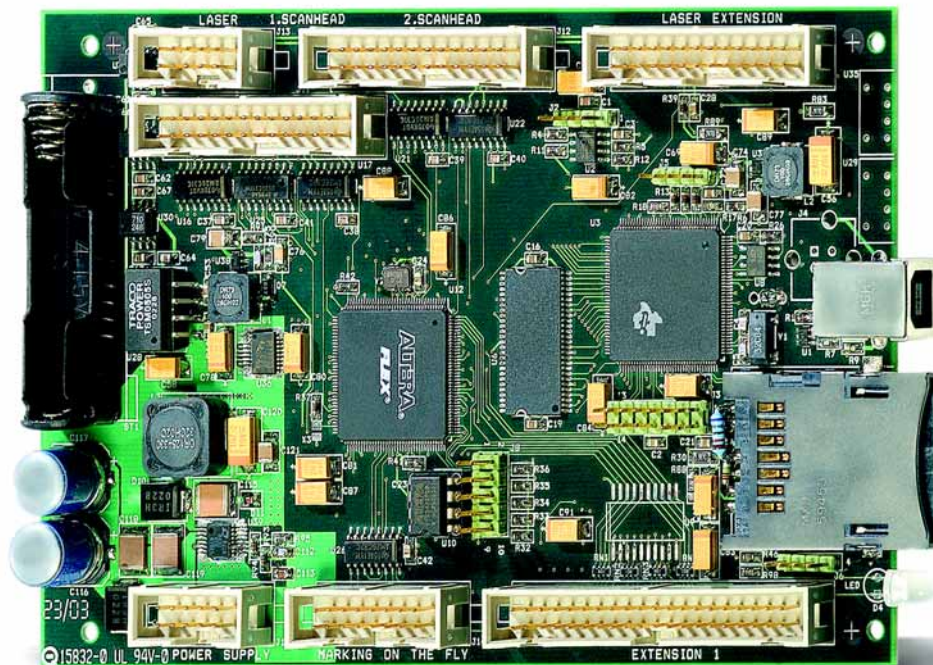




## Installation and Operation

### RTC<sup>®</sup> SCANalone Board

for PC-Independent 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](mailto:info@scanlab.de)  
[www.scanlab.de](http://www.scanlab.de)

© SCANLAB AG 2010

(Doc. Rev. 1.3 e - March 5, 2010)

SCANLAB reserves the right to change the information in this document without notice.

No part of this manual may be processed, reproduced or distributed in any form (photocopy, print, microfilm or by any other means), electronic or mechanical, for any purpose without the written permission of SCANLAB.

All mentioned trademarks are hereby acknowledged as properties of their respective owners.



## Contents

<b>1</b>	<b>Delivered Product .....</b>	<b>6</b>
1.1	Package Contents .....	6
1.2	Manufacturer .....	6
1.3	About This Operating Manual .....	6
<b>2</b>	<b>Product Overview .....</b>	<b>7</b>
2.1	Intended Use .....	7
2.2	System Requirements .....	7
2.3	Board Dimensions And Layout .....	8
2.4	Notes For RTC® 4 Users .....	10
	Hardware Changes .....	10
	Installation Tips .....	10
	Changed Commands .....	10
<b>3</b>	<b>Safety During Installation And Operation .....</b>	<b>12</b>
3.1	Steps For Safe Operation .....	12
3.2	Laser Safety .....	12
<b>4</b>	<b>Principle Of Operation .....</b>	<b>13</b>
4.1	Software Concept .....	13
	Standalone and PC Operation .....	13
	List Commands And Control Commands .....	14
	List Handling .....	15
	Automatic List Handling .....	15
	External Control Inputs .....	16
	Synchronization Of Processing .....	16
4.2	Scan Head And Laser Control .....	17
	Vector Commands .....	17
	Arc Commands .....	17
	Microsteps .....	17
4.3	Delays .....	19
	Laser Delays .....	19
	Scanner Delays .....	20
	Notes On Optimizing The Delays .....	27
4.4	Image Field Size .....	32
4.5	Image Field Correction .....	33
4.6	Laser Control .....	35
	CO <sub>2</sub> Mode .....	35
	YAG Modes .....	36
	Softstart Mode .....	37
	Laser Mode 4 .....	40
4.7	Status Monitoring and Diagnostics .....	41

<b>5</b>	<b>Advanced Programming</b>	<b>42</b>
5.1	Coordinate Transformations	42
5.2	Wobbel Function	43
5.3	Using Two Different Correction Files	44
	Double Scan Head Configuration	44
	Using Two Correction Files In A Single Scan Head System	44
5.4	Structured Programming	45
	Input Pointer	45
	List Jumps	45
	Conditional List Jumps	45
	Output Pointer	45
	Programming Examples	46
5.5	Scanning Raster Images (Bitmaps)	47
5.6	Timed Jump And Mark Commands	51
5.7	Marking the Date, Time and Serial Numbers	51
<b>6</b>	<b>Electrical Connections</b>	<b>53</b>
6.1	Power Supply	53
	Power Requirements	53
	Reset	53
6.2	USB Connection to a PC	53
6.3	MMC Memory Card	53
6.4	Laser Connector	54
	Analog Output Ports	54
	Laser Signals	54
	External Control Signals	54
6.5	Laser Extension Connector	55
	Digital Output Port	55
	Laser Signals	55
6.6	Primary Scan Head Connector	56
	Control Signals	56
	Status Signals	56
	Data Cable	56
6.7	Secondary Scan Head Connector (Optional)	57
6.8	Optical Data Interface (Optional)	57
6.9	Digital 16-bit Input and Output	58
6.10	10-bit Analog Inputs	58
6.11	Processing-on-the-fly (Optional)	59
6.12	Status LEDs	59
6.13	Battery-Powered Clock and Calender	59
<b>7</b>	<b>Options</b>	<b>60</b>
<b>8</b>	<b>Installation And Start-Up</b>	<b>61</b>
8.1	Jumper Settings Overview	61
8.2	Changing The Jumper Settings	62
	TTL Laser Signal Level	62
	Laser / Analog Output Ports	62
	"LASER EXTENSION/Digital Output Port	63
8.3	Installing the RTC® SCANalone Board	64
8.4	Connecting to the PC and Power Supply	64
8.5	Installing the Software Driver	64
8.6	Connecting to the Scan System and Laser	64

<b>9 Software</b>	<b>65</b>
9.1 Installing the Software	65
9.2 DLL Calling Convention	65
Pascal	65
Basic	65
C	66
Automatic Assignment of Access Rights	66
9.3 Initializing the RTC® SCANalone	67
9.4 Demo Program	67
9.5 Functionality Test	67
<b>10 Commands And Functions</b>	<b>68</b>
10.1 Overview	68
Control Commands	69
List Commands	70
10.2 Data Types	71
10.3 Command Set	72
10.4 Supported and Unsupported RTC® Commands	128
<b>11 Customer Service</b>	<b>134</b>
11.1 Servicing and Repairs	134
11.2 Warranty	134
11.3 Contacting SCANLAB	134
11.4 Product Disposal	134
<b>12 Technical Specifications</b>	<b>135</b>

## 1 Delivered Product

### 1.1 Package Contents

The contents of the package and the RTC® SCANalone 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® SCANalone 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® SCANalone. 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® SCANalone software and firmware:

- DLL: RTC\_SCANalone.DLL, version 500 or higher,
- DSP program (already installed on the RTC® SCANalone board):  
(hex-)version 2.500 / 3.500 or higher,
- RTC® SCANalone firmware: version 128 or higher.

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

- **get\_dll\_version** (page 77);
- **get\_hex\_version** (page 78) and
- **get\_rtc\_version** (page 79)

#### Supplements To This Manual

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

- "3D Software" manual
- "Processing-On-The-Fly Software" manual
- "correXion and CFMP" manual

## 2 Product Overview

### 2.1 Intended Use

The RTC® SCANAlone board from SCANLAB is designed to provide real-time control of scan systems and lasers without requiring a PC. The board's high-performance signal processor and extensive internal memory make this possible.

Marking data can be loaded via a removable MMC memory card or by using the built-in USB 1.1 interface. The board's internal memory can accommodate up to one million list commands – a capacity that meets the needs of both today's and tomorrow's complex applications.

An external control signal can be used to start or otherwise influence the execution of applications loaded in memory. For this purpose, the board is equipped with an additional 16-bit digital input and 16-bit digital output.

Control commands to scan systems are issued every 10 µs as 16-bit digital output signals.

The RTC® SCANAlone can also be operated via a PC connected to the board's USB interface. In this mode, the RTC® SCANAlone offers the same functionality as an RTC® 4 PC interface board.

The RTC® SCANAlone's software interface and hardware connection capabilities are largely compatible with those of the RTC® 4.

In its standard configuration, the RTC® SCANAlone consists of a base board with the following features:

- Laser port ("LASER") providing programmable laser signals, two 10-bit analog outputs and inputs for external control signals
- Laser extension interface with 8-bit digital output and laser signals ("LASER EXTENSION")
- Primary scan head connector ("1.SCAN HEAD")
- 16-bit digital input and output ("EXTENSION1")
- Two 10-bit analog inputs
- Status-LED for PWROK and BUSY
- USB 1.1 interface to PC
- MMC memory card
- Power requirement of +7...+30 V DC
- Battery-powered clock/calender (1.5 V AAA alkaline battery)

The RTC® SCANAlone is optionally available in a 19-inch rack-mount version. The RTC® SCANAlone can also be ordered with an extension board containing D-SUB connectors that facilitate front panel mounting. The header pin signals on the RTC® SCANAlone's base board are available at the extension board's D-SUB connectors.

Additionally obtainable from SCANLAB are the following optional extensions:

- Control signals for simultaneous control of a second scan head, with individual image field correction for each scan head
- Control signals for Processing-on-the-fly applications
- Control signals for a third axis, e.g. a varioSCAN
- Optical Data Interface
- Opto-decoupled laser signals

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

The dimensions of the RTC® SCANAlone's base board and extension board are indicated in [figure 1 on page 8](#) and [figure 2 on page 9](#).

### 2.2 System Requirements

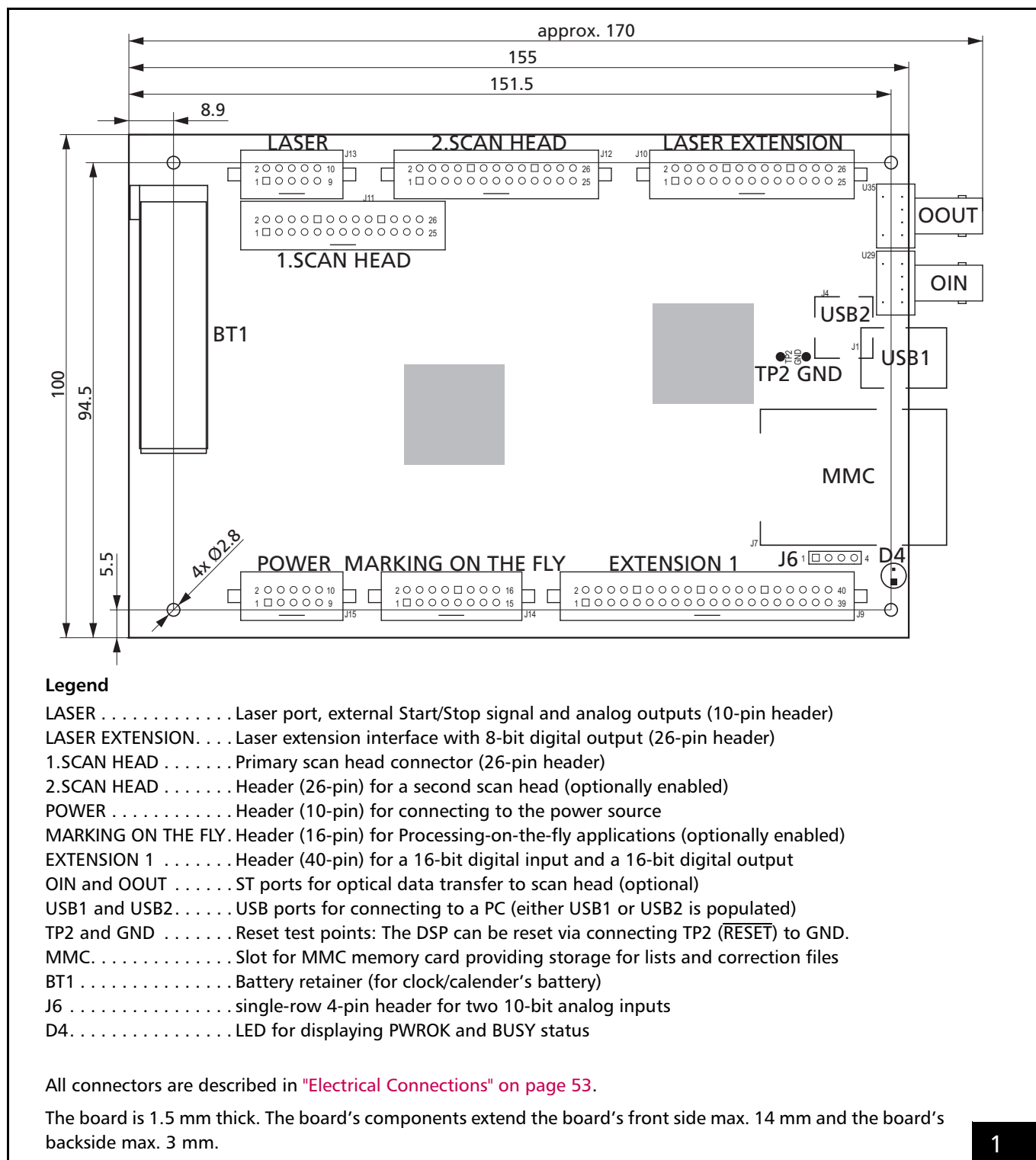
#### Hardware

The RTC® SCANAlone can be connected to any IBM-compatible personal computer equipped with a USB 1.1 interface.

#### Software

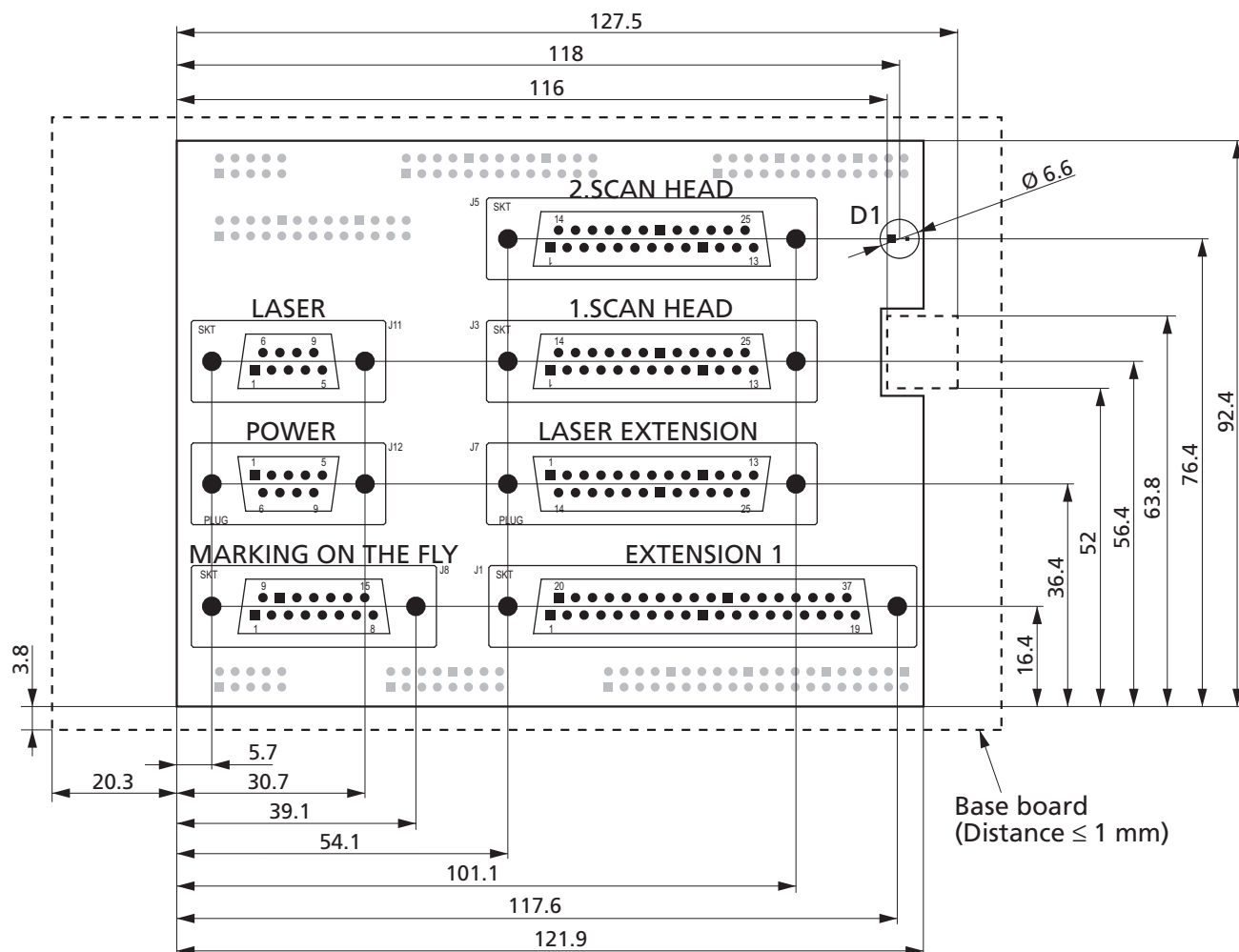
Software drivers for WINDOWS Vista / XP / 2000 and WINDOWS ME / 98 are included in the package.

## 2.3 Board Dimensions And Layout



Front side of the RTC® SCANAlone's base board with dimensions (all measurements in mm)





## Legend

- LASER . . . . . Laser port, external Start/Stop signal and analog outputs (9-pin D-SUB socket)
- LASER EXTENSION . . . . . Laser extension interface with 8-bit digital output (25-pin D-SUB plug)
- 1.SCAN HEAD . . . . . Primary scan head connector (25-pin D-SUB socket)
- 2.SCAN HEAD . . . . . D-SUB socket (25-pin) for a second scan head (optionally enabled)
- POWER . . . . . D-SUB plug (9-pin) for connecting to the power source
- MARKING ON THE FLY . . . . . D-SUB socket (15-pin) for Processing-on-the-fly applications (optionally enabled)
- EXTENSION 1 . . . . . D-SUB socket (37-pin) for a 16-bit digital input and a 16-bit digital output
- D1 . . . . . LED for displaying PWROK and BUSY status (optional)

All connectors are described in "Electrical Connections" on page 53.

## 2.4 Notes For RTC®4 Users

This section summarizes the hardware and software differences between RTC® SCANalone boards and RTC® PC interface boards. Tips are also provided to RTC® PC interface board users on how to install an RTC® SCANalone board.

### Hardware Changes

Because it is a standalone board, a few conceptual aspects of the RTC® SCANalone differ from those of an RTC® PC interface board:

- The RTC® SCANalone requires an external power supply (see ["Power Supply", page 53](#)).
- Connection to a controller (PC, laptop) is via a USB interface (see ["USB Connection to a PC", page 53](#)).
- The RTC® SCANalone provides a larger list buffer and an MMC memory card slot (see ["MMC Memory Card", page 53](#)).

The RTC® SCANalone Board's functionality is comparable to that of the RTC®4 PC Interface Board. Like an RTC®4, the RTC® SCANalone is therefore also equipped with such features as the EXTENSION 1 connector that provides a 16-bit digital input, a 16-bit digital output and access to the BUSY status signal (see ["Digital 16-bit Input and Output", page 58](#)). The RTC® SCANalone can also be optionally equipped with an optical data interface (see ["Optical Data Interface \(Optional\)", page 57](#)). In contrast to the RTC®4, only *one* scan head is controllable via the RTC® SCANalone's optical interface.

The RTC® SCANalone is additionally equipped with two 10-bit analog inputs, a battery-powered clock/calender and status LEDs (see ["Electrical Connections", page 53](#)).

### Installation Tips

As a standalone unit, the RTC® SCANalone board requires connection to a power supply and, when necessary, a controller (PC). The RTC® SCANalone must also be securely mounted (also see ["Installing the RTC® SCANalone Board", page 64](#)).

The RTC® SCANalone requires installation of an extra driver (see ["Installing the Software Driver", page 64](#)), but the RTC® SCANalone and other RTC® drivers do not conflict with another. Therefore, the RTC® SCANalone is capable of being driven by a computer already equipped with RTC® PC interface boards that are in use.

Software installation and DLL calls are identical to those of the RTC® PC interface boards, with the exception that no DSP program files need be loaded - they are already permanently pre-installed on the RTC® SCANalone (also see ["Installing the Software", page 65](#)).

### Changed Commands

The RTC® SCANalone's command set is based on that of the RTC®4 PC-Interfacekarte. If an RTC® SCANalone is connected to a PC (PC operation), it can be operated like an RTC®4 PC-Interfacekarte. In PC operation, control commands can be used, for example, for influencing program flow even during processing of command lists.

The RTC® SCANalone has a large list buffer and an MMC memory card, both of which enable it to operate as a standalone unit. Command lists transmitted from a controller (PC) to the RTC® SCANalone can be stored for standalone operation. An external start signal can be used to trigger operations in standalone mode.

Note that creation of command lists for standalone operation differs from that for PC operation (or RTC® PC interface board operation) in the following way: control commands for standalone operation cannot be used for influencing program flow, because control commands (in contrast to list commands) are not stored on the MMC.

However, initializations via control commands in PC operation (e.g. loading correction files, setting the laser mode) are storable on the MMC. This way, standalone initialization can be established for the RTC® SCANalone after a new start.

Additionally, some control command functionality is available via analogous list commands so that many important setting changes can be incorporated into command lists for standalone operation.

Also refer to ["Standalone and PC Operation", page 13](#) and ["List Commands And Control Commands", page 14](#).

## New Commands

- Commands for storing command lists on MMC memory cards (see "[Standalone Operation](#)", [page 14](#))
- Commands for [Marking the Date, Time and Serial Numbers](#) (see [page 51](#)).
- Control commands for querying the two 10-bit analog inputs ([read\\_io\\_port](#), [page 97](#))
- Control command for defining the list start position that should be triggered via the next external start signal ([set\\_extstartpos](#), [page 104](#))
- Control command for verifying correct command transmission capability between a controller and the RTC<sup>®</sup> SCANalone ([usb\\_status](#), [page 124](#))

As with the RTC<sup>®</sup>4, [Status Monitoring and Diagnostics](#) (see [page 41](#)) commands are available as well as I/O commands for accessing the 16-bit digital input and 16-bit digital output (see "[Digital 16-bit Input and Output](#)", [page 58](#) and "[Conditional List Jumps](#)", [page 45](#)).

## Unsupported Commands

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

The RTC<sup>®</sup> SCANalone cannot be expanded via an I/O board. Commands requiring an I/O board are therefore not usable.

Additionally, the following RTC<sup>®</sup> commands and functions are not supported by the RTC<sup>®</sup> SCANalone:

- Multi-board commands
- Circular queue mode
- Automatic self-calibration
- Loading and starting DSP program files

The section "[Supported and Unsupported RTC<sup>®</sup> Commands](#)", [page 128](#) lists all emulated RTC<sup>®</sup>2 software commands as well as unsupported RTC<sup>®</sup> commands.

### 3 Safety During Installation And Operation

Please read these operating instructions completely before you proceed with installing and operating the RTC® SCANalone 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 electro-magnetic fields and static electricity. These can damage the electronics on the RTC® SCANalone board. For storage, always use the antistatic bag the RTC® SCANalone is delivered in.
- The allowed operating temperature range is 15 °C to 60 °C.
- The storage temperature should be between –20 °C and +60 °C.



##### 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".

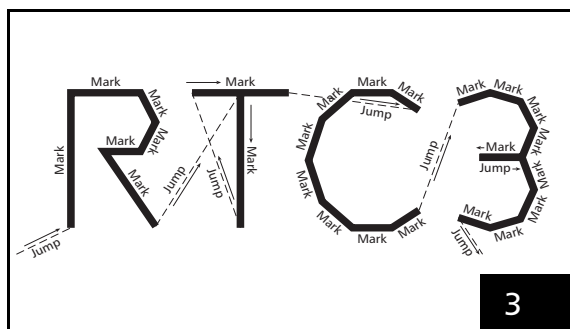
- **Always turn on the RTC® SCANalone and the power supply for the scan head first before turning on the laser. Otherwise there is the danger of uncontrolled deflection of the laser beam.**

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<sup>(1)</sup>. The image is made up of straight line segments or vectors. The RTC® SCANAlone 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® SCANAlone 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® SCANAlone 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® SCANAlone, also in accordance with the XY2-100 (or XY2-100-O) protocol.

The RTC® SCANAlone 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.

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

### Standalone and PC Operation

The RTC® SCANAlone can be used like an RTC®4 ("PC operation") by connecting a PC to the USB 1.1 port. User applications are then written to the RTC® SCANAlone's list buffer and started via software commands or external start signals. The correct execution of PC-to-RTC® SCANAlone command transfer can be verified via the **usb\_status** (see page 124) command.

