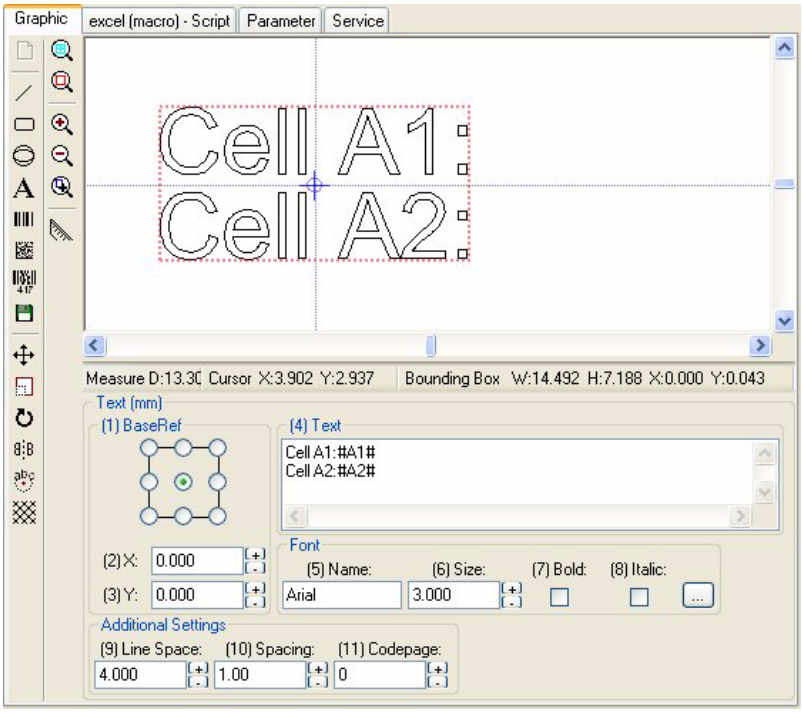


Fig. 64 Graphic Excel example

The text in the multi line text box (Cell A1:#A1#) or (Cell A2:#A2#) does not have a default value in this example, that is why none is shown in the graphic window.
However, this has no effect on the function.
In this case the text of each line contains the following components:

1. A fixed text (Cell A1:) and
2. a format specification (#A1#)

For further information on format commands see also Format specifications on page 132 and Special format specifications on page 133.

In this case the script program has the task of reading out the cell that was specified by the format specification, from an Excel table; in this case the A1 and A2 cells.

Script part

```
'Version 1.0
Option Explicit

Dim objExcel As Object
```

```
Sub Main
   Set objExcel = CreateObject("Excel.Application")
   objExcel.Workbooks.Open("C:\Test.xls")
   LC.Mark("", True)
   objExcel.Quit

End Sub
```

```
Public Function LC_Formatter(sFormat As String, sString As String,
sDefault As String) As String
   LC_Formatter = CStr(objExcel.Range(sString & ":" & sString).Value)
End Function
```

Sample 2 Excel dates

Main program

The first line is a comment and gives information about the version of this script program.
The second line, if at hand, forces the programmer to also declare all variables used in the program (Dim ...), this procedure can only be recommended!

An object (objExcel) is declared in the next line, this will be used later to access Excel.

The actual program consists of two parts:

1. The main program **Main** and
2. The CallBack function LC_Formatter. See also LC_Formatter(...) on page 109.

Main is called up (started) when the program is started and carries out the following actions:

1. Initialising the Excel object.
2. Opening the Excel file.
3. Starting the actual marking (LC.Mark("",True)).
4. Closing Excel.

CallBack function

The CallBack function (LC_Formatter...) is called up automatically during the marking and works as follows:

In the header (`LC_Formatter(sFormat As String, sString As String, sDefault As String) As String`) three values are passed:

1. In **sFormat** is the entire format string (#A1#) or (#A2#), respectively.
2. In **sString** is the identification (A1) or (A2), respectively.
3. And in **sDefault** is the default value () or (), respectively.
   In this case empty character strings.

In the CallBack function the required Excel cell is read out (`objExcel.Range(sString & ":" & sString).Value`) and then passed (`LC_Formatter = CStr(objExcel.Range(sString & ":" & sString).Value)`) to the CallBack function as a character string (`CStr(`...).

In addition, further test outputs (Debug.Print...) were added to the example program that is on your computer, to enable you to get an overview more easily.

Furthermore, the program on your computer also writes the date and time in the Excel cells A1 and A2.

3.9 Parameter area

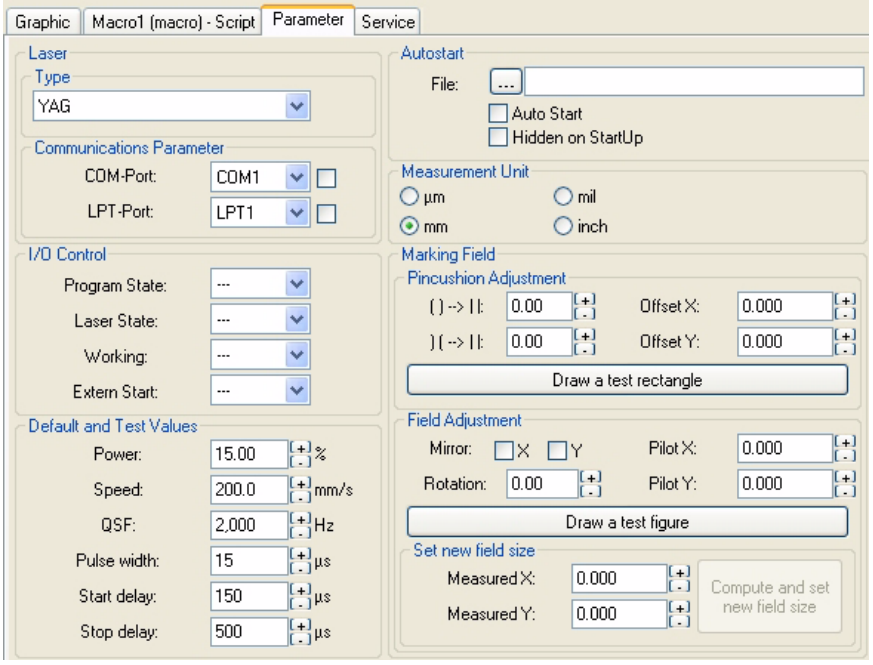


Fig. 65 Parameter area

The **Parameter Area** can be fully accessed with a Level 1 password and permits making settings for various laser parameters.

Laser

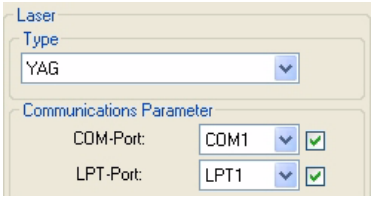


Fig. 66 Laser parameter

The type of the laser used and the connection ports, with which the software establishes the connection to the laser, are determined in the laser parameter:

COM-Port: If activated, then the COM port to be used can be set here.

LPT-Port: Selection of an optional LPT port to accelerate the marking.

Annotation: The respective connections are only active if the corresponding box is marked with a check mark.

Auto start

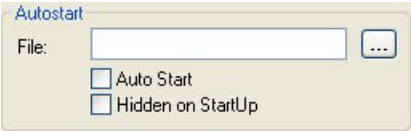


Fig. 67 Auto start

These settings influence the starting behavior of the software.

File: The file specified here is loaded after the software has been started.
This file can be either a script (*.bas) or a graphic file (*.las).
To browse for the file, use the ... button.

Auto start: If **Auto Start** is selected, then the file specified under File will start once it has been loaded.

Hidden on Startup: If the program is to start without a visible window, activate **Hidden on StartUp**.

Measurement unit

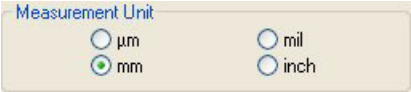


Fig. 68 Measurement unit

Here you can select the measurement unit with which you want to work.
In detail, the following can be set:

µm: µm (1/1000 mm) is used as the measurement unit.

mm: mm is used as the measurement unit.

mil: mil (1/1000 inch) is used as the measurement unit.

inch: inch (25.4 mm) is used as the measurement unit.

I/O control

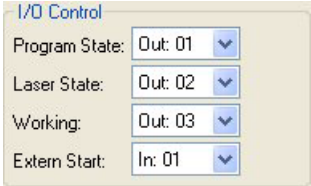


Fig. 69 I/O control

If you want the software to convey information on various states to the outside world, then you can configure the corresponding ports here accordingly.

Program state: State of the program. If the software is running, the selected outlet is set to **1**.

Laser state: If the following conditions are fulfilled:
The laser must be switched on!
The chamber and interlock must be closed!
There may be no current errors!
Then the outlet is set to **1**.

Working: This outlet is set to **1** for the duration of the marking process.

Extern start: The marking process is started when there is a logical change from **0** to **1** at this inlet.
The prerequisite is that **ExternStart** has been activated.

NOTE!
In DEMO MODE the program outlet is fixed at 0!

Default and test values

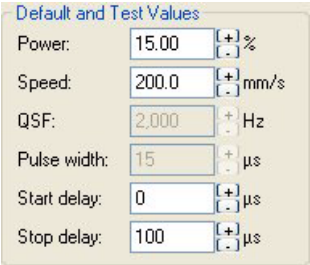


Fig. 70 Default and test values

On the one hand, the values specified here are default values for new graphic objects parameters and, on the other hand, serve as laser parameters for the text output of the **Marking field** group.
In detail the following settings can be set:

Power: Laser output in %.

Speed: The speed at which the laser beam is guided over the surface.

QSF: The frequency in Hz, which determines the pulse repetition frequency of the laser.
This parameter is only available with YAG lasers.

Pulse width: Specifies the laser pulse width.
This parameter is only available with YAG lasers.

Start delay: The delay from the beginning of the mirror motion until the activation of the laser.

Stop delay: Time extension, at which the laser remains switched on, beginning with the braking of the mirror motion.

Marking field

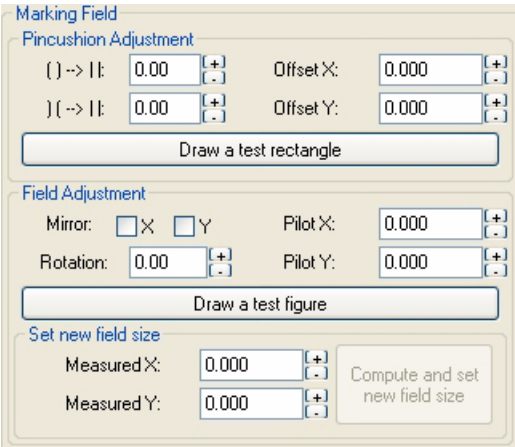


Fig. 71 Marking field

The laser type and the parameters for the marking field are determined in the **Marking field** group:

Pincushion Adjustment: With **()->||** and **)(->||** the pillow- or barrel-shaped distortions can be compensated. **Offset X** and **Y** serve to align unequally shaped distortions.
To check the settings you can laser a large rectangle with **Draw a Test Rectangle**.

Field Adjustment: With the help of **Draw a Test Figure**, which lasers a test image, the **Mirror X-** and -**Y** as well as **Rotation** can be set in such a way, that the test figure is output according to your wishes.
If your laser is equipped with a pilot laser, then this can be set with **Pilot-X** and -**Y** in such a way that it precisely hits the small circle in the centre of the test figure.
The **Field size** is set in the following manner:
1. Output test figure.
2. Measure X and Y and
3. enter at **Measured X** and **Y**.
4. Activate **Compute and set new field size**.

To check you can now laser the test figure again, the X and Y lengths should now be exactly 50 mm or 2".

3.10 Service area

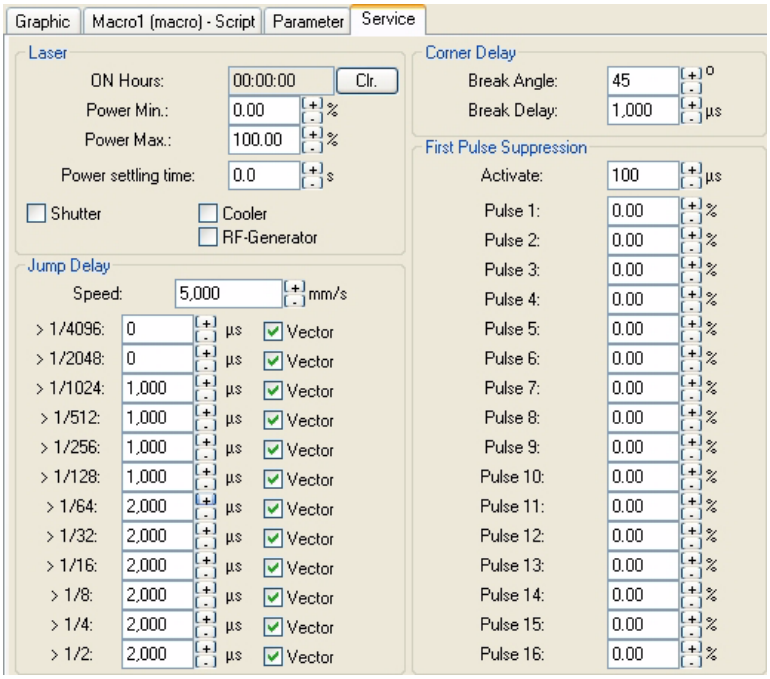


Fig. 72 Service area

The **Service** area is only accessible with **password level Service** and permits the setting of various laser-specific parameters.

