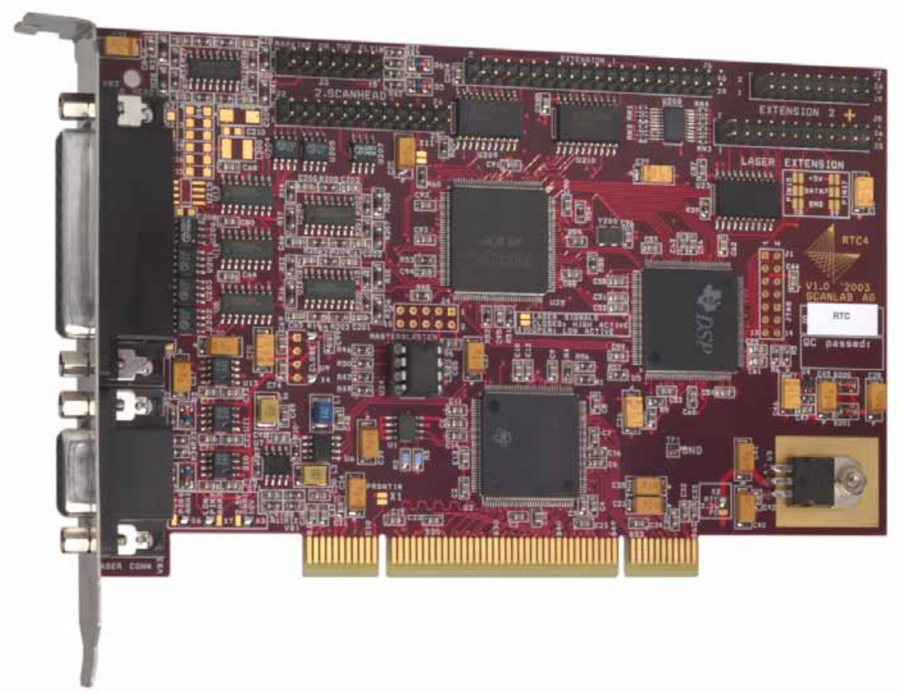
Installation and Operation

The RTC®4 PC Interface Board

for Real Time Control of Scan Heads and Lasers



SCANLAB AG
Siemensstr. 2a
82178 Puchheim
Germany

Tel. +49 (89) 800 746-0
Fax: +49 (89) 800 746-199

info@scanlab.de
www.scanlab.de

(Doc. Rev. 1.3 e - September 14, 2006)

Contents

1 Delivered Product

1.1 Package Contents

The contents of the package and the RTC®4 board's article number and configuration are listed in detail in an extra "Package Description" file.

- Check the package for damage and confirm that all parts have been delivered. If anything is missing, please contact SCANLAB.
- Keep the packaging, including the antistatic bag the RTC®4 board is delivered in, so that in case of repair the board can be properly repackaged and returned to SCANLAB.

1.2 Manufacturer

SCANLAB AG
Siemensstr. 2a
82178 Puchheim
Germany

Tel. +49 (89) 800 746-0
Fax: +49 (89) 800 746-199

info@scanlab.de
www.scanlab.de

1.3 About This Operating Manual

This operating manual is a part of the product. Please read these instructions carefully before you proceed with installation and operation of the RTC®4. If there are any questions regarding the contents of this manual, please contact SCANLAB (see previous section).

Keep the manual available for servicing, repairs and disposal. This manual should accompany the product if ownership changes hands.

This manual refers to the following versions of the RTC®4 software and firmware:

- DLL: `RTC4DLL.DLL`, version 400 or higher,
- DSP program files: `RTC4D2.HEX` / `RTC4D3.HEX`, version 2.400 / 3.400 or higher,
- RTC®4 firmware: version 128 or higher.

The following commands return the current software and firmware version numbers:

- **get_dll_version** (page 82);
- **get_hex_version** (page 83) and
- **get_rtc_version** (page 85)

Supplements To This Manual

The following RTC®4 supplementary manuals are available from SCANLAB:

- "3D Software" manual
- "Processing-On-The-Fly Software" manual
- "I/O Extension Board" manual
- "correXion and CFMP" manual

2 Product Overview

2.1 Intended Use

The RTC®4 PC Interface Board from SCANLAB is designed for real-time control of scan heads and lasers via an IBM-compatible PC with a PCI bus interface.

The RTC®4 is based on a fast digital signal processor (DSP) system providing full real-time control for a wide range of applications, including enhanced double scan head control and runtime image transformations.

The included software driver provides an extensive set of control commands. The RTC®4 stores and processes these commands completely independently of the host PC. This makes it possible to meet the stringent demands of real-time scan head and laser control even if the PC must simultaneously respond to other tasks such as machine control and network communication.

The RTC®4 offers various laser control signals for all commonly used laser types. The user can select from four different laser control modes. In addition, the output signals can be customized to meet special requirements.

The standard version of the RTC®4 includes a 16-bit digital input port, a 16-bit digital output port, an 8-bit digital output port and two general purpose analog output ports with 10-bit resolution.

The following options are available from SCANLAB:

- control signals for simultaneous control of two scan heads, with individual image field correction (on-board) for each scan head
- control signals for Processing-on-the-fly applications (on-board)
- control signals for a third axis, e.g. a vario*SCAN* (on-board)
- I/O Extension Board
- Optical Data Interface

Only hardware extensions from SCANLAB should be used in combination with the RTC®4.

2.2 System Requirements

Hardware

The RTC®4 requires an IBM-compatible personal computer with a PCI bus interface and at least one free PCI slot. The dimensions of the RTC®4 board are shown in figure 1 on page 8.

A second PCI slot is required if the optional I/O Extension Board is to be used.

SCANLAB offers additional PCI slot covers with D-SUB connectors for the following options:

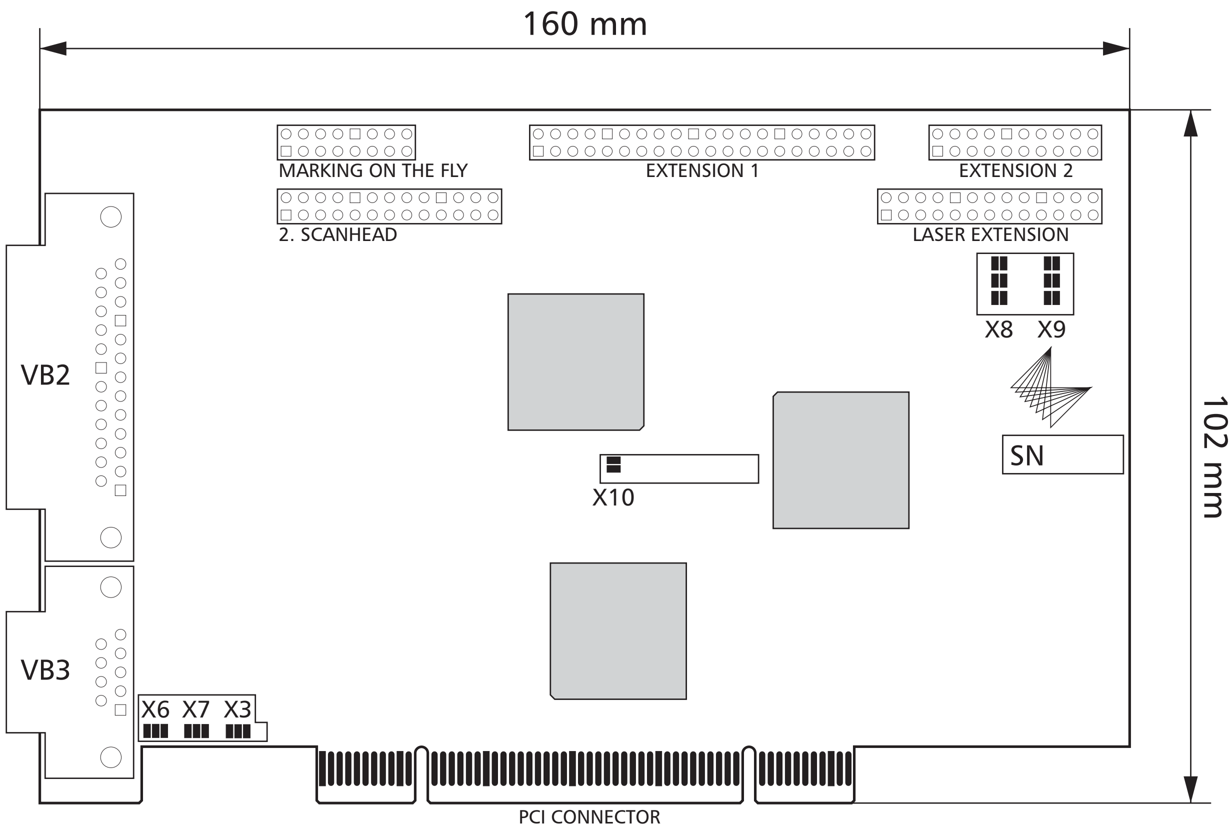
- second scan head
- Processing-on-the-fly
- laser extension connector with 8-bit digital output port

Software

Software drivers for the Microsoft operating systems WINDOWS XP and WINDOWS 2000 are included in the package. They support the RTC®4's plug and play facility and simultaneously drive up to eight RTC®4 boards.

The RTC®4 software package also includes discontinued software drivers for the Microsoft operating systems WINDOWS XP SP1 / 2000 / NT 4 and WINDOWS ME / 98 / 95. However, please note this driver and DLL version is no longer supported by SCANLAB. Functionality upgrades will be implemented in the new driver version only.

2.3 Board Dimensions And Layout



Legend

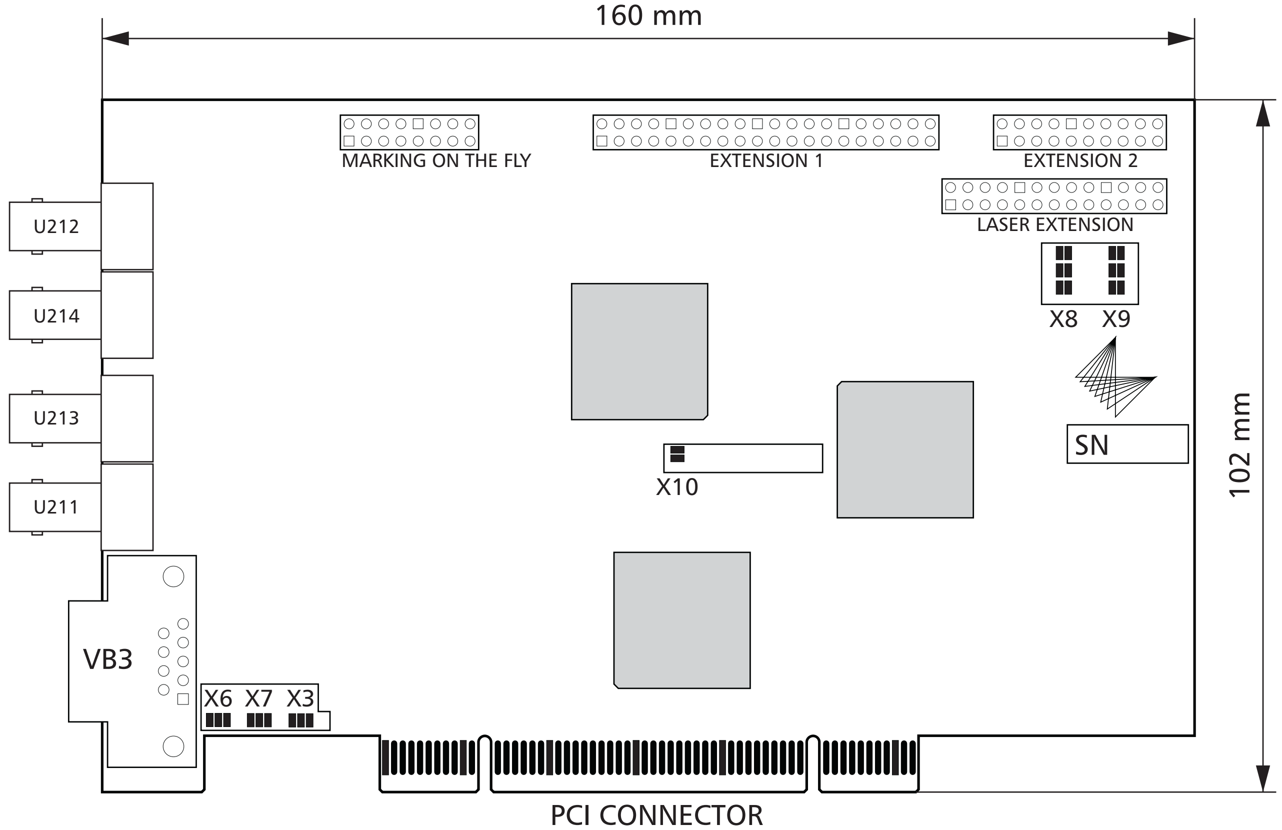
VB2	Primary scan head connector (D-SUB 25-pin female)
VB3	Laser connector and analog output ports (D-SUB 9-pin female)
MARKING ON THE FLY	Connector for Processing-on-the-fly applications (optional) – please refer to the supplement manual "Processing-On-The-Fly Software"
2. SCANHEAD	Secondary scan head connector (optional)
EXTENSION 1	Connector with a 16-bit digital input and a 16-bit digital output (optional: connector for the RTC®4 I/O Extension Board)
EXTENSION 2	Connectors for the RTC®4 I/O Extension Board (optional) – please refer to the supplement manual "I/O Extension Board"
LASER EXTENSION	Additional laser connector with 8-bit digital output port
X6/X7/X3	Jumper bank for laser connector VB3
X10	Jumper for setting the laser signal level
X8/X9	Jumper bank for the LASER EXTENSION connector
SN	Serial number

All connectors are described in section 6 on page 52.
The jumper settings are described in detail in section 8.1 on page 58.

1

Layout of the RTC®4 standard version

Optionally, the RTC®4 PC interface board can be delivered with an optical data interface.



Legend

U213/U211 . . . . Primary scan head connector (ST connectors)
U212/U214 . . . . Secondary scan head connector (ST connectors, optional)
VB3 . . . . Laser connector and analog output ports (D-SUB 9-pin female)
MARKING ON THE FLY . . . . Connector for Processing-on-the-fly applications (optional) – please refer to the supplement manual "Processing-On-The-Fly Software"
EXTENSION 1. . . . . Connector with a 16-bit digital input and a 16-bit digital output (optional: connector for the RTC®4 I/O Extension Board)
EXTENSION 2. . . . . Connectors for the RTC®4 I/O Extension Board (optional) – please refer to the supplement manual "I/O Extension Board"
LASER EXTENSION . . . . Additional laser connector with 8-bit digital output port
X6/X7/X3 . . . . Jumper bank for laser connector VB3
X10 . . . . Jumper for setting the laser signal level
X8/X9 . . . . Jumper bank for the LASER EXTENSION connector
SN . . . . Serial number

All connectors are described in section 6 on page 52.
The jumper settings are described in detail in section 8.1 on page 58.

Layout of the RTC®4 with optical data interface

2.4 Notes For RTC®3 Users

This section summarizes how the RTC®4's hardware and software differ from that of the RTC®3 PC Interface Board and provides the RTC®3 user with tips on installing an RTC®4.

Hardware Changes

- Standard RTC®4 boards are equipped with an EXTENSION 1 connector. This connector provides a 16-bit digital input and a 16-bit digital output (previously only available via an optional RTC®3 I/O Board) as well as access to the BUSY status signal (see "EXTENSION 1 Connector", page 56).
- The RTC®4 PC Interface Board is optionally available with an optical data interface (see "Optical Data Interface (Optional)", page 56). This eliminates the need for an add-on board to implement optical data transmission between the PC interface board and the scan system's (optional) integrated optical data interface.

Installation Tips

- **RTC®4 and RTC®3 boards in the same computer cannot be operated simultaneously.**
- RTC®4 and RTC®3 boards use the same Windows driver. Nevertheless, even if an RTC®3 board has already been installed in a computer, the setup program must still be called again. On the other hand, after an RTC®4 is installed, an RTC®3 board can be operated without newly installing the driver.

Also follow the guidance in "Installation And Start-Up", page 58.

Changes in the Command Set

New Commands

- The new Arc Commands (see page 15) support marking with basic arcs.
- Commands for Status Monitoring and Diagnostics (see page 38) are suitable for both monitoring the operational status and diagnosing the scan system.
- The 16-bit digital input and 16-bit digital output at the EXTENSION 1 connector can be queried or set via I/O commands (see "16-Bit Digital Input and Output", page 56). Some of these I/O commands can execute conditional jumps dependent on the current value of the digital input, thereby expanding support for structured programming (see "Conditional List Jumps", page 45). These commands were previously only available with the optional RTC®3 I/O Board.

Changed Commands

The RTC®3 command **rtc3_count_cards** contains the designation "rtc3". The RTC®4 version of the command is called **rtc4_count_cards** (see page 102).

Unsupported Commands

A group of RTC®2 commands still usable on the RTC®3 are not supported by the RTC®4. However, these unsupported commands can be replaced by equivalent RTC®4 commands.

The section Supported and Unsupported RTC®2 Commands (see page 126) lists all emulated RTC®2 software commands, unsupported RTC®2 commands and RTC®4 equivalents.

3 Safety During Installation And Operation

Please read these operating instructions completely before you proceed with installing and operating the RTC®4 PC Interface Board.

If there are any questions regarding the contents of this manual, please contact SCANLAB.

The following symbols are used in this manual:

Instructions that may affect a person's health are marked with a warning triangle next to the word "**Danger**".

Instructions that recommend appropriate use of this device or warn of damage that may occur to it are labeled by a circle with an "X" through it, next to the word "**Caution**".

3.1 Steps For Safe Operation

Caution!

- Carefully check your application program before running it. Programming errors can cause a break down of the system. In this case neither the laser nor the scan head can be controlled.
- Protect the board from humidity, dust, corrosive vapors and mechanical stress.
- For storage and operation, avoid electromagnetic fields and static electricity. These can damage the electronics on the RTC®4 board. For storage, always use the antistatic bag the RTC®4 is delivered in.
- The allowed operating temperature range is 25 °C ± 10 °C.
- The storage temperature should be between −20 °C and +60 °C.

3.2 Laser Safety

The RTC®4 is intended for controlling a laser scan system. Therefore all relevant laser safety directives must be known and applied before installation and operation. The customer is solely responsible for ensuring the laser safety of the entire system.

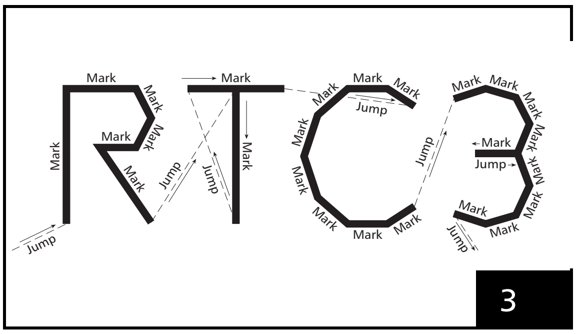
Danger!

- All applicable laser safety directives must be adhered to. Safety regulations may differ from country to country. It is the responsibility of the customer to comply with all local regulations.
- Please observe all laser safety instructions as described in your scan head or scan module manual, chapter 3 "Safety during Installation and Operation".
- **Always turn on the PC and the power supply for the scan head first before turning on the laser. Otherwise there is the danger of uncontrolled deflection of the laser beam.** SCANLAB recommends the use of a shutter to prevent uncontrolled emission of laser radiation.

4 Principle Of Operation

4.1 Software Concept

Figure 3 shows a simple laser marking sample[1]. The image is made up of straight line segments or vectors. The RTC®4 software driver provides a set of jump commands and mark commands for drawing such vector images. Each of these commands describes one vector. The RTC®4 software driver provides arc commands for producing circular arcs. Additional software commands are available for controlling the laser during the marking process.



A laser marking sample

The RTC®4 processes the commands it receives and precisely transmits the required marking signals to the scan head using a pre-defined 10 µs time raster and to the laser. The scan head's galvanometer scanners accurately position their deflection mirrors in synchronization with the incoming control signals. The control signals are transferred to the galvanometer scanners digitally in accordance with the industry standard XY2-100 (or for optical data transfer in accordance with the XY2-100-O protocol).

Current scan head status information can be queried via the RTC®4, also in accordance with the XY2-100 (or XY2-100-O) protocol.

The RTC®4 provides various analog and digital signal outputs freely available for tailoring laser control to customer-specific requirements. The customer assumes responsibility for use of these signals.

List Commands And Control Commands

The RTC®4 command set consists of *control commands* and *list commands*.

Control commands are executed immediately. They are used for controlling execution of lists and for setting some general parameters. Other control commands are provided for direct laser and scan head control.

Before a **list command** can be sent to the RTC®4, a list must be opened via a control command. List commands sent to the RTC®4 afterwards are not executed immediately, but stored in a list buffer. Only after the list is closed and started, the RTC®4 reads the commands from the list buffer and processes them in real time. The RTC®4 provides two list buffers. Each list buffer can hold one list ("list 1" or "list 2") with up to 4000 list commands. If an application needs only one list, then the RTC®4's entire memory can be treated like a single list with a capacity for 8000 commands. In this case, list 1 can be loaded with up to 8000 commands (but list 2 must not be used). This also makes possible the use of structured programming.

List commands include jump commands, mark commands and arc commands, as well as commands for setting various scanning parameters such as laser power, jump speed and marking speed.

Some commands exist in two versions: as a list command and as a control command. Among these dual-version commands are the I/O commands.

All RTC®4 commands are described in detail in chapter 10 "Commands And Functions". An overview is provided in chapter 10.1, page 69.

(1) In this manual, laser marking is mentioned only as an example of the many possible laser materials processing applications.

List Handling

The command **set_start_list** (see page 117) opens a list buffer for writing. After finishing the data input, the list must be closed with the command **set_end_of_list**.

As soon as a list is closed, it can be started either with the command **execute_list** (see page 82) or by an external start signal (see the section "External Control Inputs", right).

During the execution of one list, the other list buffer can be loaded with the next list. The host computer and the RTC®4 then work in parallel. The second list can only be started after execution of the first list has finished. During execution of a list, **execute_list** commands and external start signals are ignored.

Execution of a list can be stopped at any time, e.g. for implementing an emergency shutdown or for any other purpose. Use of the **stop_execution** command (see page 120) or an external stop signal will immediately abort the currently running list and turn off all RTC®4 laser signals.

To examine the current status of the lists, the commands **get_status** (page 86) or **read_status** (page 101) can be used.

Automatic List Handling

Continuous Transfer

The commands **auto_change** and **auto_change_pos** (see page 75) activate an automatic list change. That causes the next sequence of commands to start automatically when the current list is finished. In this manner, continuous data transfer to the scan head can be achieved without non-productive pauses.

The command **auto_change** can be called when working with two lists (each with a maximum capacity of 4000 commands). This command should only be called while a list is executing or after a list has finished. Additionally, the next list should have already been loaded and closed (by a call of the command **set_end_of_list**).

The **auto_change_pos** command can be called when the RTC®4's entire memory is to be treated like a single list. The start position (address in list memory) of the next command sequence is specified by the command.

Repeating Output

Alternatively, continuous data transfer can be achieved by alternately repeating output of the two lists:

- Load and close the two lists.
- Start one of the lists with the command **execute_list**.
- While the first list is running, call the command **start_loop** (see page 120). This causes a continuous, automatic and alternately repeating output of both lists from the RTC®4 to the scan head, until you choose to call the command **quit_loop**, which will terminate the continuous output as soon as the current list is finished.

Circular Queue

A completely different mode of operation for continuous data transfer is described in chapter 5.5 "Circular Queue Mode", page 44.

External Control Inputs

To also enable start and stop execution of a list via external signals, the RTC®4 provides two external control inputs, /START and /STOP (TTL active-low). These control inputs are accessible via the 9-pin D-SUB laser connector VB3 on the RTC®4 board – see figure 23 on page 52. Both signal inputs are internally connected to +5 V via pull-up resistors (10 kΩ).

Both inputs are active-LOW, i.e. the corresponding pin must be set to the LOW level (0 V) to start or to stop execution.

STOP Signal

If the /STOP input is at the LOW level for at least 10 µs, execution of the currently running list is aborted immediately and the RTC®4 laser signals are turned off. This is equivalent to calling the command **stop_execution** (see page 120).

The /STOP input is always enabled. It can, for instance, be used to implement an emergency shutdown.

START Signal

Before the /START input can be used, it has to be enabled with the command **set_control_mode** (see page 104). Subsequent START requests will start the loaded list. The list will only be started by the external START request if it is closed and no list is executing at the moment.

The desired list can be selected via the command **select_list** (page 103).
Alternatively, the list command **set_extstartpos_list** (see page 106) can be used.

To check whether a list was successfully started, the command **get_startstop_info** (see page 86) can be used.

Notes

- An explicit call of the command **stop_execution** disables the external /START input. An external /STOP signal also (at least temporarily) disables the external /start input, i.e. as long as the /STOP input is LOW. The command **set_control_mode** (see page 104) defines whether or not the /START input also stays disabled when the /STOP signal is no longer active.
- Please note that the /START input is *edge* sensitive (HIGH to LOW level *transition*), whereas the /STOP input is *level* sensitive.

Synchronization Of Processing

The command **set_wait** (see page 119) makes it possible to set *wait markers* (break points) within a list. Each marker is associated with a number greater than zero.

When the RTC®4 reaches a wait marker, processing of the list is temporarily interrupted and the laser is turned off.

The command **get_wait_status** (see page 89) ascertains whether processing is currently interrupted at a marker. If processing is interrupted, the command **get_wait_status** returns the number (wait_word) of the corresponding marker. Otherwise the command returns zero.

The wait markers are provided for synchronization purposes. The application program should perform a handling routine for each wait marker.

When that handling routine is finished, processing of the list can be resumed via the control command **release_wait** (see page 102).

4.2 Scan Head And Laser Control

Vector Commands

The basic commands for scan head control are the jump commands and the mark commands. These commands require as parameters the X and Y coordinates of the *end* point of the corresponding vector(1). Each vector starts at the *current output position*, which is usually the end point of the preceding vector. The initial output position at start-up (or after a reset of the RTC®4) is the center of the image field, i.e. the point (0|0).

Please refer to chapter 4.4 "Image Field Size", page 29 for a description of the image field coordinate system.

Jump Commands

A jump command (**jump_abs** or **jump_rel**) causes a (usually) fast movement of the scanner mirrors. The laser focus "jumps" to the new position. In general, the laser is switched off during the jump. The jump speed can be defined with the list command **set_jump_speed** (see page 108).

If the laser system does not allow fast switching, the jump speed must be set high enough to prevent a visible marking effect on the workpiece. Also see the command **home_position** (page 90).

Mark Commands

The RTC®4 automatically turns on the laser at the beginning of a mark command. During a mark command (**mark_abs** or **mark_rel**), the laser focus moves along the specified vector with a constant *marking speed*, producing a straight mark on the workpiece.

If another mark (or arc) command follows immediately afterward, the RTC®4 leaves the laser on. Therefore, a sequence of individual mark (and arc) commands creates a polyline mark. The laser is turned off after the last mark (or arc) command in a series of mark (or arc) commands, or if the end of the current list is reached.

The list command **set_mark_speed** (see page 110) defines the marking speed. The marking speed can be changed anywhere in a list.

(1) The coordinates must be specified as digital control values (without units). To avoid confusion with coordinates in [mm], SCANLAB uses the expression "coordinate values [in Bits]".

Arc Commands

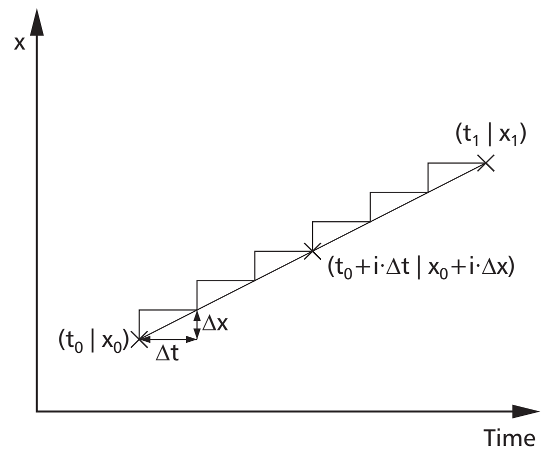
The RTC®4 software driver provides arc commands for marking circular arcs. These commands require parameters for the X and Y coordinates of the arc center and the arc angle. The arc starts at the current output position.

At the beginning of an arc command, the RTC®4 also automatically turns on the laser. During an arc command (**arc_abs** or **arc_rel**), the laser focus moves with the defined marking speed along the specified arc. The laser is turned off after the last arc (or mark) command in a series of arc (or mark) commands, or if the end of the current list is reached.

Microsteps

Each vector defined by a jump, mark or arc command is divided into a number of small steps by the RTC®4. These microsteps are transferred to the scan head at a constant time rate (*output period* Δt) In controlling its galvanometer scanners, the scan head implements the steps via an analog servo loop.

Figure 4 shows how the X component of a vector is divided into microsteps. The Y component is split up in the same way.



x_0 x coordinate of the current output position before scanning the vector

x_1 x coordinate of the end position of the vector

Δt step period

4

The X component of a vector is split up into microsteps.

The length Δs of each microstep is

$$\Delta s = v \cdot \Delta t,$$

where *v* is the current jump speed (marking speed).

The output period Δt of the position update is usually fixed at 10 µs. It is the same for 2D and for 3D applications. The output period cannot be set by the user.

The 16-bit data output width of the RTC®4 allows up to 2^{16} microsteps per vector or arc. If the marking speed is quite low and the vector or arc is very long, a step period of 10 µs would possibly lead to more than 2^{16} microsteps. In this case, the RTC®4 automatically increases the output period to 20 µs (or to a suitable multiple of 10 µs, if necessary).

Marking Time

The marking time consumed by any particular marking process can be measured by calling the command **save_and_restart_timer** (see page 102) before and after the marking process. This command saves the current value of the RTC®4's integrated timer and resets the timer value to 0. The measured time can be read via the command **get_time** (see page 87), which returns the timer value saved during the most recent call of **save_and_restart_timer**.

4.3 Delays

The timing of the scan head and laser control signals must be compatible with the dynamic behavior of the scan system, i.e. the response of the scanners and the laser, and the specific interaction between the workpiece and laser radiation.

To accomplish this, the user can set the following *delays*:

- Laser On delay
- Laser Off delay
- Jump delay (optional variable)
- Mark delay
- Polygon delay (optional variable)

All delays are described in detail in this chapter.

Laser Delays

- There are two different laser delays: *LaserOn delay* and *LaserOff delay*.
- The laser delays affect when the laser is turned on or off before or after a mark or arc command or a series of mark and arc commands. Laser delays do *not* affect the total marking time, except when they are negative.

The LaserOn delay and the LaserOff delay are set by the list command **set_laser_delays** (see page 108). The time resolution for the laser delays is **1 µs**.

The delays must be appropriate for the defined jump speed and marking speed (also see "Notes On Optimizing The Delays", page 24).

LaserOn Delay

The LaserOn delay defines the moment when the RTC®4 turns on the laser. *LaserOn delay* is automatically inserted at the start of a mark or arc command (first microstep). Thus, the laser is switched on only after execution of the first few microsteps. This delay can be used for several purposes:

- Many applications require laser marking without variations of intensity, especially without burn-in effects at the start or end of a vector. To achieve homogenous marking results, it is essential to scan the vectors with a constant velocity.

  At the beginning of a mark or arc vector, however, the mirrors first have to be accelerated up to the defined marking speed. Figure 7 on page 19 shows that the laser focus initially moves only very slowly. A burn-in effect may occur.

  To avoid this, the *LaserOn delay* must be set to a suitable, *positive* value. Thus the mirrors will have already reached a certain angular velocity when the laser eventually turns on. However, if the LaserOn delay is too long, the first part of the vector will be cut off.
- Some materials take some time until they react to the exposure to laser radiation. In this case, it can be useful to "preheat" the starting point of a mark or arc vector before marking. This can be done by setting the LaserOn delay to a *negative* value.

  A negative LaserOn delay extends the total marking time, because it is inserted *before* the actual mark or arc command.

LaserOff Delay

- Due to the acceleration phase at the beginning of the movement, a difference (*lag*) occurs between the set position and the real position of each mirror – see figure 7 on page 19.

  After execution of a mark or arc command, the laser should not be turned off immediately because the scanners have not yet reached the final set position. Therefore a *LaserOff delay* is inserted automatically before the laser is turned off.

Scanner Delays

- There are three different types of scanner delays: *jump delay*, *mark delay* and *polygon delay*. After each vector or arc command, the RTC®4 inserts one of these delays before the next command is started.

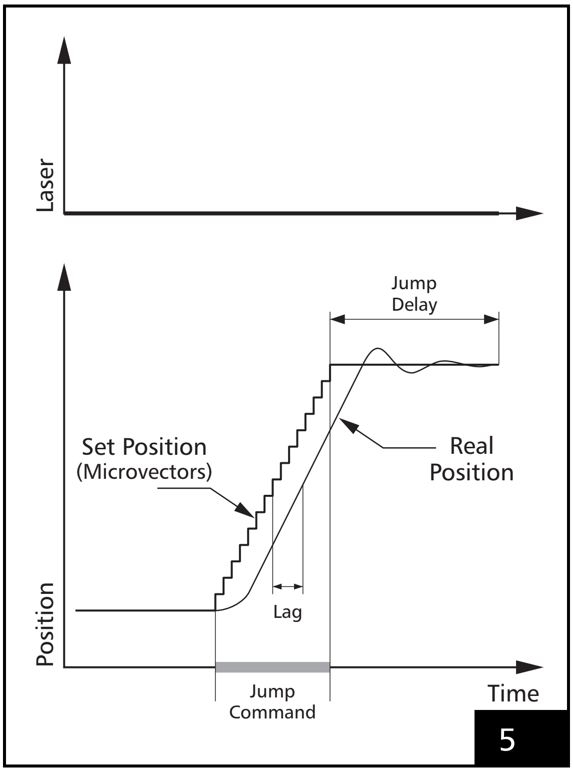
The command **set_scanner_delays** (see page 114) defines the scanner delays. The time resolution for the scanner delays is **10 µs**.

Jump Delay

When executing a jump command, the mirrors first have to be accelerated up to the defined jump speed. Because of the inertia of the mirrors, a *lag* occurs between the set position and the real position – see figure 5 on page 18.

At the end of the jump, a certain settling time is necessary for the mirrors to reach the set position within some accuracy. To allow for the settling time and for the lag, the RTC®4 inserts a *jump delay* after each jump command, before the next command is executed.

Note that the necessary settling time depends on the selected jump speed. A higher jump speed usually requires a longer jump delay. The total time needed for the entire jump command is the sum of the actual jump time and the jump delay. It can be minimized by optimizing the jump speed and the jump delay.



Scan head control timing during a jump command with a jump delay. The laser is not turned on.

Variable Jump Delay

During a jump vector, the laser focus (output position) usually moves with a constant linear velocity, the *jump speed*. Since the jump speed is the same for each jump vector, a constant *jump delay* is required for settling of the mirrors.