Before the RTC® SCANAlone can be used independently of a PC ("standalone operation"), a user application must first be stored onto the MMC memory card. However, a PC cannot directly write or read an MMC card. Instead, user programs are usually transferred from the PC to the RTC® SCANAlone's list buffer and then stored onto the MMC via a software command.

If the RTC® SCANAlone is started up with an MMC that contains a user program, then this program will be automatically loaded (in standalone or PC operation) into list memory.

If the RTC® SCANAlone is started up without a PC (in standalone operation) then after the user program is loaded into the list buffer, the external start signal input will also be enabled. The program will run when an appropriate start signal is transferred to the RTC® SCANAlone.

User programs should first be tested in PC operation before they are stored on an MMC and started in standalone operation.

## List Commands And Control Commands

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

### PC Operation

In PC operation, **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® SCANalone in PC operation, a list must be opened via a control command. List commands sent to the RTC® SCANalone afterwards are not executed immediately, but stored in a list buffer. Only after the list is closed and started, the RTC® SCANalone reads the commands from the list buffer and processes them in real time. The RTC® SCANalone provides two list buffers. Each list buffer can hold one list ("list 1" or "list 2") with up to 500000 list commands. If an application needs only one list, then the RTC® SCANalone's entire memory can be treated like a single list with a capacity for 1000000 commands. In this case, list 1 can be loaded with up to 1000000 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® SCANalone commands are described in detail in [chapter 10 "Commands And Functions"](#). An overview is provided in [chapter 10.1, page 68](#).

The command descriptions in [chapter 10 "Commands And Functions"](#) provide guidance on command usage in PC and standalone operation.

### Standalone Operation

Command lists (**list commands** stored as lists in the RTC® SCANalone's internal memory) can be transferred to the MMC card via the **store\_on\_mmc** (see [page 121](#)) command. This command overwrites all data previously stored on the MMC card.

Writing (and reading) the MMC's entire list capacity (1000000 list positions) can take up to several minutes. Therefore, only list commands from position 0 to the final entry are stored on the MMC. Unnecessary memory transfer time results if both lists are written and list 1 only contains a few list entries, because list 1 will then be stored in its entirety (500000 list positions). It is therefore preferable to use only one list (with up to 1000000 commands).

List memory contents are retained until they are either overwritten or deleted via the **clear\_list** (see [page 74](#)) command. Unneeded MMC list entries (e.g. from previous test runs) can be avoided by completely erasing the MMC with **clear\_list** prior to final PC-to-MMC downloading of list memory (a process which appends to MMC memory – list memory contents are retained until they are either overwritten or deleted via **clear\_list**).

Unlike list commands, **control commands** (transferred from the PC to the RTC® SCANalone) are not storable on the MMC. They cannot be used in standalone operation. They can only be used in PC operation.

However, some control commands specify fundamental settings (e.g. setting the laser control mode or specifying the correction file) needed to initialize RTC® SCANalone in standalone operation. Therefore, the **store\_on\_mmc** command can be used to store the RTC® SCANalone's current state onto the MMC. When the RTC® SCANalone is later booted up in standalone or PC operation, it will read the MMC and reestablish the specified state that had previously been achieved with control commands.

## List Handling

Command lists can only be created in PC operation. In PC operation, a number of commands are available for handling lists and querying their status.

### PC Operation

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 76) or by an external start signal (see the section "External Control Inputs" on page 16).

During the execution of one list, the other list buffer can be loaded with the next list. The host computer and the RTC® SCANAlone 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® SCANAlone laser signals.

To examine the current status of the lists, the commands **get\_status** (page 81) or **read\_status** (page 98) can be used.

### Standalone Operation

In standalone operation, command lists can be started via an external start signal (see "External Control Inputs" on page 16). The previously-described handling and status commands cannot be used. However, as in PC operation, control of a program's execution can be achieved via external control signals (see "Conditional List Jumps" on page 45).

## Automatic List Handling

Because of the extensive storage capacity of the RTC® SCANAlone's list memory, SCANLAB recommends loading only *one* list (with up to 1 000 000 list commands) into the list memory. The automatic list changing commands described below are only intended for PC operation. (The RTC® 3/RTC® 4's circular queue mode is not available on the RTC® SCANAlone.

### Continuous Transfer

The commands **auto\_change** and **auto\_change\_pos** (see page 73) 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 500 000 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® SCANAlone's entire memory is to be treated like a single list. The start position (address in the list buffer) 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® SCANAlone 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.



## External Control Inputs

To also enable start and stop execution of a list via external signals (in PC operation or in standalone operation), the RTC® SCANalone provides two external control inputs, /START and /STOP (TTL active-low). These control inputs are accessible via the 10-pin “LASER” connector or the 9-pin D-SUB “LASER” connector on the RTC® SCANalone board – see [figure 25](#) and [figure 26 on page 54](#). 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® SCANalone laser signals are turned off. This is equivalent to calling the command [stop\\_execution](#) (see [page 120](#)) in PC operation.

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 in PC operation, it has to be enabled with the command [set\\_control\\_mode](#) (see [page 102](#)). 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.

In standalone operation, the /START is automatically enabled after boot-up.

In the default setting, List 1 is started via an external start signal. The desired list can be selected via the control commands [select\\_list](#) ([page 100](#)) or [set\\_extstartpos](#) ([page 104](#)) or with the list command [set\\_extstartpos\\_list](#) (see [page 104](#)).

To check whether a list was successfully started, the command [get\\_startstop\\_info](#) (see [page 80](#)) can be used (in PC operation only).

### 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 102](#)) 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 following commands can only be used in PC operation.

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

The command [get\\_wait\\_status](#) (see [page 83](#)) 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 98](#)).



## 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<sup>(1)</sup>. Each vector starts at the *current output position*, which is usually the end point of the preceding vector.

If the RTC® SCANalone starts up in PC operation without an MMC, then the initial output position will be the center of the image (0|0). If the RTC® SCANalone starts up with an MMC installed, then the initial output position is determined by the corresponding commands that were stored on the MMC.

Please refer to [chapter 4.4 "Image Field Size", page 32](#) 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 106](#)).

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 84](#)).

### Mark Commands

The RTC® SCANalone 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® SCANalone 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.

(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]".

The list command [set\\_mark\\_speed](#) (see [page 109](#)) defines the marking speed. The marking speed can be changed anywhere in a list.

### Arc Commands

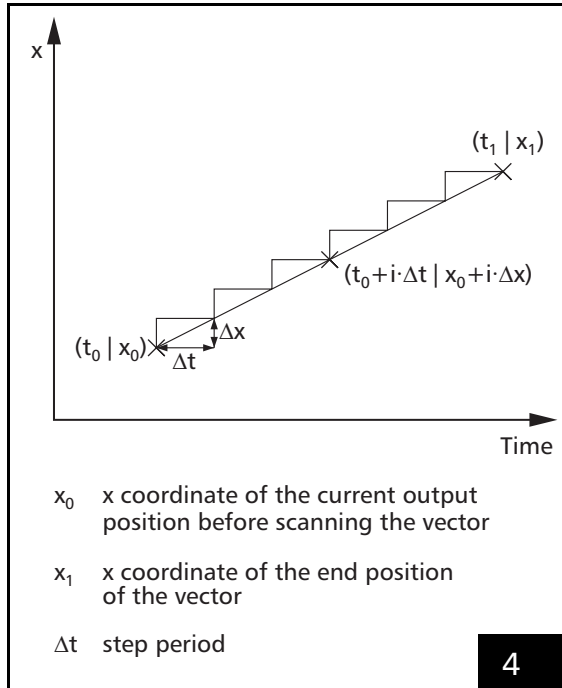
The RTC® SCANalone software driver provides arc commands for marking circular arcs. These commands require parameters for the X and Y coordinates of the arc center and the arc angle. The arc starts at the current output position.

At the beginning of an arc command, the RTC® SCANalone 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® SCANalone. These microsteps are transferred to the scan head at a constant time rate (*output period*  $\Delta t$ ). 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.



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

The length  $\Delta s$  of each microstep is

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

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

The output period  $\Delta t$  of the position update is usually fixed at 10  $\mu s$ . 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<sup>®</sup> SCANalone 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  $\mu s$  would possibly lead to more than  $2^{16}$  microsteps. In this case, the RTC<sup>®</sup> SCANalone automatically increases the output period to 20  $\mu s$  (or to a suitable multiple of 10  $\mu s$ , if necessary).

## Marking Time

The following commands can only be used in PC operation.

The marking time consumed by any particular marking process can be measured by calling the command **save\_and\_restart\_timer** (see page 99) before and after the marking process. This command saves the current value of the RTC<sup>®</sup> SCANalone's integrated timer and resets the timer value to 0. The measured time can be read via the command **get\_time** (see page 81), 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 work-piece 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 107). The time resolution for the laser delays is 1  $\mu$ s.

The delays must be appropriate for the defined jump speed and marking speed (also see "Notes On Optimizing The Delays", page 27).

### LaserOn Delay

The LaserOn delay defines the moment when the RTC® SCANalone 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 22** 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 22**. 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® SCANalone inserts one of these delays before the next command is started.

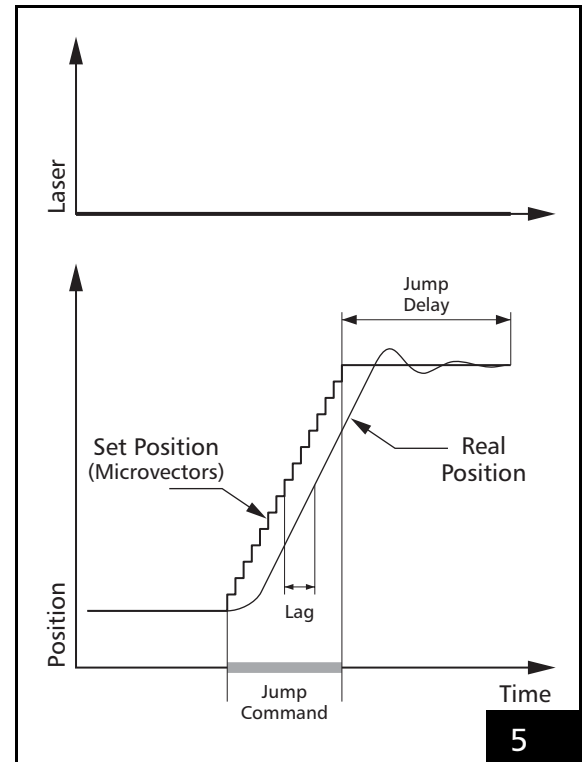
The command **set\_scanner\_delays** (see page 113) defines the scanner delays. The time resolution for the scanner delays is **10 µs**.

### 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 20.

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® SCANalone 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.

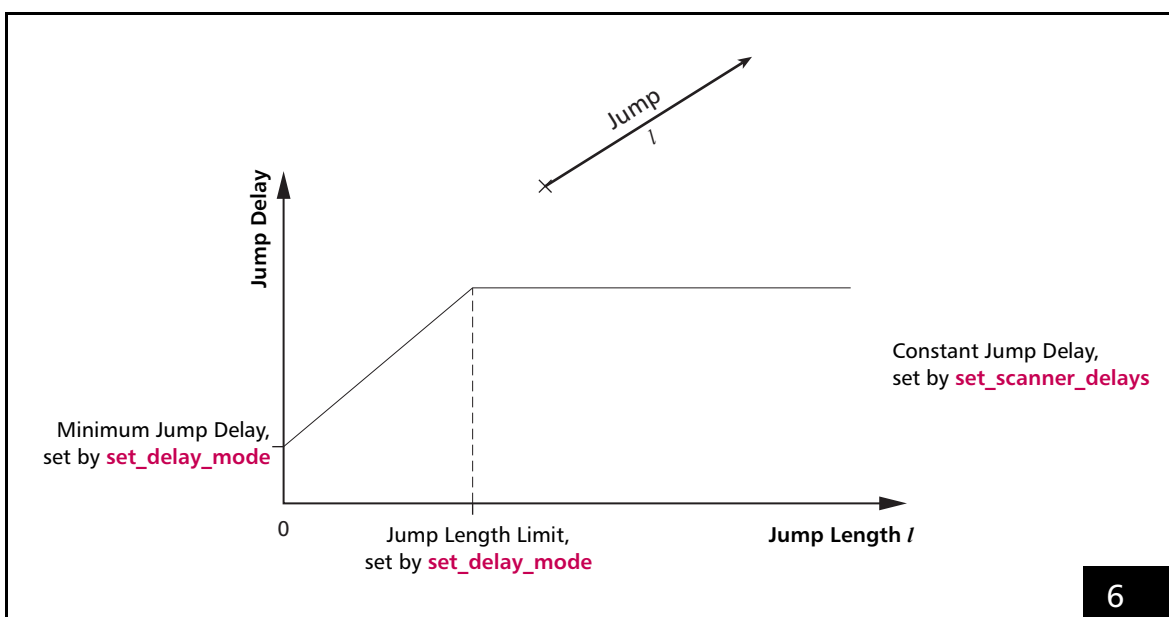
However, if a jump vector is very short, the scanners might not reach the full jump speed during the jump because of the inertia of the mirrors. In this case, a shorter jump delay might be sufficient for settling.

To make use of this, the RTC® SCANAlone 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 21](#). 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 103](#)).

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

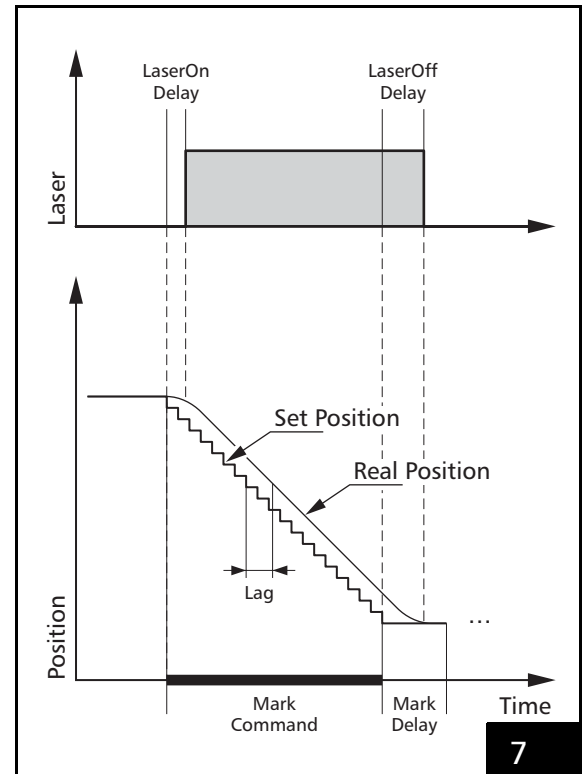
## Mark Delay

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® SCANalone 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 22](#).

### 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.

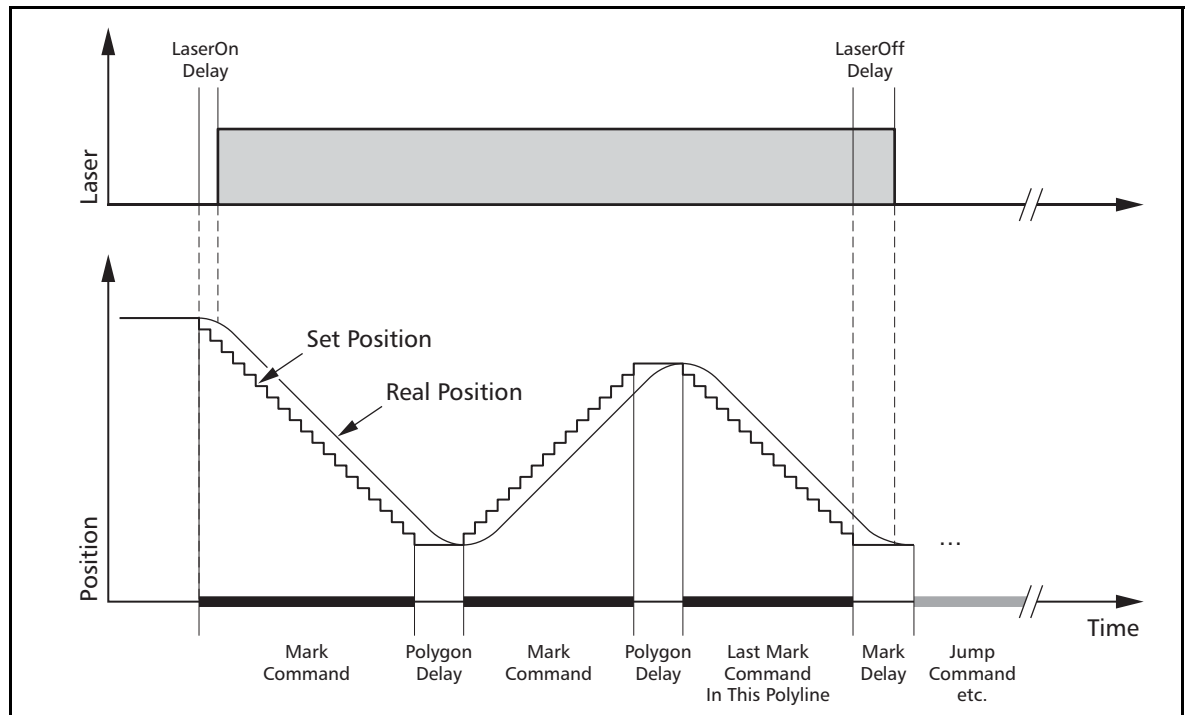


Scan head and laser control timing during a mark or arc command with a mark delay. Grey shaded areas indicate that the laser is on.

## 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 23](#).

The mark delay and the polygon delay can be set independently. In addition, the RTC® SCANAlone 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 24](#) for details.



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

## Variable Polygon Delay

A variable polygon delay mode can be activated via the command **set\_delay\_mode** (page 103). In this mode, the RTC® SCANAlone allows varying the length of the polygon delay, depending on the angle  $\phi$  between the two successive marking vectors – see figure 9 on page 24, below.

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

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

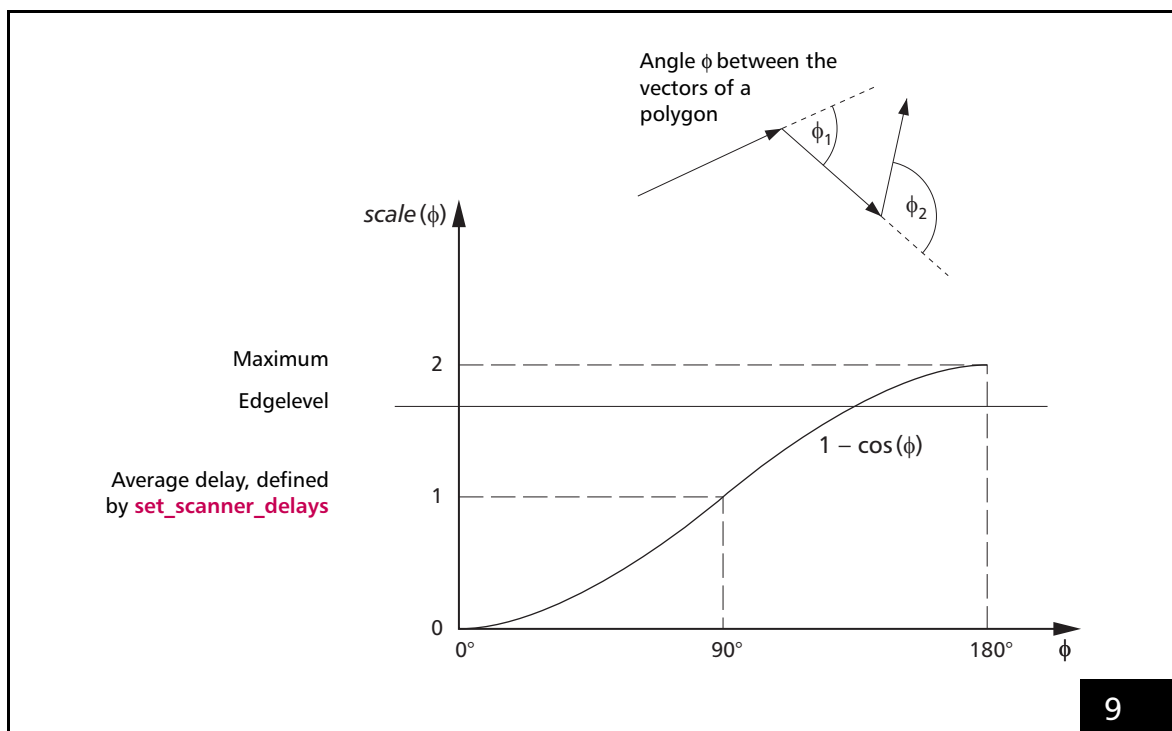
where  $scale(\phi)$  is a scaling function ( $0 \leq scale(\phi) \leq 2$ ). The parameter  $polygon\_delay$  is set by the command **set\_scanner\_delays**.

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 26.

## Edgelevel

Figure 9 shows that the variable polygon delay becomes quite long if the angle  $\phi$  is close to  $180^\circ$ . 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® SCANAlone 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 25.

For further details see **set\_delay\_mode**, page 103.

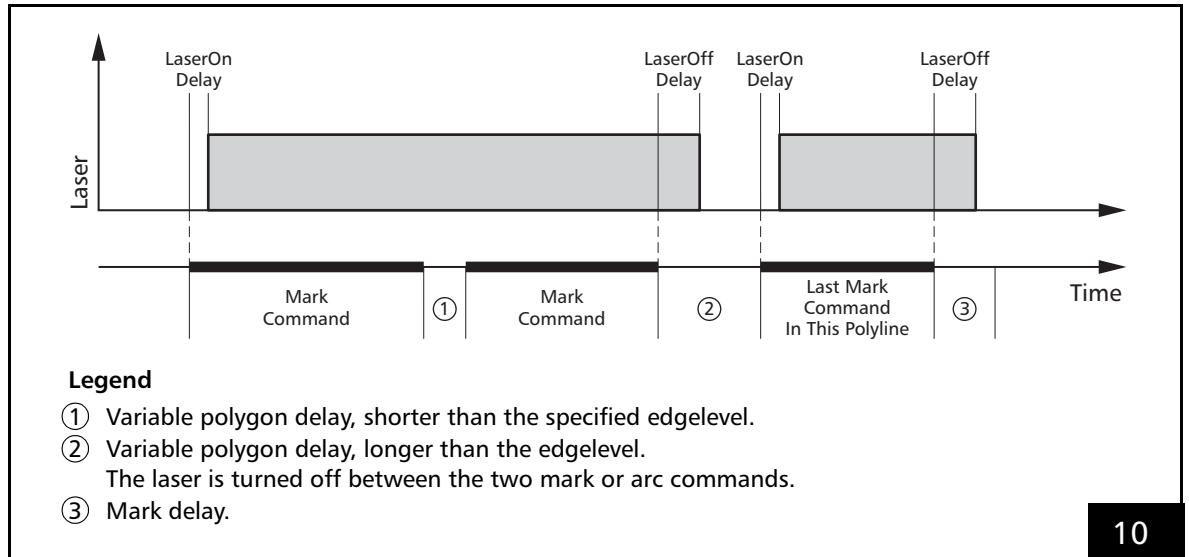


### Variable Polygon Delay

Top: Definition of the angle  $\phi$

Bottom: Variation of the polygon delay (default curve)





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  $scale(\phi)$  from an ASCII text file (with the filename extension \*.STB). Each STB file can contain one or more tables. See [load\\_varpolydelay \(page 92\)](#) for details.

The table contains up to 50 data points ( $\phi$  |  $scale(\phi)$ ) for various angles  $\phi$ . The RTC® SCANalone determines the scaling function  $scale(\phi)$  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 ( $\phi$  |  $scale(\phi)$ ) 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  $\phi$  (in degrees) and the scaling factor must be specified as floating point numbers. Use the period (.) as the decimal point.