Laser

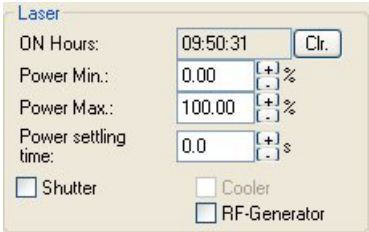


Fig. 73 Laser

The **Laser** area permits the setting of various laser-specific parameters:

ON Hours: The laser's runtime hours. By pressing **Clr.** the display can be set to 0.

Power min.: Lowest power value, which should be set at the desired power output of 0 %.

Power max.: Maximum power output value, which should be applied at the desired power output of 100 %.

Power setting time: The time that should elapse when the power setting of the laser changes.
The time entered here is the time for a power jump from 0 % to 100 %.
Corresponding less time elapses with smaller power jumps.

Shutter: The shutter can be activated for testing purposes.

Cooler: When the laser is switched off, the cooling system can be switched on for testing purposes.

RF-Generator: The RF generator can be switched on or off, respectively, for testing purposes.

Jump delays

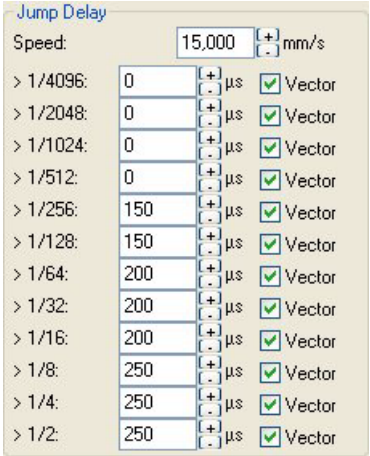


Fig. 74 Jump delays

In the group **Jump delays** all jump relevant parameters of the scanning control are defined.

Speed: If a new position is to be run to by means of a vector, then the speed set here is applied.

> 1/xxxx: Delays in µs, which are inserted after a jump or a vector used for positioning.
> 1/2: is shown for a positioning that is larger than half of the marking field. The further values stand for correspondingly smaller jumps.

Vector: If **Vector** is selected, then a positioning (the laser is deactivated at this time) by means of a vector (line) takes place with the speed stated in **Speed**.
If the relevant delay is set to 0, then there will also be a positioning by means of a vector, but the "normal" marking speed then applies.
Otherwise, the laser is positioned at the new position with a jump.

Corner delay

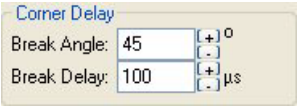


Fig. 75 Corner delay

The **Corner delay** group is for setting the delays that can be set if the angle between to vectors is too acute.

Break angle: The delay becomes applicable for angles that are smaller (more acute) than are entered here.

Break delay: The delay to be inserted.

First pulse suppression

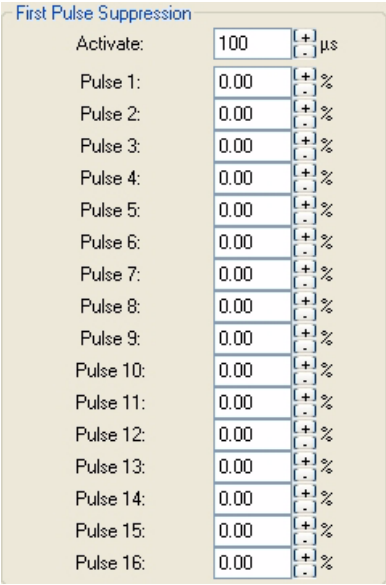


Fig. 76 First pulse suppression

The **First pulse suppression** group permits influencing the first 16 pulses of a YAG laser.

Activate: Time in µs that the laser must be off, in order for the influence to be activated.

Pulse x: Separate setting possibilities for the pulse suppression of the first 16 pulses of a vector in %. The last value should normally be 0 %, as all following pulses are oriented toward the last one in this list.

4 Programming

4.1 Basics

General information

The script programming language is compatible with Visual Basic Version 6.

Using this language, complex program sequences can be created with little effort.

The following description will explain the most important commands used in the context of laser marking.

Program documentation

Each program should be documented adequately. This makes troubleshooting, later changes or adaptations by other persons easier.

Specifically, tasks and characteristics of variables should be described adequately.

Comments are introduced with the " ' " symbol.

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  LC.TimerStart() 'Start the user timer
  Wait 3 'Wait some Time
  LC.TimerStop() 'Stop the user timer
End Sub
```

Sample 3 Comments

4.2 The script programming window

4.2.1 File handling

Fig. 77 Script toolbar

New

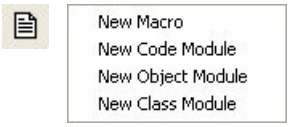


Fig. 78 New macro menu

This manual focuses mainly on creating a new executable program (new macro).
The code, object and class modules are object-oriented program

modules which do not represent programs that are executable on their own.

Open

Click on the **Open** icon.
Select the intended file from the program folder. An existing project will be opened.

Save

Click on the **Save** icon.
If the project has not yet been assigned a name, a name must be entered before saving. If the project already has a name, it can be saved without any acknowledgement.

Save All

Click on the **Save All** icon for saving all open programs.
If a project has not been assigned a name yet, a name must be entered before saving. If a project already has a name, it can be saved without any acknowledgement.

Print

Click on the **Print** icon.
The current project will be output to the standard printer connected under Windows.

4.2.2 Edit

Cut

Click on the **Cut** icon.
High-lighted parts of the program are deleted and transferred to the clipboard.

Copy

Click on the **Copy** icon.
High-lighted parts of the program are transferred to the clipboard.

Paste

Click on the **Paste** icon.
The content of the clipboard is inserted at the cursor position.

Undo

Click on the **Undo** icon.
The last action is undone. The command can be repeated until the status of the last Save has been reached.

Redo

Click on the **Redo** icon.
The action cancelled last is performed again. If you have cancelled too many changes, they can be restored by using the **Redo** command.

4.2.3 Object Catalogue

Display object

Click on the **Show Object** icon.
The object catalogue is opened with its object libraries, class libraries, classes, methods, characteristics, events and constants which can be used in the code. In addition, the modules and procedures are shown which have been defined for the project.

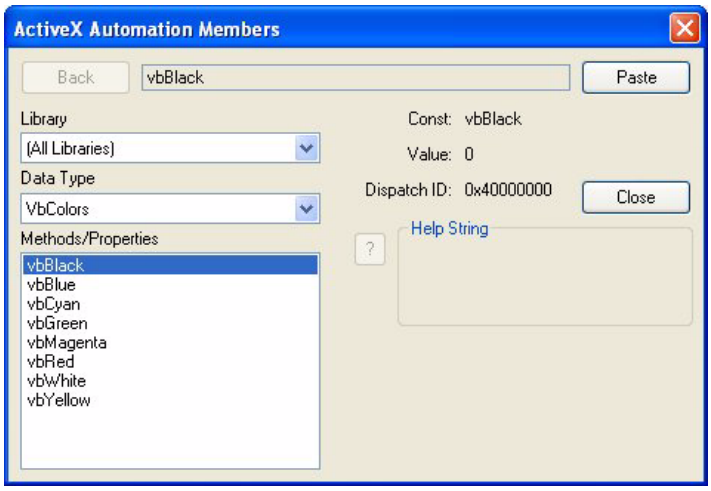


Fig. 79 Object catalogue

4.2.4 Start/Stop

Start/Continue

Click on the **Start/Continue** icon.
The current project is started under the development environment or continued to be executed after a stop.

Pause

Click on **Pause** icon.
If a program was started under the development environment, the activation of this command interrupts the program in the current location.

End

Click on the **End** icon.
The current program is closed and a change to the design mode is made.

4.2.5 Program Testing

In order to be able to test every single instruction during the execution of the program, there is an auxiliary test mode (debugger). Among others, it checks the content of variables and the correctness of the program sequence, and serves for locating programming errors.

Breakpoint On/Off

Click on the **Breakpoint On/Off** icon.
A breakpoint is inserted or deleted at the cursor position.

Evaluate Expression

Click on the **Evaluate Expression** icon.
In the break mode, the current value is shown.

Edit Point

Click on the **Show next statement** icon.
If the program is processed in the testing mode, this icon signals the current edit point.

Jumping in (individual step)

Click on the **Jump In** icon or on **F8**.
The program can be processed step by step. If the program contains procedures, it branches into them. These procedures or functions are processed step by step.

Skip (procedure step)

Click on the **Skip** icon.
The program can be processed step by step. Any procedures and functions contained will be executed.

Jump out (Finish procedure)

In break mode, click on the **Jump Out** icon.
The current procedure is executed and stopped again at the next program line to be executed.

4.2.6 User Dialog

Edit user dialog

Click on the **Edit user dialog** icon.
The **User Dialog Editor** box opens.

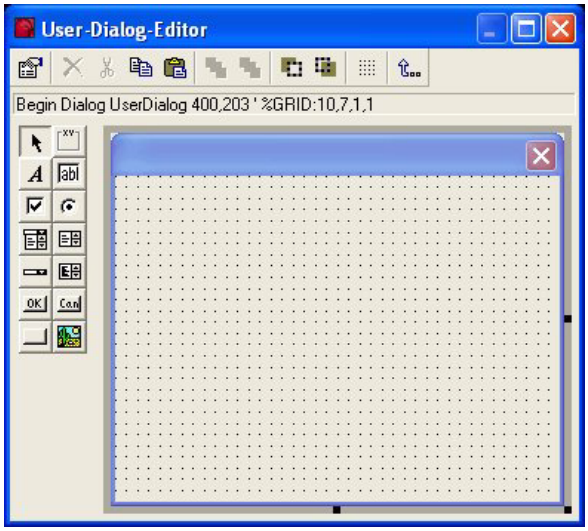


Fig. 80 User Dialog Editor

This mode is an important tool for creating user-specific programs. By using it, proprietary user interfaces in the Windows format can be created.
A detailed description is found at Edit user dialog on page 89.

4.3 Programming language

4.3.1 Variables

General information

Variables are values marked with a name whose content is changeable.

They are required for saving and/or further processing of results of calculations or entries from the program user.

Each variable should be declared; i.e., assigned a variable type (string, integer etc.)
In order to force the declaration of all variables and thus exclude sources of error, the **Option Explicit** command is used.

Naming Variables

The name of a variable should be explicit so that the function of the variable can be deduced from its name alone.

Byte

Integer according to ASCII character set.

Value range:

0 to 255

Size:

1 byte

Boolean

Status True/False

Value range:

True/False

Size:

2 bytes

Integer

Integer without decimals.

Value range:

-32768 to +32767

Size:

2 bytes

Long

Integer without decimals with greater value range as integer.

Value range:

-2 147 438 648 to +2 147 483 647

Size:

4 bytes

Single

Floating point for calculations with decimals.

Value range:

$-3.402823*10^{38}$ to $-1.401298*10^{-45}$
$1.401298*10^{-45}$ to $3.402823*10^{38}$

Size:

4 bytes

Double

Floating point for calculations with decimals.

Value range:

$-1.79769313486232*10^{308}$ to $-4.94065645841247*10^{-324}$
$4.94065645841247*10^{-324}$ to $1.79769313486232*10^{308}$

Size:

8 bytes

Currency

Mix of integers and floating point for calculations up to the fourth decimal after the point.

Value range:

-922 337 203 685 477.5808 to 922 337 203 685 477.5807

Size:

8 bytes

Date

Utilisation for date and time.

Date value range:

01.01.100 to 31.12. 9999

Time value range:

00:00:00 to 23:59:59 (8 bytes)

String

String format, any length.

Size:

10 byte + 2 byte/character

Variant

Standard type which can assume any other type. In the process, the variables are converted automatically. This declaration is used when the type may change during the execution of the program.

Numerical value size:

16 bytes

String size:

22 byte + 2 byte/character

Declaration	description
Dim Number As Integer	Declares **number** as an integer
Dim Symbol As String	Declares **symbol** as a string
Dim Value	Declares **value** as a variable of the variant type

Sample 4 Variable declaration

Variables can also be declared through icons:

Symbol	Variable type
%	Integer
&	Long
!	Single
#	Double
@	Currency
$	String

Declaration	Description
Dim number%	Declares **number** as integer
Dim symbol$	Declares **symbol** as a string

Sample 5 Declaration of variables with icons

Examples for the importance of declaring variables:

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim Number!
  Dim Result!
  Number! = 2.5
  Result! = Number! * Number!
  Debug.Print "Result = " & Result!
End Sub
```