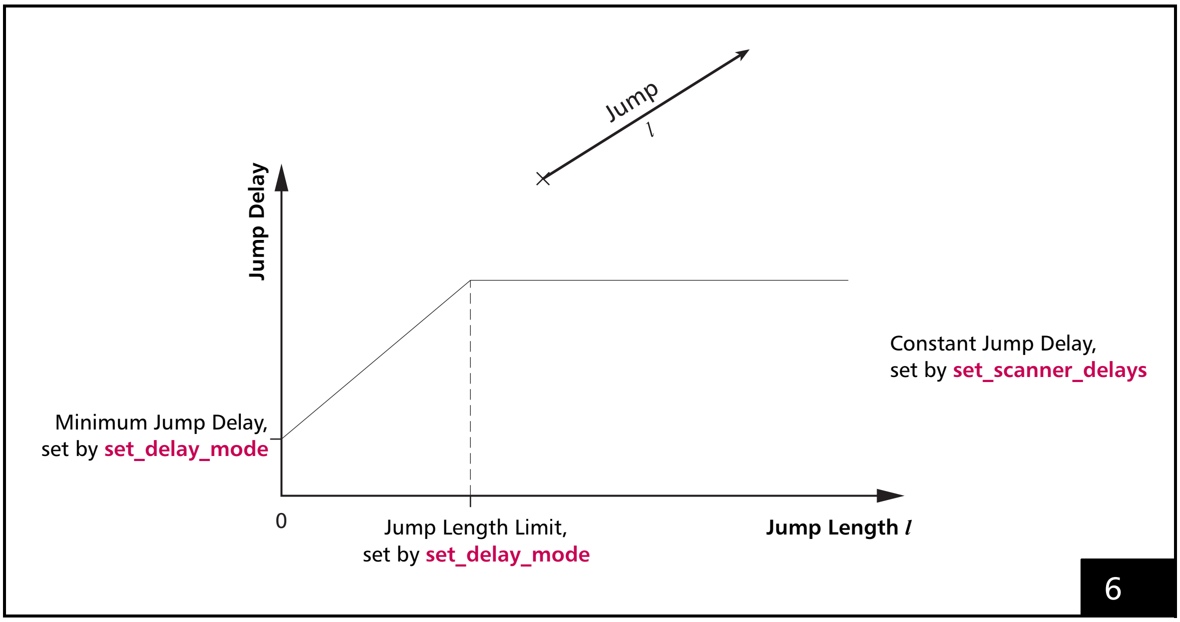
However, if a jump vector is very short, the scanners might not reach the full jump speed during the jump because of the inertia of the mirrors. In this case, a shorter jump delay might be sufficient for settling.

To make use of this, the RTC®4 offers a *variable jump delay mode*. In this mode, the jump delay for short jump vectors will be reduced in time, as shown in figure 6 on page 18. The minimum jump delay (for a jump vector of zero length) and the jump length limit are set by the user with the command **set_delay_mode** (see page 105).

When using the variable jump delay mode, total marking time can be reduced, especially in applications involving many short jumps.

Notes

- To turn *off* the variable jump delay mode, simply set the parameter `JumpLengthLimit` to zero.



Variable Jump Delay
Top: length *l* of the jump vector
Bottom: Variation of the jump delay

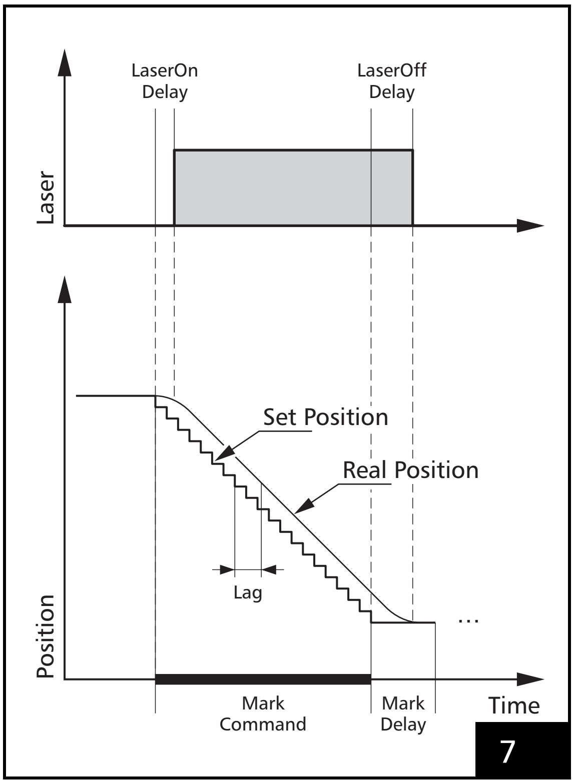
Mark Delay

Although the marking speed is usually lower than the jump speed, a lag between the set position and the real position occurs not only during a jump, but also during a mark or arc command.

To make sure that the scanners reach the final set position properly before the next command starts, the RTC®4 inserts a *mark delay* after a single mark or arc command or after the last mark or arc command of a polyline – also see figure 7 on page 19.

Notes

- If a mark or arc command is followed by a zero jump, mark or arc command, then the MarkDelay is *not* executed.
- If a jump vector is followed by a zero jump vector, then the first JumpDelay is not executed.
  In contrast, the JumpDelay *is* executed if the jump vector is followed by a zero mark or arc command.

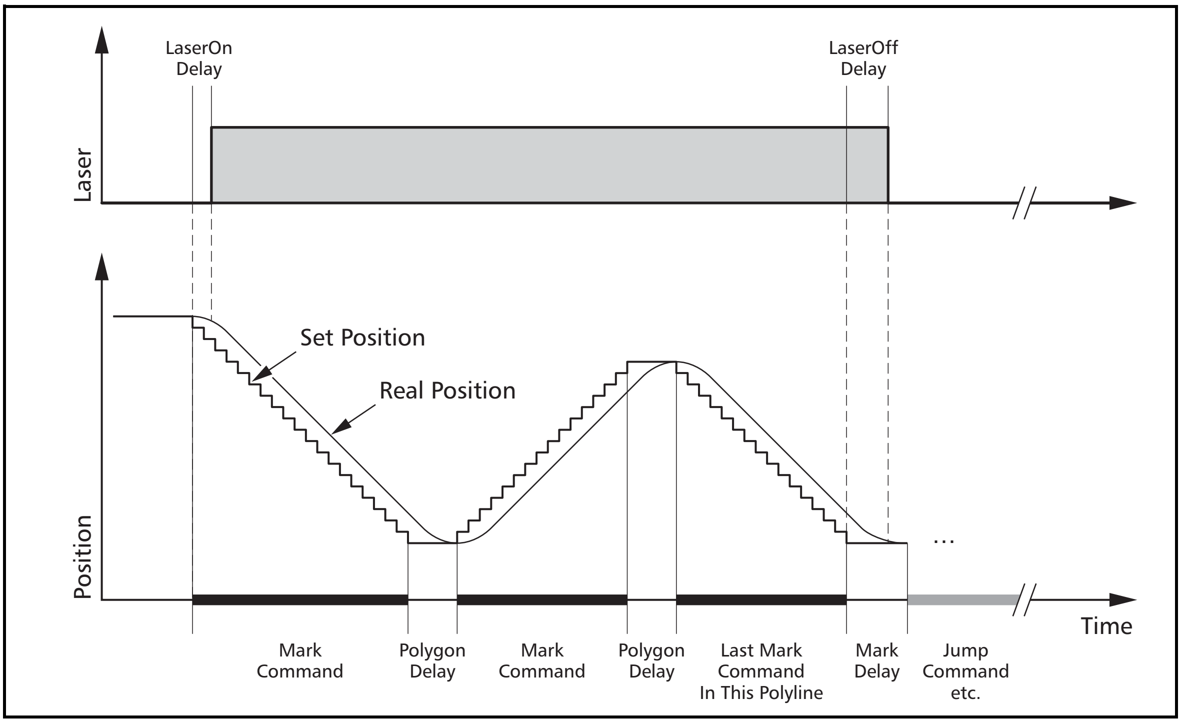


Scan head and laser control timing during a mark or arc command with a mark delay. Grey shaded areas indicate that the laser is on.

Polygon Delay

Between two *successive* mark or arc commands, there is no need for a complete stop of the scanners. Therefore the mark delay between two successive mark or arc commands is replaced by a *polygon delay* – see figure 8 on page 20.

The mark delay and the polygon delay can be set independently. In addition, the RTC®4 is able to vary the length of the polygon delay, depending on the angle between two marking vectors or the tangents of the arcs. See the section "Variable Polygon Delay" on page 21 for details.



Scan head and laser control timing during a polyline with a constant polygon delay

8

Variable Polygon Delay

A variable polygon delay mode can be activated via the command **set_delay_mode** (page 105). In this mode, the RTC®4 allows varying the length of the polygon delay, depending on the angle ϕ between the two successive marking vectors – see figure 9 on page 21, below.

For each corner of the polyline, the RTC®4 calculates the variable polygon delay $v\_delay(\phi)$ as follows:

$$v\_delay(\phi) = scale(\phi) \cdot polygon\_delay,$$

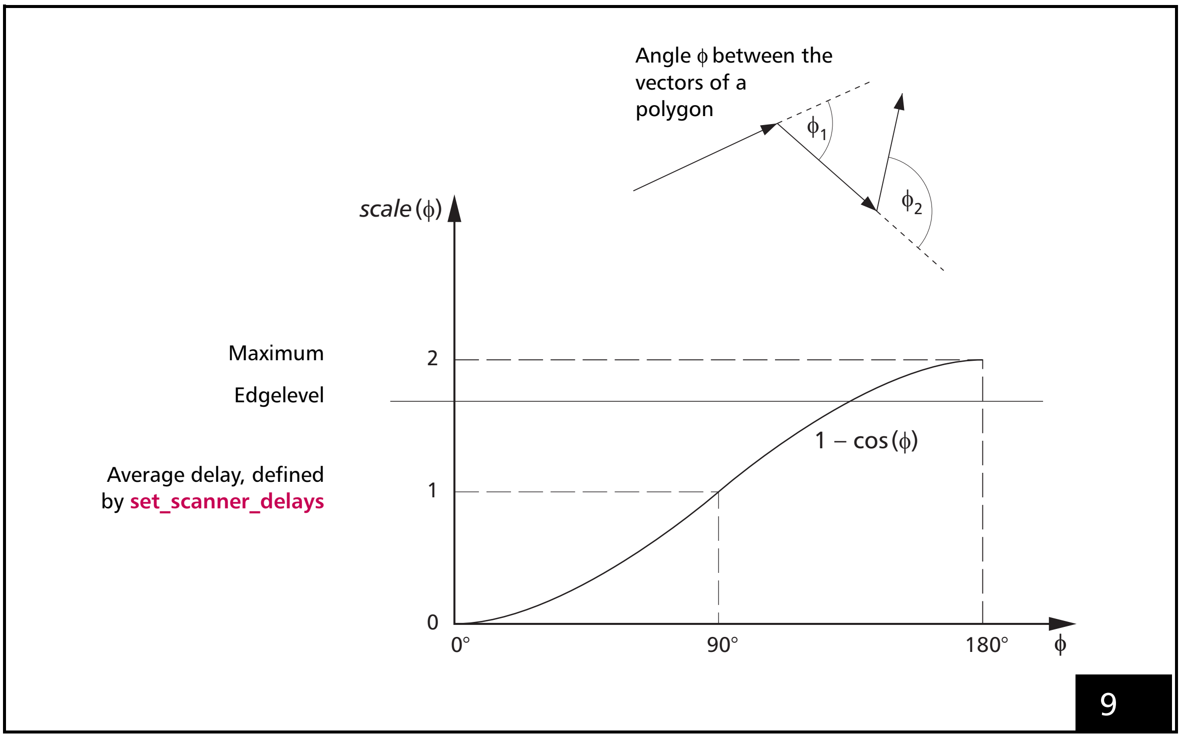
where $scale(\phi)$ is a scaling function ($0 \leq scale(\phi) \leq 2$). The parameter *polygon_delay* is set by the command **set_scanner_delays**.

Figure 9 (bottom) shows the default scaling function. This standard curve can be replaced by a customized curve. See "Customizing The Variable Polygon Delay" on page 23.

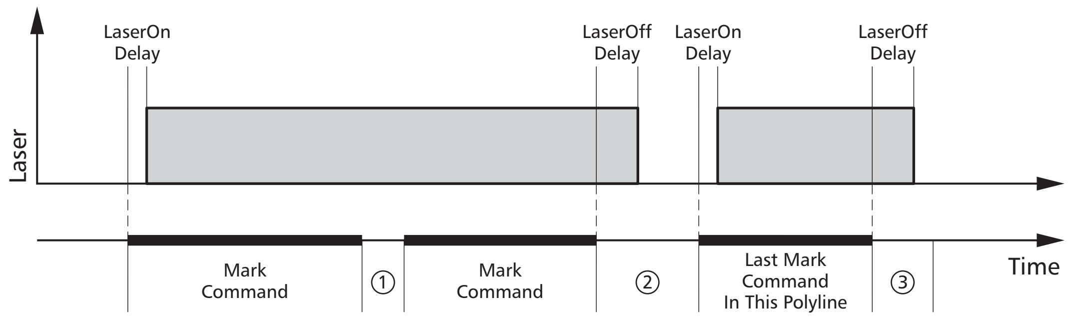
Edgelevel

Figure 9 shows that the variable polygon delay becomes quite long if the angle ϕ is close to 180°. This might lead to burn-in effects in the sharp corners of the polyline. To avoid this, the user can define a so-called *edgelevel*: If the polygon delay between two mark or arc commands is longer than or equal to this value, the RTC®4 turns the laser off after the first mark or arc command – after inserting a LaserOff delay – and starts a new polyline at the beginning of the next mark or arc command. Also see figure 10 on page 22.

For further details see **set_delay_mode**, page 105.



Variable Polygon Delay
Top: Definition of the angle ϕ
Bottom: Variation of the polygon delay (default curve)



Legend

① Variable polygon delay, shorter than the specified edgelevel.
② Variable polygon delay, longer than the edgelevel.
The laser is turned off between the two mark or arc commands.
③ Mark delay.

10

Laser control timing during a polyline with variable polygon delays.
An edgelevel was defined with the command **set_delay_mode**.

Customizing The Variable Polygon Delay

The command `load_varpolydelay(FileName, X)` loads a table for the scaling function *scale*(ϕ) from an ASCII text file (with the filename extension `*.STB`). Each STB file can contain one or more tables.
See **load_varpolydelay** (page 98) for details.

The table contains up to 50 data points (ϕ | *scale*(ϕ)) for various angles ϕ. The RTC®4 determines the scaling function *scale*(ϕ) from this data by linear interpolation.

Table Format

Each table starts with the instruction

```
[VarPolyTableX]
```

where X must be replaced by a positive integer which denotes the table number.

Each data point (ϕ | *scale*(ϕ)) is described by two instructions:

```
AngleY=....
ScaleY=....
```

where Y must be replaced by an integer ($1 \leq Y \leq 50$) which denotes the number of the data point.

The values for the angle ϕ (in degrees) and the scaling factor must be specified as floating point numbers. Use the period (`.`) as the decimal point.
The allowed ranges are:

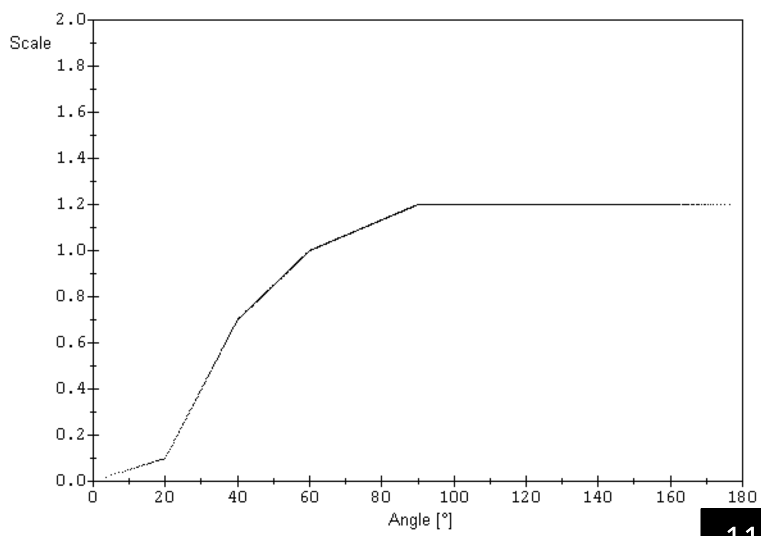
$$0° \leq \phi \leq 180° \text{ and } 0 \leq scale(\phi) \leq 2.$$

Notes

- Each instruction must be in a separate line.
- Empty lines are ignored.
- All characters following a semicolon (`;`) are ignored until the end of the line, i.e. the semicolon can be used for comments.
- The order of the data points in the table is of no importance.
- The angle ϕ = 0° means that two successive vectors are parallel and are marked in the same direction.
  If the table contains no explicit data for ϕ = 0°, the scaling factor *scale*(0°) = 0 is assumed.
- The angle ϕ = 180° means that two successive vectors are marked in the opposite direction.
  If the table contains no explicit data for ϕ = 180°, the scaling factor *scale*(180°) is set to the largest scaling factor found in the table.

Figure 11 shows a sample table and the corresponding scaling function.

```
; sample
[VarPolyTable1]
Angle1=20
Scale1=0.1
Angle2=40
Scale2=0.7
Angle3=60
Scale3=1.0
Angle4=90
Scale4=1.2
```



11

Sample table and resulting scaling function *scale*(ϕ).
The sample table contains four data points.

Notes On Optimizing The Delays

The delays have to be set with the commands **set_scanner_delays** and **set_laser_delays**. The delays have to be appropriate for the defined jump speed and the marking speed. If the delays are not optimized, the quality of the scanning results will be reduced and scanning time will be extended.

The figures on page 26 through page 28 show the various effects of non-optimized delays on the letters "RTC".

The lengths of the LaserOn delay and the LaserOff delay have no influence on the total scanning time if positive values are chosen.

The LaserOn delay and the LaserOff delay should be optimized first, followed by the delays for scanner control, i.e. the jump delay, the mark delay and the polygon delay.

When optimizing the laser delays, it is useful to set the jump delay and the mark delay to long values.

Limits For The Delays

When setting the delays, please observe the following:

(1) The LaserOff delay must be longer than the LaserOn delay. Otherwise faults in the laser control may occur.

LOffD > LOnD

(2) The mark delay must be longer than the *difference* between the LaserOff delay and the LaserOn delay.

markD > LOffD – LOnD

The same applies to the edgelevel of the variable polygon delay:

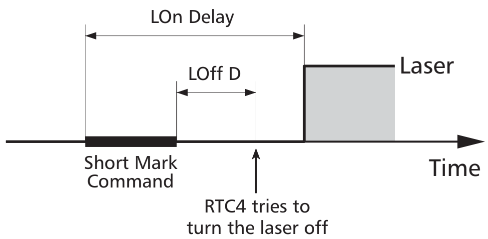
edgelevel > LOffD – LOnD

Please note that the laser delays must be specified in units of 1 µs, whereas the scanner delays (jump delay, mark delay and polygon delay) must be specified in units of 10 µs.

The reasons for the two constraints **(1)** and **(2)** can be understood as follows:

(1) Consider a very short mark command.

If the sum of the marking time and the LaserOff delay is shorter than the LaserOn delay, the LaserOff delay will be over *before the LaserOn delay has finished*.
Thus the RTC®4 will first try to turn the laser *off* and afterwards turn it *on*:

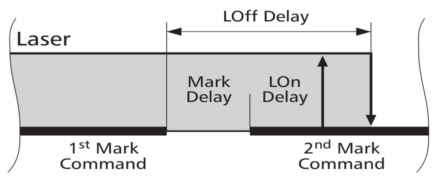


This can be avoided by always setting the LaserOff delay *longer* than the LaserOn delay.

LOffD > LOnD **[1]**

(2) Consider two subsequent mark commands.

If the LaserOff delay (after the first mark command) is longer than the sum of the mark delay and the LaserOn delay (for the second mark command), the RTC®4 will turn off the laser during the second mark command:



To avoid this, the sum of the mark delay and the LaserOn delay must be longer than the LaserOff delay.

markD + LOnD > LOffD **[2]**

In practice, the laser delays are usually optimized first. When the laser delays are fixed, equation [2] reads:

markD > LOffD – LOnD **[3]**

i.e. the mark delay must be longer than the *difference* between the LaserOff delay and the LaserOn delay.

(3) If a list change is necessary while a polyline is executed, but not **auto_change** (page 75) is used for this purpose and the second list is started immediately after complete execution of the first list, then the same as for the mark delay is required for the polygon delay:

$$\textbf{polygonD + LOnD > LOffD} \qquad \textbf{[4]}$$

Otherwise, it could occur that the laser is off for the rest of the polyline.
SCANLAB recommends to use the **auto_change** command for automatic list change or to add a small software delay of the size (LOffD - LOnD - polygonD) previously to starting the second list.

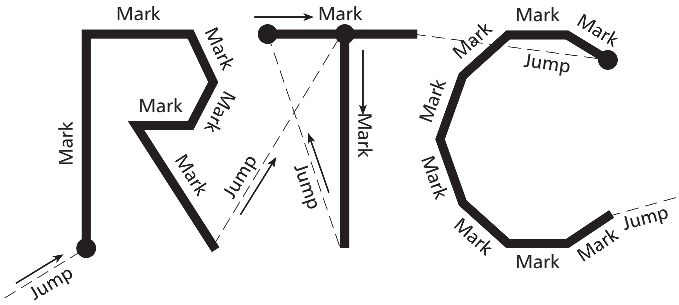
Optimizing The Delays

The following figures show the various effects of non-optimized delays on the letters "RTC".

LaserOn Delay too short

At the beginning of a mark vector the laser is switched on, even though the mirrors have not yet reached the necessary angular velocity.

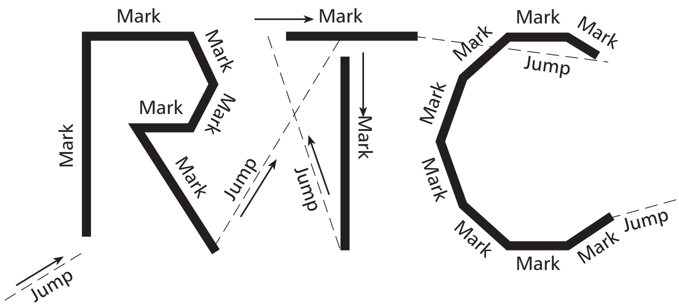
Burn-in effects at the start points of the respective vectors result.



LaserOn Delay too long

The laser is turned on too late at the beginning of a mark vector.

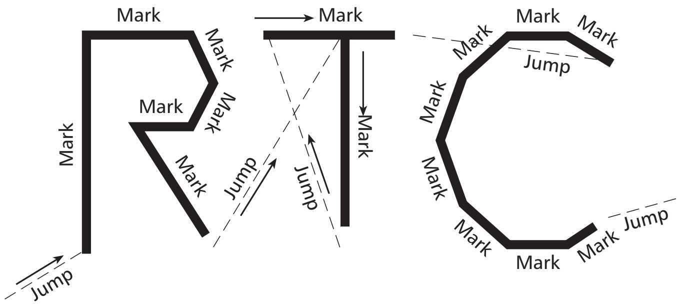
The first part of the vector is not marked.



LaserOff Delay too short

The laser is turned off after the last mark command of a line or polyline, although the mirrors have not yet reached the end position of the vectors.

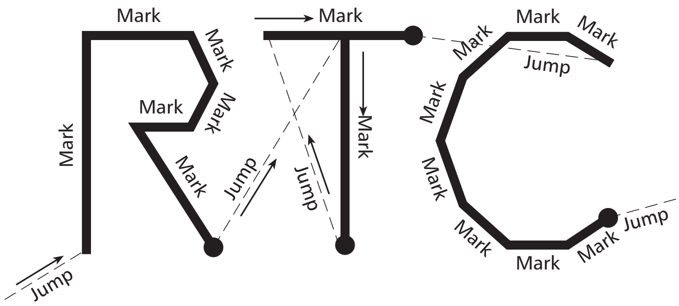
The respective vectors are not marked completely.



LaserOff Delay too long

The laser is turned off too late after the last mark command of a line or polyline. The laser is still on, even though the mirrors have already stopped or move only very slowly.

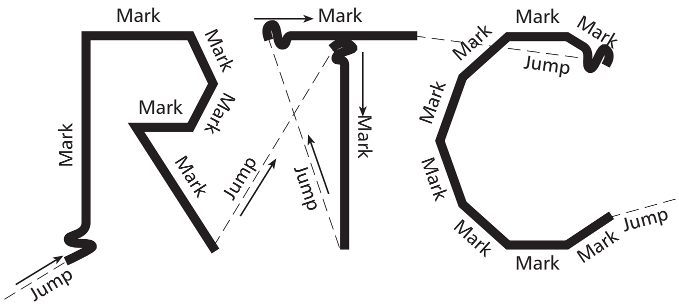
The results are burn-in effects at the end points of the respective vectors.



Jump Delay too short

After a jump, the first mark vector has already started although the scanners have not yet settled.

A running-in oscillation (overshoot) will be visible.



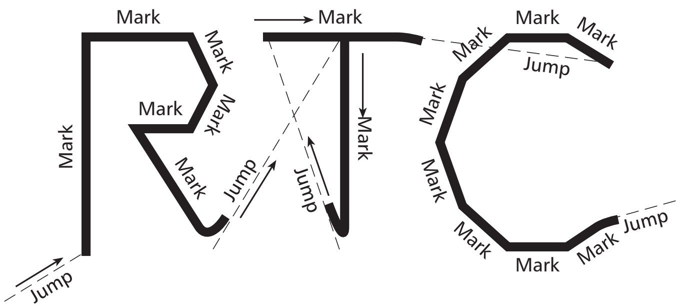
Jump Delay too long

There are no visible effects if the jump delay is too long. However, the scanning time will be extended.

Mark Delay too short

Though the mirrors have not yet reached the end position of the last vector of a line or polyline, the command for the succeeding jump vector is already executing.

The end of the mark vector is turned towards the direction of the jump vector.



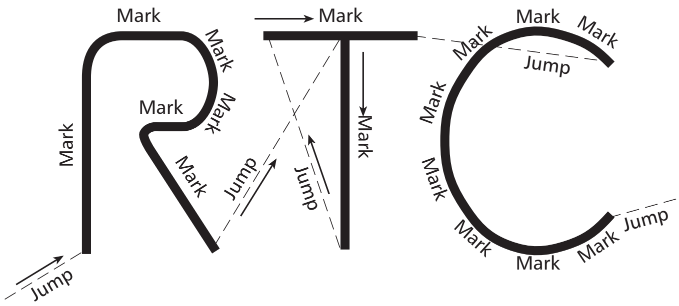
Mark Delay too long

There are no visible effects if the mark delay is too long, but the scanning time will be increased.

Polygon Delay too short

The subsequent mark command in a polyline is already executing, although the mirrors have not yet reached the end position of the preceding mark vector.

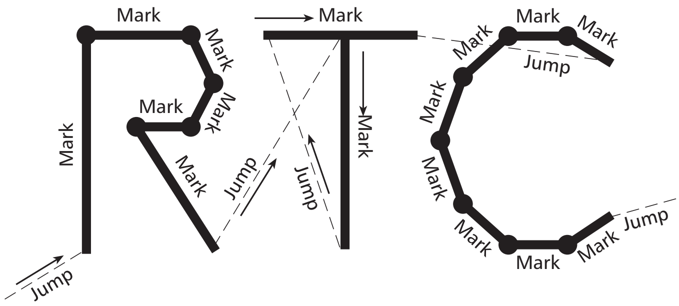
The corners of the polyline are rounded off.



Polygon Delay too long

If the polygon delay is too long, the mirrors are moving too slowly or are even stopping between subsequent mark commands.

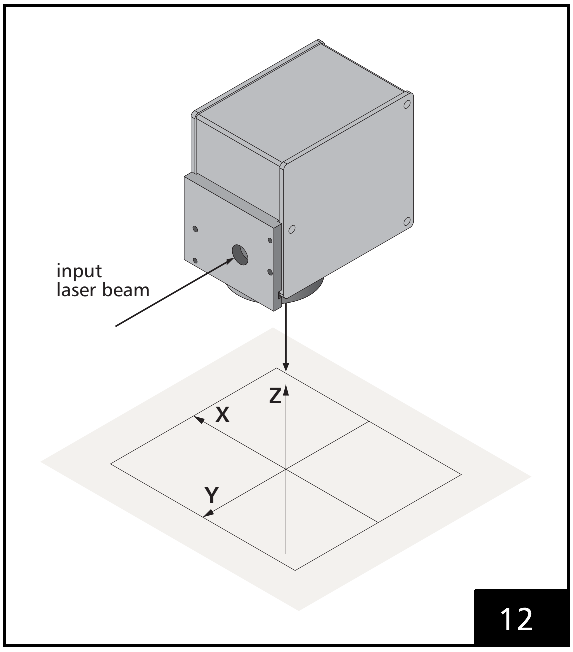
Since the laser is not turned off between these vectors, burn-in effects occur.



In the variable polygon delay mode, a maximum length ("edgelevel") can be defined. See page 21 for details.

4.4 Image Field Size

Figure 12 shows the reference system for the image field which is used by the RTC®4. The Y-axis points in the *reverse* direction of the input laser beam, the X-axis points to the right of the Y-axis. X-axis, Y-axis and Z-axis (optional) form a right-handed reference system. The origin of the reference system, i.e. the point (0|0), is in the center of the image field.



Reference system for the RTC®4 coordinates

The size of the usable image field is determined by the maximum scan angle and the focal length of the objective or the working distance (i.e. the distance between the input laser beam axis and the image field).

All X and Y vector coordinates must be specified as signed 16-bit numbers (i.e. as numbers between −32768 and +32767).

The ratio of a point coordinate in *bits*(1) and the actual position of the point in *millimeters* is defined by the calibration factor *K*.

Let a_0 denote the side length of the image field given by the maximum scan angle. The theoretical calibration factor is then $K_0 = 2^{16}/a_0$ [bits per mm(1)].

(1) The expression "bits" is here synonymous with "digital control value" (see footnote on page 15).

(2) The calibration factor is the same for the X and Z direction and – for 3D scan systems – also for the Z direction.

SCANLAB provides a rounded value for the calibration factor *K*. This value is slightly larger than, but close to, the theoretical value.

The actual calibration factor *K* can be found in the README file that is included in the software package folder containing the correction file.(2)

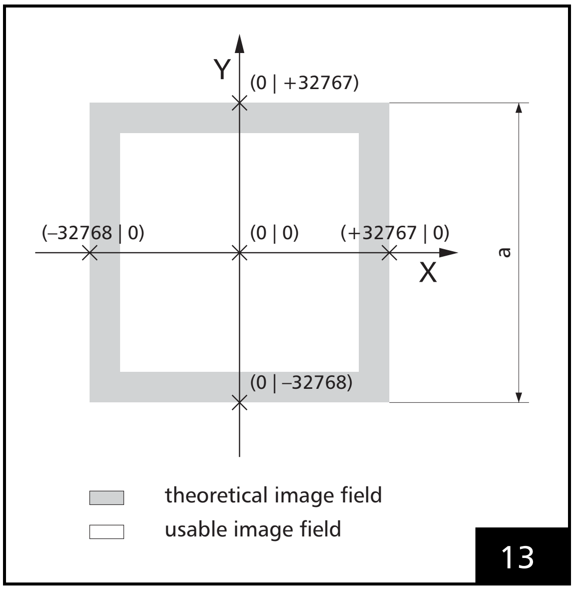
Given the calibration factor *K*, the side length *a* of the usable image field can be calculated: $a = 2^{16} / K$

Typical Image Field

In general, the objective and the optical configuration of the scan head further reduce the size of the usable image field.

The scan head has a "typical image field". If the laser focus moves outside this typical image field, some *vignetting* of the laser beam can occur. The interior of the scan head can be damaged due to excessive absorption of laser power. Please refer to your scan head operating manual's section on objectives.

- Compare the calculated side length *a* (as described above) with the side length *b* of the typical image field given in the technical specifications of your scan head manual.
- If the laser focus shall be restricted to points within the typical image field, the absolute values of the X and Y coordinates (in bits) must be smaller than the maximum value *M*, where *M* is the calibration factor *K* multiplied by *half* the side length of the typical image field: $M = K \cdot b/2$



Size of the usable image field

4.5 Image Field Correction

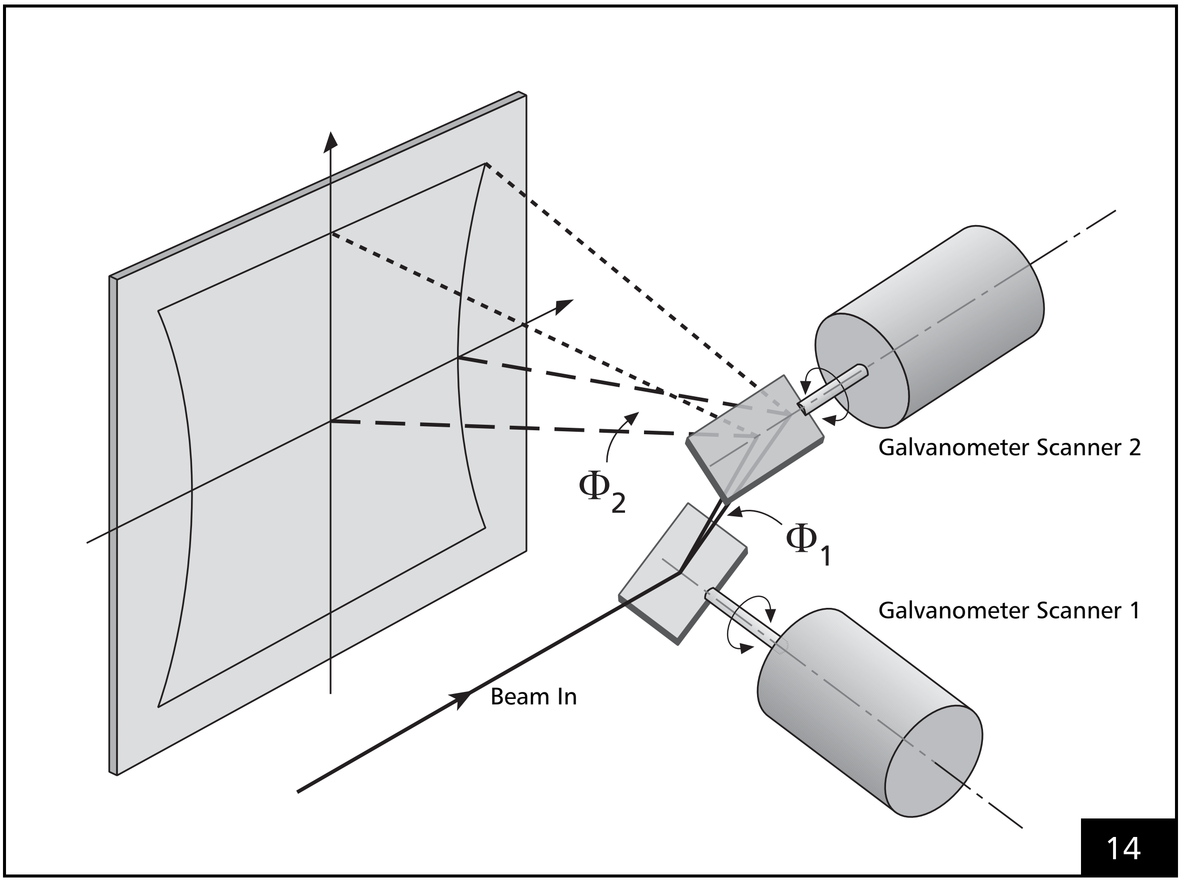
The deflection of a laser beam with a two-mirror system results in three effects:

(1) The arrangement of the mirrors leads to a certain distortion of the image field – see figure 14 on page 30.

This distortion arises from the fact that the distance between mirror 1 and the image field depends on the size of the scan angles of mirror 1 and mirror 2. A larger scan angle leads to a longer distance.

(2) The distance in the image field is not proportional to the scan angle itself, but to the tangent of the scan angle. Therefore, the marking speed of the laser focus in the image field is not proportional to the angular velocity of the corresponding scanner.

(3) If an ordinary lens is used for focusing the laser beam, the focus lies on a sphere. In a flat image field, a varying spot size results.