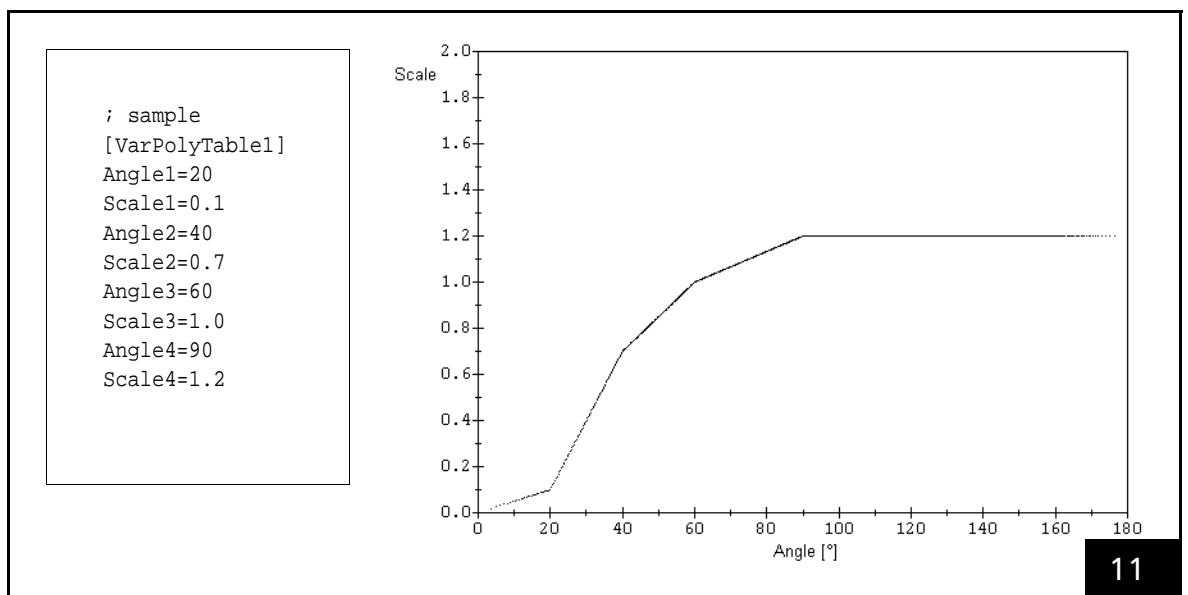
The allowed ranges are:

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

### 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  $\phi = 0^\circ$  means that two successive vectors are parallel and are marked in the same direction.  
If the table contains no explicit data for  $\phi = 0^\circ$ , the scaling factor  $scale(0^\circ) = 0$  is assumed.
- The angle  $\phi = 180^\circ$  means that two successive vectors are marked in the opposite direction.  
If the table contains no explicit data for  $\phi = 180^\circ$ , the scaling factor  $scale(180^\circ)$  is set to the largest scaling factor found in the table.

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



Sample table and resulting scaling function  $scale(\phi)$ .  
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 29](#) through [page 31](#) 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.

$$\text{LOffD} > \text{LOnD}$$

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

$$\text{markD} > \text{LOffD} - \text{LOnD}$$

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

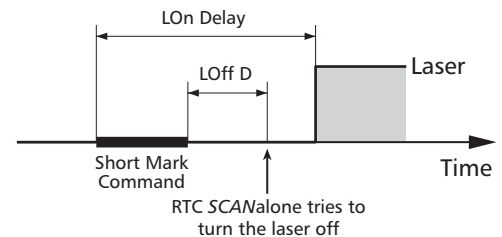
$$\text{edgelevel} > \text{LOffD} - \text{LOnD}$$

Please note that the laser delays must be specified in units of 1  $\mu\text{s}$ , whereas the scanner delays (jump delay, mark delay and polygon delay) must be specified in units of 10  $\mu\text{s}$ .

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® SCANalone will first try to turn the laser off and afterwards turn it on:

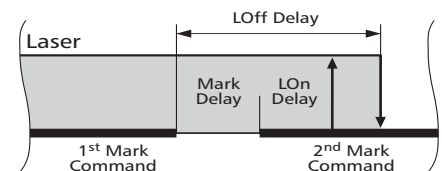


This can be avoided by always setting the LaserOff delay *longer* than the LaserOn delay.

$$\text{LOffD} > \text{LOnD} \quad [1]$$

- (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® SCANalone 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.

$$\text{markD} + \text{LOnD} > \text{LOffD} \quad [2]$$

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

$$\text{markD} > \text{LOffD} - \text{LOnD} \quad [3]$$

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 73) 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:

$$\text{polygonD} + \text{LOnD} > \text{LOffD} \quad [4]$$

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  $(\text{LOffD} - \text{LOnD} - \text{polygonD})$  previously to starting the second list.

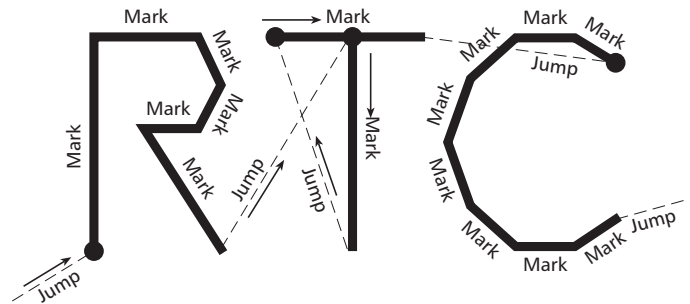
## Optimizing The Delays

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.

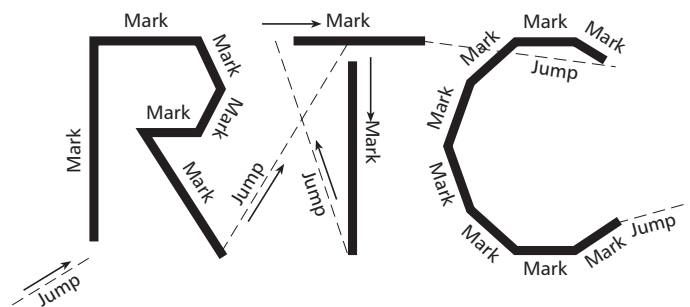
Burn-in effects at the start points of the respective vectors result.



### LaserOn Delay too long

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

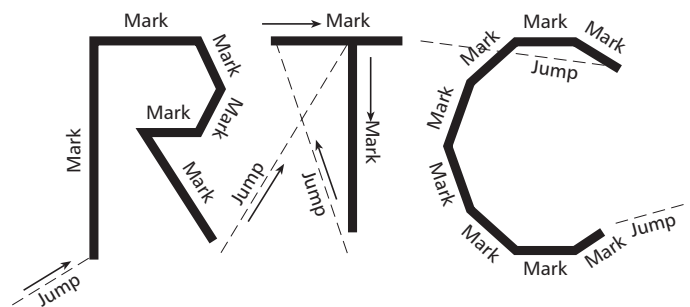
The first part of the vector is not marked.



### 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.

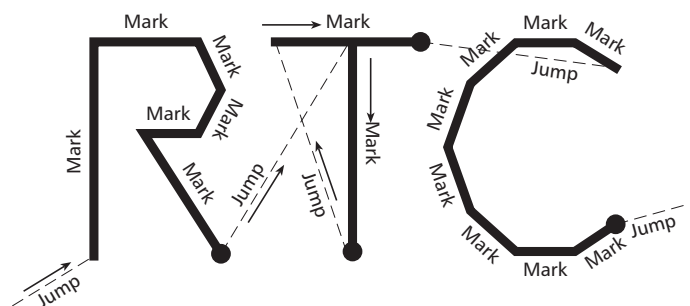
The respective vectors are not marked completely.



### 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.

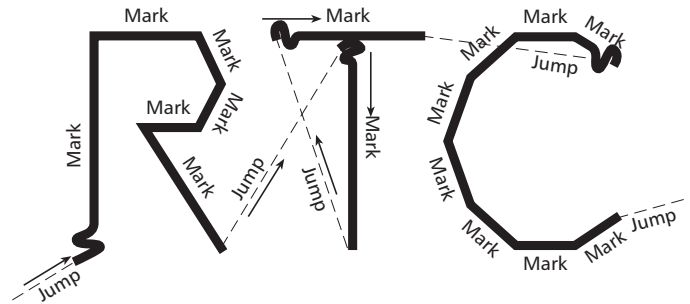
The results are burn-in effects at the end points of the respective vectors.



### 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.



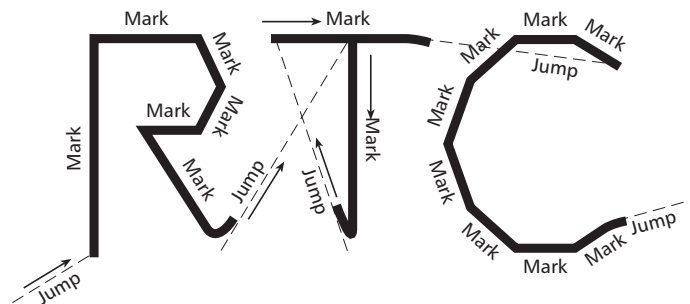
### Jump Delay too long

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.



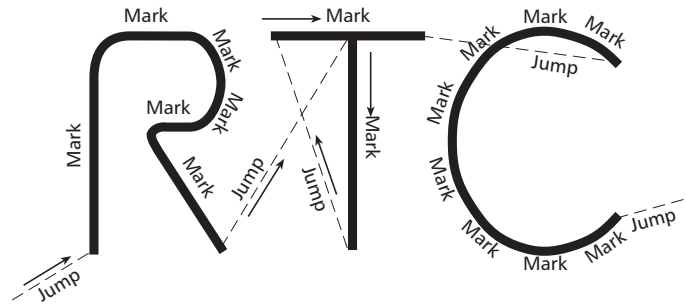
### Mark Delay too long

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.

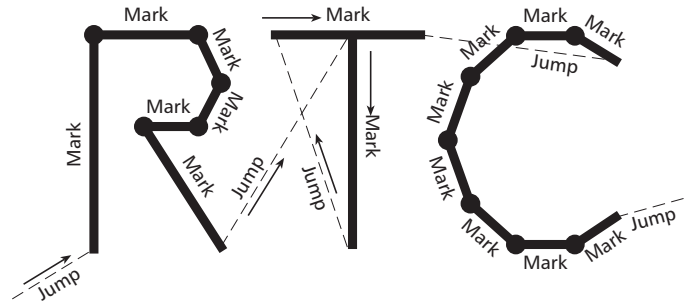
The corners of the polyline are rounded off.



## 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.

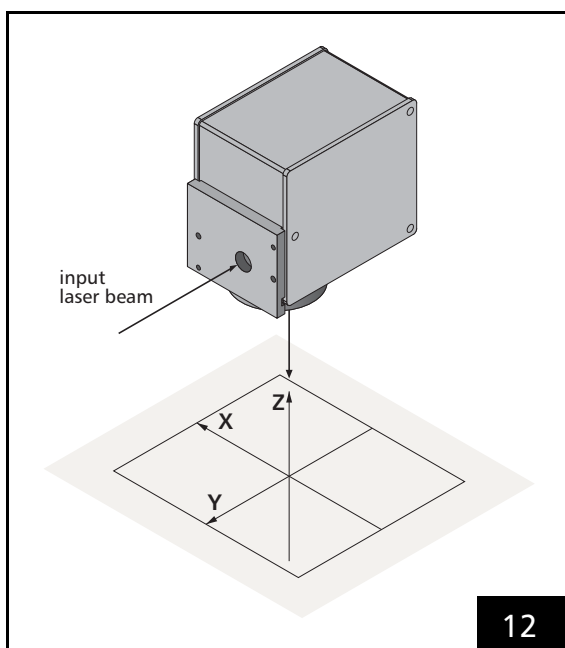
Since the laser is not turned off between these vectors, burn-in effects occur.



In the variable polygon delay mode, a maximum length ("edgelevel") can be defined. See [page 24](#) for details.

## 4.4 Image Field Size

Figure 12 shows the reference system for the image field which is used by the RTC® SCANalone. 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® SCANalone 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*<sup>(1)</sup> 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<sup>(1)</sup>].

(1) The expression "bits" is here synonymous with "digital control value" (see footnote on page 17).

(2) The calibration factor is the same for the X and Y 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 containing the correction file.<sup>(2)</sup>

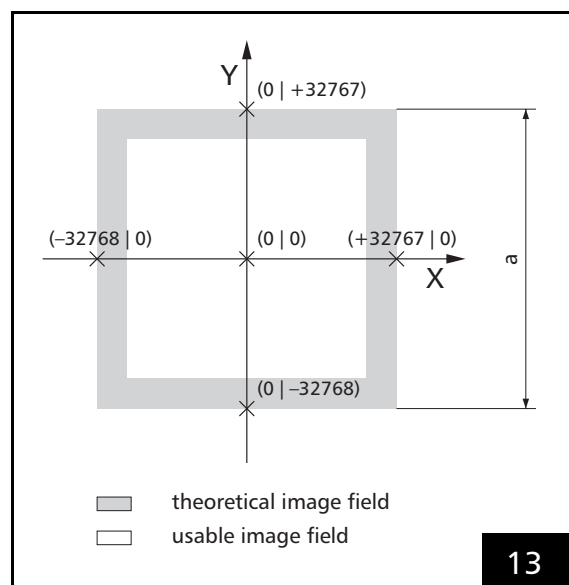
Given the calibration factor *K*, the side length *a* of the usable image field can be calculated:  $a = 2^{16} / K$

### 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



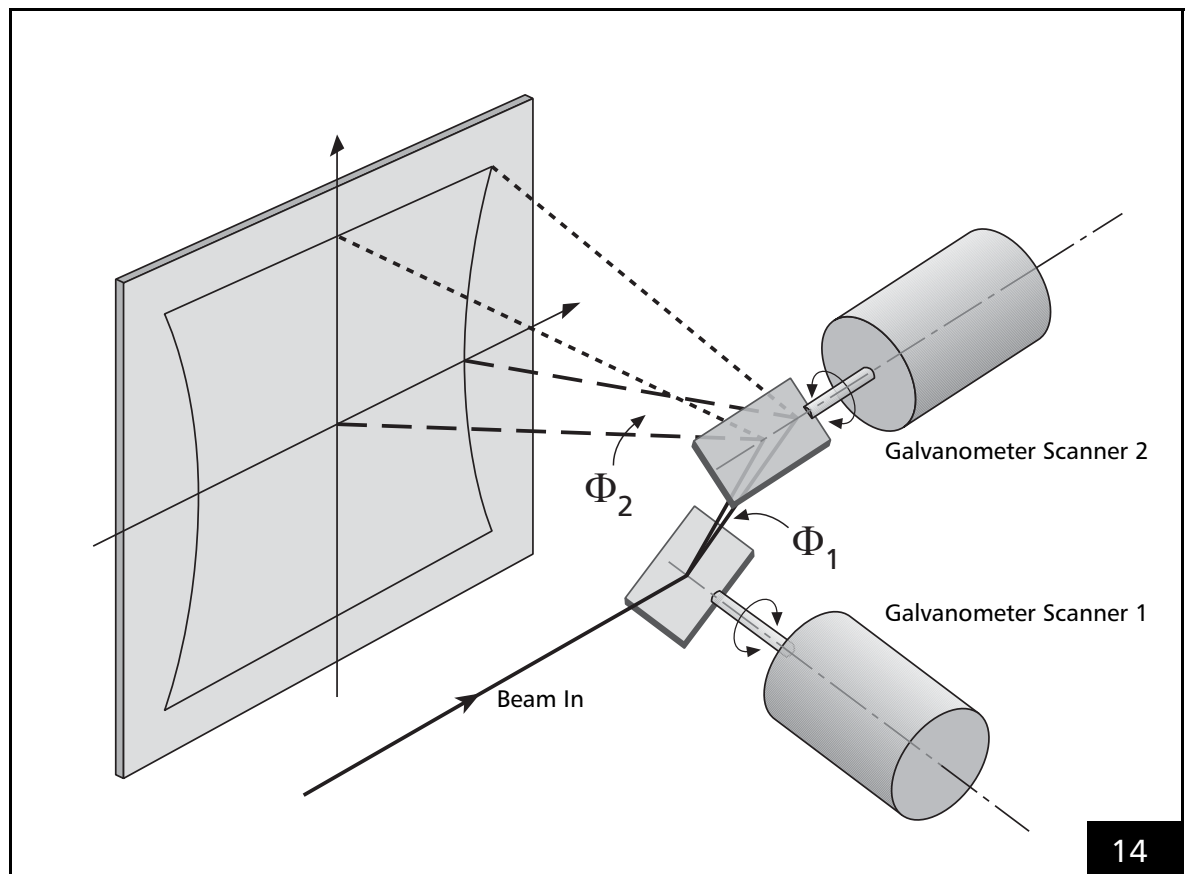
## 4.5 Image Field Correction

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 33](#).

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 34](#), center. This distortion is added to the pillow-shaped distortion described in section [\(1\) on page 33](#) (see [figure 15 on page 34](#), left), resulting in a so-called "barrel-pillow-shaped" distortion of the image field – see [figure 15 on page 34](#), right.

## Field Correction Algorithm

The RTC® SCANAlone 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® SCANAlone 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 17](#).)

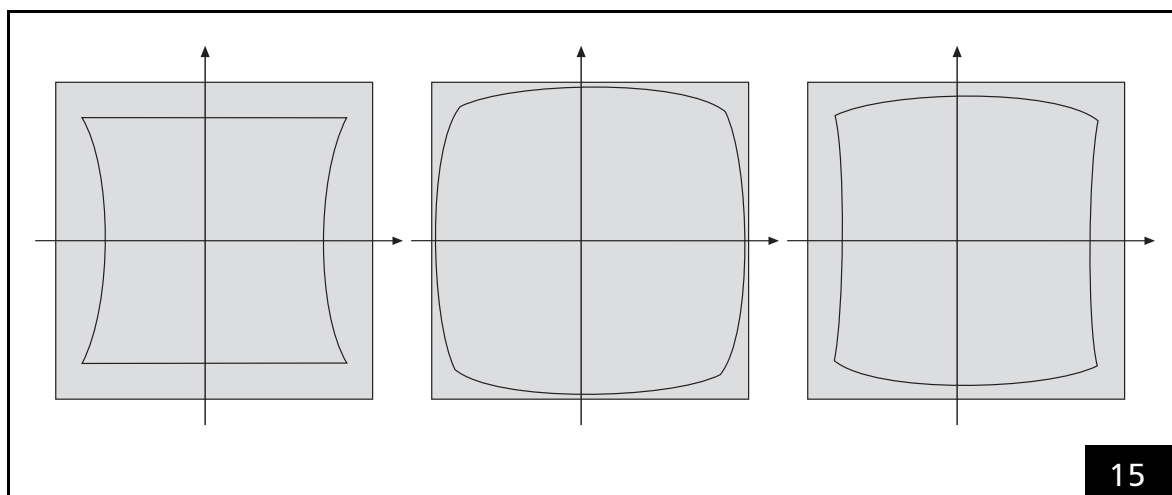
By default, SCANLAB creates an individual correction table for every delivered system. It is stored in a file named \*.CTB in the RTC® SCANAlone software package. Also see the command [load\\_correction\\_file](#), [page 90](#).

### Note

RTC® SCANAlone boards featuring the optional functionality of controlling the third axis of a 3-axis system (3D system) need a 3D correction file. SCANLAB names these files D3\_\*\*\*.CTB. All other RTC® SCANAlone board versions need 2D correction files (named D2\_\*\*\*.CTB).

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® SCANAlone 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).



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® SCANAlone 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 107), the user can choose one of five different laser control modes:

- CO<sub>2</sub> Mode (conceived of for controlling CO<sub>2</sub> lasers)
- YAG Modes 1, 2 and 3 (three variants conceived of for controlling Nd:YAG and related lasers)
- Laser Mode 4 generates continuously-running control signals.

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 62, 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 54.

## CO<sub>2</sub> Mode

The command `set_laser_mode(0)` selects the CO<sub>2</sub> laser mode. In this mode, the following laser signals are available (see figure 16 on page 35):

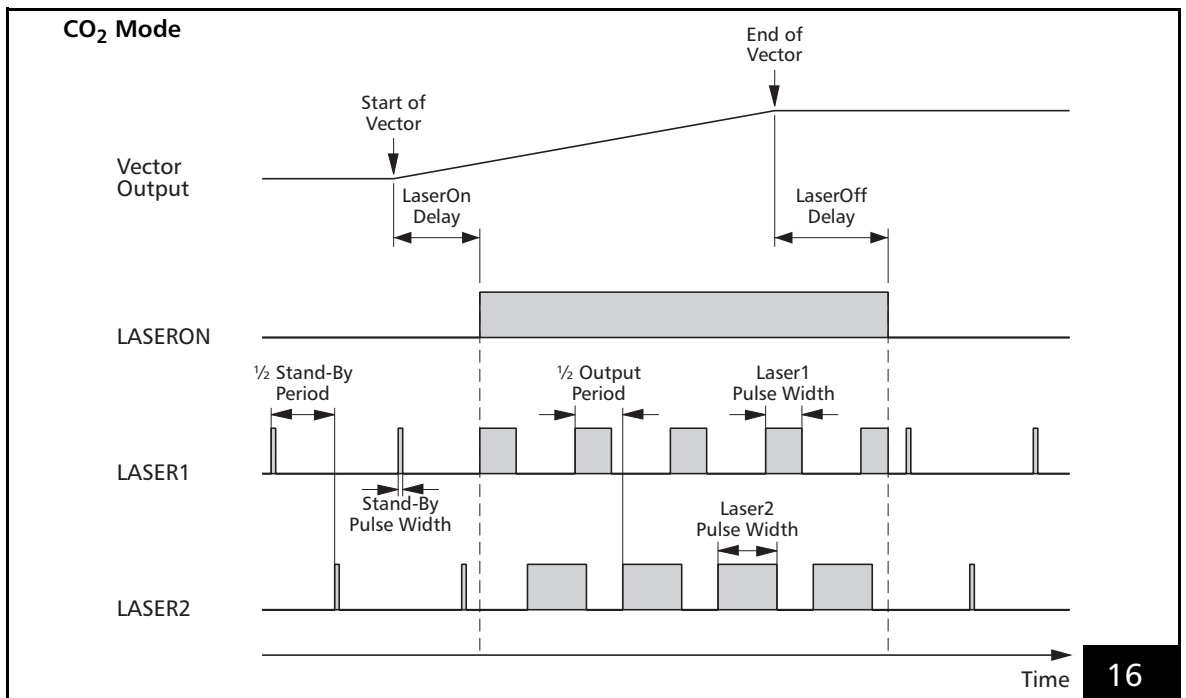
- a LASERON signal
- two alternating modulation signals with variable pulse width and frequency (LASER1 and LASER2)

The signals are available via the 9-pin D-SUB laser connector – see page 54. 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 108). Please note that *half* of the output period must be specified, i.e. the shift between the two laser signals.



Laser control timing diagram (CO<sub>2</sub> Mode)

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 108). 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).

### 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® SCANAlone 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 54. 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 108). 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 108). 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® SCANAlone 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 105).

## 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 39](#)).

- 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 25](#) and [figure 26 on page 54](#). 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 27](#) and [figure 28 on page 55](#)). The commands for setting the 8-bit port are **write\_8bit\_port** and **write\_8bit\_port\_list**.

Please refer to "Laser Extension Connector", page 55, 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.

In the softstart mode, various control values (`level(0) ... level(number)`, `number ≤ 1000`) for as many as the first 1001 pulses (at the beginning of a marking process) can be defined. Either pulsewidth values for the laser signal at the LASER1 output (see [figure 26, page 54](#)) or analog voltage levels for variable laser power can be defined. Analog voltage softstart values can be output either at the ANALOG OUT1 or ANALOG OUT2 output ports (see [figure 26, page 54](#)).

Initialization of the softstart mode is performed via the command **set\_softstart\_mode** (see page 116). Here, the type of softstart value is defined. If analog voltage values are to be output, then the analog output port can also be defined. Afterwards the individual softstart control values can be defined via **set\_softstart\_level** (see page 115).

After the laser control signals are switched on, the defined softstart values (max. 1001) will be output at the selected output port simultaneously with the laser pulses of the LASER1 signal – always with the leading edge of the laser pulse. The **set\_softstart\_mode** command allows specification of a time period (`restartdelay`) for which the laser must have been switched off before softstart will be activated at the next switch-on.