Result = 6.25

Sample 6 Program with variable declaration

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim Number%
  Dim Result!
  Number% = 2.5
  Result! = Number% * Number%
  Debug.Print "Result = " & Result!
End Sub
```

Result = 4

Sample 7 Program with variable declaration

The two programs lead to different results due to the different declaration of variables. In Sample 6 floating points are multiplied with each other, while in integers are Sample 7 multiplied with each other. When doing so, values up to x.50 are rounded down.

4.3.2 Constants

Constants are named values whose content cannot be changed. They may assume any type.

Const Pi! = 3.14: The constant Pi is assigned the value 3.14.
Const Pi% = 3.14: The constant Pi is assigned the fixed value 3.

Sample 8 Definition of constants

4.3.3 Fields

Fields are lists of variables.
Fields can be one- or multi-dimensional.
If the value of a field is constant during object time, it is called a static field.

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim Text(5) As String

  Text(0) = "Part A"
  Text(1) = "Part B"
  Text(2) = "Part C"
  Text(3) = "Part D"
  Text(4) = "Part E"
  Debug.Print Text(0)
  Debug.Print Text(1)
  Debug.Print Text(2)
  Debug.Print Text(3)
  Debug.Print Text(4)
End Sub
```

Part A
Part B
Part C
Part D
Part E

Sample 9 Using a one-dimensional field

In Sample 9 the one-dimensional **Text** field has been assigned five elements of the String type.

x	0	1	2	3	4
Value Text(x)	Part A	Part B	Part C	Part D	Part E

```
Option Explicit 'so all variables must be declared

Sub Main
  Dim x, y As Integer
  Dim Product(3,4) As Integer

  for x = 0 to 2
    for y = 0 to 3
      Product(x,y) = x * y
      Debug.Print Product(x,y)
    next y
  next x
End Sub
```

0
0
0
0
0
1
2
3
0
2
4
6

Sample 10 Multiplication

In Sample 10 the two-dimensional **Product** field has been assigned 3 times 4 elements of the type Integer.

		x		
		0	1	2
y	0	0	0	0
	1	0	1	2
	2	0	2	4
	3	0	3	6

```
Option Explicit 'so all variables must be declared

Sub Main
  Dim Field(2) As Variant

  Field(0) = "This is a text"
  Field(1) = 100

  Field(0) = Field(0) & " and not a number"
  Field(1) = Field(1) * 5

  Debug.Print Field(0)
  Debug.Print Field(1)
End Sub
```

This is a text and not a number
500

Sample 11 Using the Variant variable type

4.3.4 Loops

For Next

The For-next loop is a counting loop.

It is defined by a start value, end value, and optionally, by the step size.

The step size may be positive or negative; i.e., it is counted up or down. If no step size is stipulated, it will be set to the value of 1.

By using the **Exit** command, the loop can be exited prematurely when a certain criterion occurs.

Syntax:

For Variable = Start To End [Step Step size]

...

[Exit]

...

Next Variable

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim x!
  Dim Result!

  For x! = 0 To 1 Step 0.2
    Result! = x! * x!
    Debug.Print Result!
  Next x!
End Sub
```

0
0.04
0.16
0.36
0.64
1

Sample 12 For Next loop

While

The While loop is a Boolean loop.

The loop is repeated as long as the statement is true.

It may occur that the loop is never left.

A practical application example is waiting for a Laser Start signal which initiates the marking procedure.

Syntax:

While condition

...

Wend

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim i%

  i% = 5
  While i% > 2
    i% = i% - 1
    Debug.Print i%
  Wend
End Sub
```

4
3
2

Sample 13 While loop

Do

The Do loop is a Boolean loop.

The loop is repeated as long as the statement is true. I.e, it will be executed at least once.

It may occur that the loop is never left.

Syntax:

Do
...
[Exit]
Loop Until Condition

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim i%

  i% = 5
  Do
    i% = i% - 1
    Debug.Print i%
  Loop Until i%<2
End Sub
```

4
3
2
1

Sample 14 Do loop

A special form of the Do loop is the endless loop.

Syntax:

Do
...
Loop

This loop should be avoided since it makes a controlled program exit impossible.

4.3.5 Branching

Branches serve to distinguish cases during object time.

If Then Else

The If-Then-Else branch is a Yes/No decision.

Syntax 1:

```
If condition Then instruction
    ...
End If
```

Syntax 2:

```
If condition Then instruction Else instruction
    ...
End If
```

Syntax 3:

```
If condition Then
    ...
    Else if Condition Then
    ...
    Else
    ...
End If
```

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim i!
  i! = 5

  While i! > -2
    If i! > 2 Then
      Debug.Print "i is " & i! & "; thus greater than 2!"
    Else
      Debug.Print "i is " & i! & "; thus smaller than 2!"
    End If
    i! = i! - 1.5
  Wend
End Sub
```

i is 5; thus greater than 2!
i is 3.5; thus greater than 2!
i is 2; thus smaller than 2!
i is 0.5; thus smaller than 2!
i is -1; thus smaller than 2!

Sample 15 Branching with If Then Else

Select Case

The Select-Case decision is a decision between several options.

Syntax:

```
Select Case expression
    Case 1
        ...
    case 2
        ...
    Case 3
        ...
    Case Else
        ...
End Select
```

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim Workpiece As Integer

  Workpiece = 1 'selection no. 1
  Select Case Workpiece
    Case 1
    Marking1 'Marking1 function call
    Case 2
    Marking2 'Marking2 function call
    Case 3
    Marking3 'Marking3 function call
  End Select
End Sub
```

```
Sub Marking1
  'Branch to marking1
End Sub
```

```
Sub Marking2
  'Branch to marking2
End Sub
```

```
Sub Marking3
  'Branch to marking3
End Sub
```

Sample 16 Labelling depends on the **Workpiece** variable

4.3.6 Procedures and Functions

In order to make the program code clearer, grouping self-contained program parts into functions or procedures is recommended. The individual functions should be kept clear and short.

A function or procedure can often be divided into small functions or procedures.

Advantages:

- Programmers can find their way around in the code fast.
- Errors can be delimited fast.
- Program extensions can be realised easily.
- Functions and procedures can be used multiple times.

Procedures

Variables can be transferred to the procedures.

Syntax:

Sub Procedure name (Param1 As ... - ParamN As...)

...

End Sub

In Sample 16 several procedures were used:
In the main program, the Marking1 procedure is invoked. This is where the program branches into the Marking1 procedure and executes it.

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim Radius%
  Radius%=15

  Marking1(Radius%) 'Marking1 function call
End Sub
```

```
Sub Marking1(Radius%)
  Debug.Print Radius%
End Sub
```

15

Sample 17 Transfer of the **Radius** variable to the **Marking1** procedure

Functions

Functions are procedures which return a value.

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim i%
  Dim Result As Double

  i% = 10
  Result = Sqr (Sum(i%)) 'root of the function Sum
  Debug.Print "Result = " & Result
End Sub
```

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

10
Result = 4.47213595499958

Sample 18 Function

In our example, the square root of the returned value of the sum function is calculated. In this function then the sum of the variable i with itself is calculated.

4.3.7 Editing Texts

The functions described are important for editing and manipulating text. They represent only a portion of the full functional scope.

Len

Calculates the length of a string.

Syntax:

Len (String)

Type:

String

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim Value As Integer
  Debug.Print Len("Laser")
  Value = Len("Laser")* 2
  Debug.Print Value
End Sub
```

5
10

Sample 19 Calculating the length of a string

Left

Creates a string of the length Len, starting on the left.

Syntax:

Left (String, Len)

Type:

String

```
Option Explicit 'so all variables must be declared
```

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

La

Sample 20 Creation of a string starting on the left

Mid

Creates a string of the length Len, starting at the Index position.

Syntax:

Mid (String,Index,[Len])

Type:

String

```
Option Explicit 'so all variables must be declared
```

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

se

Sample 21 Creation of a string starting from any location

Right

Creates a string of the length Len, starting on the right.

Syntax:

Right (String,Len)

Type:

String

```
Option Explicit 'so all variables must be declared
```

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

er

Sample 22 Creation of a string starting on the right

Str

Creates a string from a numerical variable.

Syntax:

Str (Num)

Type:

String

```
Option Explicit 'so all variables must be declared
```

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

-40

Sample 23 Creation of a string from a numerical variable

StrReverse

Creates a string inverse to the start string.

Syntax:

StrReverse (String)

Type:

String

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Debug.Print StrReverse("Laser")
End Sub
```

resaL

Sample 24 Creation of an inverse string

UCase

Converts all lower case letters of a string into upper case.

Syntax:

UCase (string)

Type:

String

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Debug.Print UCase("Laser")
End Sub
```

LASER

Sample 25 Conversion of lower case letters into upper case

4.3.8 Mathematical Operations

Sin, Cos, Tan, Atn

These functions return the trigonometric functions of numerical values.

Syntax:

Sin (Num)
Cos (Num)
Tan (Num)
Atn (Num)

```
Option Explicit 'so all variables must be declared

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

Sample 26 Calculation of trigonometric functions

Exp, Log, Sqr

Calculation of the following functions:

- Exponential function,
- Logarithmic function,
- Square root.

Syntax:

Exp (Num)
Log (Num)
Sqr (Num)

```
Option Explicit 'so all variables must be declared

Sub Main
  Debug.Print Exp(1)
  Debug.Print Log(1)
  Debug.Print Sqr(81)
End Sub
```

2.71828182845905
0
9

Sample 27 Calculation of exponent, logarithm and square root

Abs, Fix, Int, Round, Sgn

Abs:	Forms the absolute value.
Fix:	Forms an integer variable by removing all decimals.
Int:	Forms an integer variable by rounding.
Round:	Rounds a numerical value to the number of decimals required.
Sgn:	Returns an Algebraic sign value.

Syntax:

Abs (Num)
Fix (Num)
Int (Num)
Round (Num,[Digits])
Sgn (Num)

```
Option Explicit 'so all variables must be declared
```

```
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

Sample 28 Calculation of Abs, Fix, Int, Round

4.3.9 Operators

n: Numeric value
s: String

- n1	Change of algebraic sign from n1.
n1 ^ n2	Takes n1 to the power of 2.
n1 * n2	Multiplies n1 and n2.
n1 / n2	Divides n1 by n2.
n1 \ n2	Divides the 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 to n2.
n1 < n2	Returns True if n1 less than n2.
n1 <= n2	Returns True if n1 less than or equal n2.
n1 > n2	Returns True if n1 greater than n2.
n1 >= n2	Returns True if n1 greater than or equal n2.
n1 = n2	Returns True if n1 equal n2.
n1 <> n2	Returns True if n1 unequal n2.
s1 = s2	Returns True if s1 equal s2.
s1 <> s2	Returns True if s1 unequal s2.
n1 And n2	Bitwise AND conjunction of n1 with n2.
n1 Or n2	Bitwise OR conjunction of n1 with n2.

```
Option Explicit 'so all variables must be declared

Sub Main
  Dim N1,N2 As Integer
  Dim S1$,S2$
  N1 = 10
  N2 = 3
  S1$ = "asdfg"
  S2$ = "hjkl"
  Debug.Print -N1 '-10
  Debug.Print N1 ^ N2 '1000
  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
```

Sample 29 Working with Operators

4.3.10 Type Conversion Functions

These functions convert expressions into certain data types.

If the value of an expression lies outside the range of the data type into which it is to be converted, an error will occur.

Syntax:

Conversion function (expression)

Function	Return Type	Range of the Expression argument
CBool	Boolean	A valid string or a valid numerical expression.
CByte	byte	0 to 255.
CCur	Currency	-922 337 203 685 477.5808 to 922 337 203 685 477.5807.
CDate	Date	Any valid date expression.
CDbl	Double	-1.79769313486231E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values.

Function	Return Type	Range of the Expression argument
CDec	Decimal	+/-79 228 162 514 264 337 593 543 950 335 for scaled integers; i.e., numbers without decimals. For numbers with 28 decimals the range... is valid +/-7.9228162514264337593543950335. The smallest possible number unequal Zero is 0.0000000000000000000000000001.
CInt	Integer	-32 768 to 32 767; Decimals are rounded.
CLng	Long	-2 147 483 648 to 2 147 483 647; Decimals are rounded.
CSng	Single	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.
Cvar	Variant	Numerical values in the range of the Double type. Non-numerical values in the range of the String type.
CStr	String	Return for CStr depends on Expression argument.

4.3.11 Working with Files

A file is a structured amount of data which, e.g., can be stored on the hard drive of a computer.

In order to be able to access the records of a file for input and/or output, it must be opened.

Access to files allows, e.g., logging marking processes in files or using data from files for laser marking.

File names

The ***Script*** language executes input and output operations by means of file numbers. This number is assigned to a file or unit when it is opened using the **Open** command.

The physical file is described by a string.

Syntax:

[Unit:] [Path] File name

Parameter:

Unit:	Input/Output unit
Path:	Path, directory in which the file is located.
File name:	File name

4.3.12 Sequential Files

Sequential files can be easily created.

The data is stored in the order it is transferred.

When reading from such a file, the file must be processed from the beginning. This is an essential disadvantage of sequential files.

Open

Opens or creates a File

Syntax:

Open File For Mode As # File Number [record length]

Parameter:

Mode:	Output	File is opened for writing or created, content is deleted.
	Append	File is opened for adding.
	Input	File is opened for reading, error if file does not exist.
	Random	Access to entire records, length of sentence must be stated.

Close

Closes a file

Syntax:

Close # File number

Print

Writes records to an opened file.

Syntax:

Print # File number Expression list

The values of the list of expressions are written to the file consecutively **without** separators, as one record.

Write

Writes records to an opened file.