Field distortion in a two-mirror deflection system

F-Theta Objectives

By focusing the deflected laser beam with an F-Theta objective, two of the three effects can be avoided:

- A direct proportionality between scan angle and distance in the image field is obtained.
- Additionally, the focus lies on a flat surface.

The F-Theta objective, however, causes a barrel-shaped distortion of the image field, as depicted in figure 15 on page 31, center. This distortion is added to the pillow-shaped distortion described in section **(1)** on page 30 (see figure 15 on page 31, left), resulting in a so-called "barrel-pillow-shaped" distortion of the image field – see figure 15 on page 31, right.

Field Correction Algorithm

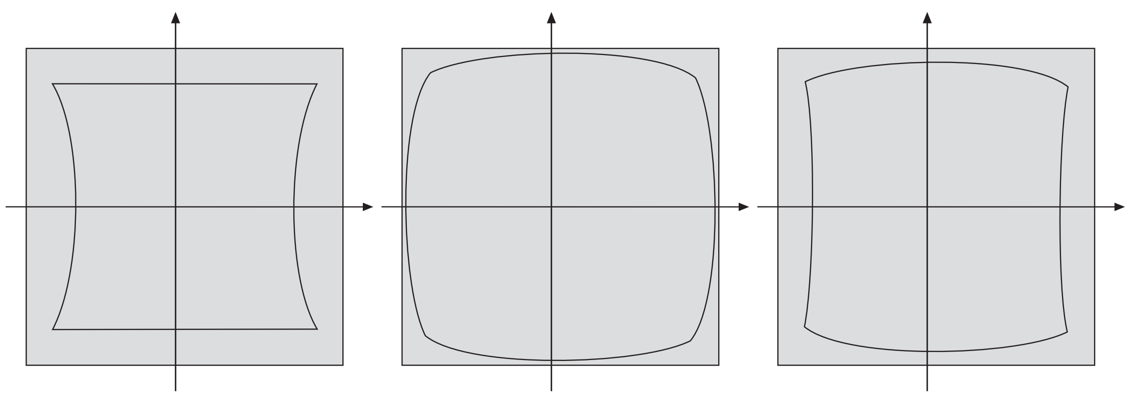
The RTC®4 board provides a correction algorithm to compensate for the field distortion. The algorithm is based on a correction table.

An orthogonal grid of 65 x 65 points is superimposed on the ideal square image field. The adjusted X and Y coordinates for the correct output of these grid points are stored in a correction table. To move the focus to any point within the image field, the RTC®4 calculates the correct coordinates by interpolating from the grid points in the correction table. The result is transmitted to the scan head. The correction algorithm is executed for every single microstep. (See the section "Microsteps", page 15.)

By default, SCANLAB creates an individual correction table for every delivered system. It is stored in a file named `*.CTB` in the RTC®4 software package. Also see the command **load_correction_file**, page 96.

The calculation of the correction table is based solely on general system data (such as mirror geometry, calibration and the objective's specifications). Standard correction files therefore reflect neither the unique properties possessed by the customer's individual system, nor alignment errors.

For those customers requiring more accurate correction tables tailored to the unique properties of their particular scan systems, SCANLAB offers two programs: CFMP and correXion. These programs generate RTC®4 correction files based on data derived from actual test measurements performed by the customer (for further information, refer to the "correXion and CFMP" manual or contact SCANLAB).



15

Left: "pillow-shaped" field distortion caused by the arrangement of the mirrors
Center: "barrel-shaped" distortion caused by the F-Theta objective
Right: resulting, "barrel-pillow-shaped" field distortion

4.6 Laser Control

The RTC®4 provides several laser control signals with programmable pulse width and frequency. These signals can be used for controlling various types of lasers.

Via the command **set_laser_mode** (see page 109), the user can choose one of four different laser control modes:

- CO_2 Mode (conceived of for controlling CO_2 lasers)
- YAG Modes 1, 2 and 3 (three variants conceived of for controlling Nd:YAG and related lasers)
- Laser Mode 4 generates a continuously-running control signal.

Depending on the selected mode, different laser control signals are available. These signals are described in the following sections.

All laser signals are TTL signals. They can be set to either active-high or active-low with a jumper. See "TTL Laser Signal Level", page 59, for details.

Notes:

- The command **set_laser_mode** must be called during initializing the RTC®4 for selecting the laser control mode. After the initialization the laser control is activated. During operation the selected laser signals can be deactivated via the control command **disable_laser** (page 80) and again reactivated via the control command **enable_laser** (page 81). The command **get_startstop_info** (see page 86) (Bit #9) provides information about the current status of the laser control.
- All laser signals are TTL signals. They can be set to either active-high or active-low with a jumper. See "TTL Laser Signal Level", page 59, for details.
  The maximum current load for all laser signals is 10 mA. If jumper X6 has been configured to replace the ANALOG OUT 1 signal with +5 V, then the maximum current load at pin (4) of the laser connector is 100 mA.
  The signals are available via the 9-pin D-SUB laser connector – see page 52.

CO_2 Mode

The command `set_laser_mode(0)` selects the CO_2 laser mode. In this mode, the following laser signals are available (see figure 16 on page 33):

- a LASERON signal
- two alternating modulation signals with variable pulse width and frequency (LASER1 and LASER2)

The signals are available via the 9-pin D-SUB laser connector – see page 52. The LASER1 and LASER2 signals are additionally available at the LASER EXTENSION connector.

The signals LASER1 and LASER2 have a constant relative phase shift of 180° (half an output period). LASER2 can be used for the control of a second laser tube.

To control the laser power, the pulse widths of both laser signals can be varied independently. However, the output *frequency* of both signals is the same.

The output frequency and the pulse widths have to be specified with the list command **set_laser_timing** (see page 109). Please note that *half* of the *output period* must be specified, i.e. the shift between the two laser signals.

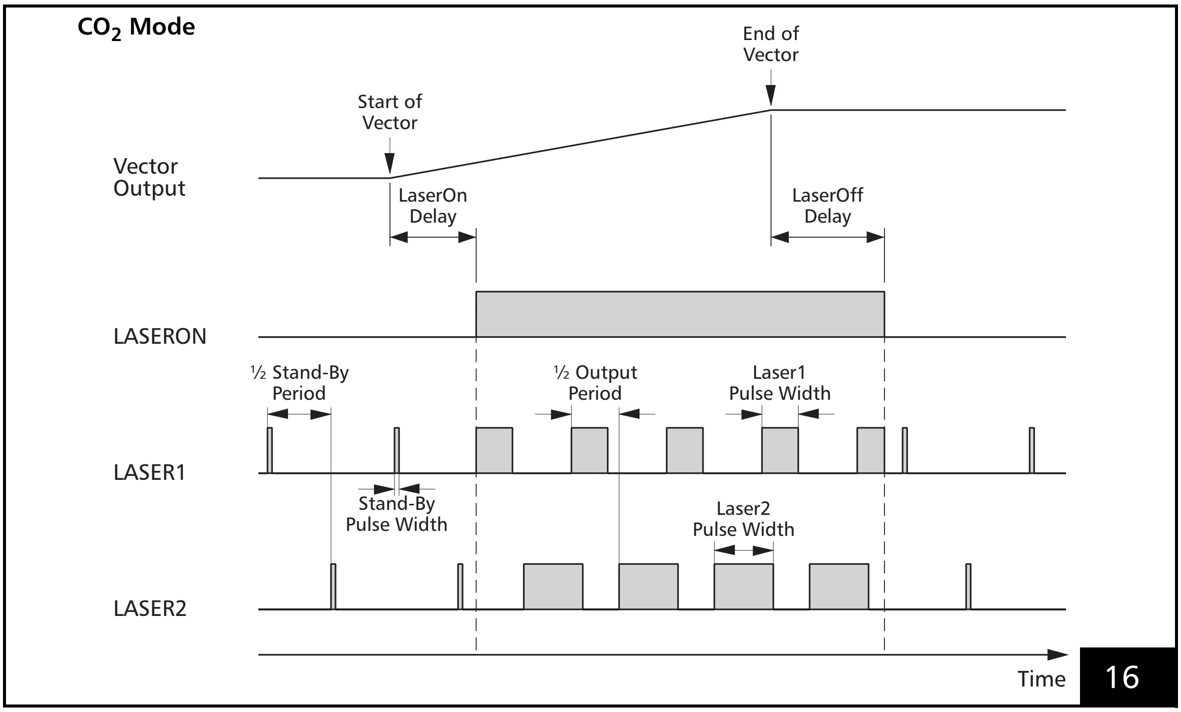
The command **set_laser_timing** is a *list* command, i.e. it can be used anywhere in a list. This allows, for instance, changing the laser power at any time within a list.

- The actual output period and the pulse widths of laser signals LASER1 and LASER2 depend on the time base, which has to be specified with the command **set_laser_timing** (see page 109). The time base can be set to 1 MHz or 8 MHz.
  In general, it is recommended to set the time base to 8 MHz (with `set_laser_timing(..,..,..,1)`). A time base of 1 MHz should only be chosen if necessary.

Stand-By Mode

While the LASERON signal is inactive, the output of signals LASER1 and LASER2 is reduced to stand-by pulses. The pulse width and the output frequency of these stand-by pulses are set with the command **set_standby** (see page 117). They are equal for both laser signals.

The stand-by pulses can be turned off by setting the stand-by pulse width to zero (default setting).



Laser control timing diagram (CO_2 Mode)

YAG Modes

With the commands `set_laser_mode(1)`, `set_laser_mode(2)` or `set_laser_mode(3)` one can choose between three different YAG laser control modes. In all three YAG modes, the RTC®4 supplies the following signals:

- a LASERON signal
- a **Q-Switch signal** with variable pulse width and frequency (LASER1)
- a programmable **FirstPulseKiller signal** (LASER2)

The signals are available via the 9-pin D-SUB laser connector – see page 52. The LASER1 and LASER2 signals are additionally available at the LASER EXTENSION connector.

Q-Switch Signal

The Q-Switch signal is available for control of the laser's Q switch. The Q-Switch period and pulse width are set with the command **set_laser_timing** (see page 109). Note that *half* of the output period must be specified. The parameter *pulse_width2* (for the LASER2 signal) has no effect in the YAG modes.

- Please note that the actual Q-Switch period and the pulse width depend on the time base specified via the command **set_laser_timing** (see page 109).
  The time base can be set to 1 MHz or 8 MHz.
  In general, it is recommended to set the time base to 8 MHz (command `set_laser_timing(..,..,..,1)`).
  A time base of 1 MHz should only be chosen if necessary.

FirstPulseKiller Signal

The FirstPulseKiller signal is available for reduction of the laser pulse power at the beginning of a pulse train. The initial pulses of a pulse train often have a higher energy than the following pulses. These intensity variations are to be avoided in most laser marking applications. Therefore the laser should be equipped with a FirstPulseKiller device that reduces the power of the first laser pulses.

The RTC®4 provides a control signal for the FirstPulseKiller (TTL level). The FirstPulseKiller signal is started together with the LASERON signal. While the FirstPulseKiller signal is active, the energy of the pulses should be reduced adequately.

The length of the FirstPulseKiller signal is set with the command **set_firstpulse_killer** or **set_firstpulse_killer_list** (see page 106).

Differences Between YAG Modes 1 - 3

The three YAG laser modes only differ in the relative start time of the first Q-Switch pulse (also see the timing diagrams in figure 17 on page 36).

- In **YAG Mode 1** the first Q-Switch pulse starts at the *beginning* of the FirstPulseKiller signal.
- In **YAG Mode 2** the first Q-Switch pulse starts at the *end* of the FirstPulseKiller signal. This mode is provided for certain YAG laser types.
- In **YAG Mode 3** the first Q-Switch pulse starts 10 µs after the beginning of the FirstPulseKiller signal.

Lamp Current (Laser Power)

To control the lamp current of a YAG laser, the analog output signal ANALOG OUT1 (10-bit signal) can be used. This signal is available via the 9-pin D-SUB laser connector – see figure 23 on page 52. To set the output signal, use the command **write_da_1** or **write_da_1_list**.

Alternatively, the lamp current can be controlled digitally via the 8-bit digital output port on the LASER EXTENSION connector (figure 24 on page 53). The commands for setting the 8-bit port are **write_8bit_port** and **write_8bit_port_list**.

Please refer to "Laser Extension Connector", page 53, for details.

Softstart Mode

For some applications it's important to control the laser intensity at the beginning of a marking process. SCANLAB provides a convenient solution in the form of the softstart mode, which can be used for all laser modes.

With the command **set_softstart_level** (see page 115), a list of control values (Level[0]...Level[*n*], $n < 1000$) for the first *n* pulses can be defined: digital values corresponding to various pulse widths or analog values corresponding to variable laser power levels.

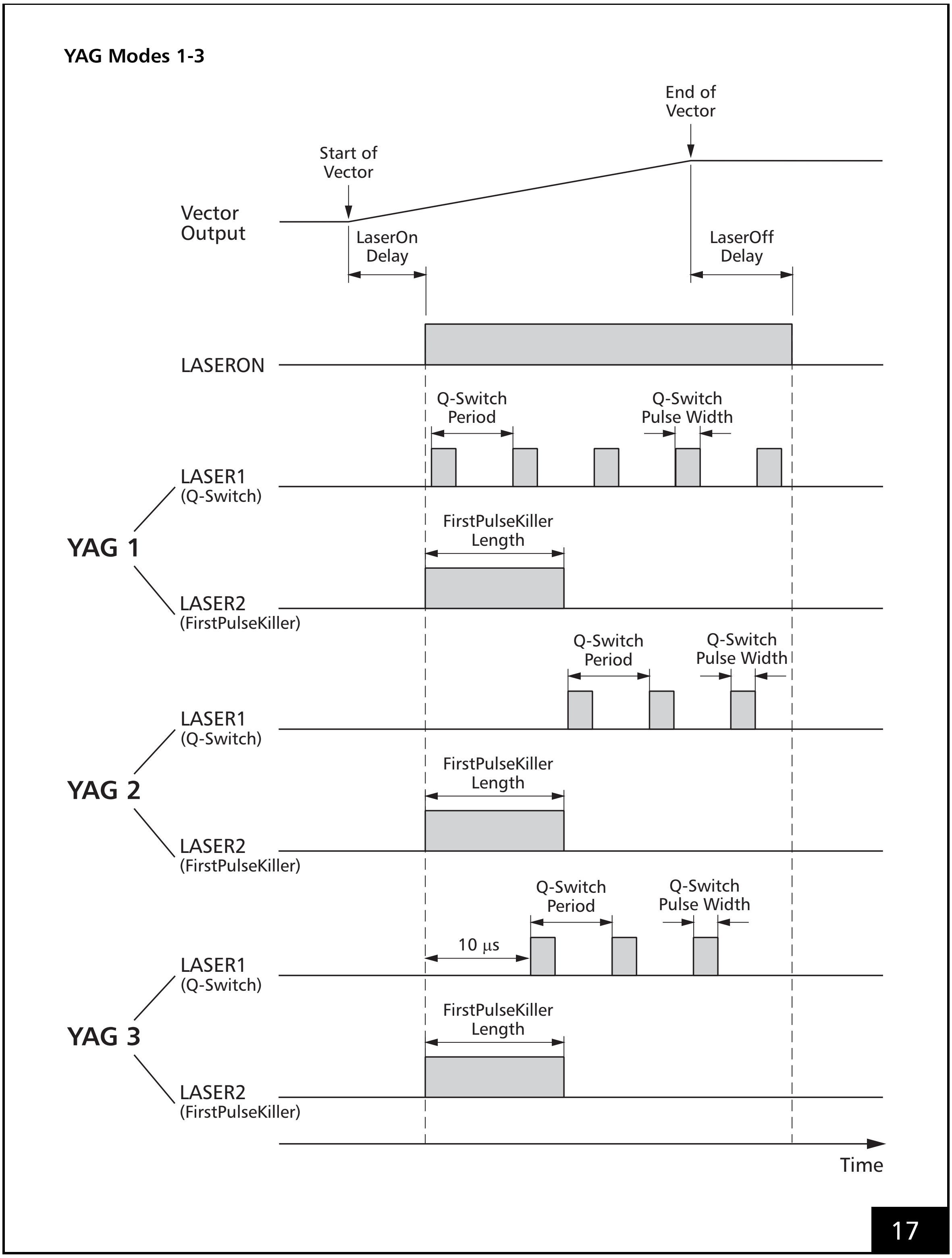
Initialization of the softstart mode is performed via the command **set_softstart_mode** (see page 116). The `set_softstart_mode` command determines, among other things, whether the `softstart` value should be interpreted as an analog value or a digital value. Therefore, the command must be called *before* first-time use of the `set_softstart_level` command.

As soon as the laser remains switched off longer than a certain predefined time (*restart delay*), the value Level [0] is assigned to the output port (digital values to LASER1 or analog values to ANALOG OUT1/2). Simultaneously with the first *n* laser pulses (selectable at either the leading or trailing edges of the laser pulses) the corresponding values (Level [1...*n*]) will then be automatically assigned to the output port.

Notes

(1) With a large enough value for the restart delay time, one can avoid the softstart mode being also activated after very short jump commands.

(2) The pulse width or power of the first laser pulse is determined by level*[0]* if the values are assigned with the *leading* edge of the laser pulse selected, or by level*[1]* if the *trailing* edge of the laser pulse is selected.



Laser control timing diagram (YAG Modes 1, 2 and 3)

Laser Mode 4

The command `set_laser_mode(4)` selects the laser mode 4. In this mode, the following laser signals are available (see figure 18 on page 37):

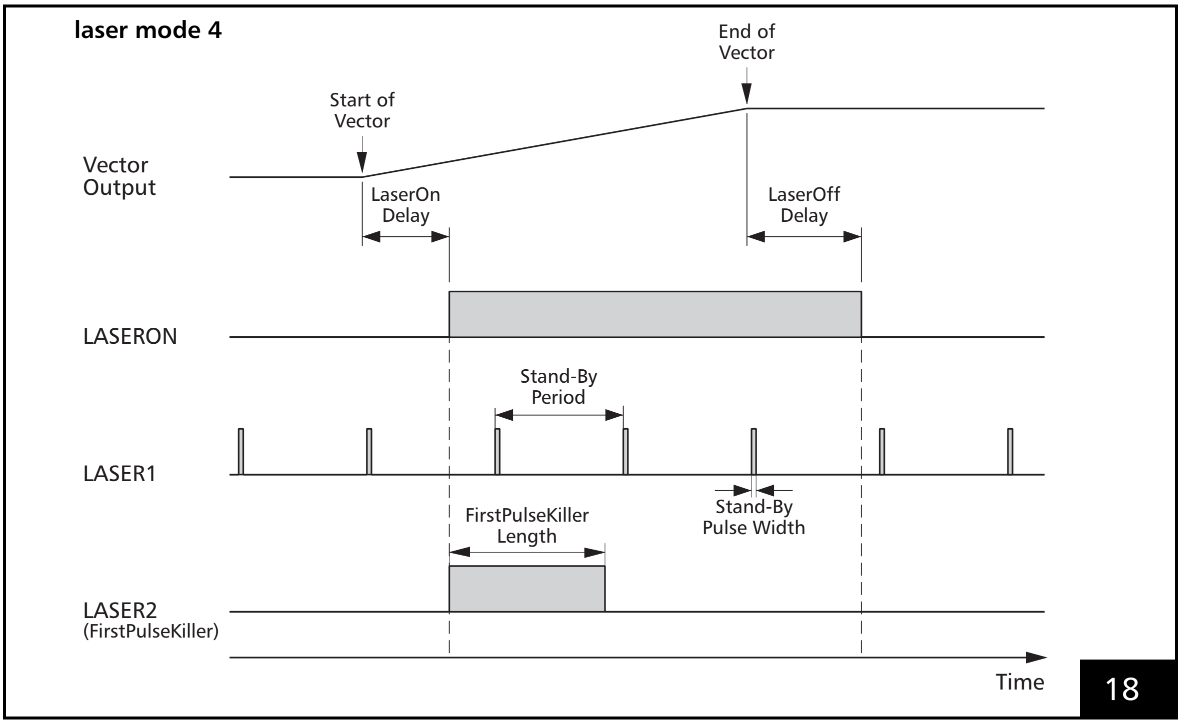
- a LASERON signal
- a continuously-running modulation signal with variable pulse width and frequency (LASER1). The signal is provided for an active LASERON signal as well as for an inactive LASERON signal (as standby signal).
- a programmable FirstPulseKiller signal (LASER2)

The signals are available via the 9-pin D-SUB laser connector – see page 52. The LASER1 and LASER2 signals are additionally available at the LASER EXTENSION connector.

The pulse width and the output period of the LASER1 modulation signal are set with the command **set_standby** (see page 117) or **set_standby_list** (see page 117). In the default setting the pulse width is set to zero. Please note that *half* of the *output period* must be specified. The maximum allowed pulse width is the *half* stand-by period.

The FirstPulseKiller signal is started together with the LASERON signal. The length of the FirstPulseKiller signal is set with the command **set_firstpulse_killer** or **set_firstpulse_killer_list** (see page 106).

In laser mode 4 the time base for the signals LASER1 and LASER2 is always 1/8 µs.



Laser control timing diagram (laser mode 4)

4.7 Status Monitoring and Diagnostics

For status monitoring and diagnostic purposes, the command **get_value** can be used to read a variety of signals:

- XY2-100-compliant (or XY2-100 Enhanced-compliant status) signals returned by the scan system
  - XY2-100-compliant status signals can be queried via the **get_head_status** command, too.
  - As specified by the XY2-100 Enhanced Protocol, intelli*SCAN*® scan systems allow various data signals for each axis to be transmitted separately to the RTC®4 for evaluation (see chapter 4.8, page 38).
- the current value of the LASERON signal
- the current cartesian output values (coordinate transformations defined by **set_matrix**, **set_matrix_list**, **set_offset** or **set_offset_list** will have already been applied to these output values)
- the actual (digital) output values currently being transmitted from the RTC®4 to the scan system (image field correction will have already been applied to these values in accordance with the loaded correction file)

The command **set_trigger** can be used on each of these signals to record their values over time – and with a selectable sample period. The recorded values are stored on the RTC®4. The command **get_waveform** can be used to transfer the recorded values to the PC.

The current status of a measurement session started with **set_trigger** can be obtained by calling the command **measurement_status**.

4.8 intelli*SCAN*®- Additional Functions

The XY2-100 Enhanced Protocol

The intelli*SCAN*®'s digital servo architecture provides enhanced functionality not possible with the XY2-100 protocol. To accommodate these new features, a superset of the XY2-100 Protocol has been created: the XY2-100 Enhanced Protocol. Essentially, the enhancement extends the two receiving channels to three receiving channels to allow separate, simultaneous evaluation of the X and Y status signals (and also – when required – separate evaluation of the Z-axis status signals).

The enhanced protocol is downward-compatible with the XY2-100 Protocol. Thus, the intelli*SCAN*® can also be operated via all SCANLAB RTC® PC interface boards to thereby obtain the standard functionality typical of traditional SCANLAB scan heads.

Achieving the intelli*SCAN*®'s full functionality, though, requires use of the XY2-100 Enhanced Protocol and an RTC®4 board.

Selecting Data Signals

The intelli*SCAN*®'s digital servo architecture allows a wide variety of data signals to be returned to the control board. Each axis is assigned its own status channel which transmits data to the controller board every 10 µs: The STATUS channel is provided for the X axis (galvanometerscanner 2) and the STATUS1 channel for the Y axis (galvanometerscanner 1). This opens up possibilities such as monitoring the galvanometer scanners' actual values during execution of an application or carrying out comprehensive troubleshooting in case of operational malfunction.

The **control_command** command allows selection of which data the scan head should return to the control board. The available data signals are described in detail in the intelli*SCAN*® Manual and in the **control_command** section of the command reference. The selected data sources will be transmitted until another source is selected.

Five seconds after every reboot or reset, a status word compliant with the XY2-100 protocol is transmitted (on all receive channels).

Querying Data

Data received by the RTC®4 can be read asynchronously at any time via the commands **get_value** or **get_head_status** or synchronously via the command **set_trigger** (see "Status Monitoring and Diagnostics", page 38 and the corresponding command references). Please note that switching of the data source is followed by a short (serial transmission-related) delay before the first data is transmitted. Therefore, after switching data sources you should wait at least 50 µs before reading.

The value ranges of transmitted data and the possible status states are described in the command reference of the **control_command** command.

Note: Together with an intelli*SCAN*® scan system, the **get_head_status** command can only be used to get the status signals (Statusword) directly after a new start or reset or when the Statusword is selected to be transmitted by the scan system via the **control_command** command.

Setting Control Values

The **control_command** command can also be used to set the PosAcknowledge threshold value. By default (after each reset) the value is set to 0.28% of the full position range (i.e. 0.28% of 2^{16} counts). If other threshold values are desired, they must be separately set for each axis. SCANLAB recommends to set only threshold values above 0.03% of 2^{16} counts. Lower values can lead to frequent system safety shut-downs due to Position Acknowledge time outs (set position not reached for long time).

5 Advanced Programming

5.1 Coordinate Transformations

The RTC®4 has the ability to transform the output coordinates by the following linear transformation for each vector defined with a jump, mark or arc command:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_o \\ y_o \end{bmatrix}$$

The coefficients m_{11} … m_{22} of the (2 x 2) transformation matrix are set with the command **set_matrix** (see page 111). The offset ($x_o | y_o$) in the X and Y directions is defined with **set_offset** (see page 112).

Any previously defined matrix and offset definitions will be thereby overwritten.

The list commands **set_matrix_list** and **set_offset_list** work in a similar way. See **set_matrix_list** (page 111) and **set_offset_list** (page 112) for details.

The coordinate transformation defined with these commands is applied to the original coordinates of all specified vectors (i.e. before splitting into microvectors and before image field correction). The transformed vectors are scanned with the correct marking speed, even if the length of a vector has changed during the transformation.

In a double scan head configuration, the coordinate transformation defined with these commands applies to **both scan heads**(1) in the same way.

The transformation matrix can be used for scaling, rotating, flipping or skewing an object.

Examples:

1. Rotation by the angle α :

$$\begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$$

For instance, the command `set_matrix(0.5, -0.866, 0.866, 0.5)` defines a rotation by 60° (counterclockwise).

2. Scaling by the factors k_X and k_Y :

$$\begin{bmatrix} k_X & 0 \\ 0 & k_Y \end{bmatrix}$$

3. Flipping in the X direction (mirroring):

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

4. Flipping in the Y direction:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

5. Exchanging the X and Y axes:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

6. Skewing in the X direction by the angle α (slanting):

$$\begin{bmatrix} 1 & -\sin\alpha \\ 0 & 1 \end{bmatrix}$$

Example: `set_matrix(1, -0.25, 0, 1)`

To combine several transformations, the corresponding matrices are multiplied (in the correct order). The resulting matrix is used for the command **set_matrix**.

(1) Also see the command **load_correction_file** (page 96).

5.2 Wobbel Function

The wobbel function allows varying the line width during laser marking.

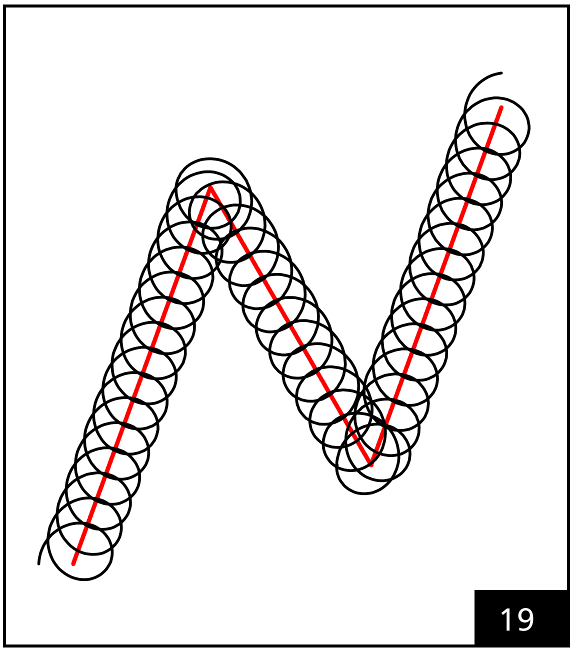
Basically, a circular movement is added to the regular, linear movement of the output position, resulting in a spiral movement of the laser focus in the image field.

A broadening of the original line is obtained by choosing suitable values for the amplitude and the frequency of the wobbel movement.

For optimum marking results, the wobbel frequency should be appropriate for the specified marking speed. In some cases it can be useful to adjust the marking speed when the wobbel function is used.

Calling the command **set_wobbel** sets the wobbel phase to a defined starting value (which is independent of the direction of the marking vector).

Please refer to the command **set_wobbel**, page 119 for further details.



19

Principle of the wobbel function

5.3 Using Two Different Correction Files

Double Scan Head Configuration

The RTC®4 (2D version) can store two different correction files at the same time. For two scan heads controlled from a single RTC®4 board, each scan head can thereby receive its own separate image field correction.

Each of the two correction files can be assigned to either of the two scan head control ports by the command **select_cor_table**.

See page 54 for details about the two scan head control ports of the RTC®4.

The following steps describe how to use two correction files in a double scan head configuration:

- Load the first correction file by the command `load_correction_file(FileName1, 1, ...)`.
- Load the second correction file by the command `load_correction_file(FileName2, 2, ...)`.
  Specify additional gain, rotation and offset to align the two image fields precisely with respect to each other. Please refer to the command **load_correction_file** (page 96) for details.
- Afterwards, you have to assign the first correction file to the first scan head and the second correction file to the second scan head by calling the command `select_cor_table(1,2)`.

Notes

- The default setting for **select_cor_table** is `(1,0)`, i.e. correction table #1 is used for the first scan head. The output signals for the second scan head are turned **off**.
- The RTC®4 returns to this setting after every reset (by one of the commands **load_program_file** or **dsp_start**).
  If a different setting is to be used, the command **select_cor_table** must be called again after the reset.

Using Two Correction Files In A Single Scan Head System

It can also be useful to work with two different correction files in a *single scan head* system, for example if a pointer laser and a main laser with a different wavelength are used.

- Load the two correction files as described in the section "Double Scan Head Configuration", left.
- Afterwards, use the commands
  `select_cor_table(1,0)` and
  `select_cor_table(2,0)`
  to switch from one file to the other.

Note: The RTC®4 3D version can store only *one* correction file.

5.4 Using Multiple RTC®4 Boards In One Computer

The RTC®4 driver DLL supports simultaneous control of up to eight RTC®4 boards in one PC.

The RTC®4 boards work completely independently of each other. Command lists for each board can be loaded and executed at any time.

Command Set

There are two different methods for writing application programs using several RTC®4 boards:

(A) Multi-Board Programming

In this programming method, so-called *multi-board* versions of the RTC®4 commands are used. Each of these multi-board commands allows specifying the number of the RTC®4 board that shall receive the command.

> To use a multi-board command, simply
>
> - add the prefix `n_` to the command name, and
> - include the number of the RTC®4 board to which the command shall be sent as the **first** parameter (unsigned 16-bit value). The remaining parameters of the command – if any – are the same as for the normal (single board) command.

The installed RTC®4 boards are numbered in the order given by the PCI bus (from 1 to a maximum of 8). The command **n_get_serial_number** (see page 85) can be used to determine which RTC®4 boards have been assigned numbers 1 to 8 (see example 3 below). The command **rtc4_count_cards** (see page 102) can be used to check how many boards are detected by the driver DLL.

All multi-board commands are listed in chapter 10.1, page 69.

Examples: (Pascal)

1. `n_jump_abs(1, 500, 500)`

   writes a jump command to the point (500, 500) into the current list of RTC®4 board #1.

2. `n_execute_list(RTC4_no, list_no)`

   executes list number `list_no` (1 or 2) on the RTC®4 board with the number specified by the variable `RTC4_no`.

3. `sn_1 := n_get_serial_number(1)`

   returns the serial number of board #1.

(B) Sequential Programming

For sequential programming, the command **select_rtc** (see page 104) activates one of the installed RTC®4 boards. In this programming method, the normal (single board) commands can be used. The multi-board commands are not affected by the command `select_rtc`. All commands following the `select_rtc` command are sent to the selected board until the command `select_rtc` is used again. The specified RTC®4 number must not be larger than the total number of RTC®4 boards. Also see **rtc4_count_cards** (page 102).

Care must be taken if several programs or processes are running on one system simultaneously, because the command `select_rtc` immediately redirects the output of *all* currently running processes to the specified RTC®4 board.

In multi-tasking or multi-threaded applications, this can result in programming errors. For such applications, only the multi-board commands described in section **(A)** should be used.

> **Caution!**
>
> - The command `select_rtc` defines the active RTC®4 board for all programs (threads) that are currently running.
>   In multi-tasking or multi-threaded applications, this can result in programming errors. For such applications, only multi-board commands should be used.

SCANLAB
innovators for industry

5.5 Circular Queue Mode

Instead of using the two list buffers as described in chapter 4.1, the RTC®4 memory can also be used as a *circular queue*. This mode allows list commands to be sent to the RTC®4 continuously without any limit. New commands are added by the application at one point of the circular queue, while commands from another point of the circular queue are executed at the same time.

Initialization

The circular queue mode is enabled by the command `set_list_mode(1)` (see page 110).

> This command should be called at the beginning of the application program, i.e. before writing any list commands.

Start

After enabling the circular queue mode, the user can immediately start writing list commands to the RTC®4. The command **set_start_list** must not be used.

Execution starts automatically when the number of stored commands exceeds 4000 or if the command **set_end_of_list** is used.

While the RTC®4 executes a block of entered commands, it is possible to load the circular queue with subsequent list commands. When the end of the list buffer is reached, the RTC®4 starts over at the beginning. However, the RTC®4 overwrites only those commands which have already been executed. Also see Note **(2)**, right.

Continuous Data Transfer

After execution of the first block of commands, the RTC®4 checks if the next block of 4000 commands is ready for execution. If this is the case, the next block is executed immediately afterwards.

Usually it takes much less time to write a list than to execute it. Therefore it should be no problem to write the next block of commands during execution of the current block.

Termination

At the end of the application, usually a block of less than 4000 commands will remain. To ensure correct execution of this last block, it must be terminated with the command **set_end_of_list**.

Circular Queue Control

If it is necessary, the command **stop_execution** can be called to immediately stop execution of the circular queue. In addition, calling the command **stop_list** pauses execution and calling the command **restart_list** resumes execution.

Notes

(1) When the circular queue mode is active, only the following list handling commands can be used:

- **set_end_of_list**
- **stop_execution**
- **stop_list**
- **restart_list**.

Do not use any of the other "List Handling" commands – see chapter 10.1 "Overview", page 69.

(2) Avoid writing a list command into a *full* circular queue because the call of that command will automatically wait until it can enter the circular queue. On the one hand, this guarantees that no list command will be lost, but on the other hand, the RTC®4 will not handle new control commands during that lock period – in contrast, control commands are immediately processed when the circular queue is not full.

- For preventing a lock period, call the command **get_list_space** (page 85), which returns the remaining capacity of the circular queue (which has a total capacity of 8000 commands).
- For relieving the RTC®4, it is a good idea to keep track of the remaining capacity instead of calling **get_list_space** each time before trying to write a list command into the circular queue.

5.6 Structured Programming

The command set of the RTC®4 includes a number of list commands for controlling program flow:

- **set_list_jump**
- **list_call**
- **list_call_cond**
- **list_jump_cond**
- **list_return**
- **list_nop**

- These commands use the RTC®4 list memory as a single list buffer with a total capacity of 8000 entries. Each command in the list buffer has a unique *address* in the range [0 … 7999].

Together with the control commands **set_input_pointer** and **execute_at_pointer**, this allows structured list programming and execution.

Input Pointer

To write a list of commands to a particular location in the RTC®4 list memory, the command **set_input_pointer** (see page 107) is used (instead of the commands **set_start_list_1 / _2**). The command places the input pointer at any address in the range [0 … 7999]. The next list command will be written to this address.