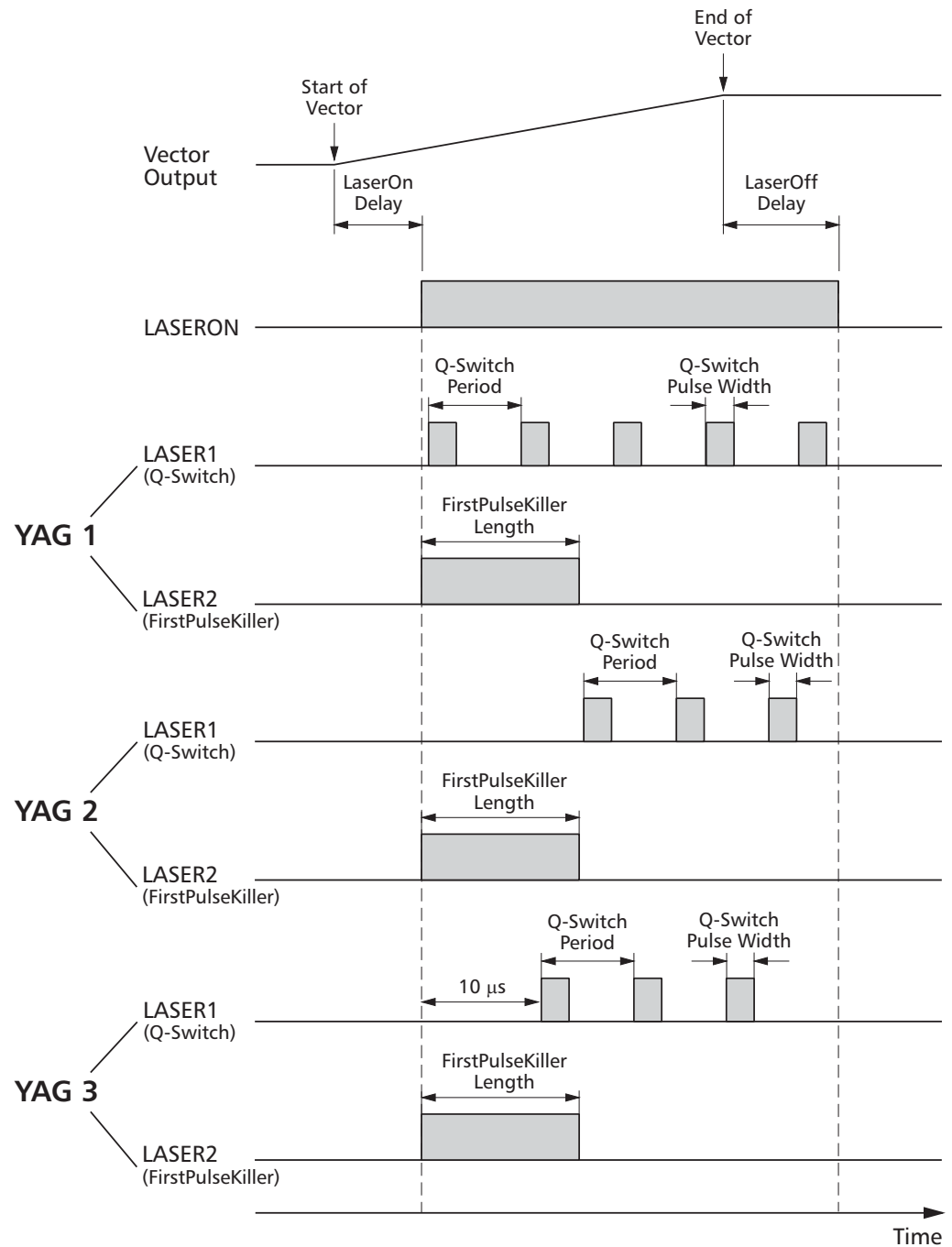
## Notes

- With a large enough value for the `restartdelay` time, one can avoid the softstart mode being also activated after very short jump commands.
- As soon as the LASERONsignal (the laser) remains switched off longer than `restartdelay`, the value `level(0)` is assigned to the output port, provided the softstart mode has not been meanwhile deactivated.
- When the LASERONsignal is switched on, then the values `level(0) ... level(number)` will be automatically assigned to the output port simultaneously with the first laser pulses. The pulse width or power of the first laser pulse is deter-

mined by `level(1)` if the values are assigned with the *leading* edge of the laser pulse selected, or by `level(0)` if the *trailing* edge of the laser pulse is selected.

- If softstart mode is inactive, then values for the analog outputs can be set via `write_da_x` or `write_da_x_list`.
- The softstart mode is also available in laser mode 4 (see [page 40](#)). In this mode, the LASER1 output only outputs standby pulses. Here, too, the defined softstart values are output simultaneously with the pulses of an internally-generated (but not output) LASER1 signal. The standby pulses are not synchronized with the internal LASER1 pulses. If the period durations of the standby and LASER1 pulses are set (via `set_laser_timing` and `set_standby`) to be equal, then the analog voltage softstart values will be output in parallel with the standby pulses, though they might have a (random) constant phase shift with respect to the standby pulses.

## YAG Modes 1-3



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 40](#)):

- a LASERON signal
- two alternating modulation signals with variable pulse width and frequency (LASER1 and LASER2). Unlike CO<sub>2</sub> Mode, both modulation signals are output independently of the LASERON signal (i.e. also as standby signals when the LASERON signal is inactive).

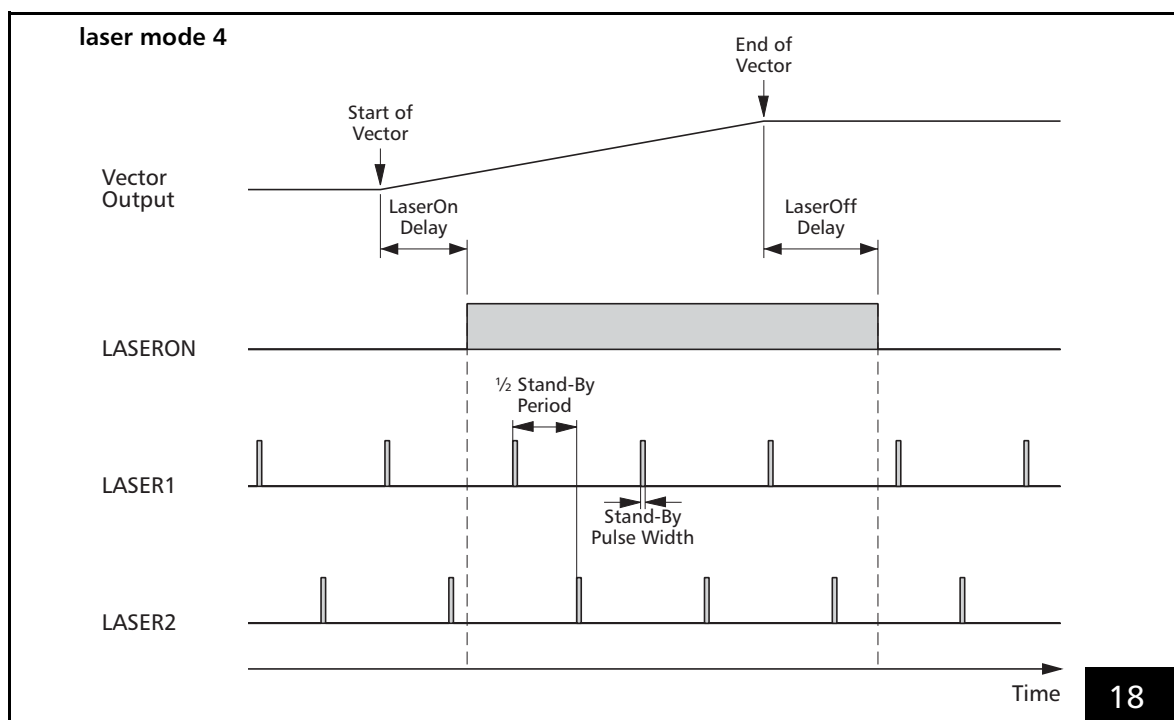
The signals are available via the 9-pin D-SUB laser connector – see [page 54](#). The LASER1 and LASER2 signals are additionally available at the LASER EXTENSION connector.

The signals LASER1 and LASER2 have a constant relative phase shift of 180° (half an output period).

The pulse width and the output frequency of the pulses are set with the command `set_standby` (see [page 117](#)) or `set_standby_list` (see [page 117](#)). They are equal for both laser signals. In the default setting the pulse width is set to zero.

Please note that *half* of the *output period* must be specified, i.e. the shift between the two laser signals.

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



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 (in PC operation only):

- XY2-100-compliant (or XY2-100-O-compliant) status signals returned by the scan system (these signals can also be read via the **get\_head\_status** command)
- 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® SCANalone 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 (in PC operation only) 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® SCANalone. 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**.

## 5 Advanced Programming

### 5.1 Coordinate Transformations

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

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

The coefficients  $m_{11} \dots m_{22}$  of the (2 x 2) transformation matrix are set with the command **set\_matrix** (see page 109). The offset ( $x_o | y_o$ ) in the X and Y directions is defined with **set\_offset** (see page 111).

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 110) and **set\_offset\_list** (page 111) for details.

The coordinate transformation defined with these commands is applied to the original coordinates of all specified vectors (i.e. before splitting into micro-vectors 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**<sup>(1)</sup> in the same way.

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

#### Examples:

1. Rotation by the angle  $\alpha$  :

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

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

(1) Also see the command **load\_correction\_file** (page 90).

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  $\alpha$  (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**.

## 5.2 Wobble Function

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

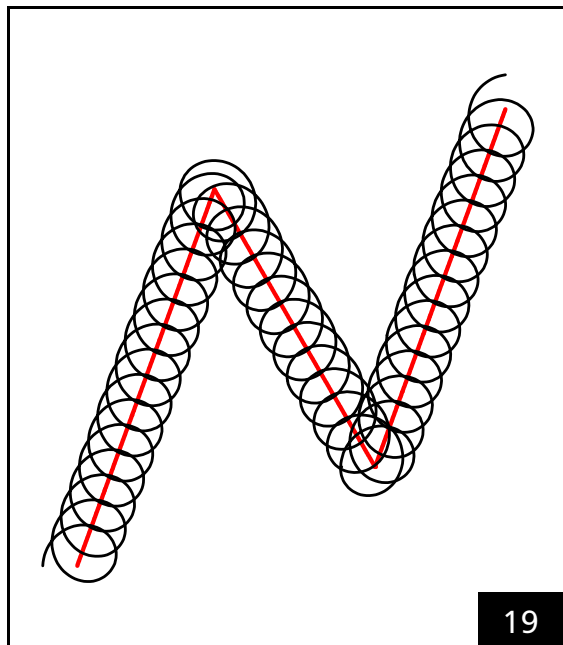
Basically, a circular movement is added to the regular, linear movement of the output position, resulting in a spiral movement of the laser focus in the image field.

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

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

Calling the command **set\_wobble** sets the wobble phase to a defined starting value (which is independent of the direction of the marking vector).

Please refer to the command **set\_wobble**, page 119 for further details.



Principle of the wobble function

## 5.3 Using Two Different Correction Files

### Double Scan Head Configuration

The RTC® SCANalone (2D version) can store two different correction files at the same time. For two scan heads controlled from a single RTC® SCANalone 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 56](#) for details about the two scan head control ports of the RTC® SCANalone.

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 90](#)) 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)`.

### 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 3D version of the RTC® SCANalone can store only *one* correction file.

### 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® SCANalone returns to the default setting after every reset in which an MMC is not installed. If a different setting is to be used, the command **select\_cor\_table** must be called again after the reset.
- Only one scan head can be controlled via the optional optical data interface.

## 5.4 Structured Programming

The command set of the RTC® SCANalone 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® SCANalone list memory as a single list buffer with a total capacity of 1 000 000 entries. Each command in the list buffer has a unique *address* in the range [0 ... 999 999].

Together with the control commands **set\_input\_pointer** and **execute\_at\_pointer** (that can be only used in PC operation), this allows structured list programming and execution.

### Input Pointer

To write a list of commands to a particular location in the RTC® SCANalone list memory, the command **set\_input\_pointer** (see page 105) is used (instead of the commands **set\_start\_list\_1 / \_2**). The command places the input pointer at any address in the range [0 ... 999 999]. 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 79).

### List Jumps

The command **set\_list\_jump** (see page 108) defines a jump to the specified address. During execution, the RTC® SCANalone 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 "Digital 16-bit Input and Output", page 58).