Syntax:

Write # File number Expression list

The values of the list of expressions are written to the file, consecutively **with commas** as separators, as one record.

```
Option Explicit 'so all variables must be declared
Sub Main
   Dim A As Integer
   A=5
   Open "c:\temp\Mark.txt" For Output As #1
   Print #1,"A=";",";A
   Close #1
End Sub
```

Sample 30 Open a file, write a record and close the file

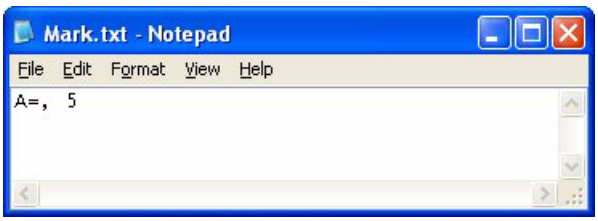


Fig. 81 Result file for Sample 30

```
Option Explicit 'so all variables must be declared
Sub Main
   Dim B As Integer
   B=10
   Open "c:\temp\Mark.txt" For Append As #1
   Print #1,"B=";",";B
   Close #1
End Sub
```

Sample 31 Open a file, add a record and close the file

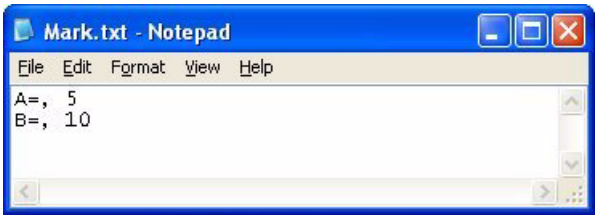


Fig. 82 Result file for Sample 31

Input

Reads a line in an opened file.

Syntax:

Input # File number List of variables

The parts of the line divided by commas are transferred to the variables of the list of variables.

Line Input

Reads a line in an opened file.

Syntax:

Line Input # File number Variable

The content of the line is transferred to the variable.

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim Text$
  Dim Value%
  Dim Lines$
  Open "c:\temp\Mark.txt" For Input As #1
  While Not EOF(1)
    Input #1,Text$,Value%
    Line Input #1,Lines$
    Debug.Print Text$;Value%
    Debug.Print Lines$
  Wend
  Close #1
End Sub
```

A= 5
B=, 10

Sample 32 Reading of the lines of the file created in Sample 31

4.3.13 Files with Direct Access

Files with direct access allow access to selected records via a record number.

Put

Writes the content of a variable into a record of a file.

Syntax:

Put File Number, [Record Number], Variable

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim V As Variant
  V="Text;"
  Open "C:\temp\sample.txt" For Random As #1 Len=15
  Put #1,1,V
  Put #1,2,V
  Put #1,3,V
  Close #1
End Sub
```

Sample 33 Generating a file with direct access.

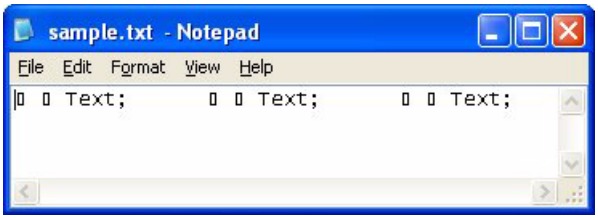


Fig. 83 Result file for Sample 33

Get

Reads the record content in a variable

Syntax

Get File number, [Record number], Variable

```
Option Explicit 'so all variables must be declared
```

```
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;

Sample 34 Read out the 1st record form the file of Sample 33

4.3.14 Creating User Dialog Windows

Edit user dialog

Click on the **Edit user dialog** icon.
The **User Dialog Editor** box will open.

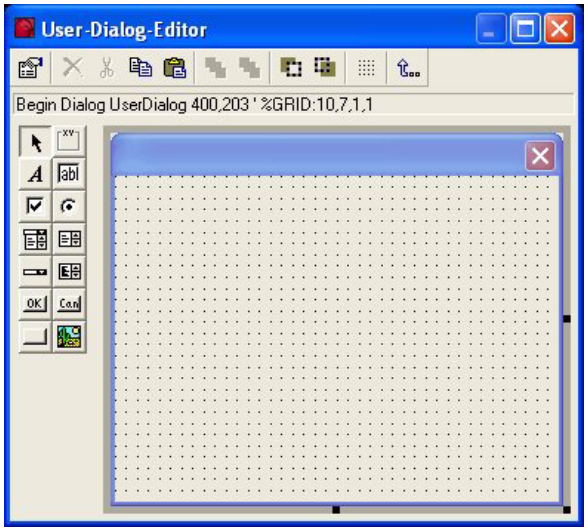


Fig. 84 User dialog editor

Example for creation of a simple user dialog box:

1. Create a new Basic project.
1. Click on the **Edit user dialog** icon.
2. The **User Dialog Editor** box will open.
3. On the left hand side, there is a number of controls. Click on the **Add Ok button** icon.
4. The mouse cursor will change into a cross hair. Move the mouse into the position of the window in which the button is to be inserted. Left-click on the mouse and define any size for the button. Release the mouse when the size appears correct.
5. If you would like to keep the window defined, click the **Accept** button.
6. The Editor will close, and you are back in program code. A few lines were inserted automatically for defining the window.

```
Option Explicit 'so all variables must be declared

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
```

Sample 35 Program code for a simple user dialog window

When the program is executed, the defined dialog window will open and will wait for an acknowledgement via a mouse click on the **OK** button.

Menu bar

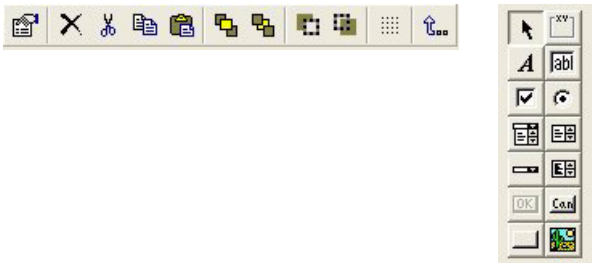


Fig. 85 Menu bar of the user dialog editor

Editing properties

Highlight a control or the entire options box (happens automatically if no control is highlighted).
Click on the **Edit Properties** icon.
The **Edit Properties** dialog box opens.
The window can also be opened by double-clicking on an element.

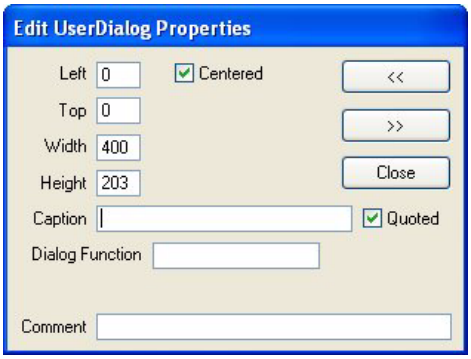


Fig. 86 Edit Properties dialog box for the user dialog

Centred:	If the check box is activated, the dialog box will be centred.
Left/Top:	Dialog box position (with centring turned off.)
Width/Height:	Window size.
Caption:	Name of window.
Quoted:	If the check box is activated, the text entered under Caption will appear as a string. Otherwise, the text entered under Caption will represent a variable.
Dialog function:	Supports execution of complex functions.
Comment:	Any comment may be entered.

View of elements

Fig. 87 Toolbar view of elements

If several elements are used in a dialog box, the functions of the Element view provide a clearer representation. If several elements overlap, the element which comes first in the program will be the top element during processing.

Select

The **Select** function allows choosing or highlighting controls.

Add group box

This element serves to organise controls. The box can be assigned a name consisting of fixed text or a variable.

Add text

This element serves to the output of text in the user dialog window. Fixed or variable text content can be given.

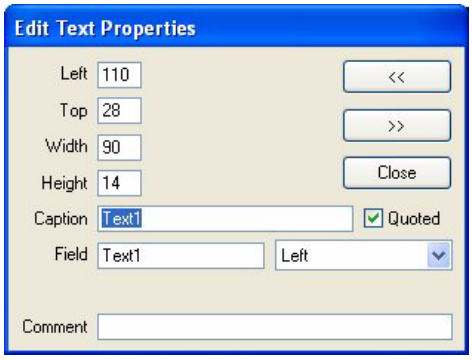


Fig. 88 Edit Text Properties dialog box

The text can be formatted (left justified, right justified, centred).

Graphically, the **Laser** script is represented as a string. If the **Quoted** check box is deactivated, Laser will become a variable. Then the content of this variable will be depicted.

```
Option Explicit 'so all variables must be declared

Sub Main
  Dim Laser As String
  Laser = "5345"
  Begin Dialog UserDialog 400,203
    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
```

Sample 36 Declaration of the marking as string or variable

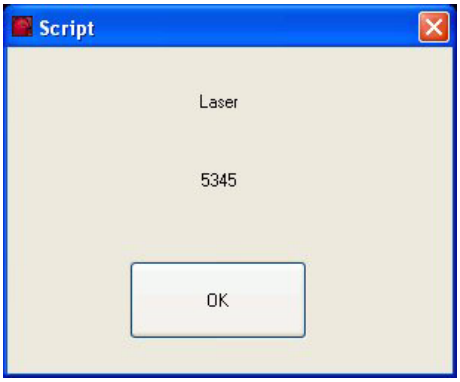


Fig. 89 Result of Sample 36

Add text box

This element allows entering data during run time.

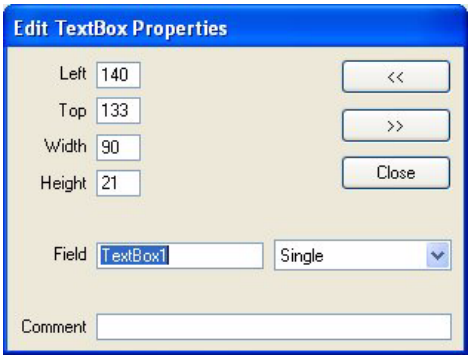


Fig. 90 Edit Text Box Properties dialog box

The content entered will be written into a variable with the name assigned under **Field**. In the further processing of the program, this variable can be accessed.

```
Option Explicit 'so all variables must be declared

Sub Main
  Dim Radius As String
  Begin Dialog UserDialog 380,175,"Circle"
    OKButton 130,112,130,49
    TextBox 230,56,60,21,.Variable
    Text 70,56,160,21,"Radius:",.Text1
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
  MsgBox (dlg.Variable)
End Sub
```

Sample 37 Output and Input of text

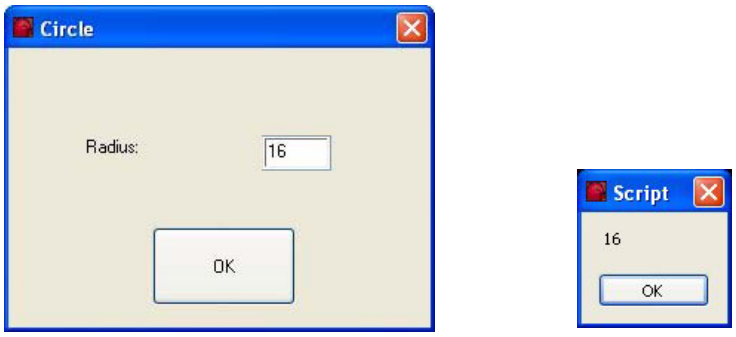


Fig. 91 User dialog box for Sample 37

Add check box

The check box allows the user to activate or deactivate various options during program processing.
A check box can assume exactly two states, true or false. If the check box is activated, the status will change to **True**; otherwise, the **False** status will remain.

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Selection
End Sub
```

```
Function Selection
  Begin Dialog UserDialog 400,203,"CheckBox"
    CheckBox 240,70,90,14,"Circle",.CheckBox1
    CheckBox 240,105,90,14,"Rectangle",.CheckBox2
    OKButton 120,140,180,35
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
  If dlg.checkbox1 = 0 and dlg.checkbox2 = 0
    Then MsgBox ("nothing")
  If dlg.checkbox1 = 1 and dlg.checkbox2 = 0 Then MsgBox ("Circle")
  If dlg.checkbox1 = 0 and dlg.checkbox2 = 1 Then MsgBox ("Rectangle")
  If dlg.checkbox1 = 1 and dlg.checkbox2 = 1
    Then MsgBox ("both")
End Function
```

Sample 38 Selecting Markings Using Check Boxes

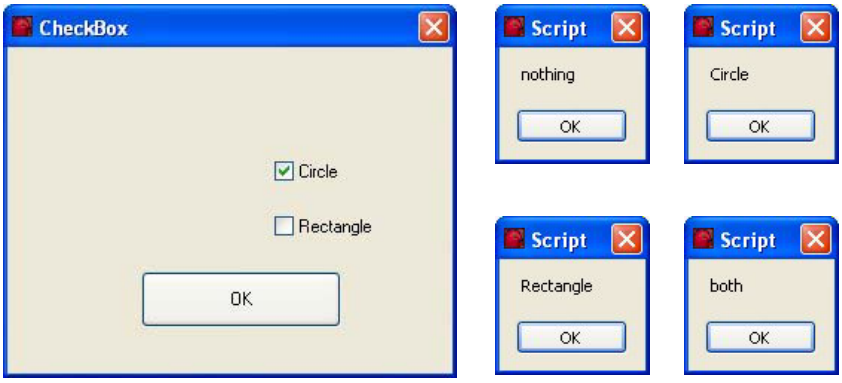


Fig. 92 User dialog box to Sample 38

A circle, rectangle, nothing or both may be selected. The query of the variables should occur correspondingly.