Caution!

- If the end of the list buffer is reached during writing of a list, the input pointer is reset to zero, i.e. the next list command is written to the address zero.
- Make sure not to overwrite any commands which are still needed by your application.

The current position of the input pointer can be interrogated by calling the command **get_input_pointer** (see page 84).

List Jumps

The command **set_list_jump** (see page 110) defines a jump to the specified address. During execution, the RTC®4 jumps to this address.

Similarly, the command **list_call** defines a jump to a subroutine (sub-list). The sub-list has to be terminated with the command **list_return**.

The main list must be terminated with the command **set_end_of_list**.

Conditional List Jumps

Conditional list jumps and subroutine calls can be programmed via the commands **list_jump_cond** and **list_call_cond**: The current value of the 16-bit digital input port at the EXTENSION 1 connector is read and, depending on its value, a subroutine is called or a jump within a list is executed (also see "Programming Examples", page 46 and "16-Bit Digital Input and Output", page 56).

Output Pointer

To start execution at a particular address, the command **execute_at_pointer** (see page 81) must be used. The RTC®4 starts execution immediately.

Execution stops when a **set_end_of_list** command is encountered. If the end of the list buffer is reached, the RTC®4 continues at the address zero.

Caution!

- If the end of the list buffer is reached during execution (and if the last command is not a **set_end_of_list** command), execution continues at the address zero.

If the external start signal (see the section "External Control Inputs", page 13) is used, the command **set_extstartpos_list** (see page 106) allows definition of the start address for the external list start.

Programming Examples

The following programming examples are written in PASCAL.

(1) Confirm a signal:

```
set_start_list(1);
   ...
   set_io_cond_list(0, 0, 1);              // set bit #0 of the 16-bit digital output port
   list_jump_cond(0, 1, get_input_pointer);   // loop until the signal is confirmed
                                              // (i.e. bit #0 of the digital input turns HIGH)
   clear_io_cond_list(0, 0, 1);            // clear bit #0 of the 16-bit output
   list_jump_cond(1, 0, get_input_pointer);   // loop until the signal is confirmed
   ...
   set_end_of_list;
execute_list(1);
```

(2) If the lower four bits of the digital input have the value (`0110`), set bit #1 of the 16-bit digital output, otherwise clear it:

```
set_start_list(2);
   ...
   list_jump_cond($0006, $0009, get_input_pointer + 3);
                                           // skip the next two commands if the state
                                           // of the 16-bit input is (xxxx xxxx xxxx 0110)
   clear_io_cond_list(0, 0, 2);            // clear bit #1 of the 16-bit output and ..
   set_list_jump(get_input_pointer + 2);   // .. skip the next command
   set_io_cond_list(0, 0, 2);              // set bit #1 of the 16-bit output
   ...                                     // (continue)
   set_end_of_list;
execute_list(2);
...
bit1 := (get_io_status AND $0002)          // returns the current state of bit #1
```

(3) Choose between 15 small subroutines:

```
   ...
   for i := 1 to 15 do
      list_call_cond(i, 15-i, i*100);      // call subroutine at address i*100
                                           // if [bit #3...bit #0] (binary) = i
   ...
```

5.7 Scanning Raster Images (Bitmaps)

The vector commands described in chapter 4.2 are intended for scanning vector based images. However, the RTC®4 also allows reproduction of raster images (or bitmaps). That means black-and-white images and even greyscale images can be created with a suitably prepared laser. Furthermore, raster and vector based images can be combined as desired.

Principle Of Operation

A raster image is created line by line, where each line consists of a number of equidistant pixels. A line is reproduced in a single scan. During this scan, the laser focus (set position) moves from one pixel to the next at a constant time rate. The pixel distance and the output period can be set by the user.

Each pixel is usually marked by one laser pulse. For black-and-white images the laser will be turned on or off at each pixel. Greyscale images can be created by modulating the laser power or the pulse width of the laser pulse for each individual pixel or, alternatively, by varying the number of laser pulses per pixel. Please refer to the section "Laser Control" on page 50 for details.

Software Commands

Before starting an image line, a **jump** command to the start position of the line must be performed. The image line itself starts with the command **set_pixel_line** (see page 114). This command turns on the pixel output mode, sets the pixel output period T and defines the distance in the X and Y directions (dx, dy) between two adjacent pixels in the line.

Pixel Output

Each pixel in the image line is then created by one **set_pixel** command. This command defines the pulse width of the LASERON signal for the pixel. In addition, an analog output signal (ANALOG OUT2, 10-bit resolution) can be specified for each pixel. The analog signal is transmitted synchronously with the pixel output (scanner position) and can be used for modulating the laser power. (See the section "Laser Control" on page 50.)

Data Input (Optional)

If the RTC®4 is used together with an optional I/O Extension Board, the command **set_pixel** also allows data to be read from an analog input port synchronous to pixel output. The input data is written to the RTC®4 list buffer, from where it can later be read. For further details please refer to the commands **set_pixel** (page 113) and **read_pixel_ad** (page 101).

Notes

- The commands **set_pixel_line** and **set_pixel** are list commands, i.e. they are written into a list.
- The command **set_pixel_line** requires two list entries in the list buffer memory.
- The number of pixels in an image line is limited only by the capacity of the RTC®4 list buffer (see page 131). It is suggested – especially for large bitmaps – to set up a new list for each image line to avoid a list change during the execution of one line.
- Each image line must start with a **set_pixel_line** command.
- The **set_pixel** commands for the individual pixels must follow immediately after the **set_pixel_line** command. The first subsequent command in the list which is *not* a **set_pixel** command turns off the pixel output mode.
- The pixel distance (dx, dy) in the X and Y directions (in bits) can be specified with floating point numbers. This allows scaling and rotating the image without rounding errors. However, the actual output coordinates (in bits) of each individual pixel are always rounded to integer values.
- The pixel output period can be any multiple of 10 µs, whereas the standard output period for the microsteps in the vector mode is always 10 µs. Also see the section "Scanner Control", page 49.

Timing

Figure 20 shows the pixel output timing diagram. Each LASERON pulse is preceded by a LaserOn delay. The ANALOG OUT2 signal is synchronous to the position update of the scanners. (Please refer to the section "Laser Control" on page 50 for details.)

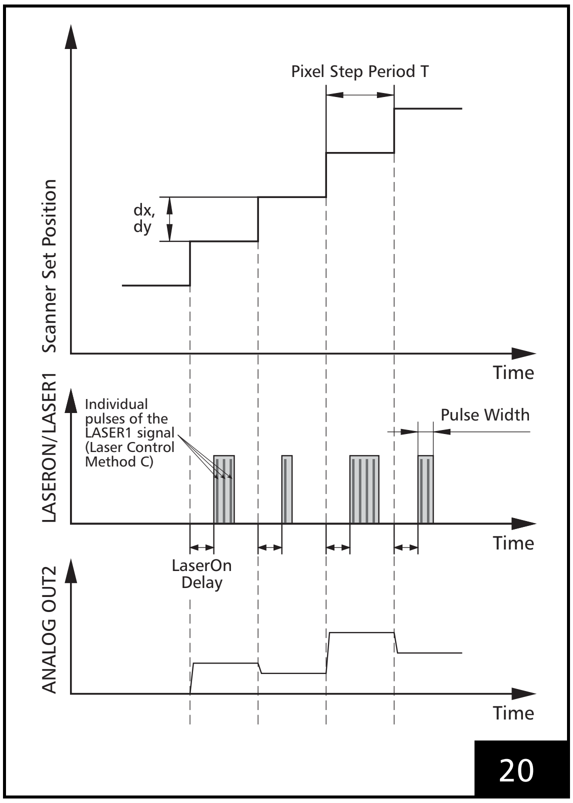
Note: The DA converter requires about 1 µs … 2 µs to produce a stable analog output signal.

The pixel step period T is set with the command **set_pixel_line**. To avoid laser control faults, make sure that the following holds true for all pixels:

T > LaserOn Delay + maximum pulse width

Note

The 9-pin D-SUB laser connector of the RTC®4 provides the signals LASER1 and LASERON *or* ANALOG OUT2. If the ANALOG OUT2 signal is to be used, jumper **X7** must be set to position 2-3. See chapter 8.2 "Changing The Jumper Settings", page 59.



Timing of the scanner positions and the laser control signals in the pixel output mode. Also see section "Laser Control", page 50.

If you need the LASERON signal *and* the ANALOG OUT2 signal, then you have the possibility (with an RTC®4 including the Processing-on-the-fly option only) of setting jumper X7 to position 1-2 and taking the signal ANALOG OUT2 from pin 14 of the MARKING ON THE FLY connector.

Laser Parameters

- The LaserOn delay must be set to a suitable value. See the section "Scanner Control" on page 49.
- The standby pulses should be turned off in pixel mode (command **set_standby**).
- Usually each pixel is marked with one pulse. To accomplish this, the pulse width and output period of the LASER1 / LASER2 signal should be set to values larger than the pixel output period T (command **set_laser_timing**).
  Thus the LASER1 / LASER2 signal will stay active and the laser is controlled by the LASERON signal only.
  Also see the section "Laser Control" on page 50.
- The FirstPulseKiller signal should be set to zero (YAG modes only).

Scanner Control

The RTC®4 supports two different modes for scanning raster image lines. The two modes differ in the way the galvanometer scanners are controlled when moving from one pixel position to the next. The scanner mode is selected with the command set_pixel_line.

Mode 0

In this mode, the scanners "jump" from one pixel to the next in one step. Depending on the pixel distance, some overshooting may occur – see figure 21 on page 49.

- In this mode, the pixel step period T must be relatively long: T ≈ 1 ms … 10 ms.
- To make sure that each pixel is marked at its exact position, the LaserOn delay must be longer than the required settling time.

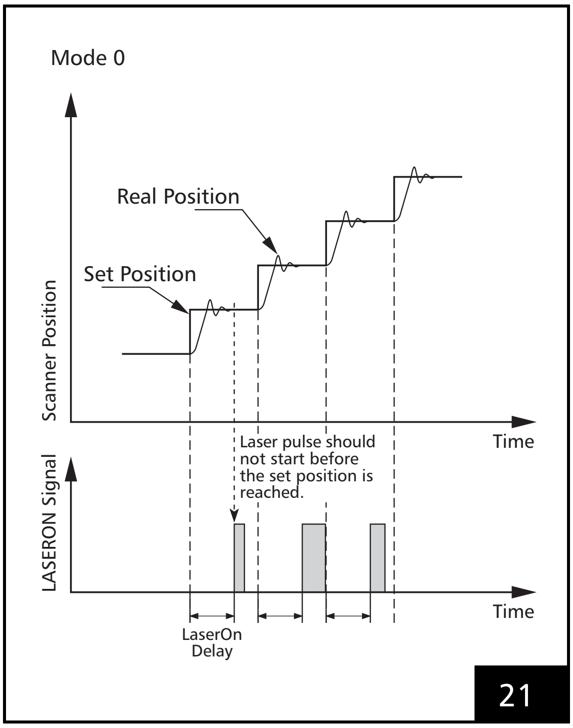
In Mode 0, each pixel will be marked at its exact position. The position does not depend on the dynamic performance of the scan head. However, the overall marking time will be quite long.

Mode 0 should always be used if one pixel is marked by more than one laser pulse.

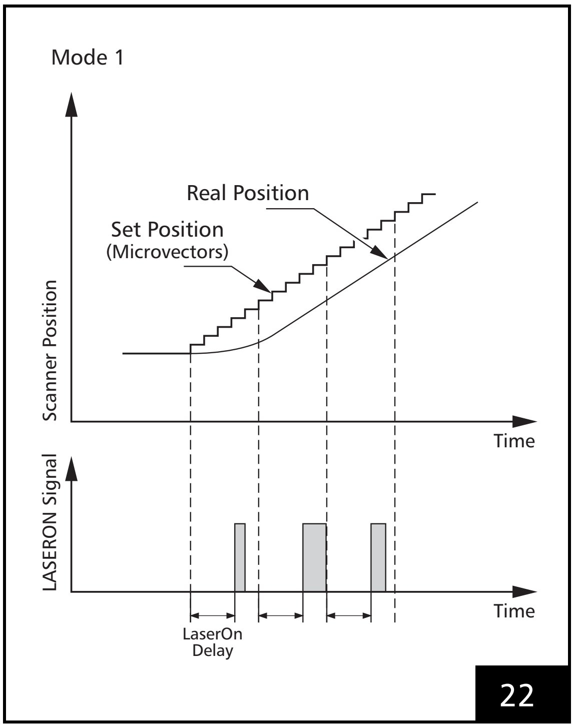
Mode 1

In Mode 1, each image line is treated like a normal vector. That means the movement is split up into a number of microsteps. After an initial acceleration phase, the laser focus will move with an approximately constant velocity along the entire image line. The pixels will be marked in passing.

- In this mode, the LaserOn delay can be set to a few microseconds.
- The pixel period T can be chosen according to the suitable laser frequency. It is typically about 50 µs.
- The entire image will be slightly shifted because of the lag between the set position and the real position. This can be compensated by adjusting the start position of each image line.
- To suppress distortions during the acceleration phase, some idle pixels (with zero pulse width) can be inserted at the beginning of each image line.



Timing of laser and scanner control in Pixel Output Mode 0



Timing of laser and scanner control in Pixel Output Mode 1
The pixel output period in this example is T = 50 µs (= 5 microsteps).

Laser Control

Black-And-White Images

For black-and-white images, the pulse width will be set either to zero ("black" pixel) or to a suitable constant value ("white" pixel).

To enhance the contrast, several pulses (instead of only one) per pixel can be set. To do this, the pulse width of the LASER1 signal (Q-Switch) has to be set accordingly – see **set_laser_timing** (page 109).

Note: If several pulses are used for one pixel, scanning mode 0 should be used (see page 49).

Greyscale Images

The laser energy discharged at each pixel position can be varied in three different ways:

- by varying the laser pulse width (A),
- by varying the laser power at each pixel (B) or
- by varying the number of laser pulses per pixel (C).

The most suitable method depends greatly on the type of laser employed.

(A) Variation Of Laser Pulse Width

Some laser types allow varying the pulse width (duration) of each laser pulse. The command **set_pixel** defines the pulse width of the LASERON signal for each pixel in units of 1/8 µs. (The pulse width of the LASERON signal is equal to the pulse width of the laser pulse, if pulse width and output period of the LASER1 / LASER2 signals are larger then the pixel step period T.)

- It is recommended that some experiments be performed to determine the appropriate pulse width range for producing a smooth greyscale. The resulting pixel "colors" (greyscale values) strongly depend on the employed material and on the laser.

(B) Variation Of Laser Power

Q-switched lasers in particular produce very short laser pulses which cannot be easily varied in length. For such lasers, it is more suitable to create greyscale values by modulating the laser pulse energy. One possibility is to vary the laser *power* with a suitable modulating device (e.g. an acousto-optical modulator).

The command **set_pixel** allows specification of a 10-bit output value for each pixel. The value is transferred to the ANALOG OUT2 port of the RTC®4 synchronously to the pixel output – also see the section "Timing", page 48.

The ANALOG OUT2 port is optionally available at the 9-pin D-SUB Laser Connector. The output range is 0 V … 10 V. Please refer to the section "Laser / Analog Output Ports (9-Pin Laser Connector)", page 59.

(C) Variation Of The Number Of Laser Pulses Per Pixel

Another way to modulate the laser pulse energy of Q-switched lasers with short laser pulses is to vary the number of laser pulses per pixel. For this purpose one can vary either the length (pulse width) of the LASERON signal, which serves as a gate for the individual laser pulses (see figure 20 on page 48) or (with **set_laser_timing**) the pulse width and the output period of the individual laser pulses.

Note: For this method, scanning mode 0 should be used (see page 49).

5.8 Timed Jump And Mark Commands

The normal vector commands (jump commands and mark commands) are processed by the RTC®4 in such a way that the laser focus moves along the surface of the image field with a defined *speed*, the *jump_speed* or the *mark_speed*. This is fine for most laser marking and laser material processing applications.

However, some applications require that each jump or mark vector consumes exactly the same amount of *time*, regardless of its length. In this case, it is practical to specify the *duration* of the jump, rather than the jump speed.

The commands (see page 121)

- **timed_jump_abs**
- **timed_jump_rel**
- **timed_mark_abs**
- **timed_mark_rel**

allow specification of the duration of the jump (mark) command with an accuracy of 10 µs (the output period of the microvectors) and in the range from 10 µs to 655350 µs (≈ 0.6 s).

The jump (mark) vector will be split up into a number of microsteps that correspond exactly to the specified time. (Also see the section "Microsteps" on page 15.) Of course, this means that the *jump (mark) speed*, i.e. the velocity of the laser focus (and the angular velocity of the movement of the mirrors) will depend on the length of the vector.

SCANLAB
innovators for industry

Notes

- After a timed jump (mark) command, a *jump delay*, a *mark delay* or a *polygon delay* is inserted, just like after a normal jump (mark) command. That means the total time for the command is the *sum* of the specified time and the corresponding delay.
  (Also see the section "Scanner Delays", page 17.)
- The timed jump and mark commands can be used for 2D vectors only.

5.9 Automatic Self-Calibration

What It Achieves

Environmental fluctuations, especially temperature changes, can cause galvanometer scanner drift, i.e. a shift (offset drift) and an increase or decrease in size (gain drift) of the working image field. Therefore, high-precision applications should only be started up when the scanners have reached their operating temperature. In addition, the magnitudes of environmental fluctuations, e.g. operating temperature changes, to which the scan head is exposed should be kept as small as possible. For applications that are subjected to unavoidable environmental fluctuations, but nevertheless require high long-term repeatability, SCANLAB provides scan heads with automatic self-calibration (optional for apertures ≥ 14 mm). Scan heads delivered with this option have an internal reference sensor system. This reference system provides the RTC®4 software with the ability to automatically calibrate the galvanometer scanner position detectors any time the user deems it appropriate. Thus, drift effects can be reliably compensated and positioning accuracy is maintained over long periods of time. Remaining long-term drift effects are the same order of magnitude as short-term repeatability.

How It Works

Automatic self calibration of the scan head is started via the command `auto_cal(...,1)`.

During calibration, the scan head automatically seeks several reference positions within the scannable area that are defined by the internal sensor system. The seek values are determined and compared with fixed reference values. From the resulting deviations, offset and gain compensation factors are calculated. These compensation factors are immediately made available for use in all future positioning tasks until the `auto_cal(...,1)` is called again or until compensation is shut off via `auto_cal(...,2)`. The entire calibration procedure takes place in about 5 seconds.

The `auto_cal(...,0)` command is available for determining the reference values. The resulting output coordinates for the various sensor positions are stored in non-volatile memory on the RTC®4 board. This way, the values are not lost when the system is shut down.

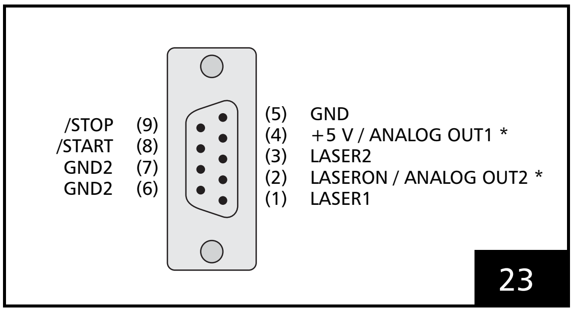
The laser must be switched off during the calibration procedure and no other commands are transferred to the scan head until the calibration is completed.

For further information, see **auto_cal**, page 74.

6 Electrical Connections

6.1 Laser Connector

Figure 23 shows the pin out of the 9-pin D-SUB connector for the laser. The signals at pins (2) and (4) are selected via jumpers. Please refer to the section "Laser / Analog Output Ports (9-Pin Laser Connector)" on page 59.



Pin out of the 9-pin female D-SUB connector for the laser (VB3)
* depends on jumper settings

Analog Output ports

The RTC®4 provides two general purpose analog output ports, ANALOG OUT1 and ANALOG OUT2.

They can be loaded directly or via a list with the commands **write_da_x** / **write_da_x_list** (see page 124), respectively. The output resolution of both ports is 10 bits.

The output voltage range of the ANALOG OUT1 port can be set to either 0 V … 2.56 V or 0 V … 10 V. The range of the ANALOG OUT2 port is fixed at 0 V … 10 V.

Both signals are referenced to PC ground (GND). The maximum current load of both signals is 5 mA.

If jumper X6 has been configured to replace the ANALOG OUT 1 signal with +5 V, then the maximum current load at pin (4) of the laser connector is 100 mA.

Further input and output ports are provided by the optional RTC®4 I/O Extension Board from SCANLAB (see "Options", page 57).

Laser Signals

The output signals LASER1 and LASER2 depend on the selected laser control mode:

	CO_2 Mode	YAG Modes 1/2
LASER1	Modulation pulse 1	Q-Switch signal
LASER2	Modulation pulse 2	FirstPulseKiller

Please refer to the laser timing diagrams on page 33 (CO_2 Mode) and page 36 (YAG Modes).

All laser outputs (LASERON, LASER1 and LASER2) are digital TTL level signals (alternatively active-high or active-low, see page 59) and are referenced to GND2. The maximum current load is 10 mA.

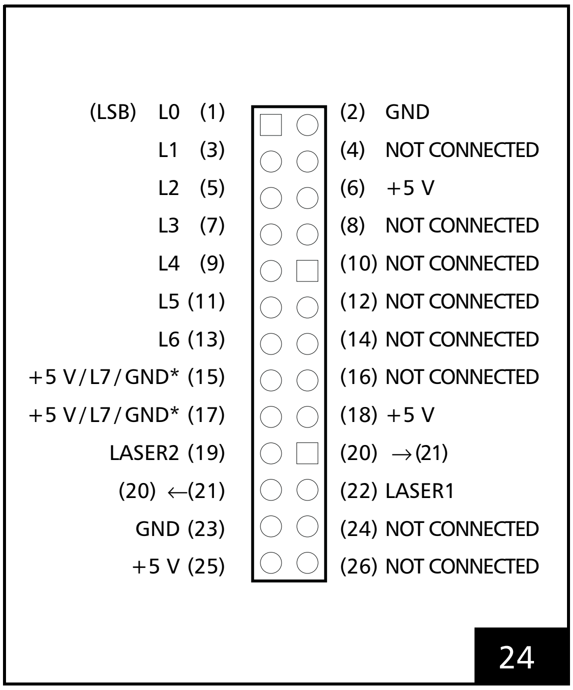
If the RTC®4 is supplied with optoelectronic couplers, GND and GND2 are optically decoupled. Otherwise GND and GND2 are identical. See chapter 7 "Options", page 57.

External Control Signals

The external control signals /START and /STOP (TTL active-low) are referenced to GND. Both input signals are connected internally to +5 V via pull-up resistors. Please refer to the section "External Control Inputs", page 13.

6.2 Laser Extension Connector

The 26-pin connector LASER EXTENSION on the RTC®4 board provides a buffered 8-bit digital output port (L0 to L7). The pins (15) and (17) must be configured with jumpers. For details please refer to the section "Digital Output Port (Laser Extension Connector)", page 60.



Pin out of the LASER EXTENSION connector
* depends on jumper settings

8-Bit Digital Output Port

The buffered 8-bit digital output port (TTL level) is intended for YAG lasers with a digital lamp current control. However, it can be used for any other purpose as well.

The commands **write_8bit_port** and **write_8bit_port_list** (see page 123) load the digital output port with an 8-bit value. The output is in high-impedance mode (tri-state) until an initial value is assigned to it.

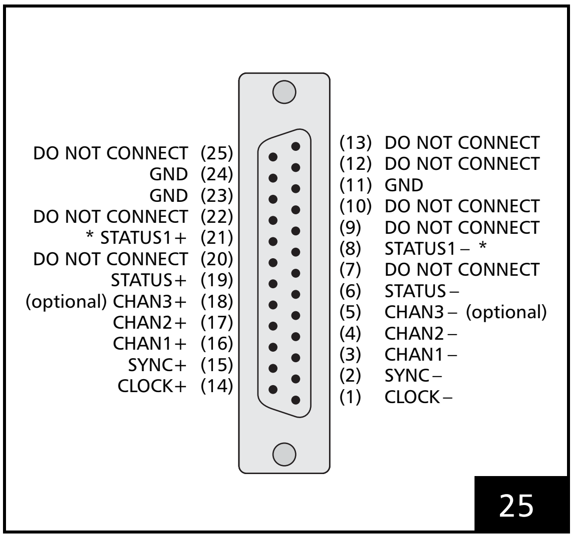
The most significant bit (MSB) (L7) of the output value can be used for other purposes, e.g. for controlling a shutter. To do this, the MSB can be assigned to an extra pin on the LASER EXTENSION connector (see page 60).

Laser Signals

Like the laser signals of the laser connector, the output signals LASER1 and LASER2 depend on the selected laser control mode (see "Laser Signals", page 52).

6.3 Primary Scan Head Connector

The 25-pin D-SUB connector for the scan head is compatible with most scan heads which use the XY2-100 standard. The pin out is shown in figure 25 on page 54.



Pin out of the 25-pin female D-SUB connector for the scan head (VB2)

* The STATUS1± channel can only be used together with intelli*SCAN*® scan systems, else: "DO NOT CONNECT"

Control Signals

Data channels CHAN1 through CHAN3 transmit control values to the scan head. The SYNC and CLOCK channels transmit synchronization and clock signals to the scan system.

The CHAN3 channel is optionally provided for controlling a third axis in a 3-axis system.
Please contact SCANLAB for further information (also see chapter 7 "Options", page 57).

For additional information, please contact SCANLAB.

Status Signals

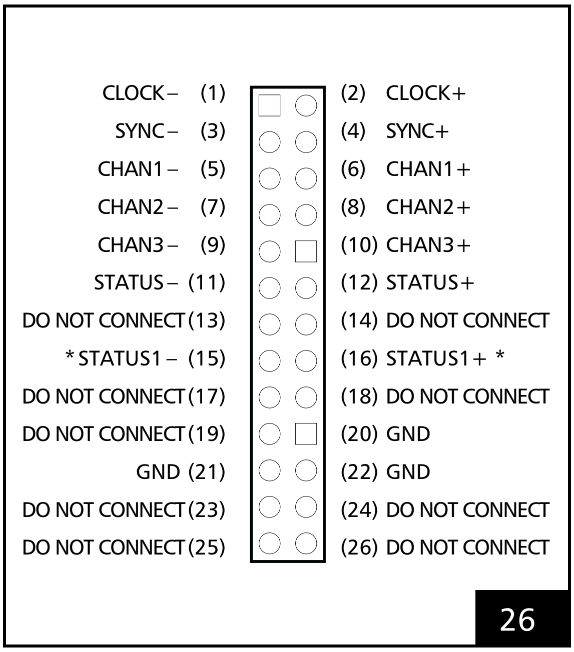
The STATUS channel receives XY2-100 standard compliant status signals returned by the scan system. Consult your scan system's operating manual to determine which status signals are generated by your scan system and how they can be applied for monitoring purposes.

6.4 Secondary Scan Head Connector (Optional)

The 26-pin connector "2. SCANHEAD" on the RTC®4 board is designed for connecting a second scan head in a double scan head configuration. The pin out of this connector is shown in figure 26 on page 54.

The signals on this connector are optional and must be enabled by SCANLAB. Please contact SCANLAB for further information. Also see chapter 7 "Options", page 57.

SCANLAB recommends using an additional slot cover for connecting the second scan head. A suitable slot cover with a 25-pin D-SUB connector – with a pin out as shown in figure 25 on page 54 – is available from SCANLAB.



Pin out of the on-board "2. SCANHEAD" connector

* The STATUS1± channel can only be used together with intelli*SCAN*® scan systems, else: "DO NOT CONNECT"

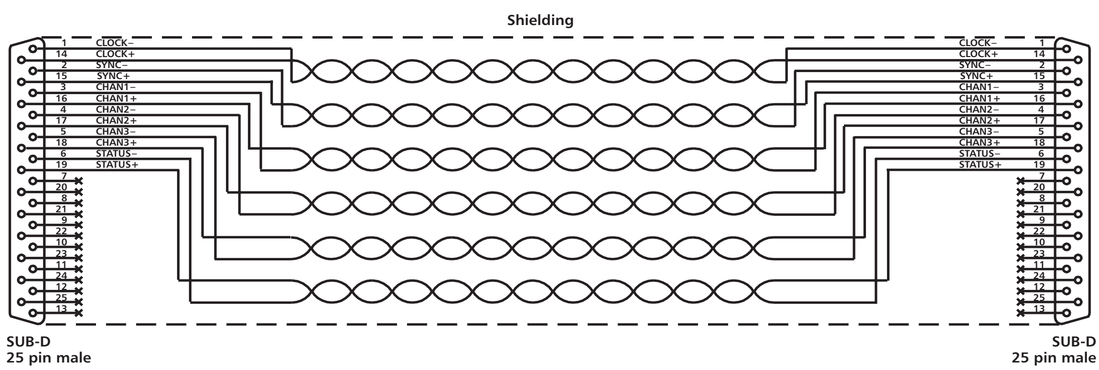
6.5 Data Cable

To connect the scan head to the RTC®4, a data cable is required. This cable is not included in the package. SCANLAB recommends the following design:

- The data cable must have identical 25-pin (male) D-SUB connectors at both ends. The cable consists of six twisted cable pairs for the following six channels: SYNC±, CHAN1±, CHAN2±, CHAN3±, STATUS± and CLOCK±. The channel CHAN3± is provided for optionally controlling a third axis.
- The data cable must have coaxial copper braided shielding.
- The data cable should not be longer than 10 m. If a longer data cable is needed, the signal timing should be adjusted to ensure correct communication between the scan head and the RTC®4. See the command **set_piso_control** (page 112) for details. The cable length must not exceed 20 m.
- The D-SUB connectors must have fully shielded metal housings.
- The electrical connection of the cable's braided shielding to the D-SUB housing should *not* be implemented as a wire. Instead, the cable's braided shielding should be *coaxially* connected to the D-SUB housing via shielded clamps.
- The data cable's controller end must be fitted with a ferrit ring (e.g. Würth WE 742 711 32).

Figure 27 shows the data cable layout and pin assignments.

Some scan heads use a single connector to provide both the power supply voltages and the data signals. For these scan heads, SCANLAB recommends implementing a cabling solution that allows the use of separate cables for data and power. The data-section of such a cabling solution should be designed to accommodate the data cable shown in figure 27.



27

Data cable layout and pin assignments

6.6 Optical Data Interface (Optional)

Optionally, the RTC®4 is equipped with an optical data interface and ST ports for data transfer via a duplex optical fiber cable (see figure 2 on page 9). This allows scan systems to be controlled via optically transferred signals. Optical data transfer provides the advantages of immunity to electrical interference (ESD and EMC) and complete galvanic separation.

The data interface converts electrical signals into optical signals with a wavelength of 650 nm. The optical signals are guided along a polymer optical fiber cable to an optical data interface (optionally) installed in the scan system, where they are converted back into electrical signals. Compliant with the XY2-100-O protocol, the optical signals are transferred bidirectionally every 10 µs over 3 data channels.

The optical fiber cable is not included in the package. SCANLAB recommends use of a 1 mm diameter duplex polymer optical fiber (POF) cable with a maximum length of 30 m. The optical fiber cable must be fitted with two ST plugs at each end. Use the optical fiber cable to cross-connect (*not* 1:1) both ST ports of the RTC®4' primary scan head connector with both ST ports of the scan system.

If the signals for a second scan head are enabled, also a second scan system can be connected via a polymer optical fiber.

DIGITAL OUT0	(1)	(2)	DIGITAL IN0
DIGITAL OUT1	(3)	(4)	DIGITAL IN1
DIGITAL OUT2	(5)	(6)	DIGITAL IN2
DIGITAL OUT3	(7)	(8)	DIGITAL IN3
DIGITAL OUT4	(9)	(10)	DIGITAL IN4
DIGITAL OUT5	(11)	(12)	DIGITAL IN5
DIGITAL OUT6	(13)	(14)	DIGITAL IN6
DIGITAL OUT7	(15)	(16)	DIGITAL IN7
DIGITAL OUT8	(17)	(18)	DIGITAL IN8
DIGITAL OUT9	(19)	(20)	DIGITAL IN9
DIGITAL OUT10	(21)	(22)	DIGITAL IN10
DIGITAL OUT11	(23)	(24)	DIGITAL IN11
DIGITAL OUT12	(25)	(26)	DIGITAL IN12
DIGITAL OUT13	(27)	(28)	DIGITAL IN13
DIGITAL OUT14	(29)	(30)	DIGITAL IN14
DIGITAL OUT15	(31)	(32)	DIGITAL IN15
DO NOT CONNECT	(33)	(34)	DO NOT CONNECT
DO NOT CONNECT	(35)	(36)	BUSYOUT
+5 V	(37)	(38)	+5 V
GND	(39)	(40)	GND

28

Pin-out of the 40-pin EXTENSION 1 connector

6.7 EXTENSION 1 Connector

The EXTENSION 1 connector provides a 16-bit digital TTL input and a buffered 16-bit digital TTL output. The pin-out is shown in figure 28.

16-Bit Digital Input and Output

The 16-bit digital output port can be used, for example, for controlling a workpiece transport system. Signals of the transport system, of workpiece recognition systems etc. can be read via the 16-bit input port. A group of related commands facilitates straight-forward incorporation of port-based operations into application programs:

The **write_io_port_list** and **write_io_port** commands specify the digital output values. The 16-bit digital output is in high-impedance mode (tri-state) until an initial value is assigned to it.

The **read_io_port** or **get_io_status** commands read the current values of the digital input or output ports.

The **set_io_cond_list** and **clear_io_cond_list** commands allow digital output values to be influenced by digital input values: Depending on the current value of the digital input, individual bits of the output port are set or cleared.

The commands **list_jump_cond** and **list_call_cond** can be used to call subroutines or jump within a list – depending on the current value of the digital input (also see "Structured Programming", page 45).

BUSY Status

The BUSY status signal, available at pin (36), is HIGH when an application is currently executing.

7 Options

The following options are available for the RTC®4. Please contact SCANLAB for further information.

Note

- The command **get_rtc_version** (see page 85) provides information about the options currently installed on your RTC®4 board.

- **I/O Extension Board**
  An optional I/O Extension Board for the RTC®4 is available from SCANLAB. It includes
  - a 16-bit digital input, 4 bits are opto-decoupled
  - a 16-bit digital output, 4 bits are opto-decoupled
  - 4 differential analog outputs, each with 10-bit resolution
  - 4 analog outputs, each with 10-bit resolution
- **Second Scan Head Connector**
  This option allows controlling two scan heads simultaneously, e.g. in a double scan head configuration.

  Also see chapter 5.3 "Using Two Different Correction Files", page 42.
- **Processing-on-the-fly**
  Control signal inputs and outputs for Processing-on-the-fly applications are optionally available on-board.
- **Controlling the Third Axis of a 3-Axis System**
  The optional third data channel can be used, for example, for controlling a vario*SCAN* module in a 3-axis scan system.
- **Optical Data Interface**
  SCANLAB's Optical Data Interface enables transmission of data via a polymer optical fiber linking the RTC®4 to an optical interface (optionally) integrated in the scan system (see the section "Optical Data Interface (Optional)" on page 56).
- **Optoelectronic Couplers**
  The digital laser control signals on the 9-pin laser connector can be optically decoupled from the PC ground (see the section "Laser Connector" on page 52).

8 Installation And Start-Up

Installation of the RTC®4 consists of the following steps:

(1) configuring the RTC®4 jumpers

(2) installing the RTC®4

(3) installing the RTC®4 software drivers

Please proceed in the order described in the following sections.

Caution!