### Output Pointer

To start execution at a particular address, the command **execute\_at\_pointer** (see page 76) can be used (only in PC operation). The RTC® SCANalone 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® SCANalone 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 16) is used, the commands **set\_extstartpos** (see page 104) and **set\_extstartpos\_list** (see page 104) allow 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(1);
...
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(1);
...
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
...
```

## 5.5 Scanning Raster Images (Bitmaps)

The vector commands described in [chapter 4.2](#) are intended for scanning vector based images. However, the RTC® SCANalone 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 113](#)). 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](#).)

### 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® SCANalone list buffer (see [page 135](#)). 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  $\mu$ s ... 2  $\mu$ s 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:

$$T \geq \text{LaserOn Delay} + \text{maximum pulse width} + 10 \mu\text{s}$$

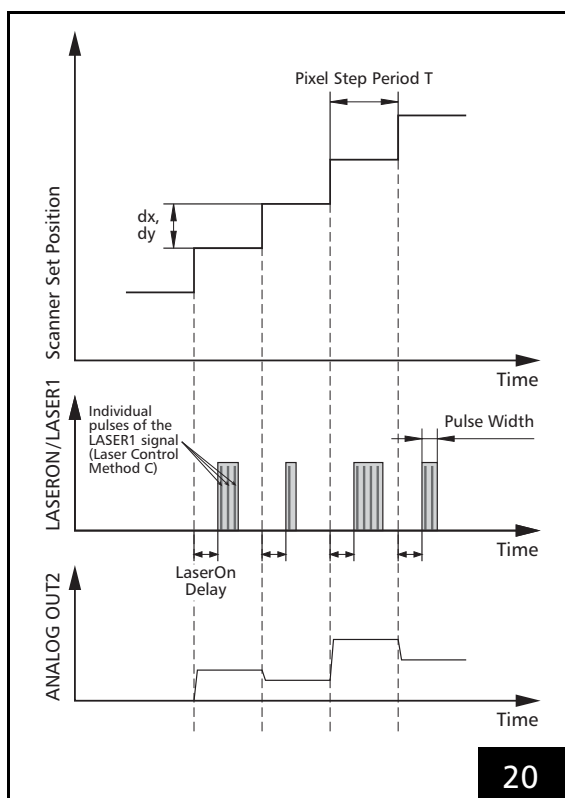
### Note

The 9-pin D-SUB laser connector of the RTC® SCANalone 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 62.

If you need the LASERON signal and the ANALOG OUT2 signal, then you have the possibility (with an RTC® SCANalone 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).



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



## Scanner Control

The RTC® SCANalone 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:  $T \approx 1 \text{ ms} \dots 10 \text{ ms}$ .
- ▶ To make sure that each pixel is marked at its exact position, the LaserOn delay must be longer than the required settling time.

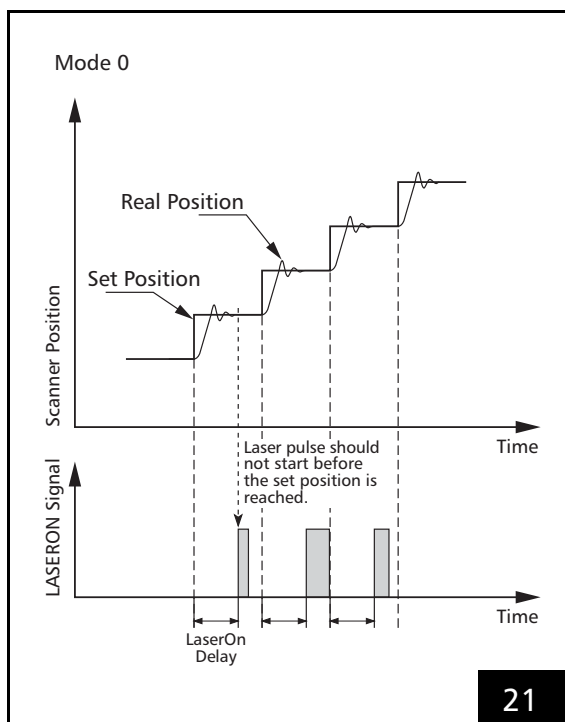
In Mode 0, each pixel will be marked at its exact position. The position does not depend on the dynamic performance of the scan head. However, the overall marking time will be quite long.

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

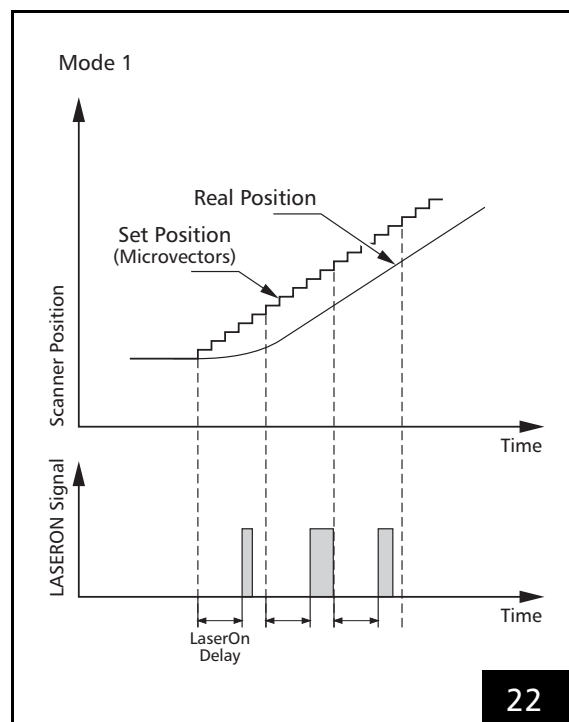
### Mode 1

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  $50 \mu\text{s}$ .
- ▶ 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  $T = 50 \mu\text{s}$  (= 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 108).

**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  $1/8 \mu\text{s}$ . (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 than 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<sup>®</sup> SCANalone 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"](#), page 62.

#### (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](#)).

## 5.6 Timed Jump And Mark Commands

The normal vector commands (jump commands and mark commands) are processed by the RTC® SCANAlone 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 122](#))

- **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 ( $\approx 0.6$  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 17](#).) 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.

### 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 20](#).)
- The timed jump and mark commands can be used for 2D vectors only.

## 5.7 Marking the Date, Time and Serial Numbers

Many applications need to mark the current time and date or product serial numbers. For this purpose, the RTC® SCANAlone features a battery-powered clock/calender and a set of supporting commands.

### Character Set Definition

Before times and dates and serial numbers can be marked, you must define a character set.

To specify how the scan system should mark the numeric characters 0...9, you need to load a separate command list per numeric character into the RTC® SCANAlone's list buffer. If month and day-of-the-week representations also contain non-numeric characters, then corresponding command lists for the months ("January" ... "December" or "Jan." or "/1/" etc.) and days of the week ("Sunday" ... "Saturday" or "Sun." ... "Sat." etc.) need to be created. The command lists also specify the text direction (writing along the x or y axes).

Character sets can be defined separately for time/date marking and for serial number marking.

The vector sequence for characters or character series can only be defined in command lists via **relative** vector commands (**jump\_rel**, **mark\_rel**, **arc\_rel**). The end position of a character or character series must be selected as the start position of any following character. Each command list must be terminated with **list\_return**.

The memory starting address of each digit, month and day command list must be queried via **get\_input\_pointer** (see [page 79](#)) and then entered into a table via **set\_char\_table** (see [page 101](#)). These table entries will later be accessed by the time and date output commands.

### Marking the Date and Time

Before times and dates can be marked, you must calibrate the clock and calender, and query the current time and date:

- The RTC® SCANAlone's calender and 24-hour clock are calibrated (only in PC operation) by comparison with the PC's time via **time\_update** (see [page 121](#)).
- The current time and date are queried and stored via **time\_fix** (see [page 121](#)).

Now the time (hours, minutes, seconds) can be marked via **mark\_time** (see page 96) and the date (year, month, day, day-of-the-week) via **mark\_date** (see page 94). Both commands use the time and date supplied via **time\_fix** and start outputting the command lists whose start addresses were stored via **set\_char\_table**.

The RTC® SCANalone is capable of marking Gregorian dates as well as Julian days.

The current serial number can't be directly queried. But it can be calculated via the starting serial number and the number of external list starts. In PC operation, the number of external list starts can be obtained with the control command **get\_counts**.



### Caution!

- For time and date functions, a 1.5 V AAA alkaline battery is required. These functions cannot be used if a battery is not installed (see page 59).

## Marking Serial Numbers

Serial numbers containing up to 10 digits can be marked via the command **mark\_serial** (see page 95). The command initiates the output of command lists for the numerals 0...9, whose starting addresses were stored via **set\_char\_table**.

The starting serial number must have been previously specified via the control command **set\_serial**. After the RTC® SCANalone Board boots with an installed MMC, this serial number will be read from the MMC and applied.



### Caution!

- If the MMC's content is not updated following a power interruption, then already-marked serial numbers will be marked again.

The serial number increments by one with each call of the **mark\_serial** command (after the actual serial number was marked).

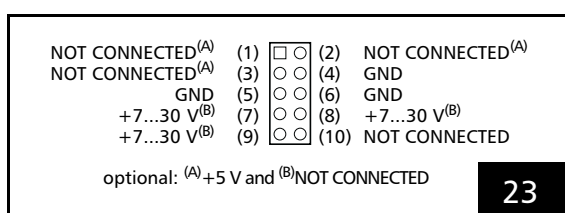
The increment size between two markable serial numbers can be indirectly controlled by issuing "markless" **mark\_serial** commands (`digits = 0`), each time incrementing the serial number by one.

The control command **set\_max\_counts** allows specification of the maximum number of external list starts and thus the maximum number of external markings.

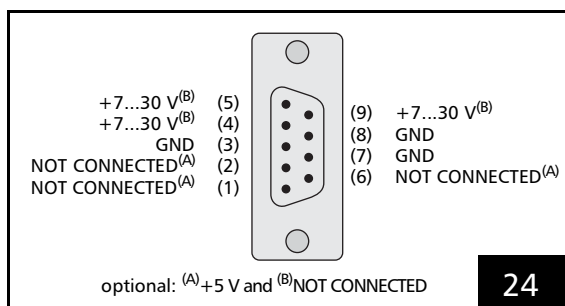
## 6 Electrical Connections

### 6.1 Power Supply

The power source should be connected to the 10-pin “POWER” header on the base board or the 9-pin D-SUB plug on the extension board. The pin-outs are shown in [figure 23](#) and [figure 24](#).



Pin-out of the 10-pin “POWER” header on the base board



Pin-out of the 9-pin “POWER” D-SUB plug on the extension board

### Power Requirements

The RTC<sup>®</sup> SCANalone requires a power source of +7...+30 V DC. Current consumption is up to 10 watts maximum, depending on which peripherals are used.

When the RTC<sup>®</sup> SCANalone is connected to a PC via the USB interface, then the grounds of the external power supply and the PC must be identical or (as in the case of a laptop) galvanically decoupled.

The voltage potential between the RTC<sup>®</sup> SCANalone and the scan system must not exceed  $\pm 5$  V.

### Reset

Test points TP2 and GND (see [figure 1 on page 8](#)) are available for resetting the RTC<sup>®</sup> SCANalone. Connecting TP2 (RESET) with GND results in a reset of the RTC<sup>®</sup> SCANalone’s DSP.

### 6.2 USB Connection to a PC

The RTC<sup>®</sup> SCANalone is equipped with a USB 1.1 port. Either horizontal connector USB1 or vertical connector USB2 is populated, as depicted in [figure 1 on page 8](#).

**Note for users of RTC<sup>®</sup> PC interface boards:**

Data transfer via a USB interface is slower than via the PCI bus.

### 6.3 MMC Memory Card

The RTC<sup>®</sup> SCANalone is equipped with a slot that accepts an MMC memory card (SD memory cards will not be accepted).

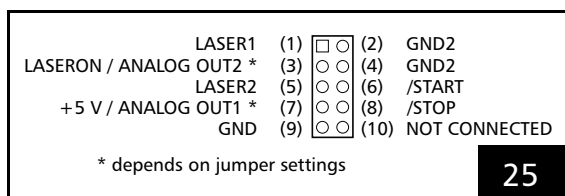
List commands and correction files can be stored on the card. The user can rapidly change between applications by swapping MMC cards.

On start-up or after a power interruption, the correction files, initialization and list commands are reloaded from the MMC card.

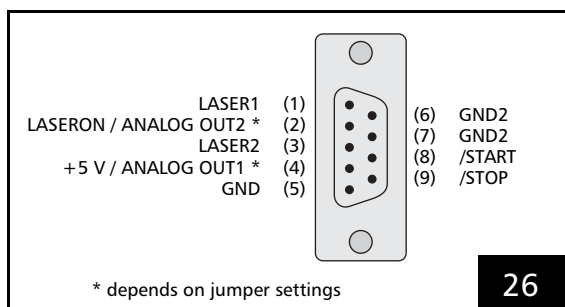
The RTC<sup>®</sup> SCANalone board’s internal memory can accommodate up to one million list commands.

## 6.4 Laser Connector

Figure 25 and figure 26 show the pin-outs of the 10-pin "LASER" header on the base board and the 9-pin "LASER" D-SUB socket on the extension board. The signals at pins (3) and (7) on the header and pins (2) and (4) on the D-SUB header are selected via jumpers (see page 62).



Pin-out of the 10-pin "LASER" header on the base board



Pin-out of the 9-pin "LASER" D-SUB socket on the base board

## Laser Signals

The output signals LASER1 and LASER2 at pins (1) and (3) depend on the selected laser control mode – please refer to the laser timing diagrams on page 35 (CO<sub>2</sub> Mode) and page 39 (YAG Modes):

	CO <sub>2</sub> Mode	YAG Modes 1 - 3
<b>LASER1</b>	Modulation pulse 1	Q-Switch signal
<b>LASER2</b>	Modulation pulse 2	FirstPulseKiller

All laser outputs (LASERON, LASER1 and LASER2) are digital TTL level signals (alternatively active-high or active-low, see page 62) and are referenced to GND2. The maximum current load is 10 mA.

If the RTC® SCANalone is supplied with optional optoelectronic couplers, GND and GND2 are galvanically decoupled from each other. Otherwise GND and GND2 are identical. See chapter 7 "Options", page 60.

## 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 16.

## Analog Output Ports

The RTC® SCANalone provides two general purpose 10-bit analog output ports, ANALOG OUT1 and ANALOG OUT2. They can be loaded with the commands `write_da_x` or `write_da_list` (see page 133).

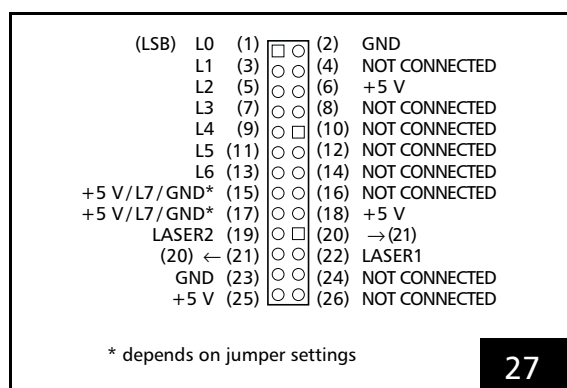
The output voltage range of the ANALOG OUT1 port can be set to either 0 V ... 2.56 V or 0 V ... 10 V (see page 62). The range of the ANALOG OUT2 port is fixed at 0 V ... 10 V.

The maximum current load of both signals is 5 mA.

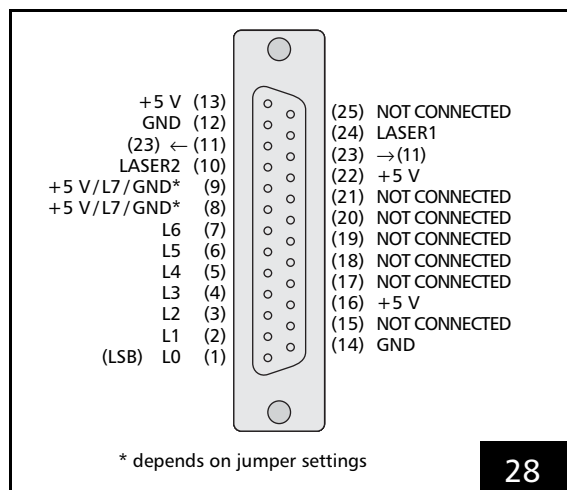
If jumper X8 has been configured to replace the ANALOG OUT 1 signal with +5 V, then the maximum current load at pin (7) of the "LASER" header or pin (4) of the "LASER" D-SUB socket is 100 mA.

## 6.5 Laser Extension Connector

The 26-pin "LASER EXTENSION" header on the base board or the 25-pin "LASER EXTENSION" D-SUB plug on the extension board provide a buffered 8-bit digital output port (L0 to L7), as shown in [figure 27](#) and [figure 28](#). Pins (15) and (17) of the header or pins (8) and (9) on the D-SUB plug must be configured via jumpers. For details please refer to "LASER EXTENSION/Digital Output Port" on [page 63](#).



Pin-out of the "LASER EXTENSION" header on the base board



Pin-out of the "LASER EXTENSION" D-SUB plug on the extension board

## 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 124](#)) load the digital output port with an 8-bit value.

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" header (see [page 63](#)).

## Laser Signals

LASER1 and LASER2 share the following characteristics with the laser signals of the "LASER" connector: they have the same specifications and their output signals depend on the selected laser control mode (see "Laser Signals", [page 54](#)).



## 6.6 Primary Scan Head Connector

The 26-pin "1.SCAN HEAD" header on the base board and the 25-pin "1.SCAN HEAD" D-SUB socket on the extension board serve to connect the scan head via a data cable (described later). The pin-out is shown in [figure 29](#) and [figure 30](#).

The 25-pin D-SUB socket is compatible with most scan heads which use the XY2-100.

CLOCK– (1)	□	(2)	CLOCK+
/SYNC– (3)	○	(4)	/SYNC+
CHAN1– (5)	○	(6)	CHAN1+
CHAN2– (7)	○	(8)	CHAN2+
(optional) CHAN3– (9)	□	(10)	CHAN3+ (optional)
STATUS– (11)	○	(12)	STATUS+
DO NOT CONNECT (13)	○	(14)	DO NOT CONNECT
DO NOT CONNECT (15)	○	(16)	DO NOT CONNECT
DO NOT CONNECT (17)	○	(18)	DO NOT CONNECT
DO NOT CONNECT (19)	□	(20)	GND
GND (21)	○	(22)	GND
DO NOT CONNECT (23)	○	(24)	DO NOT CONNECT
DO NOT CONNECT (25)	○	(26)	DO NOT CONNECT

29

Pin-out of the 26-pin "1.SCAN HEAD" header and the "2.SCAN HEAD" header on the base board

CLOCK– (1)	●	(14)	CLOCK+
/SYNC– (2)	●	(15)	/SYNC+
CHAN1– (3)	●	(16)	CHAN1+
CHAN2– (4)	●	(17)	CHAN2+
(optional) CHAN3– (5)	●	(18)	CHAN3+ (optional)
STATUS– (6)	●	(19)	STATUS+
DO NOT CONNECT (7)	●	(20)	DO NOT CONNECT
DO NOT CONNECT (8)	●	(21)	DO NOT CONNECT
DO NOT CONNECT (9)	●	(22)	DO NOT CONNECT
DO NOT CONNECT (10)	●	(23)	GND
GND (11)	●	(24)	GND
DO NOT CONNECT (12)	●	(25)	DO NOT CONNECT
DO NOT CONNECT (13)	●		

30

Pin-out of the 25-pin "1.SCAN HEAD" D-SUB socket and the "2.SCAN HEAD" D-SUB socket on the extension board

## 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 (also see [chapter 7 "Options", page 60](#)).

For additional information, please contact SCANLAB.

## Status Signals

The STATUS channel receives XY2-100 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.

## Data Cable

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

- ▶ The data cable must have 25-pin (male) D-SUB connectors with identical pin-outs at both ends. The cable consists of six twisted cable pairs for controlling the scan head via 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® SCANalone. See the command [set\\_piso\\_control \(page 111\)](#) 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 31 shows the data cable layout and pin-outs.

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 31.

## 6.7 Secondary Scan Head Connector (Optional)

The 26-pin "2. SCANHEAD" header on the base board and the 25-pin "2.SCAN HEAD" D-SUB socket on the extension board are designed for connecting a second scan head in a double scan head configuration. The pin-outs are identical with their "1.SCAN HEAD" counterparts (see figure 29 and figure 30).

The signals on this connector are optional and must be enabled by SCANLAB (see chapter 7 "Options", page 60). Please contact SCANLAB for further information.

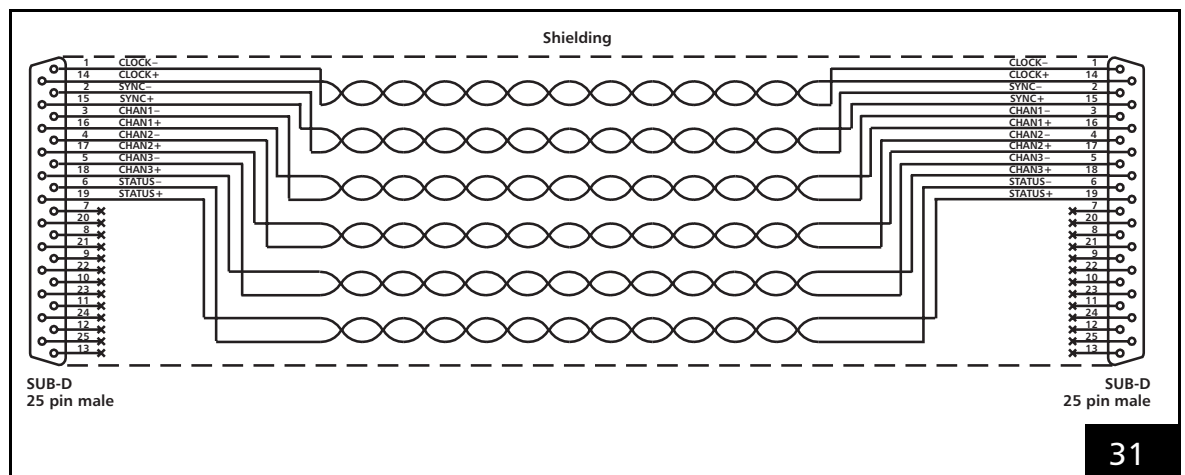
## 6.8 Optical Data Interface (Optional)

The RTC® SCANalone can be optionally equipped with the ability to optically transmit data to and from the scan head. Optical data transmission provides galvanic decoupling and makes the control process immune to electronic interference (ESD and EMV).

Transmission of data is via a duplex plastic fiber (POF = polymer optical fiber) with a wavelength of 650 nm. Compliant with the XY2-100-O protocol, the optical signals are transferred bidirectionally every 10 µs over 3 data channels.

Both ST sockets (OOUT and OIN) on the base board must be connected to the appropriate ST plugs at the scan system's (optionally installed) optical interface via a duplex plastic fiber (cross-wired, *not* 1:1). The fiber length should not exceed 30 m.

The optical data interface is designed to control only one scan head.



Data cable layout and pin assignments

## 6.9 Digital 16-bit Input and Output

A protected 16-bit digital TTL input and a buffered 16-bit digital TTL output are provided on the base board's 40-pin "EXTENSION 1" header or the extension board's 37-pin "EXTENSION 1" D-SUB socket. The pin-outs are shown in [figure 32](#) and [figure 33](#).

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
LED1 (33)	○	(34)	LED2
RESERVED (35)	○	(36)	BUSYOUT
+5 V (37)	○	(38)	+5 V
GND (39)	○	(40)	GND

32

Pin-out of the 40-pin "EXTENSION 1" header on the base board

DIGITAL OUT0 (1)	●	(20)	DIGITAL IN0
DIGITAL OUT1 (2)	●	(21)	DIGITAL IN1
DIGITAL OUT2 (3)	●	(22)	DIGITAL IN2
DIGITAL OUT3 (4)	●	(23)	DIGITAL IN3
DIGITAL OUT4 (5)	●	(24)	DIGITAL IN4
DIGITAL OUT5 (6)	●	(25)	DIGITAL IN5
DIGITAL OUT6 (7)	●	(26)	DIGITAL IN6
DIGITAL OUT7 (8)	●	(27)	DIGITAL IN7
DIGITAL OUT8 (9)	●	(28)	DIGITAL IN8
DIGITAL OUT9 (10)	●	(29)	DIGITAL IN9
DIGITAL OUT10 (11)	●	(30)	DIGITAL IN10
DIGITAL OUT11 (12)	●	(31)	DIGITAL IN11
DIGITAL OUT12 (13)	●	(32)	DIGITAL IN12
DIGITAL OUT13 (14)	●	(33)	DIGITAL IN13
DIGITAL OUT14 (15)	●	(34)	DIGITAL IN14
DIGITAL OUT15 (16)	●	(35)	DIGITAL IN15
BUSYOUT (17)	●	(36)	+5 V
+5 V (18)	●	(37)	GND
GND (19)	●		

33

Pin-out of the 37-pin "EXTENSION 1" D-SUB socket on the extension board

The BUSY status signal is available on pin (36) of the header and pin (17) of the D-SUB socket. The BUSY status signal is HIGH when an application is currently executing.

Pins (33) and (34) of the header are for connection to a dual-color LED for displaying the BUSY and PWROK status. If an application is executing (BUSY status signal = HIGH) the LED will emit one of its colors. If no application is executing and the RTC® SCANalone is operational (PWROK status signal = HIGH) then the LED will emit its other color.

The commands `write_io_port_list` and `write_io_port` can be used to load the digital output port with a value. Individual bits of the port can be set or cleared (depending on the current value of the 16-bit input) via the commands `set_io_cond_list` and `clear_io_cond_list`. 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 16-bit input (see "[Structured Programming](#)", [page 45](#)).

## 6.10 10-bit Analog Inputs

The RTC® SCANalone is equipped with two 10-bit analog inputs via the single-row 4-pin header J6. The pin-out is shown in [figure 34](#). The allowable voltage range for these inputs is 0...10 V DC.

The unsigned 10-bit output values of both DACs can be read via `read_ad_x` (see [page 97](#))

□	(1)	+10 V
○	(2)	ANALOG IN1
○	(3)	ANALOG IN2
○	(4)	GND

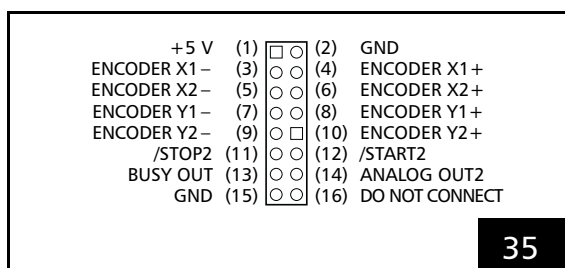
34

Pin-out of 4-pin header J6 on the base board

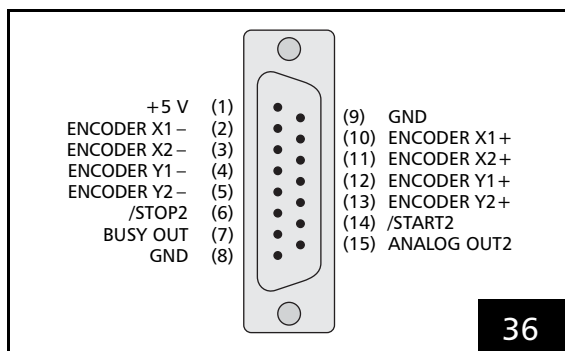
## 6.11 Processing-on-the-fly (Optional)

The 16-pin “MARKING ON THE FLY” header on the base board and the 15-pin “MARKING ON THE FLY” D-SUB socket on the extension board are designed for Processing-on-the-fly applications. The pin-outs are shown in [figure 35](#) and [figure 36](#).

The input and output control signals for these connectors are optional and must be activated by SCANLAB. Additional information can be found in the ["Processing-On-The-Fly Software"](#) manual or by contacting SCANLAB.



Pin-out of the optional 16-pin “MARKING ON THE FLY” header on the base board



Pin-out of the optional 15-pin “MARKING ON THE FLY”-D-SUB socket on the extension board

## 6.12 Status LEDs

A dual-color LED is mounted on both the base board (D4) and the extension board (D1) for displaying the state of the BUSY and PWROK status signals. A current-limiting resistor is integrated at the base board.

If an application is currently executing (BUSY status signal = HIGH) then the LED emits red. If no application is currently executing, but the RTC® SCANalone is operational (PWROK status signal = HIGH) then the LED emits green.

Upon start-up, the RTC® SCANalone becomes operational after receiving power and loading into list memory all data stored on an inserted MMC card.

## 6.13 Battery-Powered Clock and Calender

The RTC® SCANalone contains a clock/calender, useful for processing tasks such as those that require laser-marking the current time or date.

By default, the clock and calender are powered by a battery. Optionally they can alternatively be powered by the board’s external power source. Time and date functionality is unavailable if the clock/calender is not furnished with power.

For battery operation, a 1.5 V AAA alkaline battery must be installed in battery retainer BT1. Battery life is primarily a function of the battery’s self-discharge rate.

## 7 Options

The following options are available for the RTC® SCANalone.

### Note

- The command **get\_rtc\_version** (see page 79) provides information about the options currently installed on your RTC® SCANalone board.
- **Second Scan Head Connector**  
This option allows controlling two scan heads simultaneously, e.g. in a double scan head configuration.  
Also see page 44.
- **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 varioSCAN 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® SCANalone to an optical interface (optionally) integrated in the scan system (see the section "Optical Data Interface (Optional)" on page 57).  
The optical data interface is designed for controlling only *one* scan head.
- **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 54).

## 8 Installation And Start-Up

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

- (1) configure the RTC® SCANalone jumpers
- (2) connect the RTC® SCANalone board to the PC and a power supply
- (3) install the software driver via the Windows installation program
- (4) connect the RTC® SCANalone to the scan system and laser .

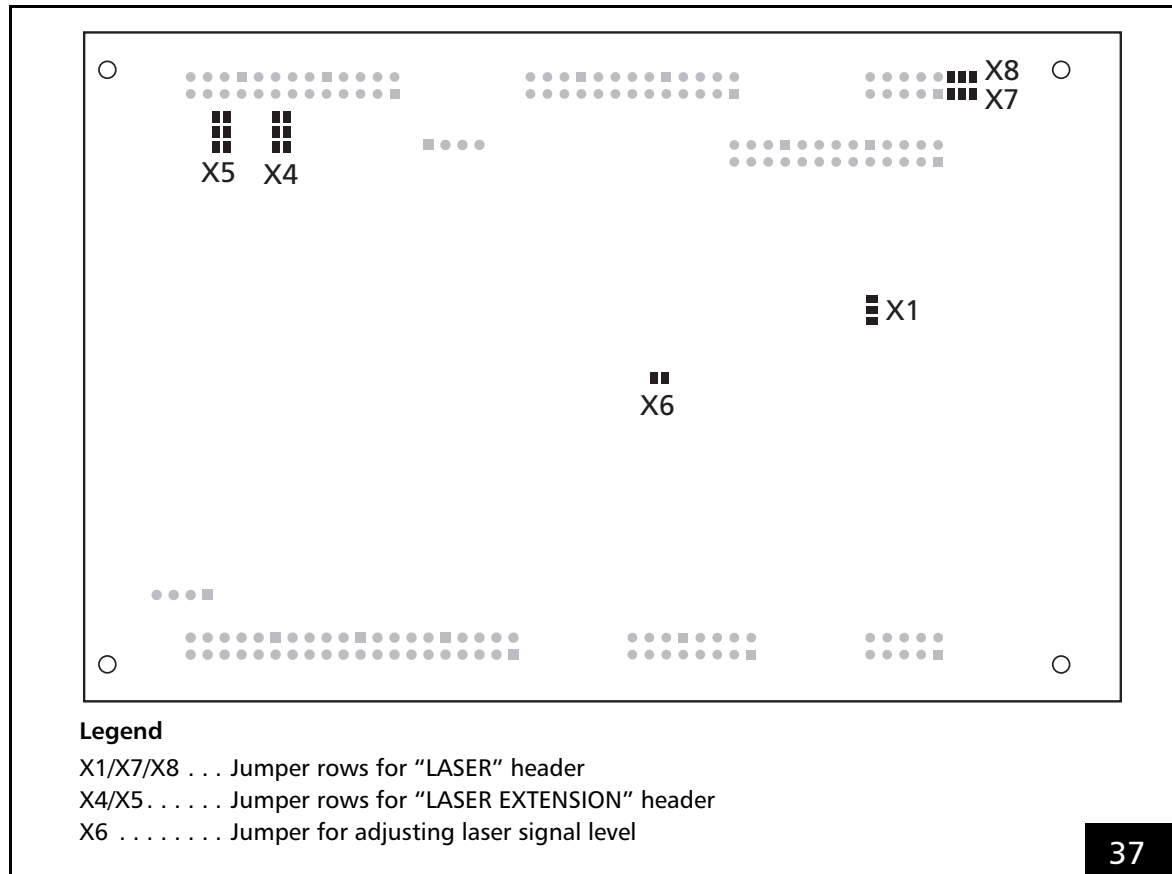
### 8.1 Jumper Settings Overview

The factory settings for the RTC® SCANalone 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 8.5 "Installing the Software Driver"](#), page 64.

The following RTC® SCANalone base board jumpers can be set by the user (see figure 37):

- Jumper for the laser signals (X6)  
(active-high or active-low)
- Jumpers for the 9-pin D-SUB laser socket or 10-pin "LASER" header (X1, X7, X8)
- Jumpers for the 8-bit digital output port on the "LASER EXTENSION" header or optional 25-pin "LASER EXTENSION" D-SUB plug (X4, X5)



Jumper positions on the RTC® SCANalone (back of base 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! Before changing jumper settings, disconnect the power source. Follow all ESD precautions.**

### TTL Laser Signal Level

The RTC® SCANalone board permanently generates the following laser signals (also see [page 35](#) through [page 40](#)):

- LASERON
- LASER1 / Q-Switch
- LASER2 / FirstPulseKiller

All laser signals can be set to either *active-low* or *active-high* logic via jumper **X6**. Thereby, you must consider whether or not the laser signals are optically decoupled from the PC ground.



Jumper X6		optically decoupled laser signals	optically non-decoupled laser signals
open		active-low	active-high
closed		active-high	active-low



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.

### Laser / Analog Output Ports

The signals at pins (3) and (7) of the 10-pin "LASER" header or pins (2) and (4) of the 9-pin "LASER" D-SUB socket are selected via jumpers **X7**, **X8** and **X1**. The following two tables summarize the jumper settings.

Jumper	Function	Position 1-2 	Position 2-3 
<b>X8</b>	selects the signal at pin (7) or pin (4)	+5 V	ANALOG OUT1
<b>X7</b>	selects the signal at pin (3) or pin (2)	LASERON	ANALOG OUT2

Jumper	Function	Position 1-2 	Position 2-3 
<b>X1</b>	sets the output range of the ANALOG OUT1 signal	0 ... 2.56 V	0 ... 10 V

#### 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 instead use pin (3) or pin (2) for the signal ANALOG OUT2. 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 header or D-SUB socket (Processing-on-the-fly option required).

#### Jumpers X8 and X1

- If you want to use the signal ANALOG OUT1 (e.g. for lamp current control), set the jumper **X8** to position 2-3. Then specify the output voltage range with the Jumper **X1** (see table).
- Alternatively, the output signal at pin (7) or pin (4) can be set permanently to +5 V (jumper **X8** in position 1-2). In this case, jumper **X1** can be ignored.

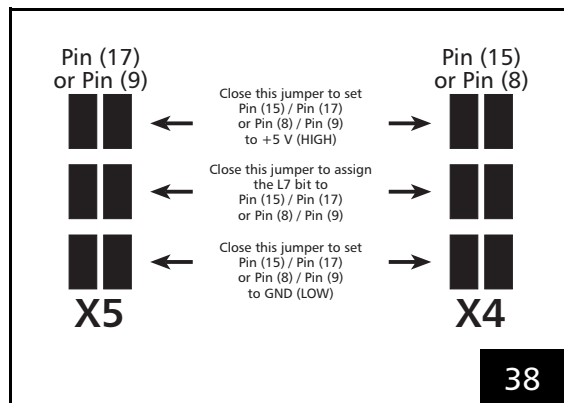
For a description of the analog output signals please refer to [chapter "Analog Output Ports", page 54](#).

## "LASER EXTENSION/Digital Output Port

The 8-bit digital output port is available via the LASER EXTENSION header or D-SUB socket – see [figure 27 on page 55](#), odd numbered pins (1) to (17) or [figure 28 on page 55](#), pins (1) to (9).

The port can be loaded with an 8-bit value by the commands **write\_8bit\_port** and **write\_8bit\_port\_list** (see [page 124](#)).

Pins (15) and (17) or pins (8) and (9) of the output port have to be configured via jumpers **X4** and **X5**. The most significant bit (L7) of the 8-bit output value can be assigned to pins (15) or (17) or pins (8) or (9). Alternatively, each of the two pins can be set permanently to +5 V (HIGH) or to GND (LOW level).



## Examples

- If the L7 bit is assigned to pin (15) or pin (8), the full 8-bit output value is available on the output port (odd numbered pins (1) to (15) of the LASER EXTENSION connector or pins (1) to (8) of the "LASER EXTENSION) D-SUB socket).
- Setting pin (15) or pin (8) permanently to HIGH results in an offset of 128 for output values and an output range of 128 ... 255.
- Setting pin (15) or pin (8) to LOW restricts the output range to 0 ... 127.
- The L7 bit can be used for other purposes by assigning it to pin (17) or pin (9).



## Caution!

- Make sure that not more than **one** of the three jumpers at **X4** is closed. Also make sure that not more than **one** of the three jumpers at **X5** is closed. Closing more than one jumper at positions **X4** or **X5** will damage the electronics on the board.

### 8.3 Installing the RTC® SCANalone Board

Install the RTC® SCANalone Board so that it is securely mounted. To avoid short circuits, ensure that adequate space separates the electronic components from mounting surfaces. Board attachment should only be via the four M2.5 screw holes, as depicted in [figure 1, page 8](#).

Additionally, SCANLAB recommends installation in an electromagnetically shielded housing that provides electronic shielding and protects against moisture and contamination.

When selecting the installation location, ensure that the RTC® SCANalone is not exposed to excessive environmental temperatures that may result in improper program execution or damage to the electronic components.

### 8.4 Connecting to the PC and Power Supply



#### Caution!

- The RTC® SCANalone should be stored in the supplied anti-static bag in an electrically neutral environment.
- Installation must be carried out at a workstation that is compliant with ESD guidelines.
- Do not touch the RTC® SCANalone's electrical contacts.
- Protect the board from moisture, dust, corrosive vapors and mechanical stress.

- ▶ Remove the RTC® SCANalone from the antistatic bag. Do not touch the contacts of the board.