Add options button

The Options button is a switch which can assume only the states of **True** or **False**. In this context, however, all elements of a group are interdependent since they are linked by an OR-function. I.e., only one Option button out of a group may assume the status **True**; the other elements will remain on **False**.
Any number of Option buttons may be organized into a group.

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Selection
End Sub
```

```
Function Selection
  Begin Dialog UserDialog 400,203,"Option Button"
    OKButton 120,140,180,35
    OptionGroup .Group1
    OptionButton 160,70,80,14,"Circle",.OptionButton1
    OptionButton 160,91,100,14,"Rectange",.OptionButton2
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
  If dlg.Group1 = 0 Then MsgBox ("Circle")
  If dlg.Group1 = 1 Then MsgBox ("Rectangle")
End Function
```

Sample 39 Selecting Markings Using Option Buttons

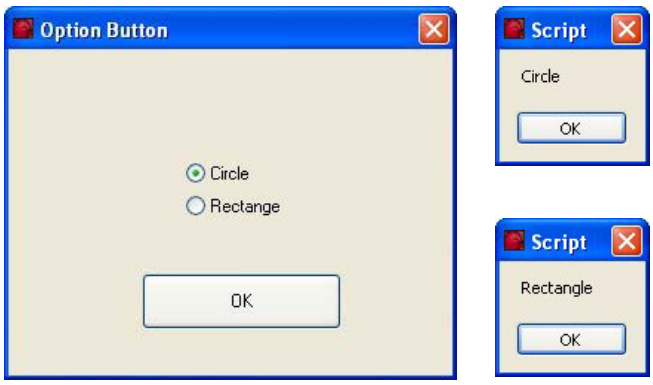


Fig. 93 User dialog box to Sample 39

Either a circle or a rectangle may be selected.

Add list box

Using the List box, lists may be shown. If the list contains more elements than can be shown, scroll bars will be added.
A list element can be selected, but there is no option for adding elements.
The List box is defined by a field.
Access to the user's selection is obtained by evaluating the variables assigned under **Field**; it contains the field number selected.

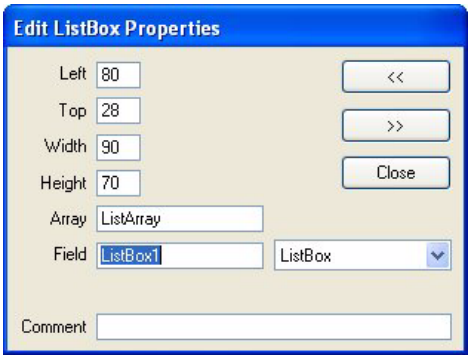


Fig. 94 Edit ListBox Properties dialog box

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim Mark(2) As String
  Mark(0)= "Circle"
  Mark(1)= "Rectangle"
  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 MsgBox ("Circle")
  If dlg.ListBox1 = 1 Then MsgBox ("Rectangle")
End Sub
```

Sample 40 Selecting Markings Using a List Box

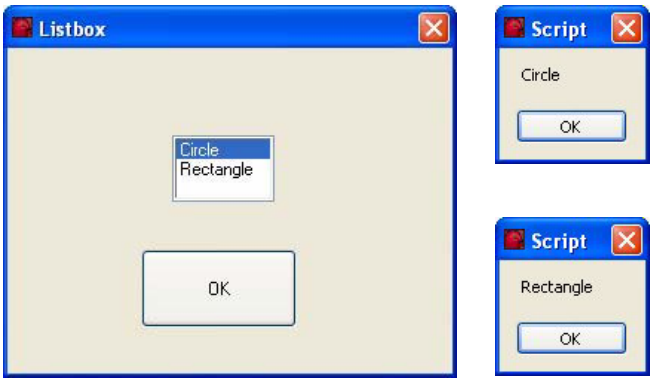


Fig. 95 User dialog box for Sample 40

Either a circle or a rectangle may be selected.

Add droplist box

Lists may be shown using the Droplist box. A line with scroll bars will be appear.
A list element may be selected or a value may be entered.
The List box is defined by a field.

Access to the user' selection is obtained by evaluating the variables assigned under **Box**; it contains the value entered or selected.

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim Mark(2) As String
  Mark(0)= "Circle"
  Mark(1)= "Rectangle"
  Begin Dialog UserDialog 400,203,"DropListBox"
    OKButton 130,119,130,56
    DropListBox 150,56,90,49,Mark(),.DropListBox1,1
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
  MsgBox (dlg.DropListBox1)
End Sub
```

Sample 41 Selecting Markings Using a Droplist Box

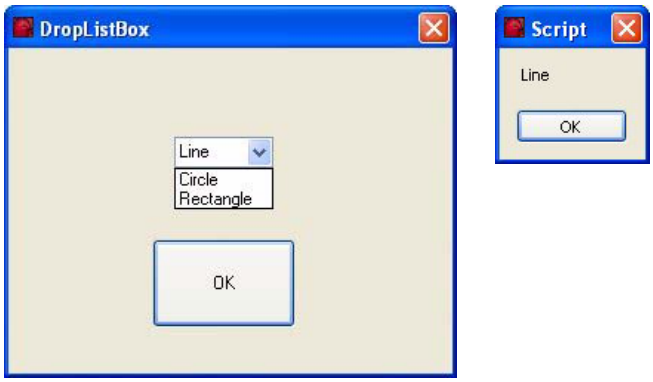


Fig. 96 User dialog box for Sample 41

A circle or a rectangle may be selected, or different text may be entered.

Add combo box

The Combo box corresponds to the Droplist box in its role; they differ in their representations.
With the Combo box, the select box is permanently open; if necessary, you can scroll through this box.
With the DropList box, the select box is not opened until activated.

Add picture

Using the **Add Picture** function, bitmap files may be linked to the user dialog box.

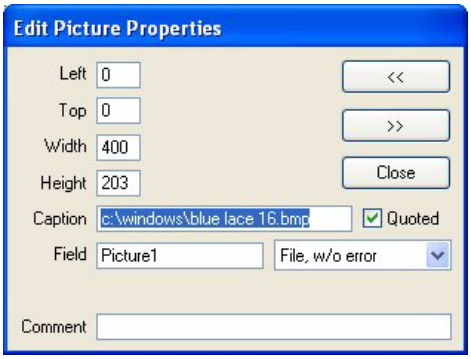


Fig. 97 Edit Picture Properties dialog box

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Begin Dialog UserDialog 400,203,"Picture"
    Picture 0,0,400,203,"c:\windows\blue lace 16.bmp",0,.Picture1
    OKButton 120,112,160,56
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
End Sub
```

Sample 42 Linking a Bitmap File as Background for a User Dialog Box

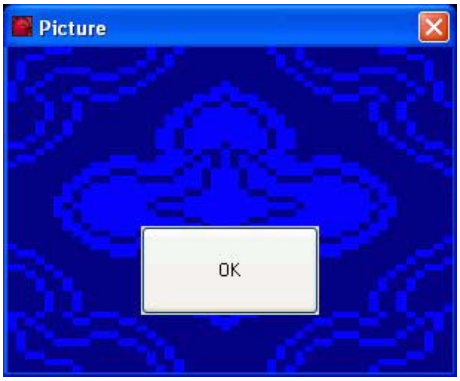


Fig. 98 User dialog box for Sample 42

Add buttons

Buttons are required for acknowledging entries or starting processes. Three different types are available:

- OK button,
- Cancel button,
- Push button.

NOTE!
For an executable dialog window, at least one of these buttons must be supplied!

```
Option Explicit 'so all variables must be declared

Sub Main
  Begin Dialog UserDialog 260,126,"Button"
    Text 60,28,140,28,"Please push a button"
    OKButton 10,91,60,21
    PushButton 90,91,60,21,"&Run"
    CancelButton 170,91,80,21
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub
```

Sample 43 User dialog box with three buttons

Fig. 99 User dialog box for Sample 43

Depending on the key pressed, Variable Dialog (dlg) will return the following results, which may be accessed when processing the program:

OK	OK-Button	-1,
Run	Push-Button	1,
Cancel	Cancel-Button	0.

Dialog function

The Dialog function in the User Dialog menu may be used, among other things, for keeping a dialog box open permanently during program processing.

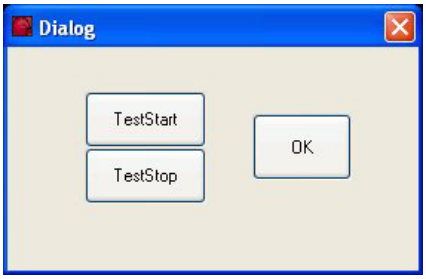
```
Option Explicit 'so all variables must be declared
```

```
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 (TestRun)
    End If
    DialogFuncStatus% = True
  End Select
End Function
```

```
Sub DOIT (TestRun)
  MsgBox("This is where the marking can be done!")
  TestRun = False
End Sub
```

Sample 44 Dialog function



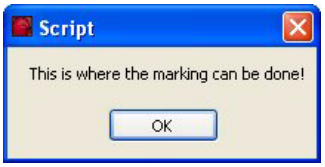


Fig. 100 User dialog box for Sample 44

By calling up User Dialog, the DialogFuncStatus dialog function is processed. In this context, a Select-Case branching will evaluate the user's actions.

4.3.15 Dialog Box

Input box

Options for the user entering data.

Syntax:

Return Value = InputBox("Message"[, "Title"][, "Default"][, XPos, YPos])

Parameter:

Return Value:	The String type variable is used to store text entered by the user.
Message:	Text which appears in the input box and prompts the user to act.
Title:	Input Box Designation
Default:	This is where an instruction for the user may be placed.
XPos:	X-position for the upper left corner of the Input box.
YPos:	Y-position for the upper left corner of the Input box.

```
Option Explicit 'so all variables must be declared

Sub Main
  Dim L$
  L$ = InputBox("Enter your name:", "Input box", "Ingmar Grote")
  Debug.Print L$
End Sub
```

Sample 45 Input box for entering a name which is then output

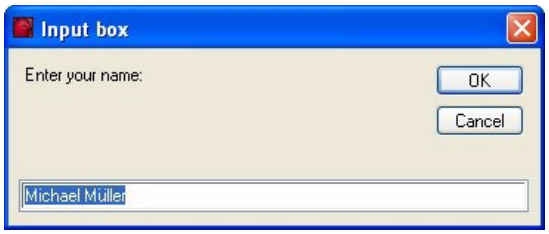


Fig. 101 User dialog box for Sample 45

Message Box

The Message box shows a pre-defined output dialog. The button can be evaluated through the return value.

Syntax:

Return Value = MsgBox("Message"[, Buttons][, "Title"])

or

MsgBox"Message"[, Buttons][, "Title"]

Parameter:

Return Value: The user input is returned in a variable of the Integer type.

Return value	Constant	selected button
1	vbOk	OK
2	vbCancel	Cancel
3	vbAbort	Abort
4	vbRetry	Repeat
5	vbIgnore	Ignore
6	vbYes	Yes
7	vbNo	No

Message: Text which appears in the input box and prompts the user to act.

Buttons: Constants or values for buttons and icons to be represented.

Constant	Value	Appearance
vbOkOnly	0	OK
vbOkCancel	1	OK, Cancel
vbAbortRetryIgnore	2	Abort, Retry, Ignore
vbYesNoCancel	3	Yes, No, Cancel
vbYesNo	4	Yes, No
vbRetryCancel	5	Retry, Cancel
	0	No icon
vbCritical	16	Stop icon
vbQuestion	32	Question symbol
vbExclamation	48	Caution symbol
vbInformation	64	Information symbol

Title: Message Box designation.

```
Option Explicit 'so all variables must be declared

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

Sample 46 Appearance of different Message boxes

The selection made can be queried:

```
Option Explicit 'so all variables must be declared

Sub Main
  If MsgBox("Program example",4,"Query")= vbYes Then
    MsgBox("Yes pressed!",16,"Information")
  Else
    MsgBox("No pressed!",32,"Information")
  End If
End Sub
```

Sample 47 Querying the reaction to a Message box

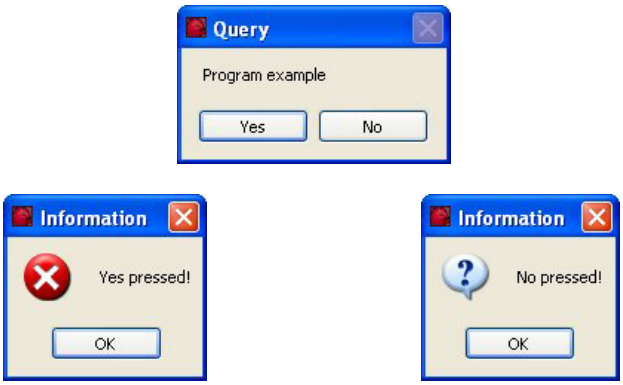


Fig. 102 User dialog boxes for Sample 47

PopUp menu

Various actions may be initiated using a PopUp menu.

Syntax:

Return Value = ShowPopupMenu (Field( ) [, Alignment][, XPos, YPos])

Parameter:

Return Value: The list number selected is returned as a variable of the Integer type.

Field: In a one-dimensional field, all selection options are determined.

Alignment: Aligning the menu with the X- and Y-coordinates.

PopupStyle	Value	Effect
vbPopupLeftTopAlign	0	Basic Alignment position
vbPopupUseLeftButton	1	Selection can be made only with the left mouse button.
vbPopupUseRightButton	2	Selection can be made with the left or right mouse button
vbPopupRightAlign	4	Menu is located on 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 in the Y-position

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Dim Selection As Integer
  Dim Items(0 To 2) As String
  Items(0) = "&Circle"
  Items(1) = "&Rectangle"
  Items(2) = "&Vector"
  Selection = ShowPopupMenu (Items)
  If selection = 0 Then circle 'Popup menu with selection circle
  If selection = 1 Then rectangle 'Popup menu with selection rectangle
  If selection = 2 Then vector 'Popup menu with selection vector
End Sub
```

```
Function circle
  MsgBox ("Circle")
End Function
```

```
Function rectangle
  MsgBox ("Rectangle")
End Function
```

```
Function vector
  MsgBox ("Vector")
End Function
```

Sample 48 Popup menu for selecting a shape for output

4.3.16 Handling of Errors

Errors may occur regularly during program processing. Therefore, during program development, care should be taken that the programs are able to tolerate potential errors.