- RTC®4 boards cannot be used simultaneously with RTC®3 boards in the same PC.
- Prior to installing software drivers for WINDOWS XP and WINDOWS 2000, previously installed drivers of the old, discontinued version must be de-installed.
- For installing the old, discontinued driver version for WINDOWS XP SP1 / 2000 / NT 4 and WINDOWS ME / 98 / 95 the software drivers must be installed ***before*** the RTC®4 board is installed into the PC.

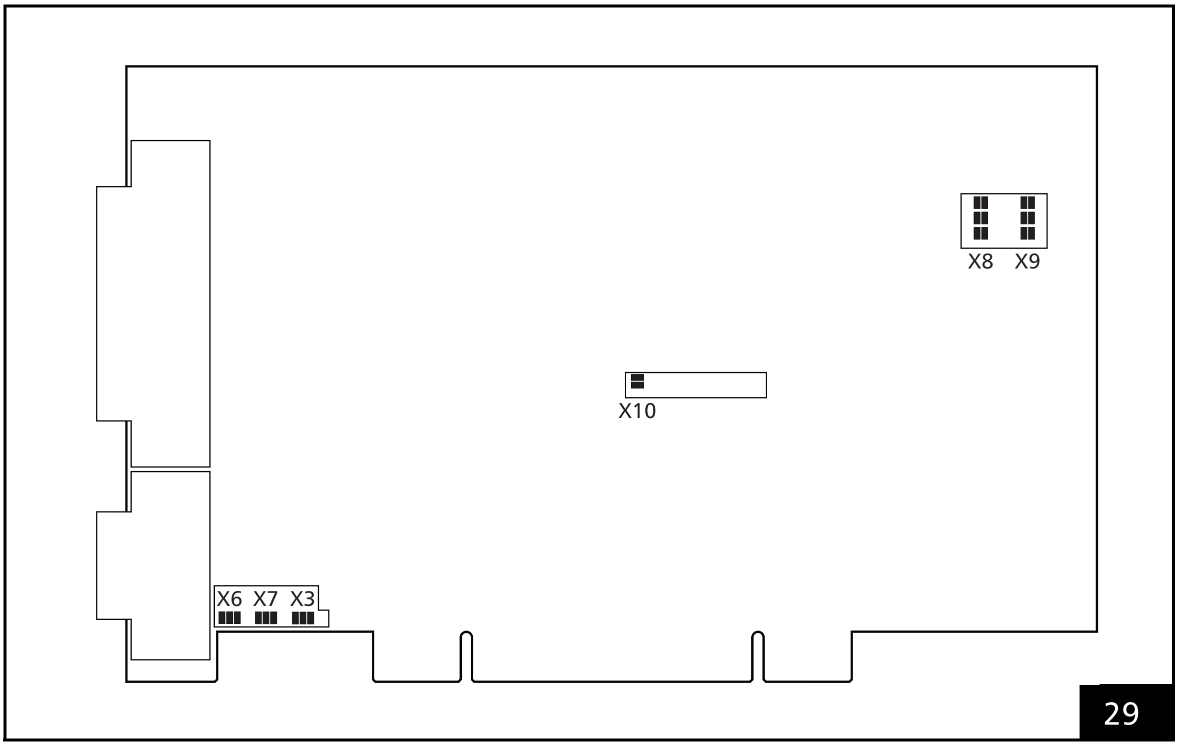
8.1 Jumper Settings Overview

The factory settings for the RTC®4 jumpers are listed on the supplement at the end of this manual.

- If you do not want to change the factory settings, proceed with chapter 8.4 "Installing the Drivers", page 61.

The following jumpers can be set by the user (see figure 29 on page 58):

- Jumper for the laser signals (**X10**) (active-high or active-low)
- Jumpers for the 9-pin D-SUB laser connector (**X6**, **X7**, **X3**)
- Jumpers for the 8-bit digital output port on the LASER EXTENSION connector (**X8**, **X9**) (optional)



Jumper positions on the RTC®4 board

8.2 Changing The Jumper Settings

Caution!

- The jumpers are soldered junctions. If you need to change the jumper settings, you'll have to use a soldering-iron.
  Please be careful not to damage the electronics on the board!

TTL Laser Signal Level

The RTC®4 board permanently generates the following laser signals (also see page 32 through page 37):

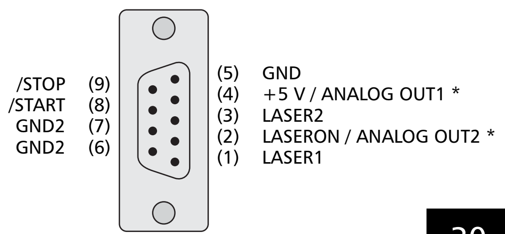
- LASERON
- LASER1 / Q-Switch
- LASER2 / FirstPulseKiller
- Stand-By1, Stand-By2

All laser signals can be set to either *active-low* or *active-high* logic via jumper **X10**:

Jumper X10	Laser Signal Logic
open	active-low
closed	active-high

Active-low means that a logical 1 ("Laser On", for instance) is represented by a LOW level (0 V, TTL). Active-high means a logical 1 is represented by a HIGH level (+5 V, TTL).

- Set the jumper according to the specifications of your laser control. Please refer to the documentation of your laser.



30

Pin out of the 9-pin female D-SUB laser output port connector
* depends on jumper settings

Laser / Analog Output Ports

(9-Pin Laser Connector)

The signals at pins (2) and (4) of the 9-pin laser output port connector (see figure 30 on page 59) must be selected via jumpers **X6**, **X7** and **X3.** The following table summarizes the jumper settings. For further information please refer to section "How To Set The Jumpers", next.

Jumper	Function	Position 1-2	Position 2-3
X6	selects the signal at pin (4)	+5 V (max. 100 mA)	ANALOG OUT1
X7	selects the signal at pin (2)	LASERON	ANALOG OUT2
X3	sets the output range of the ANALOG OUT1 signal	0 … 2.56 V	0 … 10 V

How To Set The Jumpers

Jumpers X6 and X3

- If you want to use the signal ANALOG OUT1 (e.g. for lamp current control), set the jumper **X6** to position 2-3. Then specify the output voltage range with the Jumper **X3** (see table).
- Alternatively, the output signal at pin (4) can be set permanently to +5 V (jumper **X6** in position 1-2). In this case, jumper **X3** can be ignored.

Jumper X7

- If you need the TTL signal LASERON for laser control, set the jumper **X7** to position 1-2.
- If your laser requires only the signals LASER1 and LASER2, you can use pin (2) for the signal ANALOG OUT2 instead. To do this, set the jumper **X7** to position 2-3.

  Note: The ANALOG OUT2 signal is also available via the MARKING ON THE FLY connector. (RTC®4 with Processing-on-the-fly option only)

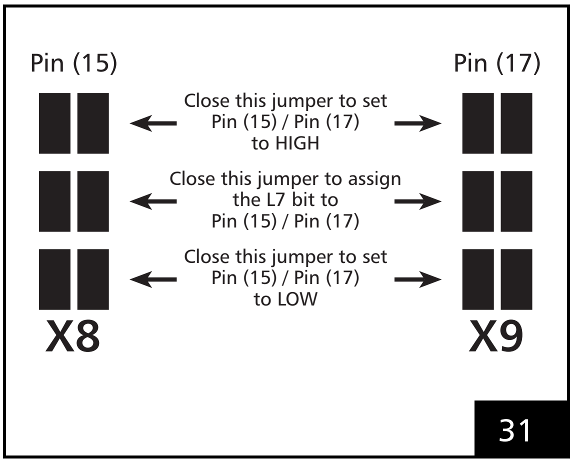
For a description of the analog output signals please refer to chapter "Analog Output ports", page 52.

Digital Output Port
(Laser Extension Connector)

The 8-bit digital output port is available via the LASER EXTENSION connector – see figure 24 on page 53, odd numbered pins (1) to (17).

The port can be loaded with an 8-bit value by the commands **write_8bit_port** and **write_8bit_port_list** (see page 123).

The pins (15) and (17) of the output port have to be configured via jumpers **X8** and **X9**. The most significant bit (L7) of the 8-bit output value can be assigned to pin (15) or to pin (17). Alternatively, each of the two pins can be set permanently to +5 V (HIGH) or to GND (LOW level).



Caution!

- Make sure that not more than **one** of the three jumpers at **X8** is closed. Also make sure that not more than **one** of the three jumpers at **X9** is closed.
  Closing more than one jumper at positions **X8** or **X9** will damage the electronics on the board.

Examples

- If the L7 bit is assigned to pin (15), the full 8-bit output value is available on the output port (odd numbered pins (1) to (15) of the LASER EXTENSION connector).
- Setting pin (15) permanently to HIGH results in an offset of 128 for the output value and restricts the output values to (128 … 255).
- Setting pin (15) to LOW restricts the output value to 0 … 127.
- The L7 bit can be used for other purposes by assigning it to pin (17).

8.3 Installing the Hardware

Caution!

- Please carry out the installation in an area that complies with the Electro Static Discharge (ESD) directives.
- RTC®4 and RTC®3 boards in the same computer cannot be operated simultaneously.
- Do not touch the contacts of the RTC®4.
- Protect the board from humidity, dust, corrosive vapors and mechanical stress.

- Remove all diskettes and CDs from your PC.
- Shut down the operating system and switch off the PC. Disconnect the PC from the mains.
- Remove the housing of the PC.
- Locate a free PCI slot and remove the slot cover.
- Remove the RTC®4 from the antistatic bag. Do not touch the contacts of the board.
- Install the RTC®4 into the PCI slot. Please observe the instructions in the manual of your PC.
- Close the housing of the PC.
- Connect the 25-pin D-SUB connector to the scan head via a data cable. Please refer to chapter 6.5 "Data Cable", page 55, for the specifications and pinouts of the data cable.
- Connect the 9-pin D-SUB connector on the RTC®4 to the laser via a suitable interface.
- Check all connections and turn on the PC.

8.4 Installing the Drivers

For installing RTC®4 drivers for WINDOWS XP and WINDOWS 2000 proceed as follows:

Note

- RTC®4 boards cannot be used simultaneously with RTC®3 boards in the same PC.
- Prior to installing software drivers for WINDOWS XP and WINDOWS 2000, previously installed drivers of the old, discontinued version must be de-installed. The de-installation procedure is described in the "Readme.txt" file of the RTC®4 software package.
- The installation procedure for installing the old, discontinued version for WINDOWS XP SP1 / 2000 / NT 4 and WINDOWS ME / 98 / 95 is also described in the "Readme.txt" file of the RTC®4 software package.

(1) After the RTC®4 board has been inserted, start the computer.

(2) If the "Add Hardware Wizard" does not come up automatically, call him from the Control Panel.

(3) In the "Add Hardware Wizard" specify the directory which includes the drivers for WINDOWS XP and WINDOWS 2000.

(4) If the old driver was previously installed (and has been de-installed prior to installing the new drivers), now restart the computer once more.

- Additionally to installing the drivers, the RTC®4 software (application-support DLL, correction files and DSP program file(s)) must be installed. Therefore, please follow the instructions in chapter 9, page 62.

8.5 Start-Up and Functionality Test

Danger!

- **Always turn on the RTC®4 and the power supply for the scan head first before turning on the laser. Otherwise, there is the danger of uncontrolled deflection of the laser beam.**

The HPGL conversion program HPGL.EXE is supplied for testing control of the scan head. This program lets you load graphics files in Hewlett Packard's HPGL format for transfer to the RTC®4.

- Copy the HPGL converter and the supplied demo file into the same directory as the driver DLL, the RTC®4 DSP program file(s) and correction file(s).
- Start the HPGL converter.
- Under `Options|Correction` select a correction file and under `File|HPGL-File select a plot file` (Demofile).
- Start output via `Mark|Start Marking`.

9 Software

9.1 Installing the Software

The RTC®4 application-support DLL supports the RTC®4 under the Microsoft operating systems WINDOWS XP and WINDOWS 2000. The DLL provides all necessary functions for operating the RTC®4. The commands and functions of the DLL can be integrated into the customer's application software as described in chapter 9.2.

To install the RTC®4 software, follow these steps:

- Copy the application-support DLL (`RTC4DLL.DLL`) included in the RTC®4 software package to the directory in which the application software will be started, or to the WINDOWS directory.
- Copy the correction file(s) (`*.CTB`) and the RTC®4 DSP program file(s) (`*.HEX`) to the harddisk of your PC (existing correction files can still be used; do not overwrite customized correction files!). SCANLAB recommends storing these files in the directory in which the application software will be started.
- In order to use the commands and functions of the DLL in your application program, you will have to initialize them. Chapter 9.2 describes the calling convention of the DLL and shows how to initialize the functions and procedures of the DLL in application programs written in Pascal, C or Visual Basic.

Caution!

- Carefully check your application program before running it. Programming errors can cause a break down of the system. In this case, neither the laser nor the scan head can be controlled.

9.2 DLL Calling Convention

The DLL calling convention is **stdcall**.

To facilitate importing the commands of the DLL into a C, Pascal or Visual Basic application, the RTC®4 software package contains corresponding utility files.

The following sections describe how to use these files in your particular software environment.

Caution!

- Some of the commands included in the utility files can only be used with the appropriate optional hardware configuration of the RTC®4 – also see chapter 7 "Options", page 57.

  Make sure to use only the commands supported by your version of the RTC®4. Please refer to the descriptions of the individual commands in chapter 10.

  The command **get_rtc_version** (page 85) provides information about the options currently installed on your RTC®4 board.

Pascal

Use the file `RTC4Import.pas` as a unit and call the RTC®4 commands you need, like

```
goto_xy(1000, 2500);
```

for performing a jump to location 1000, 2500.

Basic

Include file `RTC4Import.bas` into your project and call the RTC®4 commands you need, like

```
call goto_xy(1000, 2500)
```

for performing a jump to location 1000, 2500.

C

In C, you can choose either *implicit linking* – also known as static load or load-time dynamic linking – or *explicit linking* – also known as dynamic load or run-time dynamic linking.

Implicit Linking

To accomplishing implicit linking, include the header file `RTC4impl.h` and link with the (Visual C++) import library `RTC4DLL.LIB` for building the executable.

Call the RTC®4 commands you need like

```
goto_xy(1000, 2500);
```

for causing a jump to location 1000, 2500.

Explicit Linking

To accomplishing explicit linking, include the header file `RTC4expl.h`. Before calling any RTC®4 function, initialize the DLL by calling the function `RTC4open` (which is defined in the file `RTC4expl.cpp`).
When you are finished using the RTC®4, close the DLL by calling the function `RTC4close` (also defined in the file `RTC4expl.cpp`).

For building the executable, link with the file `RTC4EXPL.OBJ`, which you can generate from the source code `RTC4expl.cpp`.

Call the RTC®4 commands you need, like

```
goto_xy(1000, 2500);
```

for causing a jump to location 1000, 2500.

Pros and Cons of Implicit and Explicit Linking

	Implicit Linking	Explicit Linking
Necessary Files	`RTC4impl.h`, `RTC4DLL.LIB`	`RTC4expl.h`, `RTC4expl.cpp`
Advan-tage	Easiest linking method	Eliminates the need to link the application with an import library
Negative Aspect	Need to link the application with a compiler-specific import library	Need to initialize (`RTC4open`) and close (`RTC4close`) the DLL

9.3 Initializing the RTC®4

At the beginning of each RTC®4 application program, you need to perform the following steps:

(1) Download the correction file(s) to the RTC®4 (command **load_correction_file**, page 96).

See chapter 5.3, page 42, for information about using two different correction files.

(2) Download the DSP program file (command **load_program_file**, page 97).
The RTC®4 signal processor starts automatically after this command.

Note
When the RTC®4 processor starts, the scanner output position is set to the point (0|0). This point might be shifted by the image field correction table.
Therefore it is recommended to load the correction file(s) *before* loading the program file to make sure that the initial output position is correct.

(3) Set the laser mode (command **set_laser_mode**, page 109).

(4) Assign the correction file(s) to the scan head control port(s) if necessary.
The default assignment (after each reset of the RTC®4) is:

- The first scan head control port uses correction table #1.
- The second scan head control port is off.

Use the command **select_cor_table** (see page 103) to change to a different assignment.

(5) Define the scanner delay mode (variable polygon delay or constant polygon delay; command **set_delay_mode**, page 105).

(6) Load a table for the variable polygon delay if necessary (command **load_varpolydelay**, page 98).

(7) Set the FirstPulseKiller length (YAG only) (command **set_firstpulse_killer**, page 106).

(8) Set the stand-by pulses (usually CO_2 only) (command **set_standby**, page 117).

(9) Load the list(s).

(10) Enable the external start input if necessary (command **set_control_mode**, page 104).

The remaining settings (laser timing, laser delays, scanner delays, jump speed and marking speed) are set by means of list commands.

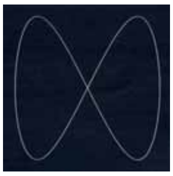
9.4 Demo Programs

The RTC®4 software package contains six different program code samples (file `DEMO.ZIP`). The purpose of these samples is to demonstrate usage of the diverse control and list commands.

The samples are written in the C language. They show the necessary calling sequences of the RTC®4 commands, which you can easily translate into your preferred programming language.

The table below gives an overview of the characteristics of the demo programs Demo1 to Demo5.

The demo program Demo_intelliSCAN.CPP demonstrates the communication between RTC®4 and intelli*SCAN*® via a simple marking example.

Filename	**Demo1**	**Demo2**	**Demo3**	**Demo4**	**Demo5**
Marking Task	Marking a square and a triangle 	Lissajous figures 	Archimedean spirals 	Raster image reproduction 	Marking squares and triangles 
Laser type	CO_2	YAG	YAG	CO_2	YAG
Marking method	vector	vector	vector	raster	vector
DLL linking	implicit	explicit	explicit	explicit	explicit
List handling	use of a single list buffer	Circular Queue Mode	Continuous Transfer	Continuous Transfer	use of both list buffers
External control inputs					✓
Structured list programming					✓
Exception handling (by using **stop_execution**, **stop_list** and **restart_list**)		✓	✓		

The source files `DEMO1.CPP` and `DEMO_intelliSCAN.CPP` are listed in the following section.
All demo programs (C sources and executables) can be found in the corresponding folder of the RTC®4 software package.

Demo 1: Marking a square and a triangle

```
//  File
//      DEMO1.cpp
//
//  Abstract
//      A console application for marking a square and a triangle
//      by using a CO2 laser
//
//  Author
//      Bernhard Schrems, SCANLAB AG
//
//  Comment
//      Besides demonstrating of how to initialize the RTC4 and how to
//      perform marking, this application also shows how to implicitly
//      link to the RTC4DLL.DLL. For accomplishing implicit linking -
//      also known as static load or load-time dynamic linking of a DLL,
//      the header file RTC4impl.H is included and for building the
//      executable, you must link with the (Visual C++) import library
//      RTC4DLL.LIB.
//      In case the operating system does not find the RTC4DLL.DLL on
//      program startup, it will respond with a corresponding error
//      message and it will terminate the program.
//
//  Necessary Sources
//      RTC4impl.H, RTC4DLL.LIB, MESSAGE.CPP
//      NOTE: RTC4DLL.LIB is a Visual C++ library.
//
//  Environment: Win32
//
//  Compiler
//      - tested with Visual C++ 5.0 and Visual C++ 6.0


// system header files
#include <stdio.h>
#include <conio.h>

// RTC4 header file
#include "rtc4impl.h"


struct polygon{
    short xval, yval;
};


#define R (20000)

polygon square[] = {
    -R, -R,
    -R,  R,
     R,  R,
     R, -R,
    -R, -R
};

polygon triangle[] = {
    -R,  0,
     R,  0,
     0,  R,
    -R,  0
};

#define POLYGONSIZE(figure) (sizeof(figure)/sizeof(polygon))


void ErrorMessage(short ErrorCode);
void draw(polygon *figure, unsigned size);


void main(void *, void *)
{
    short   ErrorCode;

    printf("Polygon Marking with a CO2 laser\n\n");


    // Initialize
    ErrorCode = load_correction_file("cor_1to1.ctb",
                    1,          // table; #1 is used by default
                    1.0, 1.0,   // scale factor
                    0.0,        // rotation in degrees, counterclockwise
                    0.0, 0.0);  // offset in bits
    if(ErrorCode) {
        printf("Correction file loading error: ");
        ErrorMessage(ErrorCode);
        return;
    }
    ErrorCode = load_program_file("RTC4d2.hex");
    if(ErrorCode) {
        printf("Program file loading error: ");
        ErrorMessage(ErrorCode);
        return;
    }
    set_laser_mode(0);      // CO2 mode selected
    set_standby(100*8,      // half of the standby period in 1/8 microseconds
            8);         // pulse width in 1/8 microseconds


    // Timing, delay and speed preset
    set_start_list(1);
    set_laser_timing(100,   // half of the laser signal period
                50,50,  // pulse widths of signals LASER1 and LASER2
                0);     // time base; 0 corresponds to 1 microsecond.
                        // Otherwise, the time base is 1/8 microseconds.
    set_scanner_delays(25,  // jump delay in 10 microseconds
                10,     // mark delay in 10 microseconds
                5);     // polygon delay in 10 microseconds
```

```
    set_laser_delays(100,   // laser on delay in microseconds
                    100);   // laser off delay in microseconds
    set_jump_speed(1000.0); // jump speed in bits per milliseconds
    set_mark_speed(250.0);  // marking speed in bits per milliseconds
    set_end_of_list();
    execute_list(1);


    // Draw
    draw(square, POLYGONSIZE(square));
    draw(triangle, POLYGONSIZE(triangle));


    // Finish
    printf("Finished - press any key to terminate ");
    while(!kbhit()) ;
    (void)getch();
    printf("\n");
    return;
}



//  draw
//
//  Description:
//
//  Function "draw" transfers the specified figure to the RTC4 and invokes
//  the RTC4 to mark that figure, when the transfer is finished. "draw" waits
//  as long as the marking of a previously transferred figure is finished before
//  it starts to transfer the specified figure.
//
//
//      Parameter   Meaning
//
//      figure      Pointer to a polygon array
//                  The first element of that array specifies the first location
//                  from where the figure will be marked until the last location,
//                  which is specified by the last array element.
//
//      size        Amount of polygons the polygon array contains
//                  In case "size" equals 0, the function immediately returns
//                  without drawing a line.
//
//  Comment
//  This function demonstrates the usage of a single list. Using list 1 only
//  means that you can utilize the space of both lists, which equals 8000 entries
//  totally.
//
//  NOTE
//      Make sure that "size" is smaller than 8000.

void draw(polygon *figure, unsigned size) {
    unsigned short  busy, position;
    unsigned        i;

    if(size) {
        do {
            get_status(&busy, &position);
        } while(busy);

        // Only, use list 1, which can hold up to 8000 entries
        set_start_list(1);
        jump_abs(figure->xval, figure->yval);
        for(i = 0, figure++; i < size - 1; i++, figure++)
            mark_abs(figure->xval, figure->yval);
        set_end_of_list();
        execute_list(1);
    }
}
```

Demo_intelliSCAN

```
//  File
//      intelliSCAN.cpp
//  Abstract
//      A console application for demonstrating the communication
//      with the intelliSCAN
//  Author
//      Gerald Schmid, SCANLAB AG
//  Comment
//      Besides demonstrating of how to initialize the RTC4 and how to
//      perform marking, this application also shows how to implicitly
//      link to the RTC4DLL.DLL. For accomplishing implicit linking -
//      also known as static load or load-time dynamic linking of a DLL,
//      the header file RTC4impl.H is included and for building the
//      executable, you must link with the (Visual C++) import library
//      RTC4DLL.LIB.
//  Necessary Sources
//      RTC4impl.H, RTC4DLL.LIB
//      NOTE: RTC4DLL.LIB is a Visual C++ library.
//  Environment: Win32
//  Compiler
//      - tested with Visual C++ 6.0

// system header files
#include <stdio.h>
#include <conio.h>

// RTC4 header file
#include "rtc4impl.h"

#define MaxWavePoints 32768// maximum number of samples per channel
#defineSamplePeriod 1// sampling period [10us]

#defineSendStatus 0x0500// command codes for changing data on the status channel
#defineSendRealPos 0x0501
#defineSendStatus2 0x0512
#defineSendGalvoTemp 0x0514
#defineSendHeadTemp 0x0515
#defineSendAGC 0x0516
#defineSendVccDSP 0x0517
#defineSendVccDSPIO 0x0518
#defineSendVccANA 0x0519
#defineSendVccADC 0x051A

#defineHeadA 1
#defineXAxis 1
#defineYAxis 2
#defineStatusAX 1
#defineStatusAY 2

void delay(void)// 100us delay
{
for (short i=0; i<10; i++)
z_out(0);// dummy control function generates a 10us delay
}

void main(void)
{
long i;
    shortErrorCode;
unsigned short WavePoints;
short Ch1[MaxWavePoints], Ch2[MaxWavePoints];

    // Initialize: load correction and program file
    ErrorCode = load_correction_file("cor_1to1.ctb",
                        1,          // table; #1 is used by default
                        1.0, 1.0,   // scale factor
                        0.0,        // rotation in degrees, counterclockwise
                        0.0, 0.0);  // offset in bits
    if(ErrorCode) {
        printf("Correction file loading error #%d\n",ErrorCode);
        return;
    }

    ErrorCode = load_program_file("RTC4D2.hex");
    if(ErrorCode) {
        printf("Program file loading error #%d\n",ErrorCode);
        return;
    }

// get some status information from the head
control_command(HeadA, XAxis, SendGalvoTemp);// send X galvo temp on X status
channel
control_command(HeadA, YAxis, SendGalvoTemp);// send Y galvo temp on Y status
channel
delay();// wait until command is transfered to the head, executed and new status
data is updated
printf("Temperature X-Galvo: %4.1f degrees centigrade\n",0.1*get_value(StatusAX));
// read X status channel
printf("Temperature Y-Galvo: %4.1f degrees centigrade\n",0.1*get_value(StatusAY));
// 1bit = 0.1 degree

// display AGC voltage of each galvo
control_command(HeadA, XAxis, SendAGC);
control_command(HeadA, YAxis, SendAGC);
delay();
printf("AGC Voltage X-Galvo: %5.2f Volts\n",0.01*get_value(StatusAX));// 1bit =
10mV
printf("AGC Voltage Y-Galvo: %5.2f Volts\n",0.01*get_value(StatusAY));

// display internal status information
control_command(HeadA, XAxis, SendStatus2);
control_command(HeadA, YAxis, SendStatus2);
delay();
printf("Status2 X-Axis: %hX\n",get_value(StatusAX));
printf("Status2 Y-Axis: %hX\n",get_value(StatusAY));

// send back the XY real position on the status channels
control_command(HeadA, XAxis, SendRealPos);   // real position
control_command(HeadA, YAxis, SendRealPos);
```

```
    printf("press any key to start ");
    while(!kbhit());
    (void)getch();
    printf("\n\n");

// mark "SCANLAB", record the measured position bend and transfer it to the PC
    set_laser_mode(1);// set YAG mode 1

    set_start_list(1);// timing, delay and speed preset
set_laser_timing(50,// half of the laser signal period
5,5,// pulse width of signal LASER1 and dummy value
0);     // time base; 0 corresponds to 1 microsecond.
// Otherwise, the time base is 1/8 microseconds.

set_firstpulse_killer_list(8*150);// pulse width of signal LASER2
// timebase is 1/8 microseconds

set_scanner_delays(25,  // jump delay in 10 microseconds
            10,     // mark delay in 10 microseconds
        5);     // polygon delay in 10 microseconds
set_laser_delays(100,   // laser on delay in microseconds
            100);   // laser off delay in microseconds
set_jump_speed(2500.0); // jump speed in bits per milliseconds
set_mark_speed(1000.0); // marking speed in bits per milliseconds
jump_abs(0,0);

save_and_restart_timer();// start marking time measurement
set_trigger(SamplePeriod, StatusAX, StatusAY);// start sampling, set sampling
period and channels

// mark "SCANLAB"
jump_abs(-3040, 380);// S
mark_abs(-3230, 570);
mark_abs(-3610, 570);
mark_abs(-3800, 380);
mark_abs(-3800, 190);
mark_abs(-3610, 0);
mark_abs(-3230, 0);
mark_abs(-3040, -190);
mark_abs(-3040, -380);
mark_abs(-3230, -570);
mark_abs(-3610, -570);
mark_abs(-3800, -380);
jump_abs(-1900, 380);// C
mark_abs(-2090, 570);
mark_abs(-2470, 570);
mark_abs(-2660, 380);
mark_abs(-2660, -380);
mark_abs(-2470, -570);
mark_abs(-2090, -570);
mark_abs(-1900, -380);
jump_abs(-1520, -570);// A
mark_abs(-1520, 190);
mark_abs(-1140, 570);
mark_abs(-760, 190);
mark_abs(-760, -570);
jump_abs(-760, 0);
mark_abs(-1520, 0);
jump_abs(-380, -570);// N
mark_abs(-380, 570);
mark_abs(380, -570);
mark_abs(380, 570);
jump_abs(760, 570);// L
mark_abs(760, -570);
mark_abs(1520, -570);
jump_abs(1900, -570);// A
mark_abs(1900, 190);
mark_abs(2280, 570);
mark_abs(2660, 190);
mark_abs(2660, -570);
jump_abs(2660, 0);
mark_abs(1900, 0);
jump_abs(3040, 0);// B
mark_abs(3610, 0);
mark_abs(3800, 190);
mark_abs(3800, 380);
mark_abs(3610, 570);
mark_abs(3040, 570);
mark_abs(3040, -570);
mark_abs(3610, -570);
mark_abs(3800, -380);
mark_abs(3800, -190);
mark_abs(3610, 0);

jump_abs(0,0);
save_and_restart_timer();// stop marking time measurement
    set_end_of_list();
    execute_list(1);// start marking

    // wait as long the execution is finished
    do {} while(read_status() & 0x010);

if (get_time() > (double)MaxWavePoints*SamplePeriod*1e-5)// marking time > buffer
time ?
WavePoints = MaxWavePoints;
else
WavePoints = (unsigned short)(get_time()/((double)SamplePeriod*1e-5));

get_waveform(1, WavePoints, &Ch1[0]);  // get the waveforms from channel 1&2
get_waveform(2, WavePoints, &Ch2[0]);

// print galvo measured positions
printf("Time [10us]  PhiX [Bit]  PhiY [Bit]\n");
printf("----------------------------------\n");
for (i=0; i<WavePoints; i++)
printf("%11d %11d %11d\n", i*SamplePeriod, Ch1[i], Ch2[i]);

    printf("press any key to terminate program ");
while(!kbhit());
    return;
}
```

10 Commands And Functions

10.1 Overview

The tables on page 70 and page 71 show the complete RTC®4 command set (control commands and list commands; also see chapter 4.1 "Software Concept", page 12). The commands are listed according to their intended use. The page numbers refer to chapter 10.3 "Command Set".

Multi-Board Commands

All commands marked (n_) also exist in a version for multiple RTC®4 boards installed in one computer. See chapter 5.4, page 43 for detailed information about these *multi-board commands*.

Compatibility

A number of commands from the RTC®2 command set are provided for compatibility. These commands are listed in chapter 10.4 "Supported and Unsupported RTC®2 Commands", page 126.

Control Commands

Other Control Commands

List Commands

Vector Commands

Arc Commands

Status Monitoring and Diagnostics

External Control Inputs

List Handling

Synchronization Of Processing

Structured Programming

Setting The Laser Parameters

Setting The Scanner Parameters

Coordinate Transformations

Direct Laser And Scan Head Control

Scanning Raster Images (Bitmaps)

I/O Commands

Other List Commands

10.2 Data Types

The following table defines the formats and ranges of the different data types used by the RTC®4 commands:

Data Format	Range	Pascal	C	Basic
64-bit IEEE floating point format		double	double	double
signed 16-bit value	[–32768; +32767]	smallint	short	integer
signed 32-bit value	[-2^{31}; $+2^{31}-1$]	longint	long	long
pointer to a null-terminated ANSI string, 1 byte per char		pchar	char*	string
unsigned 16-bit value	[0; 65535]	word	unsigned short	integer

10.3 Command Set

All commands are arranged in alphabetical order.

List Command	arc_abs	
Function	marking along the specified circular arc starting at the current position	
Parameters	`x,y`	absolute coordinates of the arc center in *bits* as signed 16-bit values
	`angle`	arc angle in ° [–360.0° … +360.0°] as 64-bit IEEE floating point value (positive angle values correspond to clockwise angles)
Integration	Pascal:	`arc_abs(x,y: smallint; angle: double);`
	C:	`void arc_abs(short x, short y, double angle);`
	Basic:	`arc_abs(ByVal x%, ByVal y%, ByVal angle#)`
Comments	• **The maximum allowed arc radius (in *bits*) is 32767.** • The marking speed is set with the command **set_mark_speed** (see page 110). • The laser is turned on at the beginning of the command after a LaserOn delay. • If another arc or mark command follows, a polygon delay is inserted, and the laser stays on. Otherwise a mark delay is inserted and the laser is turned off after a LaserOff delay. • See chapter 4.2 "Scan Head And Laser Control", page 15, for further details.	
References	**set_mark_speed**, **set_scanner_delays**, **arc_rel**	

List Command	arc_rel	
Function	marking along the specified circular arc starting at the current position	
Parameters	`dx,dy`	relative coordinates of the arc center in *bits* as signed 16-bit values
	`angle`	arc angle in ° [–360.0° … +360.0°] as 64-bit IEEE floating point value (positive angle values correspond to clockwise angles)
Integration	Pascal:	`arc_rel(dx,dy: smallint; angle: double);`
	C:	`void arc_rel(short dx, short dy, double angle);`
	Basic:	`arc_rel(ByVal dx%, ByVal dy%, ByVal angle#)`
Comments	• The arc center coordinates are relative to the current position. • **The maximum value for the *absolute* arc center coordinates is ±32767 bits.** • The marking speed is set with the command **set_mark_speed** (see page 110). • The laser is turned on at the beginning of the command after a LaserOn delay. • If another arc or mark command follows, a polygon delay is inserted, and the laser stays on. Otherwise a mark delay is inserted and the laser is turned off after a LaserOff delay. • See chapter 4.2 "Scan Head And Laser Control", page 15, for further details.	
References	**set_mark_speed**, **set_scanner_delays**, **arc_abs**	

Ctrl Command	auto_cal		
Function	starts or stops automatic self-calibration or re-calibrates the scan head (requires a scan head with *automatic self-calibration option*)		
Parameters	`head`	= 1: = 2:	calibration of scan head A (primary scan head connector) calibration of scan head B (secondary scan head connector)
	`command`	Control parameter (unsigned 16-bit value): = 0: = 1: = 2:	 get reference values for subsequent calibration routines perform calibration routine turn off compensation
Result	error code as a signed 16-bit value: 0 No error. 1 Self-calibration sensor not found. 2 Deviations during a measurement cycle are too large. 3 The command cannot be executed because a list is executing at the moment. 4 Reference data not found. 5 Calibration error. (Error during calibration or error in reference data.) 6 Parameter error. 7 Status error. (Error during data retransmission.)		
Integration	Pascal:	`function auto_cal(head, command: word): smallint;`	
	C:	`short auto_cal(unsigned short head, unsigned short command);`	
	Basic:	`function auto_cal(ByVal head%, ByVal command%)%`	
Comments	• This command can only be used in combination with a scan head equipped with sensors for automatic self-calibration. • The command is not executed if a list is executing at the moment. • The command `auto_cal(.., 0)` detects the current sensor positions and stores them as reference values for subsequent calibration routines. This should be performed under controlled, constant environmental conditions. The scan head must be at operating temperature, i.e. the warm-up time should be at least 15 minutes. The sensor positions are determined during a cycle of several measurements. If the deviations between the individual measurements are too large (maximum − minimum > 5 bits), the measurement stops and an error (error code 2) is returned. After successful determination of the reference values, the results are stored in non-volatile memory on the RTC®4 board. This way, the values are not lost when the system is shut down. • The command `auto_cal(.., 1)` performs a new calibration of the scan head. The command compares the current sensor positions with the reference values and calculates suitable compensation values (for gain and offset) for each scanner. The resulting compensation is applied to all subsequent vector outputs, until the command `auto_cal(.., 1)` is called again or the compensation is turned off by the command `auto_cal(.., 2)`. • To compensate possible drift of the scanners, the command `auto_cal(.., 1)` should be called regularly during operation. The calibration routine takes about 5 seconds (depending on the amount of drift), the initial determination of the reference values (`auto_cal(.., 0)`) takes about 30 to 50 seconds. • After initialization of the RTC®4, or after a reset, the compensation is turned off. However, the previously determined reference values are still available.		