- ▶ Connect the RTC® SCANalone's USB jack to a free USB socket on the PC (see ["USB Connection to a PC", page 53](#)).
- ▶ Connect a power supply to the RTC® SCANalone's "POWER" connector (see ["Power Supply", page 53](#)).

### 8.5 Installing the Software Driver

Microsoft's WINDOWS Vista / XP / 2000 and WINDOWS ME / 98 operating systems recognize the RTC® SCANalone as a new device. First-time connection of the RTC® SCANalone to the PC's USB port automatically causes the Windows installation program to start.

- ▶ Insert the CD containing the RTC® SCANalone software package in the CD drive.
- ▶ Follow the instructions given by the Windows installation program.

**Note:** The installation program does *not* install the RTC® SCANalone software (application support DLL and correction files). To install this software, follow the instructions in [chapter 9, page 65](#).

### 8.6 Connecting to the Scan System and Laser

- ▶ Connect the scan system to the RTC® SCANalone-board via a data cable or an optical fiber. Information on connection pin-outs and data cable layout can be found in ["Primary Scan Head Connector", page 56](#) and ["Optical Data Interface \(Optional\)", page 57](#).
- ▶ Connect the RTC® SCANalone board to the laser via an appropriate interface (see ["Laser Connector", page 54](#)).



#### Caution!

- Always turn on the RTC® SCANalone 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.



## 9 Software

### 9.1 Installing the Software

The RTC® SCANalone DLL supports RTC® SCANalone applications running under Microsoft's WINDOWS Vista / XP / 2000 and WINDOWS ME / 98 operating systems. This DLL provides access to all functions required to operate the RTC® SCANalone.

Install the RTC® SCANalone software as follows:

- ▶ Copy the driver DLL (RTC\_SCANalone.DLL) from the RTC® SCANalone software package to the directory in which the application software will be started, or to the WINDOWS directory.
- ▶ Copy the correction file(s) (\*.CTB) to the harddisk of your PC. SCANLAB recommends storing these files in the directory in which the application software will be started.  
(The RTC® SCANalone DSP file(s) (\*.HEX) don't have to be copied, because SCANLAB has pre-installed them on the RTC® SCANalone. The RTC® SCANalone's signal processor starts up automatically.)
- ▶ 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.
- Application software should only be started from the PC when the RTC® SCANalone is operational and connected to the PC.

### 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® SCANalone 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® SCANalone – also see [chapter 7 "Options", page 60](#).  
Make sure to use only the commands supported by your version of the RTC® SCANalone. Please refer to the descriptions of the individual commands in [chapter 10](#).  
The command [get\\_rtc\\_version](#) ([page 79](#)) provides information about the options currently installed on your RTC® SCANalone board.

#### Pascal

Use the file SCANalone4Import.pas as a unit and call the RTC® SCANalone commands you need, like

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

for performing a jump to location 1000, 2500.

#### Basic

Include file SCANalone4Import.bas into your project and call the RTC® SCANalone 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 `SCANalone4impl.h` and link with the (Visual C++) import library `SCANalone4DLL.LIB` for building the executable.

Call the RTC® *SCANalone* 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 `SCANalone4expl.h`. Before calling any RTC® *SCANalone* function, initialize the DLL by calling the function `SCANalone4open` (which is defined in the file `SCANalone4expl.cpp`). When you are finished using the RTC® *SCANalone*, close the DLL by calling the function `SCANalone4close` (also defined in the file `SCANalone4expl.cpp`).

For building the executable, link with the file `SCANalone4EXPL.OBJ`, which you can generate from the source code `SCANalone4expl.cpp`.

Call the RTC® *SCANalone* 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
<b>Necessary Files</b>	<code>SCANalone4impl.h</code> , <code>SCANalone4DLL.LIB</code>	<code>SCANalone4expl.h</code> , <code>SCANalone4expl.cpp</code>
<b>Advantage</b>	Easiest linking method	Eliminates the need to link the application with an import library
<b>Negative Aspect</b>	Need to link the application with a compiler-specific import library	Need to initialize ( <code>SCANalone4open</code> ) and close ( <code>SCANalone4close</code> ) the DLL

## Automatic Assignment of Access Rights

On opening the DLL by an application (e.g. via calling the function `SCANalone4open` for explicit linking in C), the application is automatically assigned access rights to the RTC® *SCANalone* board (as long as access rights are not already assigned to another application). The board *cannot* be simultaneously used by multiple applications. To prohibit simultaneous access of multiple applications to the RTC® *SCANalone* board, only the firstly-opened DLL gets the access rights to the board. If an application opens an RTC® *SCANalone* DLL, if the access rights to the board have already been assigned to another application, then **load\_correction\_file** returns error code 8. Subsequent changes to access rights are not possible.

Multi-threading is supported. However, please note that multiple threads of one application can use the board, but can not send commands to it at the same time (the DLL automatically serializes the command calls).

### 9.3 Initializing the RTC® SCANalone

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

- (1) Download the correction file(s) to the RTC® SCANalone  
(command **load\_correction\_file**, page 90).  
See chapter 5.3, page 44, for information about using two different correction files.
- (2) Set the laser mode  
(command **set\_laser\_mode**, page 107).
- (3) Assign the correction file(s) to the scan head control port(s) if necessary.  
The default assignment (after each reset of the RTC® SCANalone) is:
  - The first scan head control port uses correction table #1.
  - No correction file is assigned to the second scan head control port.
 Use the command **select\_cor\_table** (see page 100) to change to a different assignment.
- (4) Define the scanner delay mode  
(variable polygon delay or constant polygon delay; command **set\_delay\_mode**, page 103).
- (5) Load a table for the variable polygon delay if necessary (command **load\_varpolydelay**, page 92).
- (6) Set the FirstPulseKiller length (YAG only)  
(command **set\_firstpulse\_killer**, page 104).
- (7) Set values for the stand-by pulses (usually CO<sub>2</sub> only)  
(command **set\_standby**, page 117).
- (8) Load the list(s).
- (9) Enable the external start input if necessary  
(command **set\_control\_mode**, page 102).

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

### 9.4 Demo Program

The RTC® SCANalone software package contains a program code sample (file `DEMO.CPP`). The purpose of this sample is to demonstrate usage of the diverse control and list commands with an example for marking current time and date and marking serial numbers.

The sample is written in the C language. It shows the necessary calling sequences of the RTC® SCANalone commands, which you can easily translate into your preferred programming language.

### 9.5 Functionality Test



#### Caution!

- Always turn on the RTC® SCANalone 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® SCANalone.

- ▶ Copy the HPGL converter and the supplied demo file into the same directory as the driver DLL 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`.
- ▶ The RTC® SCANalone's current state can be stored onto the MMC via `File|Save`.

## 10 Commands And Functions

### 10.1 Overview

The tables on [page 69](#) and [page 70](#) show the complete RTC<sup>®</sup> SCANalone command set (control commands and list commands). The commands are listed according to their intended use. The page numbers refer to [chapter 10.3 "Command Set"](#), where the commands are ordered alphabetically and described in detail.

Control commands and some list commands can only be used in PC operation. However, control command sequences (in PC operation) can be used for standalone initialization of the RTC<sup>®</sup> SCANalone.

The RTC<sup>®</sup> SCANalone does not use multi-board commands (such as are used on RTC<sup>®</sup> PC interface boards)

#### Compatibility

The RTC<sup>®</sup> SCANalone provides additional commands for compatibility with instruction sets of the RTC<sup>®</sup>2, RTC<sup>®</sup>3, and RTC<sup>®</sup>4 PC interface boards. All supported and unsupported RTC<sup>®</sup> commands are described in [chapter 10.4 "Supported and Unsupported RTC<sup>®</sup> Commands"](#), [page 128](#).

## Control Commands

### Storage on the MMC

(1) <b>clear_list</b> .....	74
(2) <b>store_on_mmc</b> .....	121

### Initialization And Field Correction

(2) <b>load_correction_file</b> .....	90
(2) <b>select_cor_table</b> .....	100
(2) <b>load_varpolydelay</b> .....	92

### Laser Mode And Parameters

(2) <b>set_laser_mode</b> .....	107
(2) <b>set_firstpulse_killer</b> .....	104
(2) <b>set_softstart_level</b> .....	115
(2) <b>set_softstart_mode</b> .....	116
(2) <b>set_standby</b> .....	117

(Also see the correspond. list commands, [page 70.](#))

### Scanner Delay Mode

(2) <b>set_delay_mode</b> .....	103
---------------------------------	-----

### Coordinate Transformations

(2) <b>set_matrix</b> .....	109
(2) <b>set_offset</b> .....	111

### Status Monitoring and Diagnostics

(1) <b>get_head_status</b> .....	78
(1) <b>get_value</b> .....	82
(1) <b>get_waveform</b> .....	83
(1) <b>measurement_status</b> .....	96
(1) <b>usb_status</b> .....	124

### I/O Commands

(1) <b>get_io_status</b> .....	79
(1) <b>read_ad_x</b> .....	97
(1) <b>read_io_port</b> .....	97
(2) <b>write_8bit_port</b> .....	124
(2) <b>write_da_x</b> .....	125
(2) <b>write_io_port</b> .....	126

(Also see the correspond. list commands, [page 70.](#))

### External Control Inputs

(2) <b>set_control_mode</b> .....	102
(2) <b>select_list</b> .....	100
(2) <b>set_max_counts</b> .....	110
(1) <b>get_counts</b> .....	77
(1) <b>get_startstop_info</b> .....	80

(1) only usable in PC operation

(2) only usable in PC operation (where it can also be used for initializing the RTC® SCANAlone in standalone operation)

### List Handling And Status

(2) <b>set_extstartpos</b> .....	104
(1) <b>set_start_list</b> .....	117
(1) <b>execute_list</b> .....	76
(1) <b>stop_execution</b> .....	120
(1) <b>stop_list</b> .....	120
(1) <b>restart_list</b> .....	99
(1) <b>get_status</b> .....	81
(1) <b>read_status</b> .....	98

### Synchronization Of Processing

(1) <b>get_wait_status</b> .....	83
(1) <b>release_wait</b> .....	98

### Automatic List Handling

(1) <b>auto_change</b> .....	73
(1) <b>auto_change_pos</b> .....	73
(1) <b>start_loop</b> .....	120
(1) <b>quit_loop</b> .....	97

### Structured Programming

(1) <b>set_input_pointer</b> .....	105
(1) <b>get_input_pointer</b> .....	79
(1) <b>execute_at_pointer</b> .....	76

### Marking the Date, Time and Serial Numbers

(2) <b>set_char_table</b> .....	101
(2) <b>set_serial</b> .....	114
(1) <b>time_update</b> .....	121

### Direct Laser And Scan Head Control

(2) <b>disable_laser</b> .....	75
(2) <b>enable_laser</b> .....	75
(2) <b>laser_signal_on</b> .....	87
(2) <b>laser_signal_off</b> .....	86
(2) <b>goto_xy</b> .....	84
(1) <b>get_xy_pos</b> .....	83
(1) <b>z_out</b> .....	126

### Other Control Commands

(1) <b>get_dll_version</b> .....	77
(1) <b>get_hex_version</b> .....	78
(1) <b>get_rtc_version</b> .....	79
(1) <b>get_serial_number</b> .....	80
(2) <b>home_position</b> .....	84
(1) <b>get_time</b> .....	81
(2) <b>set_piso_control</b> .....	111

## List Commands

### Vector Commands

jump_abs .....	85
jump_rel .....	85
mark_abs .....	93
mark_rel .....	94
timed_jump_abs .....	122
timed_jump_rel .....	122
timed_mark_abs .....	123
timed_mark_rel .....	123

### Arc Commands

arc_abs .....	72
arc_rel .....	72

### Status Monitoring and Diagnostics

(1) set_trigger .....	118
-----------------------	-----

### External Control Inputs

set_control_mode_list .....	102
-----------------------------	-----

### List Handling

set_end_of_list .....	103
set_extstartpos_list .....	104

### Synchronization Of Processing

(1) set_wait .....	119
--------------------	-----

### Structured Programming

set_list_jump .....	108
list_call .....	87
list_return .....	89
list_nop .....	89

### Marking the Date, Time and Serial Numbers

mark_date .....	94
mark_serial .....	95
mark_time .....	96
time_fix .....	121

### Setting The Laser Parameters

set_laser_timing .....	108
set_firstpulse_killer_list .....	105
set_standby_list .....	117

### Setting The Scanner Parameters

set_laser_delays .....	107
set_scanner_delays .....	113
set_jump_speed .....	106
set_mark_speed .....	109

(1) only usable in PC operation

### Coordinate Transformations

set_matrix_list .....	110
set_offset_list .....	111

### Direct Laser And Scan Head Control

laser_on_list .....	86
laser_signal_on_list .....	87
laser_signal_off_list .....	86
z_out_list .....	127

### Scanning Raster Images (Bitmaps)

set_pixel_line .....	113
set_pixel .....	112

### I/O Commands

clear_io_cond_list .....	74
list_call_cond .....	88
list_jump_cond .....	88
set_io_cond_list .....	106
write_8bit_port_list .....	124
write_da_x_list .....	125
write_io_port_list .....	126

### Other List Commands

long_delay .....	93
(1) save_and_restart_timer .....	99
set_wobbel .....	119

## 10.2 Data Types

The following table defines the formats and ranges of the different data types used by the RTC® SCANalone 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	<b>arc_abs</b>
Function	marking along the specified circular arc starting at the current position
Use	for PC and standalone operation
Parameters	<div> <div>x,y</div> <div>absolute coordinates of the arc center in <i>bits</i> as signed 16-bit values</div> </div> <div> <div>angle</div> <div>arc angle in ° [-360.0° ... +360.0°] as 64-bit IEEE floating point value (positive angle values correspond to clockwise angles)</div> </div>
Integration	<div>Pascal:    arc_abs(x,y: smallint; angle: double);</div> <div>C:            void arc_abs(short x, short y, double angle);</div> <div>Basic:       arc_abs(ByteVal x%, ByteVal y%, ByteVal angle#)</div>
Comments	<ul style="list-style-type: none"> <li>• <b>The maximum allowed arc radius (in <i>bits</i>) is 32767.</b></li> <li>• The marking speed is set with the command <a href="#">set_mark_speed</a> (see page 109).</li> <li>• The laser is turned on at the beginning of the command after a LaserOn delay.</li> <li>• 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.</li> <li>• See <a href="#">chapter 4.2 "Scan Head And Laser Control", page 17</a>, for further details.</li> </ul>
References	<a href="#">set_mark_speed</a> , <a href="#">set_scanner_delays</a> , <a href="#">arc_rel</a>

List Command	<b>arc_rel</b>
Function	marking along the specified circular arc starting at the current position
Use	for PC and standalone operation
Parameters	<div> <div>dx,dy</div> <div>relative coordinates of the arc center in <i>bits</i> as signed 16-bit values</div> </div> <div> <div>angle</div> <div>arc angle in ° [-360.0° ... +360.0°] as 64-bit IEEE floating point value (positive angle values correspond to clockwise angles)</div> </div>
Integration	<div>Pascal:    arc_rel(dx,dy: smallint; angle: double);</div> <div>C:            void arc_rel(short dx, short dy, double angle);</div> <div>Basic:       arc_rel(ByteVal dx%, ByteVal dy%, ByteVal angle#)</div>
Comments	<ul style="list-style-type: none"> <li>• The arc center coordinates are relative to the current position.</li> <li>• <b>The maximum value for the <i>absolute</i> arc center coordinates is ±32767 bits.</b></li> <li>• The marking speed is set with the command <a href="#">set_mark_speed</a> (see page 109).</li> <li>• The laser is turned on at the beginning of the command after a LaserOn delay.</li> <li>• 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.</li> <li>• See <a href="#">chapter 4.2 "Scan Head And Laser Control", page 17</a>, for further details.</li> </ul>
References	<a href="#">set_mark_speed</a> , <a href="#">set_scanner_delays</a> , <a href="#">arc_abs</a>



Ctrl Command	<b>auto_change</b>
Function	activates an automatic list change
Use	only useful in PC operation
Integration	Pascal: <code>procedure auto_change;</code> C: <code>void auto_change(void);</code> Basic: <code>sub auto_change()</code>
Comments	<ul style="list-style-type: none"> <li>• The <b>auto_change</b> command should only be used when working with two lists (up to 500000 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.</li> <li>• With automatic list changing activated, the next list will start automatically after the current list is finished.</li> <li>• For each subsequent list, use a separate <b>auto_change</b> call.</li> <li>• The current status of both lists can be read with the commands <b>get_status</b> (page 81) or <b>read_status</b> (page 98).</li> <li>• See "Automatic List Handling", page 15.</li> </ul>
References	<b>get_status, read_status</b>

Ctrl Command	<b>auto_change_pos</b>
Function	activates an automatic list change
Use	only useful in PC operation
Parameter	start       Start position (address) of the next command sequence to be executed (signed 32-bit value) allowed range: [0 ... 999999]
Integration	Pascal: <code>procedure auto_change_pos(start: longint);</code> C: <code>void auto_change_pos(long start);</code> Basic: <code>sub auto_change_pos(ByVal start&amp;)</code>
Comments	<ul style="list-style-type: none"> <li>• Use the <b>auto_change_pos</b> command when the RTC® SCANalone's entire memory is treated like a single list buffer.</li> <li>• With automatic list changing activated, finishing of a list will automatically result in continuation of execution at the specified start position.</li> <li>• The current status of a list can be read with the commands <b>get_status</b> (page 81) or <b>read_status</b> (page 98).</li> <li>• See "Automatic List Handling", page 15.</li> </ul>
References	<b>get_status, read_status</b>

List Command	<b>clear_io_cond_list</b>
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 <i>input</i> port meets the following condition: $((IOvalue \text{ AND } mask\_1) = mask\_1) \text{ AND } (((\text{not } IOvalue) \text{ AND } mask\_0) = mask\_0)$ (i.e. the bits specified in mask_1 must be 1 and the bits specified in mask_0 must be 0).
Use	for PC and standalone operation
Parameters	mask_1,      unsigned 16-bit mask values mask_0, mask_clear
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	<ul style="list-style-type: none"> <li>The command affects only those bits of the output port that are set (=1) in the parameter mask_clear.</li> </ul>
Examples (Pascal)	<ul style="list-style-type: none"> <li>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 (DIGITAL_IN1...3) of the input port are LOW:  clear_io_cond_list(\$0001, \$000E, \$0010)</li> <li>always clear bit #15 of the output port (and leave the other bits unchanged):  clear_io_cond_list(0, 0, \$8000)</li> </ul>
References	<b>set_io_cond_list, get_io_status</b>

Ctrl Command	<b>clear_list</b>
Function	erases list buffer on the RTC® SCANalone
Use	useful only for PC operation
Integration	Pascal    procedure clear_list; C:           void clear_list(void); Basic:      sub clear_list()
Comments	<ul style="list-style-type: none"> <li>The <b>clear_list</b> command should be called before an application is loaded into the RTC® SCANalone's list buffer and subsequently stored onto the MMC via <b>store_on_mmc</b>. This command sequence prevents unneeded list entries (e.g. from prior test runs that still remain in the list buffer) from being stored onto the MMC.</li> <li>List buffer entries exist until they are overwritten or deleted via <b>clear_list</b>.</li> </ul>
References	<b>store_on_mmc</b>

Ctrl Command	<b>disable_laser</b>
Function	deactivates the laser control of the RTC® SCANalone
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
Integration	Pascal: <code>procedure disable_laser;</code>
	C: <code>void disable_laser(void);</code>
	Basic: <code>sub disable_laser()</code>
Comments	<ul style="list-style-type: none"> <li>• This command disables the laser signals LASER1, LASER2 and LASERON. Pins (1), (3) (LASERON) and (5) of the 10-pin laser connector (or pins (1), (2) (LASERON) and (3) of the 9-pin “LASER” D-SUB connector) and pins (19) and (22) of the “LASER EXTENSION” connector (or pins (10) and (24) of the “LASER EXTENSION” D-SUB connector) are then static.</li> <li>• If pin (3) of the 10-pin “LASER” connector (or pin (2) of the 9-pin “LASER” D-SUB connector) is configured as the ANALOG OUT2 signal, then the <b>disable_laser</b> command will not have an effect on this pin.</li> <li>• Use the command <b>enable_laser</b> to re-enable the laser control.</li> <li>• The command <b>get_startstop_info</b> (see <a href="#">page 80</a>) (Bit #9) provides information about the current status of the laser control.</li> <li>• After initialization of the RTC® SCANalone the laser control is <b>active</b>.</li> </ul>
References	<b>enable_laser</b> , <b>get_startstop_info</b>

Ctrl Command	<b>enable_laser</b>
Function	activates the laser control of the RTC® SCANalone
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
Integration	Pascal: <code>procedure enable_laser;</code>
	C: <code>void enable_laser(void);</code>
	Basic: <code>sub enable_laser()</code>
Comments	<ul style="list-style-type: none"> <li>• This command re-enables the laser signals LASER1, LASER2 and LASERON at the “LASER” (or “LASER” D-SUB) connector and at the “LASER EXTENSION” (or “LASER EXTENSION” D-SUB) connector.</li> <li>• The command <b>get_startstop_info</b> (see <a href="#">page 80</a>) provides information about the current status of the laser control. (Bit #9)</li> <li>• After initialization of the RTC® SCANalone, the laser control is <b>active</b>.</li> </ul>
References	<b>disable_laser</b> , <b>get_startstop_info</b>

Ctrl Command	<b>execute_at_pointer</b>
Function	starts list execution at the specified address in the RTC <sup>®</sup> SCANalone list buffer. Also see <a href="#">chapter 5.4 "Structured Programming", page 45</a> .
Use	for PC operation only
Parameter	pointer      address of the first list command to be executed (signed 32-bit value). Allowed range: [0 ... 999999]
Integration	Pascal:      procedure execute_at_pointer(pointer: longint);
	C:            void execute_at_pointer(long pointer);
	Basic:       sub execute_at_pointer(ByVal pointer&)
Comments	<ul style="list-style-type: none"> <li>• This command can be used instead of <b>execute_list</b>. For instance, <code>execute_at_pointer(0)</code> is the same as <code>execute_list(1)</code>, <code>execute_at_pointer(500000)</code> is the same as <code>execute_list(2)</code>. However, execution can be started at any other position in the list buffer as well.</li> <li>• CAUTION: If the end of the list buffer is reached, the RTC<sup>®</sup> SCANalone continues at the address zero.</li> <li>• Execution stops when a <b>set_end_of_list</b> command is encountered.</li> <li>• The command <b>execute_at_pointer</b> is ignored if a list is executing at the moment.</li> </ul>
References	<a href="#">set_input_pointer</a> , <a href="#">get_input_pointer</a>

Ctrl Command	<b>execute_list</b>
Function	starts execution of list 1 or list 2
Use	for PC operation only
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	<ul style="list-style-type: none"> <li>• The commands <b>execute_list_1</b> and <b>execute_list_2</b> (with no parameters) can be used alternatively.</li> <li>• Execution stops when a <b>set_end_of_list</b> command is encountered.</li> <li>• During execution of a particular list, the other list can be loaded. However, that other list <i>must</i> not be started by <b>execute_list</b> until the current list execution is finished. (The command <b>execute_list</b> is ignored if a list is executing at the moment.) Also see <a href="#">"List Handling", page 15</a>.</li> <li>• Use the command <a href="#">get_status (page 81)</a> to determine the current status of execution.</li> </ul>
References	<a href="#">get_status</a> , <a href="#">execute_at_pointer</a>

Ctrl Command	<b>get_counts</b>
Function	reads the counter for the external list starts
Use	for PC operation only
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	<ul style="list-style-type: none"> <li>• The counter is incremented each time a list is started via the external start signal.</li> <li>• To reset the counter, call the command <b>set_control_mode</b> (see page 102).</li> </ul>
References	<b>set_max_counts, set_control_mode, get_startstop_info</b>

Ctrl Command	<b>get_dll_version</b>
Function	returns the version number of the RTC <sup>®</sup> SCANalone driver DLL
Use	for PC operation only
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	<ul style="list-style-type: none"> <li>• The RTC<sup>®</sup> SCANalone DLL version numbers are in the range 500-599.</li> <li>• The RTC<sup>®</sup>4 DLL version numbers are in the range 400-499.</li> <li>• The RTC<sup>®</sup>3 DLL version numbers are in the range 100-399.</li> </ul>
References	<b>get_hex_version, get_rtc_version</b>

Ctrl Command	<b>get_head_status</b>
Function	returns the <i>scan head</i> status signals according to the XY2-100 or XY2-100-O protocol
Use	for PC operation only
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: <code>function get_head_status(head: word): word;</code> C: <code>unsigned short get_head_status(unsigned short head);</code> Basic: <code>function get_head_status(ByVal head%)%</code>
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	<ul style="list-style-type: none"> <li>• See <a href="#">chapter 5.3, page 44</a> for information about using two scan heads.</li> <li>• It's also important to consider all the status signal information described in your scan system's operating manual.</li> </ul>
References	<a href="#">get_status</a> , <a href="#">read_status</a> , <a href="#">set_piso_control</a>

Ctrl Command	<b>get_hex_version</b>
Function	returns the version number of the RTC® SCANalone software (program file)
Use	for PC operation only
Result	RTC® SCANalone software version number as an unsigned 16-bit value
Integration	Pascal: <code>function get_hex_version: word;</code> C: <code>unsigned short get_hex_version(void);</code> Basic: <code>function get_hex_version()%</code>
Comments	<ul style="list-style-type: none"> <li>• The RTC® SCANalone 2D-HEX version numbers are in the range 2.500-2.599</li> <li>• The RTC® SCANalone 3D-HEX version numbers are in the range 3.500-3.599</li> <li>• The RTC® 4 2D-HEX version numbers are in the range 2.400-2.499</li> <li>• The RTC® 4 3D-HEX version numbers are in the range 3.400-3.499</li> <li>• The RTC® 3 2D-HEX version numbers are in the range 2.000-2.399</li> <li>• The RTC® 3 3D-HEX version numbers are in the range 3.000-3.399</li> </ul>
References	<a href="#">get_dll_version</a> , <a href="#">get_rtc_version</a>