For this purpose, the On Error command is available which allows the program to be informed how it should respond when errors occur. In the context of this command, the commands Goto and Resume Next are used.

On Error Goto

If after entering the **On Error Goto Target Mark** command an error occurs, the program will jump to the target mark. This is where the user can be informed about the error that occurred.

Syntax:

On Error Goto Target Mark

...

Target

...

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Calculation
End Sub
```

```
Function Calculation
  Dim a!,t!
  Dim Result As Single
  On Error GoTo Errors
  a!=InputBox ("Input number")
  t!=InputBox ("Input divisor")
  Result = a!/t!
  Debug.Print Result
  Exit Function
  Errors:
  MsgBox("An error occurred.", 48, "Error")
End Function
```

Sample 49 Error Detection and Information

In the program example, input errors committed by the user are detected and reported. These may include:

- Division by Zero,
- Entering strings,
- No entry.

On Error Resume Next

This command causes the program line containing the error to be skipped. While this results in errors being avoided, it may also lead to false calculations.

Syntax:

On Error Resume Next

...

```
Option Explicit 'so all variables must be declared
```

```
Sub Main
  Calculation
End Sub
```

```
Function Calculation
  Dim a!,t!
  Dim Result As Single
  On Error Resume Next
  a!=InputBox ("Input number")
  t!=InputBox ("Input divisor")
  Result = a!/t!
  Debug.Print a!;t!;Result
  Exit Function
End Function
```

Sample 50 Skipping calculation when entry is faulty

4.4 Laser specific script extensions

4.4.1 Callback procedures

Callback procedures are automatically dealt with by the program to keep you informed about asynchronous events.
To get access to templates of such procedures select **LC** in the **Object:** drop down list, and see all the available templates in the **Proc:** drop down list.

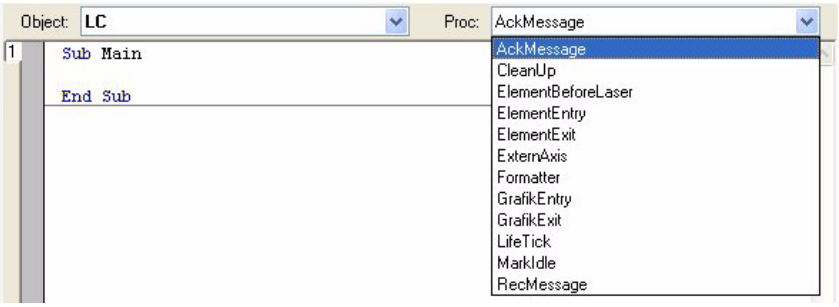


Fig. 103 Script-Object-Proc

LC_CleanUp()

Public Sub LC_CleanUp()

Is applied every time the program is stopped by the user, so the program is not allowed to reach its normal end.
In LC_CleanUp used resources should be freed.

LC_OnError()

Public Sub LC_OnError()

Is called if the program detects an error initiated by the laser.

LC_LifeTick()

Public Sub LC_LifeTick()

Is periodically activated every one second. The time can be adjusted using **LiveTickIntervall(Intervall_ms As Integer)**.

LC_AckMessage(...)

Public Sub LC_AckMessage(Host As String, Laser As String, bError As Boolean)

Is called if your Message is processed by the recipient.

LC_RecMessage(...)

Public Sub LC_RecMessage(Host As String, Laser As String, Message As String)

Is called right after a message arrived for this program.

LC_GrafikEntry()

Public Sub LC_GrafikEntry()

In the path of processing the graphic (output the graphic to the laser) this is called first.

LC_GrafikExit()

Public Sub LC_GrafikExit()

This call concludes the processing of the graphical objects.

LC_ElementEntry(...)

Public Sub LC_ElementEntry(Label As String, ArrayX As Long, ArrayY As Long)

Every time the processing of a new element starts, **LC_ElementEntry** is called.

LC_ElementExit(...)

Public Sub LC_ElementExit(Label As String)

After all vectors of an element are send to the laser this callback is called.
In **Label** you find the corresponding label assigned to the element.

LC_ElementBeforeLaser(...)

Public Sub LC_ElementBeforeLaser(Label As String, ArrayX As Long, ArrayY As Long)

Is called after all of the vectors are calculated, right before they are send to the laser.

LC_MarkIdle(...)

Public Sub LC_MarkIdle(Label As String)

Every time the laser buffer is full, so the laser is not capable of accept additional vectors **LC_MarkIdle** is called.

LC_Formatter(...)

Public Function LC_Formatter(Format As String, Text As String, Default As String, Laser As Boolean) As String

If you have used a user defined format string (#Test=123#) which can not be resolved by the system, **LC_Formatter** is called, so you can provide the necessary information.
See also Format specifications on page 132.
In **Text** you find the format text **Test** in this example.
The default value (123) is stored in ***Default***.
If the variable **Laser** is true, than the callback is called during the laser process. If it is false than we are in preview.
Use **LC_Formatter = ...** to define your answer, the result.
For your Information: The complete format string (#Test=123#), as entered, is inside the **Format** variable.

LC_ExternAxis(...)

Public Sub LC_ExternAxis(AxisX As Long, AxisY As Long, AxisZ As Long, Axis4 As Long)

Is called if one or more axis values has changed.
Use the values provided in **AxisX** and **AxisY** to move your axis system to the desired positions.

4.4.2 Internal extensions (LC.)

GetLaserConfiguration

LC.GetLaserConfiguration() As String

Parameter: ---.
Return: The name of the current Laser configuration.

GetAppPathName

LC.GetAppPathName() As String

Parameter: ---.
Return: Program path inclusive program name.

GetAppPath

LC.GetAppPath() As String

Parameter:	---.
Return:	Program path wo. name.

GetAppName

LC.GetAppName() As String

Parameter:	---.
Return:	Program name wo. path.

Hide

LC.Hide()

Parameter:	---.
Return:	---.
Function:	Hide the program.

Show

LC.Show(Tab As Integer)

Tab:	No. of the area to activate (1=Graphic, 2=Script, 3=Parameter, 4=Service).
Return:	---.
Function:	Shows the Program and switches to the desired area.

ApplicationExit

LC.ApplicationExit()

Parameter:	---.
Return:	---.
Function:	Exit the program.

LifeTickInterval

LC.LifeTickInterval(Interval_ms As Integer)

Interval_ms:	Defined the time interval used to call **Public Sub LC_LifeTick()** in ms.
Return:	---.

TimerStart

LC.TimerStart()

Parameter:	---.
Return:	---.
Function:	Activates the **user timer**, see also User timer on page 21 point 23.

TimerStop

LC.TimerStop()

Parameter: ---.

Return: ---.

Function: After call the **user timer** is stopped.
See also User timer on page 21 point 23.

StartRecMessage

LC.StartRecMessage()

Parameter: ---.

Return: ---.

Function: After call the program is ready to receive messages.

Attention: After receiving a message, a callback procedure is called.
See also Callback procedures on page 107
This function uses socket ports from 10000 to 19999.

StopRecMessage

LC.StopRecMessage()

Parameter: ---.

Return: ---.

Function: The message reception is canceled after the call of this command.

SendMessage

LC.SendMessage(Host As String, Laser As String, Message As String)

Host: The recipients computer name.
Alternative the IP-Address.

Laser: The recipients laser configurations name.

Return: ---.

Attention: After the reception of the message a callback is initiated.
See also Callback procedures on page 107
This function uses socket ports from 10000 to 19999.

LoadFile

LC.LoadFile(FileName As String)

FileName: The file name of the graphic file that should be loaded.

Return: **True**, if successful.

Attention: Is used to load a graphic file (*.LAS) at runtime.

SaveFile

LC.SaveFile(FileName As String)

FileName:	The file name to which the graphic file will be saved (**Save As**). If the Filename is empty ("") the file will be saved using its own name (**Save**).
Return:	none.
Attention:	Is used to save a graphic file (*.LAS) at runtime.

GetFileName

LC.GetFileName() As String

Return:	The name and path of the current graphic file.

StatusText

LC.StatusText(Text As String)

Text:	The text to show in the status line.
Return:	---.

Mark

LC.Mark(Label As String, Shutter As Boolean) As Boolean

Label:	The graphic or element name to start the output with. To output the complete graphic, set **Label** to an empty string ("").
Shutter:	If **True**, the shutter will be controlled automatically.
Return:	**True**, if successful.

StopMark

LC.StopMark()

Function:	Used to stop the graphic output asynchronously. Normally executed within a callback procedure.
Return:	---.

SetMoveOffset

LC.SetMoveOffset(OffsetX As Long, OffsetY As Long)

OffsetX, Y:	Used to move the complete graphic by the given values in µm.
Return:	---.
Attention:	This is a global command.

SetPowerOffset

LC.SetPowerOffset(Offset As Double)

Offset:	The given value in % is added to the laser power.
Return:	---.
Attention:	This is a global command.

GetBooleanValue

LC.GetBooleanValue(Label As String, ParameterNo As Integer) As Boolean

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to read.
Return:	The value of the parameter.

GetNumericValue

LC.GetNumericValue(Label As String, ParameterNo As Integer) As Double

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to read.
Return:	The value of the parameter.
Attention:	Size and position values are all in µm!

GetNumericValue_mm

LC.GetNumericValue_mm(Label As String, ParameterNo As Integer) As Double

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to read.
Return:	The value of the parameter in **mm**.

GetNumericValue_mil

LC.GetNumericValue_mil(Label As String, ParameterNo As Integer) As Double

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to read.
Return:	The value of the parameter in **mil**.

GetNumericValue_inch

LC.GetNumericValue_inch(Label As String, ParameterNo As Integer) As Double

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to read.
Return:	The value of the parameter in **inch**.

GetStringValue

LC.GetStringValue(Label As String, ParameterNo As Integer) As String

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to read.
Return:	The value of the parameter as string.
Attention:	Size and position values are all in µm!

SetBooleanValue

LC.SetBooleanValue(Label As String, ParameterNo As Integer, Value As Boolean)

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to modify.
Value:	The new state.
Return:	---.

SetNumericValue

LC.SetNumericValue(Label As String, ParameterNo As Integer, Value As Double)

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to modify.
Value:	The new value.
Attention:	Size and position values must be supplied in µm!
Return:	---.

SetNumericValue_mm

LC.SetNumericValue_mm(Label As String, ParameterNo As Integer, Value As Double)

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to modify.
Value:	The new value in **mm**.
Return:	---.

SetNumericValue_mil

LC.SetNumericValue_mil(Label As String, ParameterNo As Integer, Value As Double)

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to modify.
Value:	The new value in **mil**.
Return:	---.

SetNumericValue_inch

LC.SetNumericValue_inch(Label As String, ParameterNo As Integer, Value As Double)

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to modify.
Value:	The new value in **inch**.
Return:	---.

SetStringValue

LC.SetStringValue(Label As String, ParameterNo As Integer, Value As String)

Label:	The name of the element, object or modifier.
ParameterNo:	The number of the parameter to modify.
Value:	The new value as string.
Return:	---.

Refresh

LC.Refresh()

Return:	---.
Function:	Redraw's the graphic screen.

SetCheck

LC.SetCheck(Label As String, State As Boolean)

Label:	The name of the element, object or modifier.
State:	The new checkbox state.
Return:	---.
Function:	Used to enable or disable the element, object or modifier.

GetCheck

LC.GetCheck(Label As String) As Boolean

Label:	The name of the element, object or modifier.
Return:	The current checkbox value.

BoundingBox

LC.BoundingBox(Label As String) As Boolean

Label:	The name of the element.
Function:	Determined the bounding box around the objects inside the element.
Return:	**True** if successful.

ShowBoundingBox

LC.ShowBoundingBox(State As Boolean)

State:	Activate (true)/Deactivate (false) the output of the bounding box using the pilot laser.
Return:	---.

GetBBMinX

LC.GetBBMinX() As Long

Return: The minimum X value of the bounding box.

GetBBMaxX

LC.GetBBMaxX() As Long

Return: The maximum X value of the bounding box.

GetBBMinY

LC.GetBBMinY() As Long

Return: The minimum Y value of the bounding box.