Ctrl Command	auto_change	
Function	activates an automatic list change	
Integration	Pascal:	`procedure auto_change;`
	C:	`void auto_change(void);`
	Basic:	`sub auto_change()`
Comments	• The **auto_change** command should only be used when working with two lists (up to 4000 commands each). This command should only be used if a list is currently executing, or if it has already executed. Additionally, the other list should have already been loaded and closed. • With automatic list changing activated, the next list will start automatically after the current list is finished. • For each subsequent list, use a separate **auto_change** call. • The current status of both lists can be read with the commands **get_status** (page 86) or **read_status** (page 101). • See "Automatic List Handling", page 13.	
References	**get_status**, **read_status**	

Ctrl Command	auto_change_pos	
Function	activates an automatic list change	
Parameter	`start`	Start position (address) of the next command sequence to be executed (unsigned 16-bit value) allowed range: [0 … 7999]
Integration	Pascal:	`procedure auto_change_pos(start: word);`
	C:	`void auto_change_pos(unsigned short start);`
	Basic:	`sub auto_change_pos(ByVal start%)`
Comments	• Use the **auto_change_pos** command when the RTC®4's entire memory is treated like a single list buffer. • With automatic list changing activated, finishing of a list will automatically result in continuation of execution at the specified start position. • The current status of a list can be read with the commands **get_status** (page 86) or **read_status** (page 101). • See "Automatic List Handling", page 13.	
References	**get_status**, **read_status**	

List Command	**clear_io_cond_list**

Function	clears the bits of the 16-bit digital output port that are set (=1) in `mask_clear`,
	if the current `IOvalue` at the digital *input* port meets the following condition:
	`((IOvalue AND mask_1) = mask_1)  AND (((not IOvalue) AND mask_0) = mask_0)`
	(i.e. the bits specified in `mask_1` must be `1` and the bits specified in `mask_0` must be `0`).

Parameters	`mask_1`, `mask_0`, `mask_clear` unsigned 16-bit mask values
Integration	Pascal: `procedure clear_io_cond_list(mask_1, mask_0, mask_clear: word);`
	C: `void clear_io_cond_list(unsigned short mask_1, unsigned short mask_0, unsigned short mask_clear);`
	Basic: `sub clear_io_cond_list(ByVal mask_1%, ByVal mask_0%, ByVal mask_clear%)`
Comments	• The command affects only those bits of the output port that are set (=1) in the parameter `mask_clear`.

Examples (Pascal)	• clear bit #4 of the output port (DIGITAL_OUT4), if bit #0 of the input port (DIGITAL_IN0) is HIGH and bits #1 to #3 (DIGTAL_IN1…3) of the input port are LOW:
	`clear_io_cond_list($0001, $000E, $0010)`
	• always clear bit #15 of the output port (and leave the other bits unchanged):
	`clear_io_cond_list(0, 0, $8000)`

References	**set_io_cond_list**, **get_io_status**

Ctrl Command	control_command	
Function	sends a control command to an intelli*SCAN*® scan system	
Parameter	head	= 1: Head A = 2: Head B
	axis	= 1: X axis (STATUS channel, galvanometer scanner 2) = 2: Y axis (STATUS1 channel, galvanometer scanner 1) = 3: Z axis
	data	Command code with optional parameter as unsigned 16-bit value The most significant data byte ***$Code_H$*** represents a command code and the least significant data byte ***$Code_L$*** an optional parameter. For each command code the corresponding command is described and its allowed parameter values are listed below. Example: With data = 0501_H, i.e. $Code_H = 05_H$ (command *SetMode*) and $Code_L = 01_H$, the actual position is selected to be returned from the scan system.

$Code_H$	***Command and Parameter Values ($Code_L$)***
05_H	***SetMode***: This command selects the data signal to be returned from the scan system for the selected axis on the respective status channel. Each parameter value $Code_L$ corresponds to a particular data type. All returned data signals are returned as signed 16-bit values.

$Code_L$	***Returned Data Signal Type***
00_H	Statusword (Status signal as per XY2-100-Protocol) [0000_H ... $FFFF_H$] Bit #15 (MSB), Bit #7 = 1: Internal voltages normal Bit #14, Bit #6 = 1: Galvo temperature within normal range Bit #12, 11, 4, 3 = 1: X- and Y-axis position error within normal range Bit #13, 10, 8, 5, 2, 0 = 1 Bit #9, 1 = 0
01_H	Actual (angular) position / bit [–32768 ... 32767]
02_H	Set (angular) position / bit [–32768 ... 32767]
03_H	Position error (= set position - actual position) / bit [–32768 ... 32767]
04_H	Actual current (output stage current) / mA [–32768 ... 32767]
05_H	Relative galvo control / ‰ [–1000 ... 1000]
06_H	Actual (angular) velocity / (bit/ms) [–32768 ... 32767]
14_H	Galvanometer scanner temperature / 10^{-1} °C [–32768 ... 32767]
15_H	Servo board temperature / 10^{-1} °C [–32768 ... 32767]
16_H	AGC voltage (PD supply voltage) / 10^{-2} V [–32768 ... 32767]
17_H	DSP core supply voltage (1.8 V) / 10^{-2} V [–32768 ... 32767]
18_H	DSP IO voltage (3.3 V) / 10^{-2} V [–32768 ... 32767]
19_H	Analog section voltage (9 V) / 10^{-2} V [–32768 ... 32767]
$1A_H$	AD converter supply voltage (5 V) / 10^{-2} V [–32768 ... 32767]
$1B_H$	AGC current (PD supply current) / mA [–32768 ... 32767]
$1D_H$	Relative heating output of the corresponding galvanometer scanner heater / ‰ [0 ... 1000]
$1E_H$	Serial number (lower 16 bits) [0 ... 65535]
$1F_H$	Serial number (higher 16 bits) [0 ... 65535]

Ctrl Command	control_command			
		(05_H)	***Code$_L$***	***Returned Data Signal Type***
			20_H	Article number (lower 16 bits) [0 ... 65535]
			21_H	Article number (higher 16 bits) [0 ... 65535]
			22_H	Firmware version number [0 ... 65535]
			23_H	Calibration / mrad [0 ... 65535]
			24_H	Aperture / mm [0 ... 65535]
			25_H	Wavelength / nm [0 ... 65535]
			28_H	Flags B15 ... B0 [0 ... 65535]: Current operational state Bit #15 (MSB) = 1: Galvanometer scanner's output stage is on Bit #14 = 1: Galvo heater's output stage is on Bit #13 = 1: Internal voltages normal Bit #12 = 1: Position error within normal range Bit #11 = 1: Galvo and servo board temperature within normal range Bit #10 = 1: Booting process completed Bit #9 = 0: A critical error occured. The system was switched into a permanent error state. Bit #8 = 0: The external power supply voltages have – at least temporarily – dropped below the allowed value Bit #7 = 0: The temperature in the intelli*SCAN*® exceeds the maximum allowed value. The system was switched into a temporary error state. Bit #6 = 1: The AD converter was successfully initialized Bit #5 = 0: The galvanometer scanner has reached a critical edge position Bit #4 = 1: All control parameters valid Bits #1-3 Reserved Bit #0 = 0: The control is activated, as soon as all necessary flags are set
			29_H	Flags B31 ... B16 [0 ... 65535]: Current operational state Bit #31(MSB) = 1: AGC voltage (PD supply voltage) o.k. Bit #30 = 1: Analog section voltage (9 V) o.k. Bit #29 = 1: AD converter supply voltage (5 V) o.k. Bit #28 = 1: DSP IO voltage (3.3 V) o.k. Bit #27 = 1: DSP core supply voltage (1.8 V) o.k. Bit #26 = 1: Servo board operation temperature reached Bit #25 = 1: Galvanometer scanner operation temperature reached Bits #16-24 Reserved
			$2A_H$	Stop Event Code [0 ... 65535] = 0001_H: The galvanometer scanner has reached a critical edge position = 0002_H: AD converter error = 0003_H: The temperature in the intelli*SCAN*® has excceded the maximum allowed value = 0004_H: External power supply voltages have dropped below the allowed value = 0005_H: Flags are not valid = 0006_H - $000C_H$: Reserved = $000D_H$: Watchdog 10 µs time out (loop time exceeded) = $000E_H$: Position Acknowledge time out (set position not reached for long time) = $000F_H$: Reserved

Ctrl Command	control_command			
		(05_H)	***$Code_L$***	***Returned Data Signal Type***
			$2B_H$	Flags B15 ... B0 on stop [0 ... 65535]: Operational state at the moment of the most recently occured operation interruption (see data = 0528_H)
			$2C_H$	Flags B31 ... B16 on stop [0 ... 65535]: Operational state at the moment of the most recently occured operation interruption (see data = 0529_H)
			$2F_H$	Running time (seconds) / s [0 ... 59]
			30_H	Running time (minutes) / min [0 ... 59]
			31_H	Running time (hours) / h [0 ... 23]
			32_H	Running time (days) / d [0 ... 65535]
		15_H	***SetPosAcknowledgelevel***: This command sets the PosAcknowledge threshold value. The parameter value *$Code_L$* is the desired PosAcknowledge threshold value in counts [00_H ... FF_H]. By default a value of $B7_H$ is set, this corresponds to 183 counts or 0.28% of 2^{16} counts.	
Integration	Pascal:	`procedure control_command(head, axis, data: word);`		
	C:	`void control_command(unsigned short head, unsigned short axis, unsigned short data);`		
	Basic:	`sub control_command(ByVal head As Integer, ByVal axis As Integer, ByVal data As Integer)`		

Comments

General comment:

- The **control_command** command can only be used in conjunction with intelli*SCAN*® scan systems. Conventional scan systems will ignore the command.

Comments regarding the SetMode command ($Code_H = 05_H$):

- Data sources selected via the **control_command** command ($Code_H = 05_H$) will be transmitted until another source is selected.
- Data returned to the RTC®4 can be queried via the commands **get_value**, **set_trigger**, and **get_waveform**. Switching to a different data source causes a short (serial transmission-related) delay before transmission of the first data. Therefore, wait at least 50 µs before reading the data.
- Five seconds after every new start or reset or after sending the command code data = 0500_H via the **control_command** command, the statusword (XY2-100-compliant status signal) will be returned on all receiving channels. The statusword can be queried via the **get_head_status** command, too.
- During a temporary error state after the maximum allowed temperatue was exceeded (flag bit#7 = 0), the scan head's output stages of the affected axis are at least temporarily deactivated. The intelli*SCAN*® will return to normal operation as soon as the temperatur drops again below the maximum allowed temperature.
- During a "permanent error state" after a critical error, all output stages will remain deactivated. Critical errors include improper internal or external voltages and reaching critical edge positions.
- If a critical error occurs, the intelli*SCAN*® automatically enters a permanent error state, in which the output stages of the affected axis remain deactivated – even if the critical error was only temporarily present. Normal operation is *not* resumed. Flag bit#9 = 0 is only reset via a hardware reset. Critical errors are for instance improper internal voltages (flag bit#13 = 0), external power supply interruption (flag bit#8 = 0) or reaching a critical edge position (flag bit#5 = 0).
- During both temporary and permanent error states, the intelli*SCAN*® continues to transmit data to the control board. Even in these states, switching or selection of data signals for diagnostic purposes is still possible.

Ctrl Command	control_command
Comments	• The flags indicate the scan system's operational state. The scan module can return one of two flag blocks indicating the current operational state (data = 0528_H, 0529_H) or alternatively one of two further flag blocks indicating the operational state at the moment of the most recently occured operation interruption (data = $052B_H$, $052C_H$). After every successful restart – and, as long as no error has occured – all status informations of the two latter blocks (data = $052B_H$, $052C_H$) are irrelevant. Only, as soon as an error causes a switch into a temporary or permanent error state, the current status values will be saved into these two blocks. In this case, also an event code is simultaneously set, indicating which particular event caused the error state. This event code can be read out separately (data = $052A_H$). • To convert angle bit-values (actual position, set position, position error) or bit/ms values (actual speed) returned to the RTC®4 into mrad or mrad/ms, the returned values have to be multiplied by the scan system's calibration factor. The calibration factor (in mrad/bit) can be read out via data = 0523_H. • Exact values for the internal voltages referred to in the table can vary for different versions of the intelli*SCAN*®. *Comment regarding the SetPosAcknowledgelevel command (Code$_H$ = 15_H):* • After each reset, the default PosAcknowledge threshold value of $B7_H$ is set. If other threshold values are desired, they must be separately set for each axis (*Code$_H$* = 15_H). SCANLAB recommends to set only threshold values (*Code$_L$*) above 14_H (i.e. 20 counts or 0.03% of 2^{16} counts). Lower values can lead to frequent system safety shutdowns due to Position Acknowledge time outs (set position not reached for long time).
References	**get_value**, **get_head_status**, **set_trigger**, **get_waveform**

Ctrl Command	disable_laser	
Function	deactivates the laser control of the RTC®4	
Integration	Pascal:	`procedure disable_laser;`
	C:	`void disable_laser(void);`
	Basic:	`sub disable_laser()`
Comments	• This command disables the laser signals LASER1, LASER2 and LASERON. That means pins (1), (2) (LASERON) and (3) of the 9-pin laser connector and pins (19) and (22) of the LASER EXTENSION connector (on-board) are static. In case pin (2) of the 9-pin laser connector is configured as signal ANALOG OUT2, the command **disable_laser** will not have an effect on pin (2). • Use the command **enable_laser** to re-enable the laser control. • The command **get_startstop_info** (see page 86) (Bit #9) provides information about the current status of the laser control. • After initialization of the RTC®4 the laser control is ***active***.	
References	**enable_laser**, **get_startstop_info**	

Ctrl Command	dsp_start	
Function	resets the RTC®4 board	
Integration	Pascal:	`procedure dsp_start;`
	C:	`void dsp_start(void);`
	Basic:	`sub dsp_start()`
Comments	• This command is not needed for normal operation. The DSP starts automatically after the program file is loaded via the command **load_program_file** (page 97). • The command **dsp_start** resets the RTC®4. This means all parameters are reset to their default values. The laser focus is positioned in the center of the image field at the point (0|0). If any correction file(s) were loaded previously, they will stay in the RTC®4 memory, but the **select_cor_table** settings are reset to default. • After execution of the command **dsp_start**, the laser control is ***active***.	
References	**load_program_file**, **enable_laser**	

Ctrl Command	enable_laser	
Function	activates the laser control of the RTC®4	
Integration	Pascal:	`procedure enable_laser;`
	C:	`void enable_laser(void);`
	Basic:	`sub enable_laser()`
Comments	• This command re-enables the laser signals LASER1, LASER2 and LASERON at the 9-pin laser connector and at the LASER EXTENSION connector (on-board). • The command **get_startstop_info** (see page 86) provides information about the current status of the laser control. (Bit #9) • After initialization of the RTC®4, the laser control is ***active***.	
References	**disable_laser**, **get_startstop_info**	

Ctrl Command	execute_at_pointer	
Function	starts list execution at the specified address in the RTC®4 list buffer. Also see chapter 5.6 "Structured Programming", page 45.	
Parameter	`pointer`	address of the first list command to be executed (unsigned 16-bit value). Allowed range: [0 … 7999]
Integration	Pascal:	`procedure execute_at_pointer(pointer: word);`
	C:	`void execute_at_pointer(unsigned short pointer);`
	Basic:	`sub execute_at_pointer(ByVal pointer%)`
Comments	• This command can be used instead of **execute_list**. For instance, `execute_at_pointer(0)` is the same as `execute_list(1)`, `execute_at_pointer(4000)` is the same as `execute_list(2)`. However, execution can be started at any other position in the list buffer as well. • CAUTION: If the end of the list buffer is reached, the RTC®4 continues at the address zero. • Execution stops when a **set_end_of_list** command is encountered. • The command **execute_at_pointer** is ignored if a list is executing at the moment.	
References	**set_input_pointer**, **get_input_pointer**	

Ctrl Command	execute_list	
Function	starts execution of list 1 or list 2	
Parameter	`list_no`	number of the list to be executed (1 or 2)
Integration	Pascal:	`procedure execute_list(list_no: word);`
	C:	`void execute_list(unsigned short list_no);`
	Basic:	`sub execute_list(ByVal list_no%)`
Comments	• The commands **execute_list_1** and **execute_list_2** (with no parameters) can be used alternatively. • Execution stops when a **set_end_of_list** command is encountered. • During execution of a particular list, the other list can be loaded. However, that other list *must* not be started by **execute_list** until the current list execution is finished. (The command **execute_list** is ignored if a list is executing at the moment.) Also see "List Handling", page 13. • Use the command **get_status** (page 86) to determine the current status of execution.	
References	**get_status**, **execute_at_pointer**	

Ctrl Command	get_counts	
Function	reads the counter for the external list starts	
Result	counter value as a signed 32-bit value	
Integration	Pascal:	`function get_counts: longint;`
	C:	`long get_counts(void);`
	Basic:	`function get_counts()&`
Comments	• The counter is incremented each time a list is started via the external start signal. • To reset the counter, call the command **set_control_mode** (see page 104).	
References	**set_max_counts**, **set_control_mode**, **get_startstop_info**	

Ctrl Command	get_dll_version	
Function	returns the version number of the RTC®4 driver DLL	
Result	DLL version number as an unsigned 16-bit value	
Integration	Pascal:	`function get_dll_version: word;`
	C:	`unsigned short get_dll_version(void);`
	Basic:	`function get_dll_version()%`
Comments	• The RTC®4 DLL version numbers are in the range 400-499. • The RTC®3 DLL version numbers are in the range 100-399.	
References	**get_hex_version**, **get_rtc_version**	

Ctrl Command	get_head_status		
Function	returns the *scan head* status signals according to the XY2-100 or XY2-100-O protocol		
Parameter	head	= 1:	returns the status of scan head A (primary scan head connector)
		= 2:	returns the status of scan head B (secondary scan head connector)
Integration	Pascal:	`function get_head_status(head: word): word;`	
	C:	`unsigned short get_head_status(unsigned short head);`	
	Basic:	`function get_head_status(ByVal head%)%`	
Result	Status signal word (unsigned 16-bit value):		
	Bit #15 (MSB)	Power Status, 1 = OK	
	Bit #14	Temperature Status, 1 = OK	
	Bit #13	reserved	
	Bit #12	Position Acknowledge **Y**, 1 = OK	
	Bit #11	Position Acknowledge **X**, 1 = OK	
	Bit #10	reserved	
	Bit #9	0	
	Bit #8	1	
	Bits #7…#0	identical to Bits #15…#8	
Comments	• See chapter 5.3, page 42 for information about using two scan heads. • **It's also important to consider all the status signal information described in your scan system's operating manual.** • Together with an intelli*SCAN*® scan system, the **get_head_status** command can only be used to get the status signals (Statusword) directly after a new start or reset or when the Statusword is selected to be transmitted by the scan system via the **control_command** command. In addition, intelli*SCAN*® scan systems logically AND-connect the Position Acknowledge signals of the X- and Y-axis and only return a common signal at bit#3,4,11,12 (here the Position Acknowledge signals of the X- and Y-axis can be separately read out via the command **control_command** with data = 0528_H at flag bit#12).		
References	**get_status**, **read_status**, **set_piso_control**		

Ctrl Command	get_hex_version	
Function	returns the version number of the RTC®4 *software* (program file)	
Result	RTC®4 software version number as an unsigned 16-bit value	
Integration	Pascal:	`function get_hex_version: word;`
	C:	`unsigned short get_hex_version(void);`
	Basic:	`function get_hex_version()%`
Comments	• The RTC®4 2D-HEX version numbers are in the range 2.400-2.499 • The RTC®4 3D-HEX version numbers are in the range 3.400-3.499 • The RTC®3 2D-HEX version numbers are in the range 2.000-2.399 • The RTC®3 3D-HEX version numbers are in the range 3.000-3.399	
References	**get_dll_version**, **get_rtc_version**	

Ctrl Command	get_hi_data	
Function	returns the current positions of the home-in sensors for scan head A (primary scan head connector) of the current RTC®4 board.	
Result	x1, x2, y1, y2	coordinates of the current home-in positions in *bits* as unsigned 16-bit values
Integration	Pascal:	`procedure get_hi_data(var x1, x2, y1, y2: word);`
	C:	`void get_hi_data(unsigned short *x1, unsigned short *x2, unsigned short *y1, unsigned short *y2);`
	Basic:	`sub get_hi_data (x1%, x2%, y1%, y2%)`
Comments	• This command can only be used in combination with a scan head equipped with sensors for automatic self-calibration. • The command is provided for diagnostic purposes in addition to the command **auto_cal** (see page 74).	

Ctrl Command	get_input_pointer	
Function	returns the present list input pointer, i.e. the position in the RTC®4 list buffer where the next list command will be stored.	
Result	list input pointer [0 … 7999] as unsigned 16-bit value.	
Integration	Pascal:	`function get_input_pointer: word;`
	C:	`unsigned short get_input_pointer(void);`
	Basic:	`function get_input_pointer()%`
Comments	• This command is useful for reading the ADC input values from a raster image scan. See the command **read_pixel_ad** (page 101).	
References	**set_input_pointer**, **execute_at_pointer**	

Ctrl Command	get_io_status	
Function	returns the current state of the 16-bit digital *output* port on the “EXTENSION 1” connector	
Result	unsigned 16-bit value (DIGITAL_OUT0 … DIGITAL_OUT15)	
Integration	Pascal:	`function get_io_status: word;`
	C:	`unsigned short get_io_status(void);`
	Basic:	`function get_io_status()%`
Comments	• This command is conceived of for use in combination with the commands **set_io_cond_list** and **clear_io_cond_list**. • Also see "Programming Examples", page 46.	
References	**set_io_cond_list**, **clear_io_cond_list**	

Ctrl Command	get_list_space	
Function	returns the number of free list entries in circular queue mode (see chapter 5.5 "Circular Queue Mode", page 44)	
Result	number of free list entries as unsigned 16-bit value	
Integration	Pascal:	`function get_list_space: word;`
	C:	`unsigned short get_list_space(void);`
	Basic:	`function get_list_space()%`
References	**set_list_mode**	

Ctrl Command	get_rtc_version		
Function	returns the *firmware* version number of the RTC®4 board		
Result	RTC®4 version number as an unsigned 16-bit value:		
	Bit #0 (LSB) … Bit #7	Firmware version of the RTC®4 board	
	Bit #8	= 1:	Processing-on-the-fly is enabled. See supplement manual "Processing-On-The-Fly Software".
	Bit #9	= 1:	Second scan head connector is enabled. See page 54.
	Bit #10	= 1:	3D control signals are enabled. See supplement manual "3D Software".
	Bit #11	= 1:	RTC®4 I/O Extension Board is installed.
	Bits #12…#15	reserved	
Integration	Pascal:	`function get_rtc_version: word;`	
	C:	`unsigned short get_rtc_version(void);`	
	Basic:	`function get_rtc_version()%`	
Comments	• The RTC®4 firmware version numbers are in the range 128-255. If an RTC®4's version number is even, then it is equipped with an optical data interface and data transfer is compliant with the XY2-100-O protocol. For RTC®4s with odd version numbers, XY2-100 standard-compliant data transfer takes place electronically via a 25-pin female or 26-pin male connector. • The RTC®3 firmware version numbers are in the range 0-127.		
References	**get_hex_version**, **get_dll_version**		

Ctrl Command	get_serial_number	
Function	returns the individual serial number of the RTC®4 board	
Result	RTC®4 serial number as an unsigned 16-bit value	
Integration	Pascal:	`function get_serial_number: word;`
	C:	`unsigned short get_serial_number(void);`
	Basic:	`function get_serial_number()%`
Comments	• The command is helpful when using several RTC®4 boards in one computer. See chapter 5.4, page 43.	

Ctrl Command	get_startstop_info
Function	provides information about internal and external list starts and stops, as well as information about the laser signals
Result	Status signal word (unsigned 16-bit value): Bit #0 (LSB) = 1: An internal START has been executed since the last **get_startstop_info** command was called. Bit #1 = 1: An external START has been executed since the last **get_startstop_info** command was called. Bit #2 = 1: An internal STOP (command **stop_execution**) has been executed since the last **get_startstop_info** command was called. Bit #3 = 1: An external STOP has been executed since the last **get_startstop_info** command was called. Bit #4 logical AND operation of the signals /STOP and /STOP2: = 1: The external /STOP input (and the /STOP2 input of the MARKING ON THE FLY connector) are currently set to HIGH or not connected. = 0: The external /STOP input (or the /STOP2 input of the MARKING ON THE FLY connector) are currently set to LOW. Bit #9 Enable Laser Status. Bit = 1: Laser is enabled. See **enable_laser** and **disable_laser**. Bit #10 TTL Laser Signal Status. Bit = 1: Laser signals are *active-low* (Jumper **X10** is open; see page 59). Bit #12 logical AND operation of the signals /START and /START2: = 1: The external /START input (and the /START2 input of the MARKING ON THE FLY connector) are currently set to HIGH or not connected. = 0: The external /START input (or the /START2 input of the MARKING ON THE FLY connector) are currently set to LOW. The remaining bits are reserved.
Integration	Pascal: `function get_startstop_info: word;`
	C: `unsigned short get_startstop_info(void);`
	Basic: `function get_startstop_info()%`
Comments	• **The status bits #0 … #3 are reset after the command is executed.**
References	**get_counts**, **get_status**

Ctrl Command	get_status
Function	returns the current status of list execution
Result	`busy` true (≠ 0) : a list is executing at the moment false (= 0) : no list is executing at the moment
	`position` pointer to the command which is executing at the moment 0 ≤ `position` ≤ 7999.
Integration	Pascal: `procedure get_status(var busy: wordbool, var position: word);`
	C: `void get_status(unsigned short *busy, unsigned short *position);`
	Basic: `sub get_status(busy%, position%)`
Comments	• If `position` < 4000, list 1 is executing at the moment, otherwise list 2. • If `busy` = false, the variable `pointer` contains the position of the last command that was executed (usually **set_end_of_list**). • In Pascal the variable `busy` can alternatively be of the type `boolean`. • The `busy` signal is also available at pin (13) of the MARKING ON THE FLY connector (TTL level; a HIGH level indicates that a list is executing at the moment).
References	**read_status**, **get_head_status**

Ctrl Command	get_time	
Function	returns the RTC®4 timer value stored during the most recent call of **save_and_restart_timer**	
Result	timer value as 64-bit IEEE floating point value	
Integration	Pascal:	`function get_time: double;`
	C:	`double get_time(void);`
	Basic:	`function get_time ()#`
References	**save_and_restart_timer**	

Ctrl Command	get_value
Function	reads the current value of the specified signal
Parameter	`signal` specified signal represented by an unsigned 16-bit value: = 0: LASERON signal (1 = Laser On, 0 = Laser Off) = 1: StatusAX (X-axis status channel of head A) = 2: StatusAY (Y-axis status channel of head A) = 3: StatusAZ (Z-axis status channel of head A) = 4: StatusBX (X-axis status channel of head B) = 5: StatusBY (Y-axis status channel of head B) = 6: StatusBZ (Z-axis status channel of head B) = 7: SampleX (X-axis cartesian output value) = 8: SampleY (Y-axis cartesian output value) = 9: SampleZ (Z-axis cartesian output value) = 10: SampleAX_Corr (X-axis output value of head A) = 11: SampleAY_Corr (Y-axis output value of head A) = 12: SampleAZ_Corr (Z-axis output value of head A) = 13: SampleBX_Corr (X-axis output value of head B) = 14: SampleBY_Corr (Y-axis output value of head B) = 15: SampleBZ_Corr (Z-axis output value of head B)
Result	Current value of the specified signal as an unsigned 16-bit value
Integration	Pascal: `function get_value(signal: word): smallint;`
	C: `short get_value(unsigned short signal);`
	Basic: `function get_value (ByVal signal As Integer) As Integer`
Comments	• The type of scan system being used determines which status signals will be generated and returned via the status channels. Specific information can be found in your scan system's operating manual. • For scan systems with only one status channel, the status signals are only readable if `signal = 1` or `signal = 4`. • Coordinate transformations defined by **set_matrix**, **set_matrix_list**, **set_offset** or **set_offset_list** are already reflected in the SampleX, SampleY and SampleZ cartesian output values. • The SampleAX_Corr..SampleBZ_Corr output values are the (digital) output values actually transmitted from the RTC®4 to the scan system. The RTC®4 computes these values while taking into account the SampleX, SampleY and SampleZ output values as well as the selected correction file. • To observe the specified signal over a long time period, use **set_trigger** to start a corresponding measurement session.
References	**set_trigger**, **control_command**

Ctrl Command	get_wait_status	
Function	returns the wait state of the RTC®4	
Result	wait state as an unsigned 16-bit value	
Integration	Pascal:	`function get_wait_status: word;`
	C:	`unsigned short get_wait_status(void);`
	Basic:	`function get_wait_status()%`
Comments	• If processing has stopped at a wait marker, the command **get_wait_status** returns the number of this marker. See **set_wait** (page 119). • If no wait marker was reached, the command **get_wait_status** returns zero. • Processing of the list can be resumed by calling the command **release_wait**.	
References	**set_wait**, **release_wait**	

Ctrl Command	get_waveform	
Function	transfers to the PC the data that was measured and stored onto the RTC®4 via **set_trigger**	
Parameters	`channel`	measurement channel (1 or 2); specified as an unsigned 16-bit value
	`stop`	number of measured values [1..32768] to transfer; specified as an unsigned 16-bit value (values of measurement positions 1..stop will be transferred)
	`memptr`	unsigned 16-bit pointer to a location in the PC's memory to where the measured values should be transferred
Integration	Pascal:	`procedure get_waveform(channel, stop: word; memptr: pint);`
	C:	`void get_waveform(unsigned short channel, unsigned short stop, signed short *memptr);`
	Basic:	`sub get_waveform (ByVal channel As Integer, ByVal stop As Integer, ByVal memptr As Integer)`
References	**set_trigger**	

Ctrl Command	get_xy_pos	
Function	returns the current scanner set position	
Result	`xpos, ypos`	current output position in *bits* as signed 16-bit values
Integration	Pascal:	`procedure get_xy_pos(var xpos, ypos: smallint);`
	C:	`void get_xy_pos(short *xpos, short *ypos);`
	Basic:	`sub get_xy_pos(xpos%, ypos%)`
Comments	• The command returns the current output position (before the image field correction is applied). • If an image transformation was defined with the commands **set_matrix** / **set_offset**, the actual output coordinates are generally *not* equal to the original input coordinates.	

Ctrl Command	goto_xy	
Function	direct jump to the specified position	
Parameters	xpos, ypos	coordinates of the jump position in *bits* as signed 16-bit values
Integration	Pascal:	`procedure goto_xy(xpos, ypos: smallint);`
	C:	`void goto_xy(short xpos, short ypos);`
	Basic:	`sub goto_xy(ByVal xpos%, ByVal ypos%)`
Comments	• With DSP program files RTC4D2.HEX / RTC4D3.HEX, version 2.417 / 3.417 or higher the jump speed is set with the commandl **set_jump_speed** (see page 108). With older DSP program file versions, a **goto_xy** command jump will always be executed at a speed of **50000 bits/ms**. • Usually scan systems are optimized for scanning vectors (not for jumps). Therefore, after executing a jump via the **goto_xy** command with a high jump speed, a large overshoot may occur. Then – depending on the jump distance – the set positions may be reached only after a longer settling time. Therefore an appropriate low jump speed should be previously set via the **set_jump_speed** command. As with DSP program files RTC4D2.HEX / RTC4D3.HEX, version 2.416 / 3.416 or lower the jump speed is always set to maximum value of 50000 bits/ms, jumps should be realized via the list commands **jump_abs** or **jump_rel** with this DSP program file versions. • The command will be ignored if a list is executing at the moment. • Image field correction will be applied. • If an image transformation is defined with the commands **set_matrix** / **set_offset**, it will be applied.	
References	**get_xy_pos**, **set_jump_speed**, **jump_abs**, **jump_rel**	

Ctrl Command	home_position	
Function	activates the home jump mode and defines the home position	
Parameters	xhome, yhome	coordinates of the home position in *bits* as signed 16-bit values.
Integration	Pascal:	`procedure home_position (xhome, yhome: smallint);`
	C:	`void home_position (short xhome, short yhome);`
	Basic:	`sub home_position (ByVal xhome%, ByVal yhome%)`
Comments	• This command is intended for a laser system that does not allow fast switching of the laser. After calling the command, the laser focus moves to the specified home position whenever no list is executing. • At the beginning of the next list, the laser focus automatically jumps to the start point of the first list vector. The RTC®2 command **field_jump** is no longer needed. • A *beam dump* should be placed in the home position. • The home jump mode is deactivated with the command `home_position(0, 0)`.	

List Command	jump_abs	
Function	fast movement of the mirrors ("jump") to the specified position	
Parameters	`xval`, `yval`	absolute coordinates of the vector end point in *bits* as signed 16-bit values.
Integration	Pascal:	`procedure jump_abs(xval, yval: smallint);`
	C:	`void jump_abs(short xval, short yval);`
	Basic:	`sub jump_abs(ByVal xval%, ByVal yval%)`
Comments	• The jump speed is set with the command **set_jump_speed** (see page 108). • The laser is off during the jump. • After a jump command, a jump delay is inserted. • If a jump vector is followed by a zero jump vector, then the first JumpDelay is not executed. In contrast, a JumpDelay *is* executed if a jump vector is followed by a zero marking vector.	
References	**set_jump_speed**, **set_scanner_delays**, **jump_rel**	

List Command	jump_rel	
Function	fast movement of the mirrors ("jump") to the specified position	
Parameters	`dx, dy`	*relative* coordinates of the vector end point in *bits* as signed 16-bit values.
Integration	Pascal:	`procedure jump_rel(dx, dy: smallint);`
	C:	`void jump_rel(short dx, short dy);`
	Basic:	`sub jump_rel(ByVal dx%, ByVal dy%)`
Comments	• The coordinates are relative to the current position. • The maximum value for the *absolute* coordinates is ±32767 bits. • The jump speed is set with the command **set_jump_speed** (see page 108). • The laser is off during the jump. • After a jump command, a jump delay is inserted. • If a jump vector is followed by a zero jump vector, then the first JumpDelay is not executed. In contrast, a JumpDelay *is* executed if a jump vector is followed by a zero marking vector.	
References	**set_jump_speed**, **set_scanner_delays**, **jump_abs**	

List Command	laser_on_list	
Function	turns on the laser for a specified time interval	
Parameter	`delay`	time interval in *bits* as an unsigned 16-bit value. **1 bit equals 10 µs**. Allowed range: 0 ≤delay ≤65500
Integration	Pascal:	`procedure laser_on_list(delay: word);`
	C:	`void laser_on_list(unsigned short delay);`
	Basic:	`sub laser_on_list(ByVal delay%)`
Comments	• While the laser is turned on, the set position of the scanners is not changed. The next list command will be executed when the programmed time interval has passed. • The current settings for the laser delays are applied: – At the beginning of the programmed time interval, the laser turns on after a *LaserOn delay*. – At the end of the time interval, the laser turns off with the corresponding *LaserOff delay*. • The command is useful for marking separate dots.	