Ctrl Command	<b>get_input_pointer</b>
Function	returns the present list input pointer, i.e. the position in the RTC® SCANalone list buffer where the next list command will be stored.
Use	for PC operation only
Result	list input pointer [0 ... 999999] as signed 32-bit value.
Integration	Pascal:     function get_input_pointer: longint;
	C:           long get_input_pointer(void);
	Basic:       function get_input_pointer()&
References	<b>set_input_pointer, execute_at_pointer</b>

Ctrl Command	<b>get_io_status</b>
Function	returns the current state of the 16-bit digital <i>output</i> port on the "EXTENSION 1" connector
Use	for PC operation only
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	<ul style="list-style-type: none"> <li>This command is conceived of for use in combination with the commands <b>set_io_cond_list</b> and <b>clear_io_cond_list</b>.</li> <li>Also see "Programming Examples", page 46.</li> </ul>
References	<b>set_io_cond_list, clear_io_cond_list</b>

Ctrl Command	<b>get_rtc_version</b>										
Function	returns the <i>firmware</i> version number of the RTC® SCANalone board										
Use	for PC operation only										
Result	<p>RTC® SCANalone version number as an unsigned 16-bit value:</p> <table border="0"> <tr> <td>Bit #0 (LSB) ... Bit #7</td><td>Firmware version of the RTC® SCANalone board</td></tr> <tr> <td>Bit #8</td><td>= 1: Processing-on-the-fly is enabled. See supplement manual "Processing-On-The-Fly Software".</td></tr> <tr> <td>Bit #9</td><td>= 1: Second scan head connector is enabled. See page 57.</td></tr> <tr> <td>Bit #10</td><td>= 1: 3D control signals are enabled. See supplement manual "3D Software".</td></tr> <tr> <td>Bits #11...#15</td><td>reserved</td></tr> </table>	Bit #0 (LSB) ... Bit #7	Firmware version of the RTC® SCANalone 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 57.	Bit #10	= 1: 3D control signals are enabled. See supplement manual "3D Software".	Bits #11...#15	reserved
Bit #0 (LSB) ... Bit #7	Firmware version of the RTC® SCANalone 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 57.										
Bit #10	= 1: 3D control signals are enabled. See supplement manual "3D Software".										
Bits #11...#15	reserved										
Integration	Pascal:     function get_rtc_version: word;										
	C:           unsigned short get_rtc_version(void);										
	Basic:       function get_rtc_version()%										
Comments	<ul style="list-style-type: none"> <li>The RTC® SCANalone and RTC®4 firmware version numbers are in the range 128-255. If an RTC® SCANalone'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® SCANalones with odd version numbers, XY2-100 standard-compliant data transfer takes place electronically via a 25-pin female or 26-pin male connector.</li> <li>The RTC®3 firmware version numbers are in the range 0-127.</li> </ul>										
References	<b>get_hex_version, get_dll_version</b>										

Ctrl Command	<b>get_serial_number</b>
Function	returns the individual serial number of the RTC <sup>®</sup> SCANalone board
Use	for PC operation only
Result	RTC <sup>®</sup> SCANalone 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()%

Ctrl Command	<b>get_startstop_info</b>
Function	provides information about internal and external list starts and stops, as well as information about the laser signals
Use	for PC operation only
Result	<p>Status signal word (unsigned 16-bit value):</p> <p>Bit #0 (LSB)   = 1: An internal START has been executed since the last <b>get_startstop_info</b> command was called.</p> <p>Bit #1         = 1: An external START has been executed since the last <b>get_startstop_info</b> command was called.</p> <p>Bit #2         = 1: An internal STOP (command <b>stop_execution</b>) has been executed since the last <b>get_startstop_info</b> command was called.</p> <p>Bit #3         = 1: An external STOP has been executed since the last <b>get_startstop_info</b> command was called.</p> <p>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.</p> <p>Bit #9         Enable Laser Status. Bit = 1: Laser is enabled. See <b>enable_laser</b> and <b>disable_laser</b>.</p> <p>Bit #10        TTL Laser Signal Status. Bit = 1: Laser signals are <i>active-low</i> (Jumper X10 is open; see <a href="#">page 62</a>).</p> <p>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.</p> <p>The remaining bits are reserved.</p>
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	<b>get_counts</b> , <b>get_status</b>



Ctrl Command	get_status
Function	returns the current status of list execution
Use	for PC operation only
Result	busy      true ( $\neq 0$ )    : 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 $0 \leq \text{position} \leq 999999$ .
Integration	Pascal:    procedure get_status(var busy: wordbool, var position: longint);
	C:           void get_status(unsigned short *busy, long *position);
	Basic:     sub get_status(busy%, position&)
Comments	<ul style="list-style-type: none"> <li>• If position &lt; 500000, list 1 is executing at the moment, otherwise list 2.</li> <li>• If busy = false, the variable position contains the position of the last command that was executed (usually <b>set_end_of_list</b>).</li> <li>• In Pascal the variable busy can alternatively be of the type boolean.</li> <li>• 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).</li> </ul>
References	<b>read_status</b> , <b>get_head_status</b>

Ctrl Command	get_time
Function	returns the RTC® SCANalone timer value stored during the most recent call of <b>save_and_restart_timer</b>
Use	for PC operation only
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	<b>save_and_restart_timer</b>

Ctrl Command	get_value
Function	reads the current value of the specified signal
Use	for PC operation only
Parameter	<p>signal      specified signal represented by an unsigned 16-bit value:</p> <ul style="list-style-type: none"> <li>= 0:      LASERON signal (1 = Laser On, 0 = Laser Off)</li> <li>= 1:      StatusAX (X-axis status channel of head A)</li> <li>= 2:      StatusAY (Y-axis status channel of head A)</li> <li>= 3:      StatusAZ (Z-axis status channel of head A)</li> <li>= 4:      StatusBX (X-axis status channel of head B)</li> <li>= 5:      StatusBY (Y-axis status channel of head B)</li> <li>= 6:      StatusBZ (Z-axis status channel of head B)</li> <li>= 7:      SampleX (X-axis cartesian output value)</li> <li>= 8:      SampleY (Y-axis cartesian output value)</li> <li>= 9:      SampleZ (Z-axis cartesian output value)</li> <li>= 10:     SampleAX_Corr (X-axis output value of head A)</li> <li>= 11:     SampleAY_Corr (Y-axis output value of head A)</li> <li>= 12:     SampleAZ_Corr (Z-axis output value of head A)</li> <li>= 13:     SampleBX_Corr (X-axis output value of head B)</li> <li>= 14:     SampleBY_Corr (Y-axis output value of head B)</li> <li>= 15:     SampleBZ_Corr (Z-axis output value of head B)</li> </ul>
Result	Current value of the specified signal as an unsigned 16-bit value
Integration	Pascal: <code>function get_value(signal: word): smallint;</code>
	C: <code>short get_value(unsigned short signal);</code>
	Basic: <code>function get_value (ByVal signal As Integer) As Integer</code>
Comments	<ul style="list-style-type: none"> <li>• 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.</li> <li>• For scan systems with only one status channel, the status signals are only readable if <code>signal = 1</code> or <code>signal = 4</code>.</li> <li>• Coordinate transformations defined by <b>set_matrix</b>, <b>set_matrix_list</b>, <b>set_offset</b> or <b>set_offset_list</b> are already reflected in the SampleX, SampleY and SampleZ cartesian output values.</li> <li>• The SampleAX_Corr..SampleBZ_Corr output values are the (digital) output values actually transmitted from the RTC® SCANalone to the scan system. The RTC® SCANalone computes these values while taking into account the SampleX, SampleY and SampleZ output values as well as the selected correction file.</li> <li>• To observe the specified signal over a long time period, use <b>set_trigger</b> to start a corresponding measurement session.</li> </ul>
References	<b>set_trigger</b>

Ctrl Command	<b>get_wait_status</b>
Function	returns the wait state of the RTC® SCANalone
Use	for PC operation only
Result	wait state as an unsigned 16-bit value
Integration	Pascal: <code>function get_wait_status: word;</code> C: <code>unsigned short get_wait_status(void);</code> Basic: <code>function get_wait_status()%</code>
Comments	<ul style="list-style-type: none"> <li>• If processing has stopped at a wait marker, the command <b>get_wait_status</b> returns the number of this marker. See <b>set_wait</b> (page 119).</li> <li>• If no wait marker was reached, the command <b>get_wait_status</b> returns zero.</li> <li>• Processing of the list can be resumed by calling the command <b>release_wait</b>.</li> </ul>
References	<b>set_wait, release_wait</b>

Ctrl Command	<b>get_waveform</b>						
Function	transfers to the PC the data that was measured and stored onto the RTC® SCANalone via <b>set_trigger</b>						
Use	for PC operation only						
Parameters	<table border="0"> <tr> <td>channel</td><td>measurement channel (1 or 2); specified as an unsigned 16-bit value</td></tr> <tr> <td>stop</td><td>number of measured values [1..32768] to transfer; specified as an unsigned 16-bit value (values of measurement positions 1..stop will be transferred)</td></tr> <tr> <td>memptr</td><td>unsigned 16-bit pointer to a location in the PC's memory to where the measured values should be transferred</td></tr> </table>	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
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: <code>procedure get_waveform(channel, stop: word; memptr: pint);</code> C: <code>void get_waveform(unsigned short channel, unsigned short stop, signed short *memptr);</code> Basic: <code>sub get_waveform (ByVal channel As Integer, ByVal stop As Integer, ByVal memptr As Integer)</code>						
References	<b>set_trigger</b>						

Ctrl Command	<b>get_xy_pos</b>
Function	returns the current scanner set position
Use	for PC operation only
Result	xpos, ypos     current output position in <i>bits</i> as signed 16-bit values
Integration	Pascal: <code>procedure get_xy_pos(var xpos, ypos: smallint);</code> C: <code>void get_xy_pos(short *xpos, short *ypos);</code> Basic: <code>sub get_xy_pos(xpos%, ypos%)</code>
Comments	<ul style="list-style-type: none"> <li>• The command returns the current output position (before the image field correction is applied).</li> <li>• If an image transformation was defined with the commands <b>set_matrix</b> / <b>set_offset</b>, the actual output coordinates are generally <i>not</i> equal to the original input coordinates.</li> </ul>

Ctrl Command	<b>goto_xy</b>
Function	direct jump to the specified position
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
Parameters	xpos, ypos    coordinates of the jump position in <i>bits</i> as signed 16-bit values
Integration	Pascal: <code>procedure goto_xy(xpos, ypos: smallint);</code>
	C: <code>void goto_xy(short xpos, short ypos);</code>
	Basic: <code>sub goto_xy(ByVal xpos%, ByVal ypos%)</code>
Comments	<ul style="list-style-type: none"> <li>• The jump will always be executed at a speed of <b>50000 bits/ms</b>.</li> <li>• The command will be ignored if a list is executing at the moment.</li> <li>• The command will not be executed, if an external stop signal (/STOP or /STOP2) is present.</li> <li>• Image field correction will be applied.</li> <li>• If an image transformation is defined with the commands <a href="#">set_matrix</a> / <a href="#">set_offset</a>, it will be applied.</li> </ul>
References	<a href="#">get_xy_pos</a>

Ctrl Command	<b>home_position</b>
Function	activates the home jump mode and defines the home position
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
Parameters	xhome, yhome    coordinates of the home position in <i>bits</i> as signed 16-bit values.
Integration	Pascal: <code>procedure home_position (xhome, yhome: smallint);</code>
	C: <code>void home_position (short xhome, short yhome);</code>
	Basic: <code>sub home_position (ByVal xhome%, ByVal yhome%)</code>
Comments	<ul style="list-style-type: none"> <li>• 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.</li> <li>• A home jump is also executed, if list execution is stopped via <a href="#">stop_execution</a> but not if list execution is stopped via an external stop signal.</li> <li>• 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 <b>field_jump</b> is no longer needed.</li> <li>• A <i>beam dump</i> should be placed in the home position.</li> <li>• The home jump mode is deactivated with the command <code>home_position(0, 0)</code>.</li> </ul>

List Command	<b>jump_abs</b>
Function	fast movement of the mirrors ("jump") to the specified position
Use	for PC and standalone operation
Parameters	xval, yval      absolute coordinates of the vector end point in <i>bits</i> 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	<ul style="list-style-type: none"> <li>• The jump speed is set with the command <b>set_jump_speed</b> (see page 106).</li> <li>• The laser is off during the jump.</li> <li>• After a jump command, a jump delay is inserted.</li> <li>• 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.</li> </ul>
References	<b>set_jump_speed, set_scanner_delays, jump_rel</b>

List Command	<b>jump_rel</b>
Function	fast movement of the mirrors ("jump") to the specified position
Use	for PC and standalone operation
Parameters	dx, dy <i>relative</i> coordinates of the vector end point in <i>bits</i> 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	<ul style="list-style-type: none"> <li>• The coordinates are relative to the current position.</li> <li>• The maximum value for the <i>absolute</i> coordinates is <math>\pm 32767</math> bits.</li> <li>• The jump speed is set with the command <b>set_jump_speed</b> (see page 106).</li> <li>• The laser is off during the jump.</li> <li>• After a jump command, a jump delay is inserted.</li> <li>• 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.</li> </ul>
References	<b>set_jump_speed, set_scanner_delays, jump_abs</b>

List Command	laser_on_list
Function	turns on the laser for a specified time interval
Use	for PC and standalone operation
Parameter	delay      time interval in <i>bits</i> as an unsigned 16-bit value. <b>1 bit equals 10 µs.</b> Allowed range: $0 \leq \text{delay} \leq 65500$
Integration	Pascal: <code>procedure laser_on_list(delay: word);</code>
	C: <code>void laser_on_list(unsigned short delay);</code>
	Basic: <code>sub laser_on_list(ByVal delay%)</code>
Comments	<ul style="list-style-type: none"> <li>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.</li> <li>The current settings for the laser delays are applied: <ul style="list-style-type: none"> <li>At the beginning of the programmed time interval, the laser turns on after a <i>LaserOn delay</i>.</li> <li>At the end of the time interval, the laser turns off with the corresponding <i>LaserOff delay</i>.</li> </ul> </li> <li>The command is useful for marking separate dots.</li> </ul>

Ctrl Command	laser_signal_off
Function	turns off the laser immediately
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
Integration	Pascal: <code>procedure laser_signal_off;</code>
	C: <code>void laser_signal_off(void);</code>
	Basic: <code>sub laser_signal_off()</code>
Comments	<ul style="list-style-type: none"> <li>This command is intended for direct laser control in combination with the command <b>laser_signal_on</b> (see below).</li> <li><b>The command will be ignored if a list is executing at the moment.</b></li> </ul>
References	<b>laser_signal_on</b>

List Command	laser_signal_off_list
Function	same as <b>laser_signal_off</b> (see above), but a list command
Use	for PC and standalone operation
Integration	Pascal: <code>procedure laser_signal_off_list;</code>
	C: <code>void laser_signal_off_list(void);</code>
	Basic: <code>sub laser_signal_off_list()</code>

Ctrl Command	<b>laser_signal_on</b>
Function	turns on the laser immediately
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
Integration	Pascal: <code>procedure laser_signal_on;</code>
	C: <code>void laser_signal_on(void);</code>
	Basic: <code>sub laser_signal_on()</code>
Comments	<ul style="list-style-type: none"> <li>• The command is intended for turning on the laser directly, e.g. for alignment purposes.</li> <li>• Prior to calling the command <b>laser_signal_on</b>, the period and pulse width of the outgoing signal must be set via the list command <b>set_laser_timing</b>.</li> <li>• The laser must be turned off with the command <b>laser_signal_off</b>.</li> <li>• The command will be ignored if a list is executing at the moment.</li> <li>• <b>Check the beam path before turning on the laser!</b></li> </ul>
References	<a href="#">laser_signal_off</a>

List Command	<b>laser_signal_on_list</b>
Function	same as <b>laser_signal_on</b> , but a list command
Use	for PC and standalone operation
Integration	Pascal: <code>procedure laser_signal_on_list;</code>
	C: <code>void laser_signal_on_list(void);</code>
	Basic: <code>sub laser_signal_on_list()</code>

List Command	<b>list_call</b>
Function	defines a jump to the subroutine which starts at the specified list buffer address
Use	for PC and standalone operation
Parameter	address     jump address [0 ... 999999] as signed 32-bit value
Integration	Pascal: <code>procedure list_call(address: longint);</code>
	C: <code>void list_call(long address);</code>
	Basic: <code>sub list_call(ByVal address&amp;)</code>
Comments	<ul style="list-style-type: none"> <li>• The subroutine must be terminated with the command <b>list_return</b>.</li> <li>• Within a subroutine, another subroutine can be called (up to a depth of 30 calls).</li> <li>• Also see <a href="#">chapter 5.4 "Structured Programming"</a>, <a href="#">page 45</a>.</li> </ul>
References	<a href="#">list_return</a> , <a href="#">set_list_jump</a>

List Command	list_call_cond
Function	<p><i>Conditional subroutine call:</i> During execution of a list, this command causes a jump to a subroutine at the specified list buffer address [0 ... 999999], if the current IOvalue at the 16-bit digital input port meets the following condition:</p> $((IOvalue \text{ AND } mask\_1) = mask\_1) \text{ AND } (((\text{not } IOvalue) \text{ AND } mask\_0) = mask\_0)$ <p>(i.e. the bits specified in mask_1 must be 1 and the bits specified in mask_0 must be 0).</p>
Use	for PC and standalone operation
Parameters	mask_1, unsigned 16-bit masks mask_0
	address jump address [0 ... 999999] as signed 32-bit value
Integration	Pascal: list_call_cond(mask_1, mask_0: word; address: longint);
	C: void list_call_cond(unsigned short mask_1, unsigned short mask_0, long address);
	Basic: sub list_call_cond(ByVal mask_1%, ByVal mask_0%, ByVal address&)
Comments	<ul style="list-style-type: none"> <li>The subroutine must be terminated with the command <b>list_return</b> (see the <a href="#">chapter 5.4 "Structured Programming", page 45</a>).</li> <li>Also see "<a href="#">Programming Examples</a>", <a href="#">page 46</a> (sample #<a href="#">(3)</a>)</li> </ul>
References	<b>list_call</b> , <b>list_return</b>

List Command	list_jump_cond
Function	<p><i>Conditional list jump:</i> During execution of a list, this command causes a jump to the specified list buffer address [0 ... 999999], if the current IOvalue at the 16-bit digital input port meets the following condition:</p> $((IOvalue \text{ AND } mask\_1) = mask\_1) \text{ AND } (((\text{not } IOvalue) \text{ AND } mask\_0) = mask\_0)$ <p>(i.e. the bits specified in mask_1 must be 1 and the bits specified in mask_0 must be 0).</p>
Use	for PC and standalone operation
Parameters	mask_1, unsigned 16-bit masks mask_0
	address jump address [0 ... 999999] as signed 32-bit value
Integration	Pascal: list_jump_cond(mask_1, mask_0: word; address: longint);
	C: void list_jump_cond(unsigned short mask_1, unsigned short mask_0, long address);
	Basic: sub list_jump_cond(ByVal mask_1%, ByVal mask_0%, ByVal address&)
Comments	<ul style="list-style-type: none"> <li>See the <a href="#">chapter 5.4 "Structured Programming", page 45</a>.</li> </ul>
Examples (Pascal)	<ul style="list-style-type: none"> <li>wait until bit #3 of the input port turns HIGH (= loop while the bit is <i>LOW</i>): list_jump_cond(0, \$0008, get_input_pointer);</li> <li>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);</li> <li>Also see "<a href="#">Programming Examples</a>", <a href="#">page 46</a>.</li> </ul>
References	<b>set_list_jump</b>



List Command	<b>list_nop</b>
Function	inserts a null operation (no operation) into the list buffer
Use	for PC and standalone operation
Integration	Pascal: <code>procedure list_nop;</code>
	C: <code>void list_nop(void);</code>
	Basic: <code>sub list_nop()</code>
Comments	<ul style="list-style-type: none"> <li>• Null operations serve as place markers. The execution of a null operation needs 10 <math>\mu</math>s.</li> </ul>

List Command	<b>list_return</b>
Function	terminates a list subroutine and jumps to the command following the <b>list_call</b> command
Use	for PC and standalone operation
Integration	Pascal: <code>procedure list_return;</code>
	C: <code>void list_return(void);</code>
	Basic: <code>sub list_return()</code>
Comments	<ul style="list-style-type: none"> <li>• See <a href="#">chapter 5.4 "Structured Programming"</a>, page 45.</li> </ul>
References	<b>list_call</b>

Ctrl Command	load_correction_file																				
Function	loads the specified field correction file into RTC® SCANalone memory (as table #1 or #2) and performs additional scaling, rotation and translation of the correction file																				
Use	for PC operation only; in PC operation, the command must also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ). Output is not possible without a correction file.																				
Parameters	<div> <div>FileName</div> <div>name of the correction file as a pointer to a null-terminated ANSI string</div> </div>																				
	<div> <div>cor_table</div> <div>determines whether the file shall be stored as correction table #1 or #2 (see command <a href="#">select_cor_table</a>, <a href="#">page 100</a>)</div> </div>																				
	<div> <div>kx, ky</div> <div>Gain values for scaling the correction table. Allowed range: [−1.2 ... −0.8, +0.8 ... +1.2]. (See comments below.)</div> </div>																				
	<div> <div>phi</div> <div>Rotational angle in degrees Allowed values: [−10° ... +10°] + i · 90° ; i integer Positive angles: rotation is counterclockwise</div> </div>																				
	<div> <div>x_offset, y_offset</div> <div>additional offset in <i>bits</i> allowed range: [−30000 ... +30000]</div> </div>																				
Result	<p>error code as a signed 16-bit value:</p> <table> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td>0</td><td>Success.</td></tr> <tr> <td>3</td><td>File error.</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 (access rights assigned to another application).</td></tr> <tr> <td>10</td><td>Parameter error.</td></tr> <tr> <td>11</td><td>RTC® SCANalone not found.</td></tr> <tr> <td>12</td><td>A 3D correction file could not be loaded, because the 3D option is not installed. See <a href="#">chapter 7 "Options"</a>, <a href="#">page 60</a>.</td></tr> <tr> <td>14</td><td>correction file has out-of-date format</td></tr> <tr> <td>15</td><td>wrong correction file</td></tr> </tbody> </table>	Value	Description	0	Success.	3	File error.	4	Verify error.	8	System driver not found, or the system driver is locked by another application (access rights assigned to another application).	10	Parameter error.	11	RTC® SCANalone not found.	12	A 3D correction file could not be loaded, because the 3D option is not installed. See <a href="#">chapter 7 "Options"</a> , <a href="#">page 60</a> .	14	correction file has out-of-date format	15	wrong correction file
Value	Description																				
0	Success.																				
3	File error.																				
4	Verify error.																				
8	System driver not found, or the system driver is locked by another application (access rights assigned to another application).																				
10	Parameter error.																				
11	RTC® SCANalone not found.																				
12	A 3D correction file could not be loaded, because the 3D option is not installed. See <a href="#">chapter 7 "Options"</a> , <a href="#">page 60</a> .																				
14	correction file has out-of-date format																				
15	wrong correction file																				
Integration	<div> <div>Pascal:</div> <div>function load_correction_file(FileName: pchar; cor_table: smallint; kx, ky, phi, x_offset, y_offset: double): smallint;</div> </div>																				
	<div> <div>C:</div> <div>short load_correction_file(const char* FileName, short cor_table, double kx, double ky, double phi, double x_offset, double y_offset);</div> </div>																				
	<div> <div>Basic:</div> <div>function load_correction_file(ByVal FileName\$, ByVal cor_table%, ByVal kx#, ByVal ky#, ByVal phi#, ByVal x_offset#, ByVal y_offset#)%</div> </div>																				

Ctrl Command	load_correction_file
Comments	<ul style="list-style-type: none"> <li>• This command should be called at the beginning of an application program, before loading the program file. See note in <a href="#">chapter 9.3 "Initializing the RTC® SCANalone", page 67</a>.</li> <li>• The RTC® SCANalone (only the 2D version of the RTC® SCANalone) 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 <a href="#">select_cor_table</a> (page 100). Also see <a href="#">chapter 5.3 "Using Two Different Correction Files", page 44</a>.</li> <li>• 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® SCANalone.</li> <li>• The parameters <i>kx</i>, <i>ky</i>, <i>phi</i>, <i>x_offset</i> and <i>y_offset</i> <i>have to be</i> specified. If no additional transformation is required, the parameters must be set to (., ., 1, 1, 0, 0, 0).</li> <li>• By setting the gain factor <i>kx</i> or <i>ky</i> to a <i>negative</i> value, the corresponding axis is flipped.</li> <li>• Modification of 3D correction files is only reliable for <i>Z</i> = 0. For other <i>Z</i> values and parameters other than (., ., 1, 1, 0, 0, 0), the RTC® SCANalone calculates output values which may deviate from the expected values.</li> <li>• RTC® SCANalone boards featuring the optional functionality of controlling the third axis of a 3-axis system (3D system) need a 3D correction file. SCANLAB names these files D3_***.CTB. All other RTC® SCANalone board versions need 2D correction files (named D2_***.CTB). Correction files with wrong format cannot be loaded. The command <a href="#">load_correction_file</a> thereby only returns a corresponding error code.</li> </ul>
References	<a href="#">select_cor_table</a>

Ctrl Command	load_varpolydelay														
Function	loads a table for the variable polygon delay into the RTC <sup>®</sup> SCANalone. See the section " <a href="#">Customizing The Variable Polygon Delay</a> " on page 26.														
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC <sup>®</sup> SCANalone (see <a href="#">page 14</a> ).														