GetBBMaxY

LC.GetBBMaxY() As Long

Return: The maximum Y value of the bounding box.

ZoomBoundingBox

LC.ZoomBoundingBox()

Function: Zoom the graphic view to show the content of the bounding box.

ZoomAll

LC.ZoomAll()

Function: Zoom the graphic view to show the content inside the marking area.

4.4.3 External Extensions (EX.)

Formatter

EX.Formatter(Format As String) As String

Format: The Format string to use, see also Format specifications on page 132.

Return: The processed string.

ReadIniFormat

EX.ReadIniFormat(SectionName As String, KeyName As String, DefaultString As String, FileName As String) As String

SectionName: The INI-File section to read.

KeyName: The key inside the section to read.

DefaultString: If it is not possible to read the INI-Value then the **DefaultString** is returned.

FileName: The file to read.

Return: The content of the key requested or the **DefaultString**.

WriteIniFormat

EX.WriteIniFormat(SectionName As String, KeyName As String, ValueString As String, FileName As String)

SectionName:	The section to write.
KeyName:	The key to write.
ValueString:	The content to write.
FileName:	The file name to write to.
Return:	---.

ReadXmlFormat

EX.ReadXmlFormat(NodeName As String, DefaultString As String, FileName As String) As String

NodeName:	The XML-Node to read.
DefaultString:	If the node doesn't exist **DefaultString** is returned.
FileName:	The file to read.
Return:	The content of the node or the **DefaultString**.

WriteXmlFormat

EX.WriteXmlFormat(NodeName As String, ValueString As String, FileName As String)

SectionName:	The XML-Node to write to.
ValueString:	The content to write.
FileName:	The file to write to.
Return:	---.

SerOpen

EX.SerOpen(PortNo As Integer, Param As String) As Boolean

PortNo:	The port number to open (1...64).
Param:	COM port parameter in the following format: **Baud Rate**, **Parity**, **DataBits**, **StopBits** (example: "38400, n, 8, 2").
Return:	**True** if successful.
Attention:	Reading from, and writing to the COM ports is done in background. The size of the send and receive buffer is 10 KBytes.

SerClose

EX.SerClose(PortNo As Integer)

PortNo:	The port to close.
Return:	---.

SerCloseAll

EX.SerCloseAll()

Parameter:	---.
Return:	---.
Function:	All COM ports will be closed.

SerChangeBaudRate

EX.SerChangeBaudRate(PortNo As Integer, BaudRate As Integer) As Boolean

PortNo:	The port number to change.
BaudRate:	The new Baud rate (e.g.:19200).
Return:	**True**, if successful.

SerWrite

EX.SerWrite(PortNo As Integer, Data As String) As Integer

PortNo:	The port to write to.
Data:	The data to transmit.
Return:	The byte count actually send.

SerWriteByte

EX.SerWriteByte(PortNo As Integer, Data As Byte) As Integer

PortNo:	The port to write to.
Data:	The byte to send.
Return:	The byte count actually send.

SerWritePending

EX.SerWritePending(PortNo As Integer) As Boolean

PortNo:	The port number.
Return:	**True** if the write process is still pending.

SerWritePendingWait

EX.SerWritePendingWait(PortNo As Integer, TimeOut As Integer) As Boolean

PortNo:	The port number.
TimeOut:	Time in ms to wait to complete the writing.
Return:	**True** if the write process is still pending.

SerRead

EX.SerRead(PortNo As Integer, Count As Integer) As String

PortNo:	The port number.
Count:	The byte count to read.
Return:	The already read Byte count.

SerReadByte

EX.SerReadByte(PortNo As Integer, TimeOut As Integer) As Integer

PortNo:	The port number.
TimeOut:	Time in ms to wait to complete the reading.
Return:	The read Byte or -1 if time out.

SerReadWait

EX.SerReadWait(PortNo As Integer, Count As Integer, TimeOut As Integer) As String

PortNo:	The port number.
Count:	The byte count to read.
TimeOut:	Time in ms to wait to complete the reading.
Return:	The read bytes.

SerReadUntil

EX.SerReadUntil(PortNo As Integer, Char As Byte) As String

PortNo:	The port number.
Char:	Byte to wait for.
Return:	The read bytes.

SerReadUntilWait

EX.SerReadUntilWait(PortNo As Integer, Char As Byte, TimeOut As Integer) As String

PortNo:	The port number.
Char:	The byte to wait for.
TimeOut:	Time in ms to wait to complete the reading.
Return:	The read bytes.

SerReadPending

EX.SerReadPending(PortNo As Integer) As Boolean

PortNo:	The port number.
Return:	**True** if the reading is still pending.

4.4.4 ScannerControl extensions (SC.)

All the **SC.** commands have a leading character with the following meaning:

A_:	Asynchronous command. It is possible to use this command while output to the laser is in progress.
S_:	Synchronous command. This command is inserted in the buffer list of the laser. Before execution of such a command you should check if there is enough space in the buffer, use the **A_ReadyForNextSyncCmd()** to do so.
P_:	This is a parameter command. Use this command prior to all others.

A_GetStatusDigital

SC.A_GetStatusDigital(StatusBlock As Long) As Long

StatusBlock:	The block number you like to receive.
Return:	Bits 0...15 of the addressed block.
Attention:	Please ask your supplier for information about the block numbers and there meanings.

A_GetStatusAnalog

SC.A_GetStatusAnalog(StatusBlock As Long) As Long

StatusBlock:	The block number you like to get.
Return:	Value of the addressed block.
Attention:	Please ask your supplier for information about the block numbers and there meanings.

A_SetLaserState

SC.A_SetLaserState(State As Boolean)

State:	**True** means **ON**, **False OFF**.
Return:	---.
Function:	Switches the Laser on or off.
Attention:	Using this command, it is possible to enable the laser without asking for a password. **This is not allowed!!!** **This command must be used in conjunction with a password input box or an equivalent authorization scheme!!!**

A_SetShutterState

SC.A_SetShutterState(State As Boolean)

State:	**True** means **open**, **False close**.
Return:	---.
Function:	Controls the state of the shutter.
Attention:	If the shutter is locked, the command has no effect!

A_LockShutter

SC.A_LockShutter(State As Boolean)

State:	**True** means **locked**, **False unlocked**.
Return:	---.
Function:	After execution the shutter is closed and locked.

A_SetPilotState

SC.A_SetPilotState(State As Boolean)

State:	**True** means **ON**, **False OFF**.
Return:	---.
Function:	Controls the pilot laser, if installed in the laser.

A_GetBufferEmptyState

SC.A_GetBufferEmptyState() As Boolean

Parameter:	---.
Return:	The state of the internal buffer. **True** means buffer is empty.
Function:	Use this command to check if the marking is complete.

A_ReadyForNextSyncCmd

SC.A_ReadyForNextSyncCmd() As Boolean

Parameter:	---.
Return:	The command returns **True**, if there is space for at least one additional sync command.

A_Stop

SC.A_Stop() As Boolean

Parameter:	---.
Return:	**True**, if successful.
Function:	This command stops the running marking asynchronously.

A_GetGeneralInputDigital

SC.A_GetGeneralInputDigital(BitNo As long) As Boolean

BitNo:	Number of the bit to read.
Return:	State of the bit.
Function:	Reads the state of the digital inputs.
Attention:	This command depends on hardware which must be installed in your system in order to use this command.

A_SetGeneralOutputDigital

SC.A_SetGeneralOutputDigital(BitNo As long)

BitNo:	Number of the bit to modify.
Return:	---.
Function:	After execution of this command the addressed bit is set to **true**.
Attention:	This command depends on hardware which must be installed in your system in order to use this command.

A_ResGeneralOutputDigital

SC.A_ResGeneralOutputDigital(BitNo As long)

BitNo:	Number of the bit to modify.
Return:	---.
Function:	After execution of this command the addressed bit is set to **false**.
Attention:	This command depends on hardware which must be installed in your system in order to use this command.

A_ClrGeneralOutputDigital

SC.A_ClrGeneralOutputDigital()

Parameter:	---.
Return:	---.
Function:	After execution of this command all of the digital output bits are set to **false**.
Attention:	This command depends on hardware which must be installed in your system in order to use this command.

A_GetGeneralOutputDigital

SC.A_GetGeneralOutputDigital(BitNo As Long) As Boolean

BitNo:	Number of the bit to read.
Return:	The state of the addressed bit.
Function:	Reads the state of the digital outputs.
Attention:	This command depends on hardware which must be installed in your system in order to use this command.

A_GetPower

SC.A_GetPower() As Double

Parameter:	---.
Return:	The current laser power in use (0,0 %...100,0 %).

S_Power

SC.S_Power(Power As Double, Wait As Long)

Power:	Laser power in % (0,0...100,0).
Wait:	Time used to settle the new power. The given time is for a 100 % jump, so normally the real used time will be shorter.
Return:	---.
Function:	Change the marking power.

S_Speed

SC.S_Speed(Speed As Double)

Speed:	Marking speed in mm/s.
Return:	---.
Function:	Change the marking speed.

S_QSF

SC.S_QSF(QSF As Long)

QSF:	The laser pulse frequency.
Return:	---.
Function:	Change the pulse frequency.

S_QSF_PW

SC.S_QSF_PW(QSF_PW As Long)

QSF_PW:	The pulse frequency pulse width in µs.
Return:	---.
Function:	Change the pulse width.

S_SetStartStopDelay

SC.S_SetStartStopDelay(StartDelay As Long, StopDelay As Long)

StartDelay:	The time in µs between the start of the mirror movement and the first laser pulse.
StopDelay:	The time in µs after the mirror movement has stopped to the last laser pulse.
Return:	---.
Function:	Changes the start and stop delay values.

S_Pos

SC.S_Pos(X As Long, Y As Long)

X, Y:	Position in µm, to move the laser.
Return:	---.
Function:	The laser (pilot laser) is positioned to the given coordinate.

P_SetRotation

SC.P_SetRotation(X As Long, Y As Long, Angle As Double)

X, Y:	The rotation 0 point in µm.
Angle:	The rotation angle, positive values results in CCW rotation.
Return:	---.
Attention:	This is a global command.

P_SetSize

SC.P_SetSize(X As Long, Y As Long, SizeX As Double, SizeY As Double)

X, Y:	The size 0 point in µm.
SizeX/Y:	Size factor. (normal = 1.0).
Return:	---.
Attention:	This is a global function.

P_SetMove

SC.P_SetMove(XOffset As Long, YOffset As Long)

X, Y:	Move offset in µm.
Return:	---.
Attention:	This is a global function.

P_SetClip

SC.P_SetClip(Xmin As Long, Xmax As Long, Ymin As Long, Ymax As Long)

Xmin:	Minimum X position (left).
Xmax:	Maximum X position (right).
Ymin:	Minimum Y position (bottom).
Ymax:	Maximum Y position (top).
Return:	---.
Function:	Use this command to define a clipping rectangle. Only vectors inside the clip rectangle are output by the laser. To release the clipping function set all values to 0.
Attention:	This is a global function.

P_SetMirror

SC.P_SetMirror(X As Long, Y As Long, MirrorX As Boolean, MirrorY As Boolean)

X, Y:	Defined the 0 point of the virtual coordinate system in µm used to mirror the vectors.
MirrorX:	If true, the X axis of the virtual coordinate system is used to mirror the vectors.
MirrorY:	If true, the Y axis of the virtual coordinate system is used to mirror the vectors.
Return:	---.
Attention:	This is a global function.

4.4.5 HighlevelGraphics extensions (HG.)

InitVectorArray

HG.InitVectorArray(InitCount As Long, IncCount As Long) As Boolean

InitCount:	The possible count of vectors right after command execution (10000).
IncCount:	If the capacity in the array is too low to hold all the vectors the array size is increased by **IncCount** (10000).
Return:	**True**, if the array is initialised successfully.
Attention:	You must use **FreeVectorArray** after you are finished with the vector array!

FreeVectorArray

HG.FreeVectorArray()

Parameter:	---.
Return:	---.
Function:	Releases the vector array. To every **InitVectorArray** you must use an **FreeVectorArray**!

SendToSC

HG.SendToSC()

Parameter:	---.
Return:	---.
Function:	The vector array is sent to the laser for processing. After this command you normally use **FreeVectorArray** to release the used memory.

GetBoundingBox

HG.GetBoundingBox(ByRef Xmin As Long, ByRef Ymin As Long, ByRef Xmax As Long, ByRef Ymax As Long)

Xmin:	After execution Xmin contains the bounding box minimum X value in µm.