Ctrl Command	laser_signal_off	
Function	turns off the laser immediately	
Integration	Pascal:	`procedure laser_signal_off;`
	C:	`void laser_signal_off(void);`
	Basic:	`sub laser_signal_off()`
Comments	• This command is intended for direct laser control in combination with the command laser_signal_on (see below). • **The command will be ignored if a list is executing at the moment.**	
References	laser_signal_on	

List Command	laser_signal_off_list	
Function	same as laser_signal_off (see above), but a list command	
Integration	Pascal:	`procedure laser_signal_off_list;`
	C:	`void laser_signal_off_list(void);`
	Basic:	`sub laser_signal_off_list()`

Ctrl Command	laser_signal_on
Function	turns on the laser immediately
Integration	Pascal: `procedure laser_signal_on;`
	C: `void laser_signal_on(void);`
	Basic: `sub laser_signal_on()`
Comments	• The command is intended for turning on the laser directly, e.g. for alignment purposes. • Prior to calling the command **laser_signal_on**, the period and pulse width of the outgoing signal must be set via the list command **set_laser_timing**. • The laser must be turned off with the command **laser_signal_off**. • The command will be ignored if a list is executing at the moment. • **Check the beam path before turning on the laser!**
References	**laser_signal_off**

List Command	laser_signal_on_list
Function	same as **laser_signal_on**, but a list command
Integration	Pascal: `procedure laser_signal_on_list;`
	C: `void laser_signal_on_list(void);`
	Basic: `sub laser_signal_on_list()`

List Command	list_call
Function	defines a jump to the subroutine which starts at the specified list buffer address
Parameter	`address` jump address [0 … 7999] as unsigned 16-bit value
Integration	Pascal: `procedure list_call(address: word);`
	C: `void list_call(unsigned short address);`
	Basic: `sub list_call(ByVal address%)`
Comments	• The subroutine must be terminated with the command **list_return**. • Within a subroutine, another subroutine can be called (up to a depth of 30 calls). • Also see chapter 5.6 "Structured Programming", page 45.
References	**list_return**, **set_list_jump**

List Command	list_call_cond
Function	*Conditional subroutine call*: During execution of a list, this command causes a jump to a subroutine at the specified list buffer address [0 … 7999], if the current `IOvalue` at the 16-bit digital input port meets the following condition: `((IOvalue AND mask_1) = mask_1) AND (((not IOvalue) AND mask_0) = mask_0)` (i.e. the bits specified in `mask_1` must be `1` and the bits specified in `mask_0` must be `0`).
Parameters	`mask_1`, `mask_0` — unsigned 16-bit masks
	`address` — jump address [0 … 7999] as unsigned 16-bit value
Integration	Pascal: `list_call_cond(mask_1, mask_0, address: word);`
	C: `void list_call_cond(unsigned short mask_1, unsigned short mask_0, unsigned short address);`
	Basic: `sub list_call_cond(ByVal mask_1%, ByVal mask_0%, ByVal address%)`
Comments	• The subroutine must be terminated with the command **list_return** (see the chapter 5.6 "Structured Programming", page 45). • Also see "Programming Examples", page 46 (sample #(3))
References	**list_call**, **list_return**

List Command	list_jump_cond
Function	*Conditional list jump*: During execution of a list, this command causes a jump to the specified list buffer address [0 … 7999], if the current `IOvalue` at the 16-bit digital input port meets the following condition: `((IOvalue AND mask_1) = mask_1) AND (((not IOvalue) AND mask_0) = mask_0)` (i.e. the bits specified in `mask_1` must be `1` and the bits specified in `mask_0` must be `0`).
Parameters	`mask_1`, `mask_0` — unsigned 16-bit masks
	`address` — jump address [0 … 7999] as unsigned 16-bit value
Integration	Pascal: `list_jump_cond(mask_1, mask_0, address: word);`
	C: `void list_jump_cond(unsigned short mask_1, unsigned short mask_0, unsigned short address);`
	Basic: `sub list_jump_cond(ByVal mask_1%, ByVal mask_0%, ByVal address%)`
Comments	• See the chapter 5.6 "Structured Programming", page 45.

Examples (Pascal)	• wait until bit #3 of the input port turns HIGH (= loop while the bit is *LOW*):
	`list_jump_cond(0, $0008, get_input_pointer);`
	• skip the next two list commands if the state of the input port is `xxxx xxxx xxxx 0110` :
	`list_jump_cond(6, 9, get_input_pointer + 3);`
	• Also see "Programming Examples", page 46.

References	**set_list_jump**

List Command	list_nop
Function	inserts a null operation (no operation) into the list buffer
Integration	Pascal: `procedure list_nop;`
	C: `void list_nop(void);`
	Basic: `sub list_nop()`
Comments	• Null operations serve as place markers. The execution of a null operation needs 10 µs.

List Command	list_return
Function	terminates a list subroutine and jumps to the command following the **list_call** command
Integration	Pascal: `procedure list_return;`
	C: `void list_return(void);`
	Basic: `sub list_return()`
Comments	• See chapter 5.6 "Structured Programming", page 45.
References	**list_call**

Ctrl Command	load_correction_file
Function	loads the specified field correction file into RTC®4 memory (as table #1 or #2) and performs additional scaling, rotation and translation of the correction file
Parameters	`FileName` name of the correction file as a pointer to a null-terminated ANSI string
	`cor_table` determines whether the file shall be stored as correction table #1 or #2 (see command **select_cor_table**, page 103)
	`kx, ky` Gain values for scaling the correction table. Allowed range: [−1.2 … −0.8, +0.8 … +1.2]. (See comments below.)
	`phi` Rotational angle in degrees Allowed values: [−10° … +10°] + i · 90° ; i integer Positive angles: rotation is counterclockwise
	`x_offset, y_offset` additional offset in *bits* allowed range: [−30000 … +30000]
Result	error code as a signed 16-bit value: **Value** – **Description** 0 – Success. 3 – File error. 4 – Verify error. 8 – System driver not found, or the system driver is locked by another application. 10 – Parameter error. 11 – RTC®4 not found. 12 – A 3D correction file could not be loaded, because the 3D option is not installed. See chapter 7 "Options", page 57.
Integration	Pascal: `function load_correction_file(FileName: pchar; cor_table: smallint; kx, ky, phi, x_offset, y_offset: double): smallint;`
	C: `short load_correction_file(const char* FileName, short cor_table, double kx, double ky, double phi, double x_offset, double y_offset);`
	Basic: `function load_correction_file(ByVal FileName$, ByVal cor_table%, ByVal kx#, ByVal ky#, ByVal phi#, ByVal x_offset#, ByVal y_offset#)%`
Comments	• This command should be called at the beginning of an application program, before loading the program file. See note in chapter 9.3 "Initializing the RTC®4", page 63. • The RTC®4 (only the RTC®4 2D version) can store two different correction files at the same time, e.g. for use in a double scan head configuration. The files must be assigned to the two scan heads with the command **select_cor_table** (page 103). Also see chapter 5.3 "Using Two Different Correction Files", page 42. • To compensate possible misalignment or to align the image fields of the two scan heads precisely, each correction file can be scaled, rotated and translated (shifted) when it is loaded into the RTC®4. • The parameters `kx`, `ky`, `phi`, `x_offset` and `y_offset` *have to be* specified. If no additional transformation is required, the parameters must be set to `(.., .., 1, 1, 0, 0, 0)`. • By setting the gain factor `kx` or `ky` to a *negative* value, the corresponding axis is flipped. • Modification of 3D correction files is only reliable for Z = 0. For other Z values and parameters other than `(.., .., 1, 1, 0, 0, 0)`, the RTC®4 calculates output values which may deviate from the expected values.
References	**select_cor_table**, **load_program_file**

Ctrl Command	load_program_file		
Function	loads the specified program file into RTC®4 memory		
Parameter	FileName	name of the program file as a pointer to a null-terminated ANSI string	
Result	error code as a signed 16-bit value:		
	Value	**Description**	
	0	Success.	
	3	File error.	
	4	Verify error.	
	6	Number of bytes in Intel hex file is odd.	
	7	Checksum error in Intel hex file.	
	8	System driver not found, or the system driver is locked by another application.	
	9	Program file not complete.	
	11	RTC®4 not found.	
Integration	Pascal:	`function load_program_file(FileName: pchar): smallint;`	
	C:	`short load_program_file(const char* FileName);`	
	Basic:	`function load_program_file(ByVal FileName$)%`	
Comments	• The program file must be loaded at the beginning of each RTC®4 application program. See chapter 9.3 "Initializing the RTC®4", page 63. • The filename extension for RTC®4 program files is `*.HEX`. • The command **load_program_file** resets the RTC®4. This means all parameters are reset to their default values. The laser focus is positioned in the center of the image field at the point (0|0). If any correction file(s) were loaded previously, they will remain in RTC®4 memory, but the **select_cor_table** settings are reset to default. • After execution of the command **load_program_file**, the laser control is ***active***.		
References	**load_correction_file**, **dsp_start**		

<table>
<tr><th>Ctrl Command</th><th colspan="2">load_varpolydelay</th></tr>
<tr><td>Function</td><td colspan="2">loads a table for the variable polygon delay into the RTC®4.
See the section "Customizing The Variable Polygon Delay" on page 23.</td></tr>
<tr><td rowspan="2">Parameters</td><td>STBFileName</td><td>name of the data file (with extension *.STB) as a pointer to a null-terminated ANSI string. A data file can contain one or more tables.</td></tr>
<tr><td>TableNo</td><td>specifies which table in the data file shall be used (unsigned 16-bit value). The parameter TableNo must be identical with extension X of the command [VarPolyTableX] which denotes the desired table.</td></tr>
<tr><td>Result</td><td colspan="2">error code as a signed 16-bit value:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>≤ − 1</td><td>Success.
The absolute value of the error code is equal to the number of valid data points found in the table. Invalid entries are ignored. See the section "Customizing The Variable Polygon Delay" on page 23.</td></tr>
<tr><td>4</td><td>Verify error.</td></tr>
<tr><td>8</td><td>System driver not found, or the system driver is locked by another application.</td></tr>
<tr><td>10</td><td>Parameter error.</td></tr>
<tr><td>11</td><td>RTC®4 not found.</td></tr>
<tr><td>13</td><td>The specified table number was not found in the file; or the file was not found.</td></tr>
</table></td></tr>
<tr><td rowspan="3">Integration</td><td>Pascal:</td><td><code>function load_varpolydelay (STBFileName: pchar, TableNo: word): smallint;</code></td></tr>
<tr><td>C:</td><td><code>short load_varpolydelay (const char* STBFileName, unsigned short TableNo);</code></td></tr>
<tr><td>Basic:</td><td><code>function load_varpolydelay (ByVal STBFileName$, ByVal TableNo%)%</code></td></tr>
<tr><td>Comments</td><td colspan="2">• The table contains a number of data points for the customized variable polygon delay. When loading the table, the RTC®4 determines suitable values for the entire range of angles by linear interpolation.
Also see the section "Customizing The Variable Polygon Delay" on page 23.
• The command load_varpolydelay overwrites any previously loaded table for the variable polygon delay.
• After loading a program file by the command load_program_file, or after a reset by the command dsp_start, the RTC®4 uses the internal (default) table for the variable polygon delay (see figure 9 on page 21). That means the command load_varpolydelay is only needed if a different table should be used.
• To return to the internal polygon delay table (after a different table has been used), either the command load_program_file or the command dsp_start must be called.</td></tr>
<tr><td>References</td><td colspan="2">load_program_file, set_delay_mode</td></tr>
</table>

List Command	long_delay	
Function	stops execution of the list for the specified time	
Parameter	`delay`	delay time in *bits* as an unsigned 16-bit value. **1 bit equals 10 µs**. Allowed range: 0 ≤delay ≤65500.
Integration	Pascal:	`procedure long_delay(delay: word);`
	C:	`void long_delay(unsigned short delay);`
	Basic:	`sub long_delay(ByVal delay%)`
Comments	• This command should always be called after changing the lamp current of a YAG laser to obtain a constant laser power.	

List Command	mark_abs	
Function	marking along a straight line from the current position to the specified position	
Parameters	`xval,` `yval`	absolute coordinates of the vector end point in *bits* as signed 16-bit values.
Integration	Pascal:	`procedure mark_abs(xval, yval: smallint);`
	C:	`void mark_abs(short xval, short yval);`
	Basic:	`sub mark_abs(ByVal xval%, ByVal yval%)`
Comments	• The marking speed is set with the command **set_mark_speed** (see page 110). • The laser is turned on at the beginning of the command after a LaserOn delay. • If another mark or arc command follows, a polygon delay is inserted, and the laser stays on. Otherwise a mark delay is inserted and the laser is turned off after a LaserOff delay. • If a marking vector is followed by a zero jump or marking vector or a zero arc command, then the MarkDelay is *not* executed. • See chapter 4.2 "Scan Head And Laser Control", page 15, for further details.	
References	**set_mark_speed**, **set_scanner_delays**, **mark_rel**	

List Command	mark_rel	
Function	marking along a straight line from the current position to the specified position	
Parameters	`dx, dy`	*relative* coordinates of the vector end point in *bits* as signed 16-bit values.
Integration	Pascal:	`procedure mark_rel(dx, dy: smallint);`
	C:	`void mark_rel(short dx, short dy);`
	Basic:	`sub mark_rel(ByVal dx%, ByVal dy%)`
Comments	• The coordinates are relative to the current position. • The maximum value for the *absolute* coordinates is ±32767 bits. • The marking speed is set with the command **set_mark_speed** (see page 110). • The laser is turned on at the beginning of the command after a LaserOn delay. • If another mark or arc command follows, a polygon delay is inserted, and the laser stays on. Otherwise a mark delay is inserted and the laser is turned off after a LaserOff delay. • If a marking vector is followed by a zero jump or a marking vector or a zero arc command, then the MarkDelay is *not* executed. • See chapter 4.2 "Scan Head And Laser Control", page 15, for further details.	
References	**set_mark_speed**, **set_scanner_delays**, **mark_abs**	

Ctrl Command	measurement_status	
Function	returns the status of a measurement session started via **set_trigger**	
Result	`busy`	true (≠ 0) : a measurement session is currently in progress false (= 0) : no measurement session is currently in progress
	`position`	current position within the RTC®4's measurement storage area 0 ≤ `position` ≤ 32767
Integration	Pascal:	`procedure measurement_status(var busy: wordbool; var position: word);`
	C:	`void measurement_status(unsigned short *busy, unsigned short *position);`
	Basic:	`sub measurement_status (busy As Integer, position As Integer)`
Comments	• The **set_trigger** command will always cause exactly 32768 values per measurement channel to be measured and stored onto the RTC®4.	
References	**set_trigger**	

Ctrl Command	quit_loop	
Function	stops continuous output of the two lists started with **start_loop**	
Integration	Pascal:	`procedure quit_loop;`
	C:	`void quit_loop(void);`
	Basic:	`sub quit_loop()`
Comments	• The current list will execute completely before execution is stopped.	
References	**start_loop**	

Ctrl Command	read_io_port	
Function	returns the state of the 16-bit digital input port on the "EXTENSION 1" connector	
Result	unsigned 16-bit value (DIGITAL_IN0 … DIGITAL_IN15)	
Integration	Pascal:	`function read_io_port: word;`
	C:	`unsigned short read_io_port(void);`
	Basic:	`function read_io_port()%`
References	**write_io_port**	

Ctrl Command	read_pixel_ad
Function	returns the ADC value that was obtained at a pixel position during the output of a raster image line
Parameter	`pos` list pointer to the corresponding **set_pixel** command [0 … 7999]
Result	10-bit output value from the analog input channel (on the RTC®4 I/O Extension Board) which was specified with the **set_pixel** command (unsigned 16-bit value; the upper 6 bits contain the analog input channel number)
Integration	Pascal: `function read_pixel_ad(pos: word): word;`
	C: `unsigned short read_pixel_ad(unsigned short pos);`
	Basic: `function read_pixel_ad(ByVal pos%)%`
Comments	• This command can only be used with the RTC®4 I/O Extension Board. • The RTC®4 reads the ADC value during execution of each **set_pixel** command, and stores the result in the list (at the position of the **set_pixel** command). After the list is finished, the data can be read from the list pixel by pixel with the command **read_pixel_ad**. • The command requires the position of the corresponding **set_pixel** command in the list. To retain this position, use the command **get_input_pointer** just *before* writing the **set_pixel** command into the list.
References	**set_pixel**, **set_pixel_line**

Ctrl Command	read_status
Function	returns the list execution status
Result	RTC®4 status as an unsigned 16-bit value: **Bit #** / **Name** / **Description** Bit #0 (LSB) / Load1 / = 1: indicates that all following list commands will be stored in list 1. This bit will be set after a **set_start_list_1** command and will be reset after a **set_end_of_list** command. Bit #1 / Load2 / = 1: indicates that all following list commands will be stored in list 2. This bit will be set after a **set_start_list_2** command and will be reset after a **set_end_of_list** command. Bit #2 / Ready1 / = 1: indicates that list 1 is closed. This bit will be set after list 1 is closed by the **set_end_of_list** command. Bit #3 / Ready2 / = 1: indicates that list 2 is closed. This bit will be set after list 2 is closed by the **set_end_of_list** command. Bit #4 / Busy1 / = 1: indicates that list 1 is executing at the moment. Bit #5 / Busy2 / = 1: indicates that list 2 is executing at the moment. Bits #6…#7 / / 0 Bits #8…#15 / / 1
Integration	Pascal: `function read_status: word;`
	C: `unsigned short read_status(void);`
	Basic: `function read_status()%`
Comments	• Compare with the command **get_status** (page 86). • To read the status signals from the *scan heads*, use the command **get_head_status** (page 83).
References	**get_status**, **get_head_status**

Ctrl Command	release_wait	
Function	resumes execution of a list that was interrupted by a **set_wait** command	
Result	wait state as an unsigned 16-bit value	
Integration	Pascal:	`procedure release_wait;`
	C:	`void release_wait(void);`
	Basic:	`sub release_wait()`
Comments	• The command **release_wait** can only be used if the RTC®4 is actually in a wait state (i.e. a wait marker was reached and processing has stopped). • The command **release_wait** resets the wait_word to zero.	
References	**set_wait**, **get_wait_status**	

Ctrl Command	restart_list	
Function	enables the laser again and resumes execution of a list that was interrupted using the command **stop_list**.	
Integration	Pascal:	`procedure restart_list;`
	C:	`void restart_list(void);`
	Basic:	`sub restart_list()`
References	**stop_list**	

Ctrl Command	rtc4_count_cards	
Function	returns the number of detected RTC®4 boards	
Result	number of RTC®4 boards as an unsigned 16-bit value	
Integration	Pascal:	`function rtc4_count_cards: word;`
	C:	`unsigned short rtc4_count_cards(void);`
	Basic:	`function rtc4_count_cards()%`

List Command	save_and_restart_timer	
Function	stores the current value of the RTC®4 timer and resets it to zero	
Integration	Pascal:	`procedure save_and_restart_timer;`
	C:	`void save_and_restart_timer(void);`
	Basic:	`sub save_and_restart_timer ()`
Comments	• The stored timer value can be read by the **get_time** command. • The command is useful for measuring the marking time of a marking process.	
References	**get_time**	

Ctrl Command	select_cor_table
Function	assigns the previously loaded correction tables #1 and #2 to the scan head control ports
Parameters	`head_a` = 0: turns off the signals for scan head A (primary scan head connector) = 1: assigns correction table #1 to scan head A = 2: assigns correction table #2 to scan head A
	`head_b` = 0: turns off the signals for scan head B (secondary scan head connector) = 1: assigns correction table #1 to scan head B = 2: assigns correction table #2 to scan head B
Integration	Pascal: `procedure select_cor_table(head_a, head_b: word);`
	C: `void select_cor_table(unsigned short head_a, unsigned short head_b);`
	Basic: `sub select_cor_table(ByVal head_a%, ByVal head_b%)`
Comments	• In a double scan head system, table #1 will be used for scan head A, and table #2 will be used for scan head B: `select_cor_table(1,2)`. • However, each of the two correction tables can be assigned to any of the two scan head control ports. This allows, for example, to switch rapidly between two correction files for one scan head, e.g. one for a pointer laser and one for the main laser with a different wavelength. (Use `select_cor_table(1,0)` and `select_cor_table(2,0)` to switch from one file to the other.) • The default setting is `(1,0)`, i.e. correction table #1 will be used for scan head A, whereas the output signals for scan head B are turned off. Also see chapter 5.3 "Using Two Different Correction Files", page 42. • The RTC®3 3D version can store only *one* correction file.
References	load_correction_file

Ctrl Command	select_list
Function	selects which list will be executed upon receipt of an external start signal
Parameter	`list` = **0**: selects **list 1** = **1**: selects **list 2**
Integration	Pascal: `procedure select_list(list: word);`
	C: `void select_list(unsigned short list);`
	Basic: `sub select_list(ByVal list%)`
Comments	• By default, list 1 is selected. • A list can only be started via an external start signal if the list is closed and if no list is executing at the moment.
References	set_extstartpos_list

Ctrl Command	select_rtc	
Function	defines the active RTC®4 board in a multi-board system. See chapter 5.4, page 43.	
Parameter	`CardNo`	number of the RTC®4 board
Integration	Pascal:	`procedure select_rtc(CardNo: word);`
	C:	`void select_rtc(unsigned short CardNo);`
	Basic:	`sub select_rtc(ByVal CardNo%)`
Comments	• All subsequent commands (except for multi-board commands) are sent to the specified RTC®4 board. • By default, all commands are sent to board number 1.	

Ctrl Command	set_control_mode	
Function	enables or disables the external control input /START (and /START2, if the Processing-on-the-fly option is installed). See "External Control Inputs", page 13.	
Parameter	`control_mode` (unsigned 16-bit value):	

Bit #	Value	Description
Bit #0 (LSB)	= 1:	The external start input is enabled. The external *start* signal corresponds to the command **execute_list_1** or **execute_list_2**. See **select_list** (page 103). The external *stop* signal corresponds to the command **stop_execution**.
	= 0:	no external start signal
Bit #1		not used
Bit #2	= 1:	The *external start delay* (encoder delay) is turned off. See the supplement manual "Processing-On-The-Fly Software", commands **simulate_ext_start** (page 14) and **set_ext_start_delay** (page 10).
	= 0:	No effect. To turn on the external start delay, use the command **set_ext_start_delay** or **simulate_ext_start**.
Bit #3	= 1:	The external start input is *not* disabled by an external stop request.
	= 0:	The external start input is disabled by an external stop request.
Bits #4 … #15		not used

Integration	Pascal:	`procedure set_control_mode(control_mode: word);`
	C:	`void set_control_mode(unsigned short control_mode);`
	Basic:	`sub set_control_mode(ByVal control_mode%)`
Comments	• The command **set_control_mode** resets the counter for external list starts to zero. • If execution is aborted by the command **stop_execution**, bit #0 is reset to zero, i.e. external start inputs are disabled.	
References	**select_list**, **get_counts**, **set_max_counts**	

List Command	set_control_mode_list	
Function	similar to **set_control_mode** (see above), but a list command	
Integration	Pascal:	`procedure set_control_mode_list(control_mode: word);`
	C:	`void set_control_mode_list(unsigned short control_mode);`
	Basic:	`sub set_control_mode_list(ByVal control_mode%)`
Comments	• The counter for external list starts is ***not*** reset by this command.	
References	**set_control_mode**	

Ctrl Command	set_delay_mode
Function	turns the variable polygon delay mode and the variable jump delay mode on or off
Parameters	All parameters must be unsigned 16-bit values.

Parameter	Allowed Values	Description
varpoly	> 0	Enables the variable polygon delay mode. See page 21.
	= 0	Disables the variable polygon delay mode. (This is the default setting.)
directmove3d		This parameter effects only 3D-applications.
	> 0	The x-, y- and z-values are changed directly (linearly) to their end values during a jump.
	= 0	While the x- and y-values are changed linearly to their end values during a jump, the z-value is changed to its end-value in such a way that the focus is kept in one plane during the entire jump.
edgelevel	0 … 65500 **(1 bit equals 10 µs)**	This parameter defines a maximum "laser on" time for the corners of a polyline. If the polygon delay is longer than or equal to this value (because the angle ϕ is close to 180°, for instance), the laser is turned off (after a LaserOff delay) and a new polyline is started. This can be useful for preventing burn-in effects. The edgelevel must be smaller than twice the set value for the polygon delay, otherwise it has no effect. Also see figure 9 on page 21. **Note:** To disable this feature, set the edgelevel to 65500 (default value).
MinJumpDelay	0 … 32500 **(1 bit equals 10 µs)**	Minimum jump delay for a jump vector of zero length. See figure 6 on page 18.
JumpLengthLimit	0 … 32500	Jump length limit in *bits*. If the jump vector is *shorter* than this value, the jump delay is varied as shown in figure 6 on page 18. Otherwise the jump delay is constant. To disable the Variable Jump Delay mode, set the `JumpLengthLimit` to 0.

Integration		
	Pascal:	`procedure set_delay_mode(varpoly, directmove3d, edgelevel, MinJumpDelay, JumpLengthLimit: word);`
	C:	`void set_delay_mode(unsigned short varpoly, unsigned short directmove3d, unsigned short edgelevel, unsigned short MinJumpDelay, unsigned short JumpLengthLimit);`
	Basic:	`sub set_delay_mode(ByVal varpoly%, ByVal directmove3d%, ByVal edgelevel%, ByVal MinJumpDelay%, ByVal JumpLengthLimit%)`
References	**set_scanner_delays**, **load_varpolydelay**	

List Command	set_end_of_list	
Function	closes the currently open list	
Integration	Pascal:	`procedure set_end_of_list;`
	C:	`void set_end_of_list(void);`
	Basic:	`sub set_end_of_list()`
Comments	• A list can hold up to 4000 commands. • Also see chapter 5.5 "Circular Queue Mode", page 44. • Also see chapter 5.6 "Structured Programming", page 45.	
References	**set_start_list**	

List Command	set_extstartpos_list	
Function	defines the start position (list buffer address) of the list to be executed by the next external start signal	
Parameter	`position`	list pointer to the start address of the list [0 … 7999]
Integration	Pascal:	`procedure set_extstartpos_list(position: word);`
	C:	`void set_extstartpos_list(unsigned short position);`
	Basic:	`sub set_extstartpos_list(ByVal position%)`
Comments	• This command is a list command. When used within a list, the command "links" the current list to the next list. • `set_extstartpos_list(0)` selects list 1, `set_extstartpos_list(4000)` selects list 2. However, any other position in the list buffer can be specified as well. • The specified start address will be used for all subsequent external starts until a new address is specified either by **set_extstartpos_list** or by **select_list**.	
References	**select_list**, **set_control_mode**, **simulate_ext_start** (see the supplement manual "Processing-On-The-Fly Software")	

Ctrl Command	set_firstpulse_killer	
Function	defines the length of the FirstPulseKiller signal for a YAG laser	
Parameter	`fpk`	length of the FirstPulseKiller signal in bits: 1 bit equals 1/8 µs
Integration	Pascal:	`procedure set_firstpulse_killer(fpk: word);`
	C:	`void set_firstpulse_killer(unsigned short fpk);`
	Basic:	`sub set_firstpulse_killer(ByVal fpk%)`
Comments	• The time base for the FirstPulseKiller signal is always 8 MHz (i.e. 1 bit equals 1/8 µs). • In CO_2 mode, the command **set_firstpulse_killer** has no effect. • The laser control mode has to be set via the command **set_laser_mode** (page 109). Please refer to chapter 4.6 "Laser Control", page 32 for details. • To set the Q-Switch pulse width and period, use the list command **set_laser_timing** (page 109).	
References	**set_laser_mode**, **set_laser_timing**	

List Command	set_firstpulse_killer_list	
Function	same as **set_firstpulse_killer** (see above), but a list command	
Integration	Pascal:	`procedure set_firstpulse_killer_list(fpk: word);`
	C:	`void set_firstpulse_killer_list(unsigned short fpk);`
	Basic:	`sub set_firstpulse_killer_list(ByVal fpk%)`

SCANLAB
innovators for industry

Ctrl Command	set_input_pointer	
Function	sets the list input pointer to the specified address in the RTC®4 list buffer. The next list command will be stored at this address. Also see chapter 5.6 "Structured Programming", page 45.	
Parameter	`pointer`	list input pointer [0 … 7999] as unsigned 16-bit value
Integration	Pascal:	`procedure set_input_pointer(pointer: word);`
	C:	`void set_input_pointer(unsigned short pointer);`
	Basic:	`sub set_input_pointer(ByVal pointer%)`
Comments	• This command can be used alternatively instead of the **set_start_list** commands.	
	• CAUTION: If the end of the list buffer is reached, the list input pointer is reset to zero. Make sure not to overwrite any commands still needed by your application.	
References	**execute_at_pointer**, **get_input_pointer**	

List Command	set_io_cond_list	

Function	set the bits of the 16-bit digital output port that are set (=1) in `mask_set`,
	if the current `IOvalue` at the digital *input* port meets the following condition:
	`((IOvalue AND mask_1) = mask_1) AND (((not IOvalue) AND mask_0) = mask_0)`
	(i.e. the `IOvalue`'s bits specified in `mask_1` must be `1` and the `IOvalue`'s bits specified in `mask_0` must be `0`).	

Parameters	`mask_1`, `mask_0`, `mask_set`	unsigned 16-bit masks
Integration	Pascal:	`procedure set_io_cond_list(mask_1, mask_0, mask_set: word);`
	C:	`void set_io_cond_list(unsigned short mask_1, unsigned short mask_0, unsigned short mask_set);`
	Basic:	`sub set_io_cond_list(ByVal mask_1%, ByVal mask_0%, ByVal mask_set%)`
Comments	• The command affects only those bits of the output port that are set (=1) in the parameter `mask_set`.	

Examples (Pascal)	• set bit #4 of the output port (DIGITAL_OUT4), if bit #0 of the input port (DIGITAL_IN0) is HIGH and bits #1 to #3 (DIGTAL_IN1…3) of the input port are LOW:
	`set_io_cond_list($0001, $000E, $0010)`
	• always set bit #15 of the output port (and leave the other bits unchanged):
	`set_io_cond_list(0, 0, $8000)`	

References	**clear_io_cond_list**, **get_io_status**	

List Command	set_jump_speed	
Function	defines the jump speed for the following vector commands	
Parameter	`jump_speed`	jump speed in *bits per ms* (64-bit IEEE floating point value) Allowed range: [1 … 50000]
Integration	Pascal:	`procedure set_jump_speed(jump_speed: double);`
	C:	`void set_jump_speed(double jump_speed);`
	Basic:	`sub set_jump_speed(ByVal jump_speed#)`
Comments	• The command is written directly into the list. • By default a jump speed of 100 *bits per ms* is preset. • The specified jump speed (or the preset jump speed, if no other value has been specified) is used for all subsequent jump commands until a new value is specified. • To obtain the actual jump speed *v* in the image plane (in *meters per second*), the specified speed value (in *bits per ms*) must be divided by the calibration factor *K* of the correction file (in *bits per mm*): v_{jump} = jump_speed / K	
References	**jump_abs**, **jump_rel**, **set_mark_speed**	

List Command	set_laser_delays	
Function	sets the LaserOn delay and the LaserOff delay. **1 bit equals 1 µs.**	
Parameters	`laser_on_delay`	signed 16-bit value. Allowed range: [–8000 … +8000]
	`laser_off_delay`	signed 16-bit value. Allowed range: [+2 … +8000]
Integration	Pascal:	`procedure set_laser_delays(laser_on_delay, laser_off_delay: smallint);`
	C:	`void set_laser_delays(short laser_on_delay, short laser_off_delay);`
	Basic:	`sub set_laser_delays(ByVal laser_on_delay%, ByVal laser_off_delay%)`
Comments	• See the section "Laser Delays" on page 16, for details. • The LaserOff delay must always be longer than the LaserOn delay. See "Limits For The Delays" on page 24. • If the LaserOn delay is negative, the total marking time is extended.	
References	**set_scanner_delays**	

Ctrl Command	set_laser_mode
Function	selects the laser control mode of the RTC®4
Parameter	mode = 0: CO_2 mode = 1: YAG mode 1 = 2: YAG mode 2 = 3: YAG mode 3 = 4: laser mode 4
Integration	Pascal: `procedure set_laser_mode(mode: word);`
	C: `void set_laser_mode(unsigned short mode);`
	Basic: `sub set_laser_mode(ByVal mode%)`
Comments	• The available laser signals depend on the selected laser control mode. Please refer to chapter 4.6 "Laser Control", page 32 for a detailed description. • The command should be used only once at the program start. Also see chapter 9.3 "Initializing the RTC®4", page 63.
References	**set_laser_timing**, **set_firstpulse_killer**

List Command	set_laser_timing
Function	defines the output period and the pulse widths for the laser signals LASER1 and LASER2
Parameters	half_period — half of the output period in *bits*. 1 bit equals 1/8 µs **or** 1 µs, depending on the selected time base. Allowed range: [2 … 65500]
	pulse_width1, pulse_width2 — Pulse widths of the laser signals LASER1 and LASER2 in *bits*. 1 bit equals 1/8 µs **or** 1 µs, depending on the selected time base. Allowed range: [2 … 65500]
	time_base = 0: sets the time base to 1 MHz (1 bit equals 1 µs) ≠ 0: sets the time base to 8 MHz (1 bit equals 1/8 µs)
Integration	Pascal: `procedure set_laser_timing(half_period, pulse_width1, pulse_width2, time_base: word);`
	C: `void set_laser_timing(unsigned short half_period, unsigned short pulse_width1, unsigned short pulse_width2, unsigned short time_base);`
	Basic: `sub set_laser_timing(ByVal half_period%, ByVal pulse_width1%, ByVal pulse_width2%, ByVal time_base%)`
Comments	• The time base setting applies only for the parameters of this command. • In general, it is recommended to set the time base to 8 MHz. A time base of 1 MHz should only be chosen if necessary. • Please refer to chapter 4.6 "Laser Control", page 32, for details.
References	**set_laser_mode**, **set_firstpulse_killer**, **set_standby**, **set_standby_list**

List Command	set_list_jump
Function	defines a jump to the specified list buffer address
Parameter	`address` jump address [0 … 7999] as unsigned 16-bit value
Integration	Pascal: `procedure set_list_jump(address: word);`
	C: `void set_list_jump(unsigned short address);`
	Basic: `sub set_list_jump(ByVal address%)`
Comments	• See chapter 5.6 "Structured Programming", page 45.
References	list_call

Ctrl Command	set_list_mode
Function	enables the circular queue mode (see chapter 5.5 "Circular Queue Mode", page 44)
Parameter	`mode` = 0: circular queue mode disabled = 1: circular queue mode enabled
Integration	Pascal: `procedure set_list_mode(mode: word);`
	C: `void set_list_mode(unsigned short mode);`
	Basic: `sub set_list_mode(ByVal mode%)`
Comments	• If the circular queue mode is to be used, this command should be called at the beginning of the application program, i.e. before writing any list commands to the RTC®4.
References	get_list_space

List Command	set_mark_speed
Function	defines the marking speed for the subsequent vector and arc commands
Parameter	`mark_speed` marking speed in *bits per ms* (64-bit IEEE floating point value) Allowed range: [1 … 50000]
Integration	Pascal: `procedure set_mark_speed(mark_speed: double);`
	C: `void set_mark_speed(double mark_speed);`
	Basic: `sub set_mark_speed(ByVal mark_speed#)`
Comments	• The command is written directly into the list. • By default a marking speed of 100 *bits per ms* is preset. • The specified marking speed (or the preset marking speed, if no other value has been specified) is used for all subsequent mark and arc commands until a new value is specified. • To obtain the actual marking speed v in the image plane (in *meters per second*), the specified speed value (in *bits per ms*) must be divided by the calibration factor K of the correction file (in *bits per mm*): v_{mark} = mark_speed / K
References	mark_abs, mark_rel, set_jump_speed

Ctrl Command	set_matrix	
Function	defines a (2 x 2) transformation matrix which will be used for all subsequent vector outputs.	
Parameters	m11, m12, m21, m22	Matrix coefficients as 64-bit IEEE floating point values Allowed range: [–100 … 100]
Integration	Pascal:	`procedure set_matrix(m11, m12, m21, m22: double);`
	C:	`void set_matrix(double m11, double m12, double m21, double m22);`
	Basic:	`sub set_matrix(ByVal m11#, ByVal m12#, ByVal m21#, ByVal m22#)`
Comments	• See chapter 5.1 "Coordinate Transformations", page 40.	
References	**set_matrix_list**, **set_offset**, **set_offset_list**	

List Command	set_matrix_list	
Function	sets *one* of the four coefficients of the (2 x 2) transformation matrix during execution of a list.	
Parameters	i, j	Row index and column index [1 or 2] of the matrix coefficient to be changed
	m_ij	Matrix coefficient as a 64-bit IEEE floating point value Allowed range: [–100 … 100]
Integration	Pascal:	`procedure set_matrix_list(i, j: word; m_ij: double);`
	C:	`void set_matrix_list(unsigned short i, unsigned short j, double m_ij);`
	Basic:	`sub set_matrix_list(ByVal i%, ByVal j%, ByVal m_ij#)`
Comments	• The command **set_matrix_list** only allows changing *one* of the four coefficients at a time. To change several coefficients during execution of a list, the command has to be called repeatedly. • See chapter 5.1 "Coordinate Transformations", page 40.	
References	**set_matrix**, **set_offset**, **set_offset_list**	

Ctrl Command	set_max_counts	
Function	defines the maximum number of external list starts	
Parameter	max_counts	maximum number of external list starts as a signed 32-bit value. Allowed range: 0 ≤max_counts ≤2147483647
Integration	Pascal:	`procedure set_max_counts(max_counts: longint);`
	C:	`void set_max_counts(long max_counts);`
	Basic:	`sub set_max_counts(ByVal max_counts&)`
Comments	• If the parameter *max_counts* is set to 0, the maximum number of external start signals is unlimited. • After a reset of the RTC®4 the parameter *max_counts* is set to 0. • The counter can be read with the command **get_counts**. • When the specified number of external start signals is reached, bit 0 of the control_mode register is set to zero. Thus no further external start signals are possible. • To reset the counter, call the command **set_control_mode** (see page 104).	
References	**get_counts**, **set_control_mode**	

Ctrl Command	set_offset	
Function	defines an offset in the X and Y directions which will be added to all subsequent vector outputs.	
Parameters	`x_offset,` `y_offset`	offset in *bits* Allowed range: [−32768 ...+32767]
Integration	Pascal:	`procedure set_offset(x_offset, y_offset: smallint);`
	C:	`void set_offset(short x_offset, short y_offset);`
	Basic:	`sub set_offset(ByVal x_offset%, ByVal y_offset%)`
Comments	• See chapter 5.1 "Coordinate Transformations", page 40.	
References	**set_matrix**, **set_offset_list**	

List Command	set_offset_list	
Function	same as **set_offset** (see above), but a list command	
Integration	Pascal:	`procedure set_offset_list(x_offset, y_offset: smallint);`
	C:	`void set_offset_list(short x_offset, short y_offset);`
	Basic:	`sub set_offset_list(ByVal x_offset%, ByVal y_offset%)`
Comments	• This command allows definition of an offset within a list. • See chapter 5.1 "Coordinate Transformations", page 40.	
References	**set_matrix_list**	

Ctrl Command	set_piso_control	
Function	changes the delay which is inserted before the RTC®4 reads the Status Signal Word from the scan head	
Parameters	`L1, L2`	lengths of the data cables for scan head A and scan head B in meters
Integration	Pascal:	`procedure set_piso_control(L1, L2: word);`
	C:	`void set_piso_control(unsigned short L1, unsigned short L2);`
	Basic:	`sub set_piso_control(ByVal L1%, ByVal L2%)`
Comments	• This command provides adjustment of the timing of the (bi-directional) communication between the scan head and the RTC®4 to reflect the length of the data cable. • The command *should* be used if the data cable is longer than approximately 12 m (and if the Status Signal from the scan head is to be evaluated). See also chapter 6.5 "Data Cable", page 55. • The maximum length for the data cable is 75 m.	
References	**get_head_status**	

List Command	set_pixel		
Function	defines the output parameters (laser pulse width and ANALOG OUT2 value) for **one** pixel in an image line. In addition, one of the four analog input ports of the RTC®4 I/O Extension Board can be read during the output of the pixel (RTC®4 with I/O Extension Board only). Also see chapter 5.7 "Scanning Raster Images (Bitmaps)", page 47.		
Parameters	`pulse_width`	laser output pulse width (unsigned 16-bit value). **1 bit equals 1/8 µs.**	
	`da_value`	output value for the ANALOG OUT2 port of the RTC®4 (10-bit resolution) (unsigned 16-bit value; the upper 6 bits are ignored)	
	`ad_channel`	number of the ADC input channel to be read [0 … 4; 0 = none]. The RTC®4 writes the obtained ADC result into the current list (at the position of the **set_pixel** command). The value can be read with the command **read_pixel_ad** (see page 101) after execution of the list.	
Integration	Pascal:	`procedure set_pixel(pulse_width, da_value, ad_channel: word);`	
	C:	`void set_pixel(unsigned short pulse_width, unsigned short da_value, unsigned short ad_channel);`	
	Basic:	`sub set_pixel(ByVal pulse_width%, ByVal da_value%, ByVal ad_channel%)`	
Comments	• Each image line must start with the command **set_pixel_line**. • Each pixel in the image line is defined by one **set_pixel** command. The **set_pixel** commands must follow immediately after the command **set_pixel_line**. No other commands must be written into the list until the image line is completed.		
References	**set_pixel_line**, **read_pixel_ad**		

List Command	set_pixel_line		
Function	switches to the pixel output mode, defines various pixel output parameters and marks the beginning of a pixel line. Also see chapter 5.7 "Scanning Raster Images (Bitmaps)", page 47.		
Parameters	`pixel_mode`	= 0:	The laser focus "jumps" from one pixel to the next.
		= 1:	The laser focus moves from one pixel to the next in small steps (microvectors). Also see chapter 5.7, page 47.
	`pixel_period`	pixel output period (unsigned 16-bit value). **1 bit equals 10 µs.**	
	`dx, dy`	distance in the X and Y directions between adjacent pixels in *bits* (64-bit IEEE floating point values)	
Integration	Pascal:	`procedure set_pixel_line(pixel_mode, pixel_period: word; dx, dy: double);`	
	C:	`void set_pixel_line(unsigned short pixel_mode, unsigned short pixel_period, double dx, double dy);`	
	Basic:	`sub set_pixel_line(ByVal pixel_mode%, ByVal pixel_period%, ByVal dx#, ByVal dy#)`	
Comments	• Each image line must start with the command **set_pixel_line**. This command should be preceded by a jump command to the start point of the image line. • The command **set_pixel_line** turns on the pixel output mode of the RTC®4. The individual pixels of the image line are defined by **set_pixel** commands. The **set_pixel** commands must follow immediately after the command **set_pixel_line**. • The first subsequent command in the list which is *not* a **set_pixel** command turns off the pixel output mode. • The command **set_pixel_line** requires two list entries in the list buffer memory.		
References	**set_pixel**		

List Command	set_scanner_delays	
Function	sets the scanner delays	

Parameters	`jump_delay,`
	`mark_delay,`
	`polygon_delay`
		unsigned 16-bit values. Allowed range: [0…32767]; **1 bit equals 10 µs.**

Integration	Pascal:	`procedure set_scanner_delays(jump_delay, mark_delay, polygon_delay: word);`
	C:	`void set_scanner_delays(unsigned short jump_delay, unsigned short mark_delay, unsigned short polygon_delay);`
	Basic:	`sub set_scanner_delays(ByVal jump_delay%, ByVal mark_delay%, ByVal polygon_delay%)`
Comments	• See "Scanner Delays" on page 17 • and "Limits For The Delays" on page 24.	
References	**set_delay_mode**, **set_laser_delays**	

Ctrl Command	set_softstart_level
Function	sets the softstart values.
Parameters	All parameters must be unsigned 16-bit values.
	Parameter / **Allowed Values** / **Description** `index` / 0 … *n* / index number of the softstart table value `level` / 0 … 1023 / for softstart mode = 1, 2, 11 and 12. Value 0 corresponds to 0 V; value 1023 corresponds to 2.56 V (for jumper X3 position 1-2) or 10 V (for jumper X3 position 2-3). 2 … 2^{16}-1 / for softstart mode = 3 and 13 (see **set_softstart_mode**). 1 bit corresponds to 1 µs or 1/8 µs depending on the time base which can be specified with the command **set_laser_timing** (also see page 33).
Integration	Pascal: `procedure set_softstart_level (index, level: word);`
	C: `void set_softstart_level (unsigned short index, unsigned short level);`
	Basic: `sub set_softstart_level (ByVal index%, ByVal level%)`
Comments	• The softstart mode is enabled by the command **set_softstart_mode** (also see section "Softstart Mode" on page 35). • The **set_softstart_mode** command must be called *before* first-time use of the **set_softstart_level** command. • Please be sure to set as many softstart values as you defined by **set_softstart_mode**.
References	• **set_softstart_mode**

Ctrl Command	set_softstart_mode
Function	initializes the softstart mode.
Parameters	All parameters are unsigned 16-bit values.

Parameter	Allowed Values	Description
mode	= 0	disables the softstart mode, but does not remove previously loaded softstart values.
	= 1, 11	enables the analog output signal ANALOG OUT1.
	= 2, 12	enables the analog output signal ANALOG OUT2.
	= 3, 13	enables the LASER1 signal.
	= 1 ... 3	The change from one Level to the next occurs with the leading edge of the laser pulse.
	= 11 ... 13	The change from one Level to the next occurs with the trailing edge of the laser pulse.
number	1 … 1000	(number+1) is the number of softstart values to be transmitted.
restartdelay	0 … 65534 (1 bit equals 10 µs)	defines the laser idle time after which the softstart is executed upon restarting the laser.

Integration	Pascal:	`procedure set_softstart_mode (mode, number, restartdelay: word);`
	C:	`void set_softstart_mode (unsigned short mode, unsigned short number, unsigned short restartdelay);`
	Basic:	`sub set_softstart_mode (ByVal mode%, ByVal number%, ByVal restartdelay%)`

Comments	• The **set_softstart_mode** command must be called *before* first-time use of the **set_softstart_level** command. • The 0 ... *number* softstart values are set by calling the command **set_softstart_level** (*number*+1) times • The pulse width or power of the first laser pulse is determined by level*[0]* if the values are assigned with the *leading* edge of the laser pulse selected, or by level*[1]* if the *trailing* edge of the laser pulse is selected. • Also see section "Softstart Mode" on page 35.
References	• **set_softstart_level**

Ctrl Command	set_standby	
Function	defines the output period and the pulse width of the stand-by pulses or – in Laser Mode 4 – the continuously-running laser signals.	
Parameters	`half_period`	*half* of the stand-by output period in *bits*. **1 bit equals 1/8 µs.** Allowed range: [0 … 65500]
	`pulse_width`	Pulse width of the stand-by pulses in *bits*. **1 bit equals 1/8 µs.** Allowed range: [0 … half_period] * • If a value larger than half_period is specified, the RTC®4 driver sets the pulse_width to half_period.
Integration	Pascal:	`procedure set_standby(half_period, pulse_width: word);`
	C:	`void set_standby(unsigned short half_period, unsigned short pulse_width);`
	Basic:	`sub set_standby(ByVal half_period%, ByVal pulse_width%)`
Comments	• The time base for the stand-by pulses is always 8 MHz (i.e. 1 bit equals 1/8 µs). • The stand-by pulses are available in **all** laser modes (YAG 1/2/3, CO_2 and laser mode 4). They can be turned off by setting the stand-by pulse width to zero (default). • The laser control mode has to be set with the command **set_laser_mode** (page 109). Please refer to chapter 4.6 "Laser Control", page 32, for details. • To set the active output period and pulse width for the two laser signals, use the list command **set_laser_timing** (see page 109).	
References	**set_standby_list**, **set_laser_mode**, **set_laser_timing**	

List Command	set_standby_list	
Function	same as **set_standby** (see above), but a list command	
Integration	Pascal:	`procedure set_standby_list(half_period, pulse_width: word);`
	C:	`void set_standby_list(unsigned short half_period, unsigned short pulse_width);`
	Basic:	`sub set_standby_list(ByVal half_period%, ByVal pulse_width%)`

Ctrl Command	set_start_list	
Function	opens the list memory, or half of it, for writing a list (list 1 or list 2). All subsequent list commands are stored in the corresponding list.	
Parameter	`list_no`	number of the list to be opened for writing (1 or 2)
Integration	Pascal:	`procedure set_start_list(list_no: word);`
	C:	`void set_start_list(unsigned short list_no);`
	Basic:	`sub set_start_list(ByVal list_no%)`
Comments	• The commands **set_start_list_1** and **set_start_list_2** (with no parameter) can be used alternatively.	
References	**execute_list**, **read_status**	

List Command	set_trigger	
Function	starts measurement and storage of the specified signals	
Parameters	`sample-period`	measurement period [(1..65535) · 10 µs] as an unsigned 16-bit value
	`signal1`, `signal2`	The following signals (each represented by an unsigned 16-bit value) are specifiable for measurement channels 1 and 2: = 0: LASERON signal (1 = laser on, 0 = laser off) = 1: StatusAX (X-axis status channel of head A) = 2: StatusAY (Y-axis status channel of head A) = 3: StatusAZ (Z-axis status channel of head A) = 4: StatusBX (X-axis status channel of head B) = 5: StatusBY (Y-axis status channel of head B) = 6: StatusBZ (Z-axis status channel of head B) = 7: SampleX (X-axis cartesian output value) = 8: SampleY (Y-axis cartesian output value) = 9: SampleZ (Z-axis cartesian output value) = 10: SampleAX_Corr (X-axis output value of head A) = 11: SampleAY_Corr (Y-axis output value of head A) = 12: SampleAZ_Corr (Z-axis output value of head A) = 13: SampleBX_Corr (X-axis output value of head B) = 14: SampleBY_Corr (Y-axis output value of head B) = 15: SampleBZ_Corr (Z-axis output value of head B)
Integration	Pascal:	`procedure set_trigger(sampleperiod, signal1, signal2: word);`
	C:	`void set_trigger(unsigned short sampleperiod, unsigned short signal1, unsigned short signal2);`
	Basic:	`sub set_trigger(ByVal sampleperiod As Integer, ByVal signal1 As Integer, ByVal signal2 As Integer)`
Comments	• The **set_trigger** command will always cause exactly 32768 values per measurement channel to be measured and stored onto the RTC®4. Afterward, the **get_waveform** command can be used to transfer the measured values to the PC for evaluation. • The current status of a measurement session can be queried via the **measurement_status** command. • Once started, a measurement session can only be canceled by again calling the **set_trigger** command. Previously measured values are thereby lost. • The type of scan system being used determines which status signals will be generated and returned via the status channels. Specific information can be found in your scan system's operating manual. • For scan systems with only one status channel, the status signals are only readable if `signal1,2 = 1` or `signal1,2 = 4`. • Coordinate transformations defined by **set_matrix**, **set_matrix_list**, **set_offset** or **set_offset_list** are already reflected in the SampleX, SampleY and SampleZ cartesian output values. • The SampleAX_Corr..SampleBZ_Corr output values are the (digital) output values actually transmitted from the RTC®4 to the scan system. The RTC®4 computes these values while taking into account the SampleX, SampleY and SampleZ output values as well as the selected correction file.	
References	**get_waveform**, **measurement_status**, **control_command**	

List Command	set_wait	
Function	sets a wait marker (break point) in the list	
Parameter	`wait_word`	number of the wait marker as an unsigned 16-bit value [1 … 65535]
Integration	Pascal:	`function set_wait(wait_word: word);`
	C:	`void set_wait(unsigned short wait_word);`
	Basic:	`sub set_wait(ByVal wait_word%)`
Comments	• Processing of a list will be interrupted at each wait marker. The laser will be turned off. • If processing has been stopped at a wait marker, the command **get_wait_status** returns the number of this marker. • Processing of the list can be resumed by calling the command **release_wait**.	
References	**get_wait_status**, **release_wait**	

List Command	set_wobbel	
Function	defines a circular movement of the output position which is added to the regular marking movement	
Parameters	`amplitude`	amplitude of the circular wobbel movement in *bits* (unsigned 16-bit value) Allowed range: [1 … 5000]
	`frequency`	frequency of the wobbel movement in *Hz* (circles per second, 64-bit IEEE floating point value) Allowed range: [1 … 6000]
Integration	Pascal:	`procedure set_wobbel(amplitude: word; frequency: double);`
	C:	`void set_wobbel(unsigned short amplitude, double frequency);`
	Basic:	`sub set_wobbel(ByVal amplitude%, ByVal frequency#)`
Comments	• Calling the command set_wobbel sets the wobbel phase to a defined starting value (which is independent of the direction of the marking vector). • The wobbel function can be used for marking lines with variable line width. • A circular movement is added to the linear marking movement, resulting in a spiral movement of the laser focus in the image field. • A broad line is obtained by choosing suitable values for the amplitude and the frequency. The amplitude and the frequency must be appropriate for the current marking speed. • The wobbel function is terminated by setting the amplitude to zero.	
References	**set_mark_speed**	

Ctrl Command	**start_loop**
Function	starts an alternately repeating output of the two lists
Integration	Pascal: `procedure start_loop;`
	C: `void start_loop(void);`
	Basic: `sub start_loop()`
Comments	• The command **start_loop** can only be called if the two lists are loaded and closed, and if one of the lists is executing at the moment. See "Repeating Output" on page 13. • Both lists are alternately repeated until execution is terminated by calling the command **quit_loop**.
References	**quit_loop**

Ctrl Command	**stop_execution**
Function	stops execution of the list and turns off the laser immediately
Integration	Pascal: `procedure stop_execution;`
	C: `void stop_execution(void);`
	Basic: `sub stop_execution()`
Comments	• The mirrors will stay in the current position. Therefore, before loading a new list, the mirrors should be set to a defined position using the command **goto_xy**. • The external START inputs are disabled. The Processing-on-the-fly correction is turned off (Processing-on-the-fly option only). • A list that was interrupted with **stop_execution** *cannot* be resumed. Use the command **stop_list** if execution is to be temporarily stopped and resumed later.
References	**get_startstop_info**

Ctrl Command	**stop_list**
Function	pauses execution of the list and turns the laser off
Integration	Pascal: `procedure stop_list;`
	C: `void stop_list(void);`
	Basic: `sub stop_list()`
Comments	• The command **restart_list** *has to be used* to resume execution of the list.
References	**restart_list**, **stop_execution**

List Command	timed_jump_abs	
Function	jump to the specified position in the image field using the specified duration	
Parameters	`xval`, `yval`	absolute coordinates of the vector end point in *bits* as signed 16-bit values.
	`time`	duration of the complete jump vector in *microseconds* Allowed range: [10 … 655350] (64-bit IEEE floating point value)
Integration	Pascal:	`procedure timed_jump_abs(xval, yval: smallint; time: double);`
	C:	`void timed_jump_abs(short xval, short yval, double time);`
	Basic:	`sub timed_jump_abs(ByVal xval%, ByVal yval%, ByVal time#)`
Comments	• The parameter `time` will be rounded down to a multiple of 10 µs (within the allowed range). • A *timed* jump (mark) command will not be executed with the normal jump speed (marking speed). Instead, the speed (i.e. the number of microsteps) will be adjusted so that the vector lasts exactly as long as specified. Also see chapter 5.8 "Timed Jump And Mark Commands", page 50. • **Note:** After a timed jump, a *jump delay* is inserted, just like after a normal jump. That means the total time for the jump is the specified time *plus* the jump delay. • Subsequent **jump_abs** / **jump_rel** commands (not timed) will be executed with the normal jump speed. (See **set_jump_speed**, page 108.)	
References	**set_scanner_delays**, **timed_jump_rel**	

List Command	timed_jump_rel	
Function	jump to the specified position in the image field using the specified duration	
Parameters	`dx, dy`	*relative* coordinates of the vector end point in *bits* as signed 16-bit values.
	`time`	duration of the complete jump vector in *microseconds* Allowed range: [10 … 655350] (64-bit IEEE floating point value)
Integration	Pascal:	`procedure timed_jump_rel(dx, dy: smallint; time: double);`
	C:	`void timed_jump_rel(short dx, short dy, double time);`
	Basic:	`sub timed_jump_rel(ByVal dx%, ByVal dy%, ByVal time#)`
Comments	• The coordinates are relative to the current position. • The maximum value for the *absolute* coordinates is ±32767 bits. • See **timed_jump_abs**.	
References	**set_scanner_delays**, **timed_jump_abs**	

List Command	timed_mark_abs	
Function	marking vector to the specified position in the image field using the specified duration	
Parameters	`xval`, `yval`	absolute coordinates of the vector end point in *bits* as signed 16-bit values.
	`time`	duration of the complete mark vector in *microseconds* Allowed range: [10 … 655350] (64-bit IEEE floating point value)
Integration	Pascal:	`procedure timed_mark_abs(xval, yval: smallint; time: double);`
	C:	`void timed_mark_abs(short xval, short yval, double time);`
	Basic:	`sub timed_mark_abs(ByVal xval%, ByVal yval%, ByVal time#)`
Comments	• The parameter `time` will be rounded down to a multiple of 10 µs (within the allowed range). • A *timed* mark (jump) command will not be executed with the normal marking speed (jump speed). Instead, the speed (i.e. the number of microsteps) will be adjusted so that the vector lasts exactly as long as specified. Also see chapter 5.8 "Timed Jump And Mark Commands", page 50. • **Note:** After a timed mark command, a *mark delay* or a *polygon delay* is inserted, just like after a normal mark command. That means the total time for the command is the specified time *plus* the mark delay or polygon delay. • Subsequent **mark_abs** / **mark_rel** commands (not timed) will be executed with the normal marking speed. (See **set_mark_speed**, page 110.)	
References	**set_scanner_delays**, **timed_mark_rel**	

List Command	timed_mark_rel	
Function	marking vector to the specified position in the image field using the specified duration	
Parameters	`dx, dy`	*relative* coordinates of the vector end point in *bits* as signed 16-bit values.
	`time`	duration of the complete mark vector in *microseconds* Allowed range: [10 … 655350] (64-bit IEEE floating point value)
Integration	Pascal:	`procedure timed_mark_rel(dx, dy: smallint; time: double);`
	C:	`void timed_mark_rel(short dx, short dy, double time);`
	Basic:	`sub timed_mark_rel(ByVal dx%, ByVal dy%, ByVal time#)`
Comments	• The coordinates are relative to the current position. • The maximum value for the *absolute* coordinates is ±32767 bits. • See **timed_mark_abs**.	
References	**set_scanner_delays**, **timed_mark_abs**	

Ctrl Command	write_8bit_port	
Function	writes an 8-bit value to the digital output port on the LASER EXTENSION connector	
Parameter	value	output value for the digital output port as unsigned 16-bit value. Allowed range: [0 … 255]
Integration	Pascal:	`procedure write_8bit_port(value: word);`
	C:	`void write_8bit_port(unsigned short value);`
	Basic:	`sub write_8bit_port(ByVal value%)`
Comments	• Please refer to chapter "8-Bit Digital Output Port", page 53, and to the section "Digital Output Port (Laser Extension Connector)", page 60. • The upper 8 bits of the specified value are ignored.	
References	**write_8bit_port_list**	

List Command	write_8bit_port_list	
Function	same as **write_8bit_port** (see above), but a list command	
Integration	Pascal:	`procedure write_8bit_port_list(value: word);`
	C:	`void write_8bit_port_list(unsigned short value);`
	Basic:	`sub write_8bit_port_list(ByVal value%)`

Ctrl Command	write_da_1	
Function	see **write_da_x**	
Integration	Pascal:	`procedure write_da_1(value: word);`
	C:	`void write_da_1(unsigned short value);`
	Basic:	`sub write_da_1(ByVal value%)`

List Command	write_da_1_list	
Function	see **write_da_x_list**	
Integration	Pascal:	`procedure write_da_1_list(value: word);`
	C:	`void write_da_1_list(unsigned short value);`
	Basic:	`sub write_da_1_list(ByVal value%)`

Ctrl Command	write_da_2	
Function	see **write_da_x**	
Integration	Pascal:	`procedure write_da_2(value: word);`
	C:	`void write_da_2(unsigned short value);`
	Basic:	`sub write_da_2(ByVal value%)`

List Command	write_da_2_list	
Function	see **write_da_x_list**	
Integration	Pascal:	`procedure write_da_2_list(value: word);`
	C:	`void write_da_2_list(unsigned short value);`
	Basic:	`sub write_da_2_list(ByVal value%)`

Ctrl Command	write_da_x	
Function	writes a 10-bit value to one of the analog output ports of the RTC®4 or the RTC®4 I/O Extension Board	
Parameters	`x`	[1, 2] ANALOG OUT1 / ANALOG OUT2 ports on the RTC®4 board [3, 4, 5, 6] ANALOG OUT ports on the RTC®4 I/O Extension Board (optional)
	`value`	output value for the DA converter as unsigned 16-bit value. Allowed range: [0 … 1023]. If a value > 1023 is specified, the output value is set to 1023.
Integration	Pascal:	`procedure write_da_x(x, value: word);`
	C:	`void write_da_x(unsigned short x, unsigned short value);`
	Basic:	`sub write_da_x(ByVal x%, ByVal value%)`
Comments	• For the ANALOG OUT1 / ANALOG OUT2 ports, the commands **write_da_1** / **write_da_2** can be used alternatively (without parameter `x`). • Note that the output range of the ANALOG OUT1 port can be either 0 V … 2.56 V or 0 V … 10 V (see page 59), whereas the output range of the ANALOG OUT2 port is always 0 V … 10 V. Also see the supplement manual "I/O Extension Board".	

List Command	write_da_x_list	
Function	same as **write_da_x** (see above), but a list command	
Integration	Pascal:	`procedure write_da_x_list(x, value: word);`
	C:	`void write_da_x_list(unsigned short x, unsigned short value);`
	Basic:	`sub write_da_x_list(ByVal x%, ByVal value%)`

Ctrl Command	write_io_port	
Function	writes a value to the 16-bit digital output port on the "EXTENSION 1" connector	
Parameter	`value`	output value as unsigned 16-bit value (DIGITAL_OUT0 … DIGITAL_OUT15)
Integration	Pascal:	`procedure write_io_port(value: word);`
	C:	`void write_io_port(unsigned short value);`
	Basic:	`sub write_io_port(ByVal value%)`
Comments	• Use the commands **set_io_cond_list** and **clear_io_cond_list** to set/clear individual bits of the 16-bit digital output port, depending on the state of the *input* port.	
References	**write_io_port_list**, **read_io_port**	

List Command	write_io_port_list	
Function	same as **write_io_port**, but a list command	
Integration	Pascal:	`procedure write_io_port_list(value: word);`
	C:	`void write_io_port_list(unsigned short value);`
	Basic:	`sub write_io_port_list(ByVal value%)`
References	**write_io_port**	

Ctrl Command	z_out	
Function	sends a 16-bit value directly to the Z channel of the RTC®4 (CHAN3; see figure 25 on page 54)	
Parameters	`value`	Z output value (signed 16-bit value)
Integration	Pascal:	`procedure z_out(value: smallint);`
	C:	`void z_out(short value);`
	Basic:	`sub z_out(ByVal value%)`
Comments	• The output value is sent directly to the Z channel of the RTC®4. • The Z output value must be in the range −32768 … +32767 (signed 16-bit value). • **Note:** This command should only be used in 2D applications. In 3D applications which use the program file `RTC4D3.HEX`, the RTC®4 will overwrite the Z output value every 10 µs. Also see the supplement manual "3D Software".	
References	**z_out_list**	

List Command	z_out_list	
Function	same as **z_out**, but a list command	
Integration	Pascal:	`procedure z_out_list(value: smallint);`
	C:	`void z_out_list(short value);`
	Basic:	`sub z_out_list(ByVal value%)`

10.4 Supported and Unsupported RTC®2 Commands

The RTC®4 emulates some RTC®2 commands for compatibility. Though these emulated commands can be used with the RTC®4, new application software should only use the commands described in chapter 10.

This section also specifies RTC®4 replacements for those RTC®2 commands that are neither supported nor emulated.

The following table provides an overview of emulated RTC®2 commands and unsupported RTC®2 commands as well as appropriate RTC®4 replacements.

Ctrl Command	aut_change	
Support status	**The RTC®4 emulates this RTC®2 command**.	
Replaced by	**auto_change** (see page 75)	
Function	same as **auto_change** (see page 75)	
Integration	Pascal:	`procedure aut_change;`
	C:	`void aut_change(void);`
	Basic:	`sub aut_change()`

List Command	field_jump
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**home_position** (see page 90)

Ctrl Command	get_rtc2_mode
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	The RTC®4 has no need to query the mode, which is now be set via the software command **set_laser_mode** (see page 109) instead of hardware.

Ctrl Command	get_rtc2_version
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**get_rtc_version** (see page 85)

Ctrl Command	get_version
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**get_hex_version** (see page 83)

List Command	**home_jump**
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**home_position** (see page 90)

Ctrl Command	**laser_on**
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**laser_on_list** (see page 92)

Ctrl Command	**load_cor** *(load correction file)*	
Support status	**The RTC®4 emulates this RTC®2 command.**	
Replaced by	**load_correction_file** (see page 96)	
Function	loads the specified image field correction file into RTC®3 memory. The file is used as correction file #1.	
Parameter	`ctbfile`	name of the correction file as a pointer to a null-terminated ANSI string
Integration	Pascal:	`function load_cor(ctbfile: pchar): smallint;`
	C:	`short load_cor(char* ctbfile);`
	Basic:	`function load_cor(ByVal ctbfile$)%`

Ctrl Command	**load_pro** *(load program file)*	
Support status	**The RTC®4 emulates this RTC®2 command.**	
Replaced by	**load_program_file** (see page 97)	
Function	loads the specified image field correction file into RTC®3 memory. The file is used as correction file #1.	
Parameter	`hexfile`	name of the program file as a pointer to a null-terminated ANSI string
Integration	Pascal:	`function load_pro(hexfile: pchar): smallint;`
	C:	`short load_pro(char* hexfile);`
	Basic:	`function load_pro(ByVal hexfile$)%`

List Command	**pola_abs, polb_abs, polc_abs**
Support status	These RTC®2 commands are not supported by the RTC®4.
Replaced by	**mark_abs** (see page 99)

Ctrl Command	**set_base**
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**There is no equivalent command for the RTC®4** (because it is a plug-and-play board whose base address is automatically set).

Ctrl Command	set_co2_standby
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**set_standby** (see page 117)

List Command	set_co2_standby_list
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**set_standby** (see page 117)

List Command	set_delays
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**set_laser_delays** (page 108) and **set_scanner_delays** (page 114)

Ctrl Command	set_gain
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**set_matrix** (page 111) and **set_offset** (page 112) or **load_correction_file** (page 96)

Ctrl Command	set_mode
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**There is no equivalent command for the RTC®4**, because its image field correction algorithm is always enabled. Instead, a 1-to-1 correction file can be loaded.

Ctrl Command	set_speed
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**set_jump_speed** (page 108) and **set_mark_speed** (page 110)

Ctrl Command	set_yag_parameter
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**There is no equivalent command for the RTC®4.**

Ctrl Command	write_da
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**write_da_1** (page 123)

List Command	write_da_list
Support status	This RTC®2 command is not supported by the RTC®4.
Replaced by	**write_da_1_list** (page 123)

11 Troubleshooting

Problem	Remedy
PC does not boot	Switch off the PC and check the following: • Check if the RTC®4 board is correctly seated in the PCI slot. Please refer to the instructions in your PC manual. • Check for metal parts that may have fallen into the PC housing during installation of the RTC®4. • Check for loose cables or connectors
RTC®4 does not respond	• Check the software driver installation. See chapter 9.1, page 62. • Was the software driver mistakenly installed *after* the RTC®4 board was installed (driver installation should *precede* board installation)? • Check the DLL import declarations in your application software. See chapter 9.2 "DLL Calling Convention", page 62.
Application software fails	• Check the software driver installation. See chapter 9.1, page 62. • Check the RTC®4 initialization in the application software. • Check the DLL import declarations in your application software. See chapter 9.2 "DLL Calling Convention", page 62.
Scan head control fails	• Check if the scan head is properly connected to the RTC®4 via the data cable. Make sure to follow the specifications for the data cable. See section "Data Cable", page 55. • Check the power supply of the scan head. Please refer to your scan head operating manual. • Check your application software.
Laser control fails	• Check the interface between the RTC®4 and the laser.
Irregular marking results	• Check the laser and scanner delays. See "Notes On Optimizing The Delays", page 24.
Laser doesn’t switch off during jump commands	• Check the laser and scanner delays. See page 24.

If the problems persist, please contact SCANLAB.

12 Customer Service

12.1 Servicing and Repairs

All servicing and repairs should be performed only at SCANLAB.The warranty expires if the board has been altered.

12.2 Warranty

SCANLAB guarantees this product to be free of defects in manufacturing and material. The warranty is valid for 12 months after delivery. Repairs covered under the warranty will be performed at SCANLAB.

The scope of the warranty is limited to repair or replacement of the SCANLAB product.

SCANLAB is responsible for the return delivery of products repaired under warranty; the customer is responsible for delivery to SCANLAB.

SCANLAB will not be held responsible:

- when the product has been damaged through misuse or improper operation
- for repairs not performed by SCANLAB
- if the RTC®4 board has been altered
- for damage resulting from improper packaging of a product returned to SCANLAB
- for consequential damages

12.3 Contacting SCANLAB

For service, repairs, advice or information, simply contact SCANLAB using one of the contact possibilities listed below:

SCANLAB AG
Siemensstr. 2a
82178 Puchheim
Germany

Tel. +49 (89) 800 746-0
Fax: +49 (89) 800 746-199

info@scanlab.de

www.scanlab.de

12.4 Product Disposal

The RTC®4 can be returned to SCANLAB for a fee to be properly disposed of.

13 Technical Specifications

System Requirements

IBM-compatible PC with PCI bus interface

Operating system	Microsoft WINDOWS XP and WINDOWS 2000
Minimum free space above the PCI slot	105 mm

Dimensions

Length	160 mm
Height	102 mm

Connectors, I/O Signals

Laser

Connector	9-pin D-SUB connector, female
Maximum current load of the laser signals	10 mA

Scan Head

Connector	25-pin female connector or ST sockets
Signals	XY2-100 or XY2-100-O protocol

Scan Head Control

Number of list buffers	2
Capacity of a single list	4000 commands
Position update period (microstep period)	10 µs
Maximum range for the image field coordinates	−32768 to +32767 (16-bit signed)

Digital Input Port	16 bits
LOW level	< 0.5 V
HIGH level	2.6 V … 24 V
Input resistance	> 10 kΩ

The input signals are referenced to GND.

Digital Output Ports	8 bits, 16 bits, buffered
Maximum output current	±8 mA
Output voltage	
LOW level	< 0.4 V
HIGH level	> 2.0 V

The output signals are referenced to GND.

Analog Output Ports

ANALOG OUT1	0 V … 2.56 V or 0 V … 10 V, 10-bit resolution
ANALOG OUT2	0 V … 10 V, 10-bit resolution
Maximum current load	5 mA *

* = If jumper X6 has been configured to replace the ANALOG OUT 1 signal with +5 V, then the maximum current load at pin (4) of the laser connector is 100 mA.

Index

D

E

F

G

SCANLAB

innovators for industry

T

U

V

W

X

Y

Z

Notes