Ymin:	After execution Ymin contains the bounding box minimum Y value in µm.
Xmax:	After execution Xmax contains the bounding box maximum X value in µm.
Ymax:	After execution Ymax contains the bounding box maximum Y value in µm.
Return:	---.
Function:	Returns the current graphic bounding box.

GetBoundingBoxLast

HG.GetBoundingBoxLast(ByRef Xmin As Long, ByRef Ymin As Long, ByRef Xmax As Long, ByRef Ymax As Long)

Xmin:	After execution Xmin contains the bounding box minimum X value in µm.
Ymin:	After execution Ymin contains the bounding box minimum Y value in µm.
Xmax:	After execution Xmax contains the bounding box maximum X value in µm.
Ymax:	After execution Ymax contains the bounding box maximum Y value in µm.
Return:	---.
Function:	Returns the bounding box of the last added graphic.

Vector

HG.Vector(X1 As Long, Y1 As Long, X2 As Long, Y2 As Long)

X1, Y1:	Start point in µm of the vector.
Y1, Y2:	End point in µm of the vector.
Return:	---.
Function:	Adds a vector to the vector array.

Ellipse

HG.Ellipse(BaseRef As Integer, X As Long, Y As Long, RadiusX As Long, RadiusY As Long, AlphaStart As Double, AlphaEnd As Double)

BaseRef, X, Y:	BaseRef in µm, see also BaseRef on page 131.
RadiusX, Y:	Radius in µm along the X and Y axis.
AlphaStart:	Start angle of the ellipse/circle.
AlphaEnd:	End angle.
Return:	---.
Function:	Adds an ellipse/circle to the vector array.

Rectangle

HG.Rectangle(BaseRef As Integer, X As Long, Y As Long, Width As Long, Height As Long, RadiusX As Long, RadiusY As Long)

BaseRef, X, Y:	BaseRef in µm, see also BaseRef on page 131.
Width:	The width of the rectangle in µm.
Height:	The height of the rectangle in µm.
RadiusX, Y:	Radius in µm along the X and Y axis of the corner radius.
Return:	---.
Function:	Adds a rectangle to the vector array.

Text

HG.Text(BaseRef As Integer, X As Long, Y As Long, Text As String, FontName As String, Height As Long, FontWeight As Long, Italic As Boolean, LineDistance As Long, SpaceAdjust As Double, Quality As Integer, CodePage As Long)

BaseRef, X, Y:	BaseRef in µm, see also BaseRef on page 131.
Text:	The text to mark.
FontName:	The name of the font.
Height:	The height of the text in µm.
FontWeight:	100 = small, 400 = normal, 800 = bold.
Italic:	If **True**, than the text is output in *italic shape*.
LineDistance:	Distance from one line to the next in µm.
SpaceAdjust:	Factor to modify the inter char distance, normally 1.0.
Quality:	0 = bad, 1 = good, 2 = very good.
CodePage:	The used code page to get access to special characters, normally 0.
Return:	---.
Function:	Added text to the vector array.

TextInfo

HG.TextInfo(ByRef NextPosX As Long, ByRef NextPosY As Long)

NextPosX, Y:	This variable returns the next start point for additional text output.
Return:	---.

Barcode

HG.Barcode(BaseRef As Integer, X As Long, Y As Long, BarcodeType As Long, Data As String, AddOn As String, HeightMain As Long, HeightAddOn As Long, HeightLong As Long, ModuleWidth As Long, FillDistance As Long, Inverse As Boolean)

BaseRef, X, Y:	BaseRef in µm, see also BaseRef on page 131.
BarcodeType:	Type number of the barcode see Barcode specification on page 134.
ModuleWidth:	Width of the smallest bar in µm.
Data:	The content of the barcode. See also Barcode specification on page 134.
AddOn:	The contents of the add on area if available.
HeightMain:	The height of the barcode, see Barcode specification on page 134.
HeightAddOn:	The height of the add on area, see Barcode specification on page 134.
HeightLong:	The height of the long bars, see Barcode specification on page 134.
FillDistance:	The fill line distance in µm.
Inverse:	If **True**, then the output is inversed (light on dark).
Return:	---.
Function:	Adds a barcode to the vector array.

Datamatrix

HG.Datamatrix(BaseRef As Integer, X As Long, Y As Long, Data As String, Rows As Long, Cols As Long, ECCType As Long, Style As Long, Format As Long, Border As Long, ModuleWidth As Long, FillDistance As Long)

BaseRef, X, Y:	BaseRef in µm, see also BaseRef on page 131.
Data:	The content of the barcode.
Rows:	Specifies the row count, 0 for automatic.
Cols:	Specifies the column count, 0 for automatic.
ECCType:	For ECC200 = 26.
Style:	0 = normal, 1 = mirror mode.
Format:	Normally 6.
Border:	Normally 1.
ModuleWidth:	Width of the smallest dot in µm.
FillDistance:	The fill line distance in µm.
Return:	---.
Function:	Adds a datamatrix barcode to the vector array.

PDF417

HG.PDF417(BaseRef As Integer, X As Long, Y As Long, Data As String, Rows As Long, Cols As Long, ECCLevel As Long, Style As Long, ModuleWidth As Long, FillDistance As Long, Inverse As Boolean)

BaseRef, X, Y:	BaseRef in µm, see also BaseRef on page 131.
Data:	The content of the barcode.
Rows:	Specifies the row count, 0 for automatic.
Cols:	Specifies the column count, 0 for automatic.
ECCLevel:	0...8.
Style:	0 = normal, 1 = truncated, 0x0004...0x2004 = MicroPDF.
ModuleWidth:	Width of the smallest dot in µm.
FillDistance:	The fill line distance in µm.
Inverse:	If **True**, then the output is inversed (light on dark).
Return:	---.
Function:	Adds a PDF417 barcode to the vector array.
Attention:	An additional DLL is required to use this barcode.

HPGL

HG.HPGL(BaseRef As Integer, X As Long, Y As Long, FileName As String, PenFlag As Long, HPGL_Inc As Double)

BaseRef, X, Y:	BaseRef in µm, see also BaseRef on page 131.
FileName:	The HPGL file name.
PenFlag:	This is a bit field, every bit can be used to prevent the output of the corresponding pen. 0 = all pens are output.
HPGL_Inc:	The value that represents one HPGL unit in µm. Normally 25.0 sometimes 25.4.
Function:	Adds a HPGL drawing to the vector array.

Draw

HG.Draw(BaseRef As Integer, X As Long, Y As Long, ByRef VectorArray)

BaseRef, X, Y:	BaseRef in µm, see also BaseRef on page 131.
VectorArray:	Pointer to the vector array to output.
Function:	Adds a vector array to the vector array.

Fill

HG.Fill(Mode As Long, ParameterArray As HG_FillParameter)

Mode:	Bit field: Bit 0 = false: Left original vectors in vector array. Bit 0 = true: Delete original vectors, so only the fill is left in the vector array. Bit 1 = false: Normal fill. Bit 1 = true: Bidirectional filling.
ParameterArray:	Array of fill parameter. The array consist of: FillDistance As Long Reserve As Long FillAngle As Double.
Function:	Fills all closed polygons inside the vector array.

Move

HG.Move(X As Long, Y As Long)

X, Y:	Moved the current vectors by the given value.
Function:	Moved all the vectors currently in the vector array.

Size

HG.Size(X As Long, Y As Long, SizeX As Double, SizeY As double)

X, Y:	Defines the 0 point in µm.
SizeX, Y:	Size factor, 1.0 results in no change.
Function:	Changes the size of the current drawing inside the vector array.

Rotation

HG.Rotation(X As Long, Y As Long, Angle As Double)

X, Y:	Defines the 0 point in µm.
Angle:	The rotation angle. A positive value result in a CCW rotation.
Function:	Rotates the current drawing inside the vector array around the given point.

Mirror

HG.Mirror(X As Long, Y As Long, MirrorX As Boolean, MirrorY As Boolean)

X, Y:	Defines the 0 point of the virtual coordinate system in µm used to mirror the vectors.
MirrorX:	If true, the X axis of the virtual coordinate system is used to mirror the vectors.
MirrorY:	If true, the Y axis of the virtual coordinate system is used to mirror the vectors.

Polar

HG.Polar(X As Long, Y As Long, Radius As Long, MaxLength As Long)

X, Y:	Defines the 0 point of the virtual circle in µm.
Radius:	Defines the radius of the virtual circle in µm.
MaxLength:	Defines the maximum vector length allowed to left unbend. Useful value: 1000 µm.
Function:	All the vectors are bent using the defined virtual circle. The X axis of the normal coordinate system is bent to the given circle and than moved to the given 0 point.
Example:	You have a text along the X axis centred at the Y axis. Using polar you can generate a circle shape text, so you can laser the text onto a round piece of work.

Clip

HG.Clip(Xmin As Long, Xmax As Long, Ymin As Long, Ymax As Long) As Long

Xmin:	Minimum X position (left).
Xmax:	Maximum X position (right).
Ymin:	Minimum Y position (bottom).
Ymax:	Maximum Y position (top).
Return:	Count of the vectors left.
Function:	Use this command to define a clipping rectangle. Only vectors inside the clip rectangle are output by the laser. To release the clipping function set all values to 0.
Attention:	This command is normally used inside the **LC_ElementBeforeLaser** call back procedure. See also LC_ElementBeforeLaser(...) on page 109.

4.5 References

4.5.1 BaseRef

BaseRef defined the position used for the basis point of each graphical object:

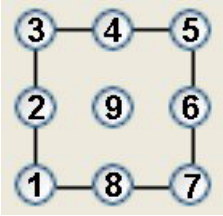


Fig. 104 BaseRef

Example:

If you have the mid point coordinate of a circle, than you set **BaseRef** to 9.
Using left justified text than you like to use the 1.
Use 8 if you like to have the text centred, etc.

4.5.2 Format specifications

Format strings and the results:

Writing	Result	Comment
Time format		
#Time#	18:40:55	PC internal format
#Time:#	18:40:55	
#Hour#	18	
#Minute#	40	
#Second#	55	
Date format		
#Date#	12/24/04	PC internal format
#DateL#	24.12.2004	
#DateS#	24.12.04	
#YearL#	2004	
#YearS#	04	
#YearD#	358	
#Month#	12	
#Week#	51	
#Day#	24	
Serial number	**First output**	**Sequence**
#SN=12#	12	12, 13, 14, 15, ...
#SN=12,2#	12	12, 14, 16, 18, ...
#SN=12,2,16,12#	12	12, 14, 16, 12, 14, ...
#SN=0,1,10,0,2#	0	0, 0, 1, 1, 2, 2, 3, ...

Writing	Result	Comment
#SND4=12#	0012	0012, 0013, 0014, ...
#SND2=2,-1#	02	02, 01, 00, -01, ...
#SNX4=9#	0009	0009, 000A, 000B, ...
#SNx3=8,2#	008	008, 00a, 00c, ...

All the time and date formats use the time/date at mark time.
Using the serial numbers the characters after the **#SN** have the following meaning:
D or **d** the number is shown in decimal notation.
X the number is shown in hexadecimal notation (0,1,..,9,A,B,C,D,E,F,10,...).
x as **X** but using lower case letters (9,a,b,...).
If the letter is followed by a number, than the resulting serial number is preceded by 0, building a number with the giving character count.
The numbers behind the **=** have the following meaning:
1st number: Current serial number.
2nd number: Step width (optional).
3rd number: Max count (optional).
4th number: Value after the maximum value has used.
5th number: How many identical number should be generated (optional).

Example using format strings:

Writing	Result
Date: #DateL#	Date: 24.12.2004
Week #Week# in the year #YearS#	Week 51 in the year 04
Time: #Time:#	Time: 18:40:55
Hour: #Hour# Minute: #Minute#	Hour: 18 Minute: 55

4.5.3 Special format specifications

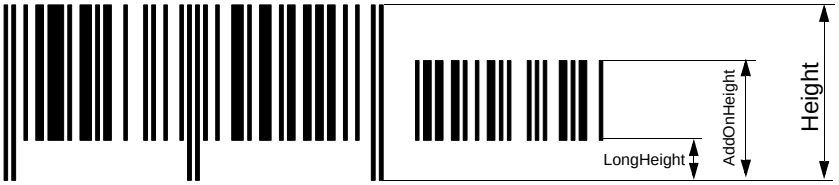
Side by side with the normal (time and date) format strings you can use special format strings.
These special format string are send to your program to get the result. See LC_Formatter(...) on page 109.
You write special format strings like the following:

#Value1=123,456#

As the normal format strings the special begin and ends also with the # char.
In front of the **=** you define the name of the format (Value1).
after the **=** you can write an optional default value (123,456) used every time your program is not running, or there is no **LC_Formmatter(...)** call-back defined.

4.5.4 Barcode specification

Definition of the barcode heights:



Characteristics:

No.	Barcode type	Numeric	Character	Length
1	2 of 5	Yes	No	Variable
2	Interleaved 2 of 5	Yes	No	Variable
3	Code 39	Yes	Uppercase	Variable
4	Code 93	Yes	Uppercase	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	Uppercase	Variable
11	EAN-128 B	Yes	Yes	Variable
12	EAN-128 C	Yes	No	Equal count
13	POSTNET(1.20)	Yes	No	Variable
14	Code-128 A	Yes	Uppercase	Variable
15	Code-128 B	Yes	Yes	Variable
16	Code-128 C	Yes	No	Equal count
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

5 Index

D

E

F

G

H

I

J

L

Q

R

S

T

U

V

W

Z

Index

ACI Laser GmbH
Österholzstraße 9
D-99428 Nohra
Fon: +49 3643 4152-0 e-mail: info@ACI-Laser.de
Fax: +49 3643 4152-77 Internet: www.ACI-Laser.de

Manual Laser Marking Software: V1.5 Build: 343