Parameters	STBFileName    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.														
	TableNo        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.														
Result	<p>error code as a signed 16-bit value:</p> <table> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <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 "<a href="#">Customizing The Variable Polygon Delay</a>" on page 26.</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<sup>®</sup> SCANalone 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> </tbody> </table>	Value	Description	≤ -1	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 " <a href="#">Customizing The Variable Polygon Delay</a> " on page 26.	4	Verify error.	8	System driver not found, or the system driver is locked by another application.	10	Parameter error.	11	RTC <sup>®</sup> SCANalone not found.	13	The specified table number was not found in the file; or the file was not found.
Value	Description														
≤ -1	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 " <a href="#">Customizing The Variable Polygon Delay</a> " on page 26.														
4	Verify error.														
8	System driver not found, or the system driver is locked by another application.														
10	Parameter error.														
11	RTC <sup>®</sup> SCANalone not found.														
13	The specified table number was not found in the file; or the file was not found.														
Integration	Pascal:    function load_varpolydelay (STBFileName: pchar, TableNo: word): smallint;														
	C:         short load_varpolydelay (const char* STBFileName, unsigned short TableNo);														
	Basic:     function load_varpolydelay (ByVal STBFileName\$, ByVal TableNo%)%														
Comments	<ul style="list-style-type: none"> <li>The table contains data points for the scaling function of the user-defined variable polygon delay. When loading the table, the RTC<sup>®</sup> SCANalone determines suitable values for the entire range of angles by linear interpolation. Also see the section "<a href="#">Customizing The Variable Polygon Delay</a>" on page 26.</li> <li>The command <b>load_varpolydelay</b> overwrites any previously loaded table for the variable polygon delay.</li> <li>As long as <b>load_varpolydelay</b> is not called and no corresponding initialization is stored on the MMC, then after a reset the RTC<sup>®</sup> SCANalone will use the standard scaling function for the variable polygon delay (see <a href="#">figure 9 on page 24</a>). That means the command <b>load_varpolydelay</b> is only needed if a different table should be used.</li> <li>To return to the standard polygon delay table (after a different table has been used), an RTC<sup>®</sup> SCANalone reset is required with no MMC card attached.</li> </ul>														
References	<a href="#">set_delay_mode</a>														

List Command	<b>long_delay</b>
Function	stops execution of the list for the specified time
Use	for PC and standalone operation
Parameter	delay      delay time in <i>bits</i> as an unsigned 16-bit value. <b>1 bit equals 10 µs.</b> Allowed range: $0 \leq \text{delay} \leq 65500$ .
Integration	Pascal:    procedure long_delay(delay: word);
	C:          void long_delay(unsigned short delay);
	Basic:     sub long_delay(ByVal delay%)
Comments	<ul style="list-style-type: none"> <li>This command should always be called after changing the lamp current of a YAG laser to obtain a constant laser power.</li> </ul>

List Command	<b>mark_abs</b>
Function	marking along a straight line from the current position to the specified position
Use	for PC and standalone operation
Parameters	xval,      absolute coordinates of the vector end point in <i>bits</i> as signed 16-bit values. yval
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	<ul style="list-style-type: none"> <li>The marking speed is set with the command <b>set_mark_speed</b> (see page 109).</li> <li>The laser is turned on at the beginning of the command after a LaserOn delay.</li> <li>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.</li> <li>If a marking vector is followed by a zero jump or marking vector or a zero arc command, then the MarkDelay is <i>not</i> executed.</li> <li>See chapter 4.2 "Scan Head And Laser Control", page 17, for further details.</li> </ul>
References	<b>set_mark_speed, set_scanner_delays, mark_rel</b>

List Command	mark_date
Function	the date stored via <b>time_fix</b> will be marked at the current position.
Use	for PC and standalone operation
Parameters	<b>part</b> this parameter (unsigned 16-bit value) specifies which part of the date should be marked: = 0:    year (only the last two digits) = 1:    month = 2:    day corresponding to the normal Gregorian date = 3:    day-of-the-week = 4:    Julian day (0..364 or 0..365)
	<b>mode</b> unsigned 16-bit value that specifies the day format ( <b>part</b> = 2 or 4); = 0:    leading zero is suppressed = 1:    always two digits ( <b>part</b> = 2) or three digits ( <b>part</b> = 4)
Integration	Pascal: <code>mark_date(part, mode: word);</code>
	C: <code>void mark_date(unsigned short part, unsigned short mode);</code>
	Basic: <code>sub mark_date(ByVal part%, ByVal mode%)</code>
Comments	<ul style="list-style-type: none"> <li>The <b>mark_date</b> command initiates the output of command lists whose starting addresses were stored via <b>set_char_table</b>. The command lists must contain marking instructions for digits 0...9 and for the month and day-of-the-week designations (see <a href="#">page 51</a>).</li> <li>The complete date can be marked via multiple calls of the <b>mark_date</b> command.</li> <li>The year is always marked as two digits (therefore the last 2 digits of the year are used). For a full 4-digit year, the first two digits must be marked separately.</li> <li>When marking Gregorian dates or Julian days, the transition to the next day occurs at 24:00 o'clock. Leap years are represented in both date styles.</li> </ul>
References	<b>time_fix</b> , <b>set_char_table</b>

List Command	mark_rel
Function	marking along a straight line from the current position to the specified position
Use	for PC and standalone operation
Parameters	<b>dx, dy</b> <i>relative</i> coordinates of the vector end point in <i>bits</i> as signed 16-bit values.
Integration	Pascal: <code>procedure mark_rel(dx, dy: smallint);</code>
	C: <code>void mark_rel(short dx, short dy);</code>
	Basic: <code>sub mark_rel(ByVal dx%, ByVal dy%)</code>
Comments	<ul style="list-style-type: none"> <li>The coordinates are relative to the current position.</li> <li>The maximum value for the <i>absolute</i> coordinates is <math>\pm 32767</math> bits.</li> <li>The marking speed is set with the command <b>set_mark_speed</b> (see <a href="#">page 109</a>).</li> <li>The laser is turned on at the beginning of the command after a LaserOn delay.</li> <li>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.</li> <li>If a marking vector is followed by a zero jump or a marking vector or a zero arc command, then the MarkDelay is <i>not</i> executed.</li> <li>See <a href="#">chapter 4.2 "Scan Head And Laser Control"</a>, <a href="#">page 17</a>, for further details.</li> </ul>
References	<b>set_mark_speed</b> , <b>set_scanner_delays</b> , <b>mark_abs</b>

List Command	<b>mark_serial</b>
Function	The current serial number is output starting at the current position and afterward the number is (optionally) incremented by one.
Use	for PC and standalone operation
Parameters	<div> <div>mode</div> <div> serial number format as an unsigned 16-bit value.  = 0: leading zeros marked (e.g. "000034"),  serial number incremented by one  = 1: leading zeros suppressed,  left-justified marking (e.g. "34"),  serial number incremented by one  = 2: leading zeros replaced by blank characters,  serial number incremented by one  = 10: as in mode 0, but the serial number is <i>not</i> incremented  = 11: as in mode 1, but the serial number is <i>not</i> incremented  = 12: as in mode 2, but the serial number is <i>not</i> incremented </div> </div> <div> <div>digits</div> <div>number [0-10] of to-be-marked characters as an unsigned 16-bit value</div> </div>
Integration	<div>Pascal: procedure mark_serial(mode, digits: word);</div> <div>C: void mark_rel(unsigned short mode, unsigned short digits);</div> <div>Basic: sub mark_rel(ByVal mode%, ByVal digits%)</div>
Comments	<ul style="list-style-type: none"> <li>The <b>mark_serial</b> command initiates the output of command lists whose starting addresses were stored via <b>set_char_table</b>. The command lists must contain marking definitions for the numerals 0...9 (see <a href="#">page 51</a>).</li> <li>The first serial number to be marked must have been previously specified via the control command <b>set_serial</b>. After the RTC® SCANalone Board boots with an installed MMC, this serial number will be read from the MMC and applied. If the MMC's contents are not updated following a power interruption, then previously marked serial numbers will be marked again.</li> <li>Optionally, the serial number increments by one with each call of the <b>mark_serial</b> command (after the actual serial number was marked). If the incremented serial number exceeds 10<sup>digits</sup>, then marking begins again at 0. The control command <b>set_max_counts</b> allows specification of the maximum number of external list starts and thus the maximum number of external markings.</li> <li>If <b>digits</b> = 0, then a "markless" marking is executed. The increment size between two markable serial numbers can be indirectly controlled by issuing "markless" <b>mark_serial</b> commands, each time incrementing the serial number by one (<b>mode</b> = 0, 1 or 2).</li> <li>The current serial number can't be directly queried. But it can be calculated via the starting serial number and the number of external list starts. In PC operation, the number of external list starts can be obtained with the control command <b>get_counts</b>.</li> </ul>
References	<b>set_char_table</b> , <b>set_serial</b> , <b>set_max_counts</b> , <b>get_counts</b>

List Command	<b>mark_time</b>
Function	The time stored via <b>time_fix</b> will be marked at the current position.
Use	for PC and standalone operation
Parameters	<b>part</b> this parameter (unsigned 16-bit value) specifies which part of the time to mark: = 0:    hours = 1:    minutes = 2:    seconds
	<b>mode</b> format, specified as an unsigned 16-bit value = 0:    leading zero is suppressed = 1:    always two digits
Integration	Pascal: <code>mark_time(part, mode: word);</code>
	C: <code>void mark_time(unsigned short part, unsigned short mode);</code>
	Basic: <code>sub mark_time(ByVal part%, ByVal mode%)</code>
Comments	<ul style="list-style-type: none"> <li>• The <b>mark_time</b> command starts outputting command lists whose start addresses were stored via <b>set_char_table</b>. The command lists must contain marking instructions for the digits 0...9 (see <a href="#">page 51</a>).</li> <li>• The complete time can be marked via multiple calls of the <b>mark_time</b> command.</li> <li>• Time marking is always in 24-hour mode. The 12-hour American variant (a.m./p.m.) is not supported.</li> </ul>
References	<b>time_fix</b> , <b>set_char_table</b>

Ctrl Command	<b>measurement_status</b>
Function	returns the status of a measurement session started via <b>set_trigger</b>
Use	only for PC operation
Result	<b>busy</b> true ( ≠ 0 ) : a measurement session is currently in progress false ( = 0 ) : no measurement session is currently in progress
	<b>position</b> current position within the RTC® SCANalone's measurement storage area 0 ≤ position ≤ 32767
Integration	Pascal: <code>procedure measurement_status(var busy: wordbool; var position: word);</code>
	C: <code>void measurement_status(unsigned short *busy, unsigned short *position);</code>
	Basic: <code>sub measurement_status (busy As Integer, position As Integer)</code>
Comments	<ul style="list-style-type: none"> <li>• The <b>set_trigger</b> command will always cause exactly 32768 values per measurement channel to be measured and stored onto the RTC® SCANalone.</li> </ul>
References	<b>set_trigger</b>



Ctrl Command	<b>quit_loop</b>
Function	stops continuous output of the two lists started with <b>start_loop</b>
Use	only for PC operation
Integration	Pascal:    procedure quit_loop; C:            void quit_loop(void); Basic:       sub quit_loop()
Comments	<ul style="list-style-type: none"> <li>The current list will execute completely before execution is stopped.</li> </ul>
References	<b>start_loop</b>

Ctrl Command	<b>read_ad_x</b>
Function	reads the current status of the selected analog-to-digital converter (ADC) on the RTC® SCANalone (also see <a href="#">page 58</a> )
Use	only for PC operation
Parameters	x            [1, 2]: number of the analog input on the RTC® SCANalone
Result	output value (unsigned 10-bit value) of the selected analog-to-digital converter (ANALOG_IN1 or ANALOG_IN2)
Integration	Pascal:    function read_ad_x(x: word): word; C:            unsigned short read_ad_x(unsigned short x); Basic:       function read_ad_x(ByteVal x%)%
References	<b>write_da_x</b>

Ctrl Command	<b>read_io_port</b>
Function	returns the state of the 16-bit digital input port on the "EXTENSION 1" connector
Use	only for PC operation
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	<b>write_io_port</b>

Ctrl Command	read_status																											
Function	returns the list execution status																											
Use	for PC operation only																											
Result	<div>RTC<sup>®</sup> SCANalone status as an unsigned 16-bit value:</div> <table><thead><tr><th>Bit #</th><th>Name</th><th>Description</th></tr></thead><tbody><tr><td>Bit #0 (LSB)</td><td>Load1</td><td>= 1: indicates that all following list commands will be stored in list 1. This bit will be set after a <b>set_start_list_1</b> command and will be reset after a <b>set_end_of_list</b> command.</td></tr><tr><td>Bit #1</td><td>Load2</td><td>= 1: indicates that all following list commands will be stored in list 2. This bit will be set after a <b>set_start_list_2</b> command and will be reset after a <b>set_end_of_list</b> command.</td></tr><tr><td>Bit #2</td><td>Ready1</td><td>= 1: indicates that list 1 is closed. This bit will be set after list 1 is closed by the <b>set_end_of_list</b> command.</td></tr><tr><td>Bit #3</td><td>Ready2</td><td>= 1: indicates that list 2 is closed. This bit will be set after list 2 is closed by the <b>set_end_of_list</b> command.</td></tr><tr><td>Bit #4</td><td>Busy1</td><td>= 1: indicates that list 1 is executing at the moment.</td></tr><tr><td>Bit #5</td><td>Busy2</td><td>= 1: indicates that list 2 is executing at the moment.</td></tr><tr><td>Bits #6...#7</td><td></td><td>0</td></tr><tr><td>Bits #8...#15</td><td></td><td>1</td></tr></tbody></table>	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 <b>set_start_list_1</b> command and will be reset after a <b>set_end_of_list</b> command.	Bit #1	Load2	= 1: indicates that all following list commands will be stored in list 2. This bit will be set after a <b>set_start_list_2</b> command and will be reset after a <b>set_end_of_list</b> command.	Bit #2	Ready1	= 1: indicates that list 1 is closed. This bit will be set after list 1 is closed by the <b>set_end_of_list</b> command.	Bit #3	Ready2	= 1: indicates that list 2 is closed. This bit will be set after list 2 is closed by the <b>set_end_of_list</b> 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
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 <b>set_start_list_1</b> command and will be reset after a <b>set_end_of_list</b> command.																										
Bit #1	Load2	= 1: indicates that all following list commands will be stored in list 2. This bit will be set after a <b>set_start_list_2</b> command and will be reset after a <b>set_end_of_list</b> command.																										
Bit #2	Ready1	= 1: indicates that list 1 is closed. This bit will be set after list 1 is closed by the <b>set_end_of_list</b> command.																										
Bit #3	Ready2	= 1: indicates that list 2 is closed. This bit will be set after list 2 is closed by the <b>set_end_of_list</b> 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	<div>Pascal:     function read_status: word;</div> <div>C:           unsigned short read_status(void);</div> <div>Basic:       function read_status()%</div>																											
Comments	<ul style="list-style-type: none"><li>• Compare with the command <b>get_status</b> (page 81).</li><li>• To read the status signals from the <i>scan heads</i>, use the command <b>get_head_status</b> (page 78).</li></ul>																											
References	<b>get_status, get_head_status</b>																											

Ctrl Command	release_wait
Function	resumes execution of a list that was interrupted by a <b>set_wait</b> command
Use	for PC operation only
Result	wait state as an unsigned 16-bit value
Integration	<p>Pascal:     <code>procedure release_wait;</code></p> <p>C:           <code>void release_wait(void);</code></p> <p>Basic:       <code>sub release_wait()</code></p>
Comments	<ul style="list-style-type: none"> <li>• The command <b>release_wait</b> can only be used if the RTC<sup>®</sup> SCANalone is actually in a wait state (i.e. a wait marker was reached and processing has stopped).</li> <li>• The command <b>release_wait</b> resets the <code>wait_word</code> to zero.</li> </ul>
References	<b>set_wait, get_wait_status</b>

Ctrl Command	<b>restart_list</b>
Function	enables the laser again and resumes execution of a list that was interrupted using the command <b>stop_list</b> .
Use	for PC operation only
Integration	Pascal: <code>procedure restart_list;</code>
	C: <code>void restart_list(void);</code>
	Basic: <code>sub restart_list()</code>
References	<b>stop_list</b>

List Command	<b>save_and_restart_timer</b>
Function	stores the current value of the RTC® SCANalone timer and resets it to zero
Use	for PC operation only
Integration	Pascal: <code>procedure save_and_restart_timer;</code>
	C: <code>void save_and_restart_timer(void);</code>
	Basic: <code>sub save_and_restart_timer()</code>
Comments	<ul style="list-style-type: none"> <li>• The stored timer value can be read by the <b>get_time</b> command.</li> <li>• The command is useful for measuring the marking time of a marking process.</li> </ul>
References	<b>get_time</b>

Ctrl Command	<b>select_cor_table</b>
Function	assigns the previously loaded correction tables #1 and #2 to the scan head control ports
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
Parameters	<div> 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 </div> <div> 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 </div>
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	<ul style="list-style-type: none"> <li>• 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: <code>select_cor_table(1,2)</code>.</li> <li>• 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 <code>select_cor_table(1,0)</code> and <code>select_cor_table(2,0)</code> to switch from one file to the other.)</li> <li>• 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 <a href="#">chapter 5.3 "Using Two Different Correction Files"</a>, <a href="#">page 44</a>.</li> <li>• The 3D version of the RTC® SCANalone can store only <i>one</i> correction file.</li> </ul>
References	<a href="#">load_correction_file</a>

Ctrl Command	<b>select_list</b>
Function	selects which list will be executed upon receipt of an external start signal
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
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	<ul style="list-style-type: none"> <li>• By default, list 1 is selected.</li> <li>• In PC operation, 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.</li> <li>• SCANLAB recommends taking advantage of the RTC® SCANalone's huge memory capacity by loading just a single list (with up to 1 million list commands) into the list buffer.</li> </ul>
References	<a href="#">set_extstartpos_list</a>

Ctrl Command	set_char_table
Function	stores the jump addresses of command lists defined as marking instructions needed for characters or character sequences.
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
Parameters	<p>ch      unsigned 16-bit value that specifies the category of the character or character sequence for which a corresponding command list with marking instructions is stored at the jump address listpos:</p> <p>= 0 ... 9:    digits (0 ... 9) for marking the time and date</p> <p>= 10 ... 21: months (January ... December)</p> <p>= 22 ... 28: days-of-the-week (Sunday ... Saturday)</p> <p>= 29:        blank character for marking serial numbers</p> <p>= 30 ... 39: digits (0 ... 9) for marking serial numbers</p>
	<p>listpos   signed 32-bit jump address in the list buffer [1 ... 999999]</p>
Integration	Pascal:    set_char_table(ch: word; listpos: longint);
	C:        void set_char_table(unsigned short ch, long listpos);
	Basic:    sub set_char_table(ByVal ch%, ByVal listpos%)
Comments	<ul style="list-style-type: none"> <li>• The jump address stored via <a href="#">set_char_table</a> is used by the <a href="#">mark_date</a>, <a href="#">mark_serial</a> and <a href="#">mark_time</a> and the corresponding command lists are started. If no jump addresses have been stored, then these three commands have no effect.</li> <li>• The command lists for the digits define the character set. The command lists for the months and days-of-the-week also determine their specific written form (e.g. "Jan."..."Dec.", "Su."..."Sa.", "/1/" etc.).</li> <li>• Character sets can be defined separately for date/time marking and for serial number marking.</li> <li>• The command lists also specify the text direction (along the x or y axes).</li> <li>• The command lists can only contain <b>relative</b> vector commands (<a href="#">jump_rel</a>, <a href="#">mark_rel</a>, <a href="#">arc_rel</a>) and must be terminated with <a href="#">list_return</a>. The end position of a character or character sequence must be selected as the start position of any character that may follow (also see <a href="#">page 51</a>).</li> <li>• The starting addresses of command lists can be queried via <a href="#">get_input_pointer</a>.</li> </ul>
References	<a href="#">mark_date</a> , <a href="#">mark_serial</a> , <a href="#">mark_time</a>

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 16.																								
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see page 14).																								
Parameter	<div>control_mode (unsigned 16-bit value):</div> <table><thead><tr><th>Bit #</th><th>Value</th><th>Description</th></tr></thead><tbody><tr><td rowspan="2">Bit #0 (LSB)</td><td>= 1:</td><td>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 100). The external stop signal corresponds to the command stop_execution.</td></tr><tr><td>= 0:</td><td>no external start signal</td></tr><tr><td>Bit #1</td><td></td><td>not used</td></tr><tr><td rowspan="2">Bit #2</td><td>= 1:</td><td>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).</td></tr><tr><td>= 0:</td><td>No effect. To turn on the external start delay, use the command set_ext_start_delay or simulate_ext_start</td></tr><tr><td rowspan="2">Bit #3</td><td>= 1:</td><td>The external start input is not disabled by an external stop request.</td></tr><tr><td>= 0:</td><td>The external start input is disabled by an external stop request.</td></tr><tr><td>Bits #4 ... #15</td><td></td><td>not used</td></tr></tbody></table>	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 100). 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
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 100). 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	<div>Pascal:    procedure set_control_mode(control_mode: word);</div> <div>C:         void set_control_mode(unsigned short control_mode);</div> <div>Basic:     sub set_control_mode(ByVal control_mode%)</div>																								
Comments	<ul style="list-style-type: none"><li>The command set_control_mode resets the counter for external list starts to zero.</li><li>If execution is aborted by the command stop_execution, bit #0 is reset to zero, i.e. external start inputs are disabled.</li></ul>																								
References	select_list, get_counts, set_max_counts																								

List Command	set_control_mode_list
Function	similar to <b>set_control_mode</b> (see above), but a list command
Use	for use in PC and standalone operation
Integration	<p>Pascal:    <code>procedure set_control_mode_list(control_mode: word);</code></p> <p>C:         <code>void set_control_mode_list(unsigned short control_mode);</code></p> <p>Basic:     <code>sub set_control_mode_list(ByVal control_mode%)</code></p>
Comments	<ul style="list-style-type: none"> <li>The counter for external list starts is <i>not</i> reset by this command.</li> </ul>
References	<b>set_control_mode</b>

Ctrl Command	set_delay_mode																				
Function	turns the variable polygon delay mode and the variable jump delay mode on or off																				
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).																				
Parameters	All parameters must be unsigned 16-bit values.																				
	<table><tr><th>Parameter</th><th>Allowed Values</th><th>Description</th></tr><tr><td>varpoly</td><td>&gt; 0 = 0</td><td>Enables the variable polygon delay mode. See <a href="#">page 24</a>. Disables the variable polygon delay mode. (This is the default setting.)</td></tr><tr><td>directmove3d</td><td>&gt; 0 = 0</td><td>This parameter effects only 3D-applications. The x-, y- and z-values are changed directly (linearly) to their end values during a jump. 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.</td></tr><tr><td>edgelevel</td><td>0 ... 65500 (1 bit equals 10 µs)</td><td>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 <a href="#">figure 9 on page 24</a>. <b>Note:</b> To disable this feature, set the edgelevel to 65500 (default value).</td></tr><tr><td>MinJumpDelay</td><td>0 ... 32500 (1 bit equals 10 µs)</td><td>Minimum jump delay for a jump vector of zero length. See <a href="#">figure 6 on page 21</a>.</td></tr><tr><td>JumpLengthLimit</td><td>0 ... 32500</td><td>Jump length limit in <i>bits</i>. If the jump vector is <i>shorter</i> than this value, the jump delay is varied as shown in <a href="#">figure 6 on page 21</a>. Otherwise the jump delay is constant. To disable the Variable Jump Delay mode, set the JumpLengthLimit to 0.</td></tr></table>	Parameter	Allowed Values	Description	varpoly	> 0 = 0	Enables the variable polygon delay mode. See <a href="#">page 24</a> . Disables the variable polygon delay mode. (This is the default setting.)	directmove3d	> 0 = 0	This parameter effects only 3D-applications. The x-, y- and z-values are changed directly (linearly) to their end values during a jump. 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 <a href="#">figure 9 on page 24</a> . <b>Note:</b> 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 <a href="#">figure 6 on page 21</a> .	JumpLengthLimit	0 ... 32500	Jump length limit in <i>bits</i> . If the jump vector is <i>shorter</i> than this value, the jump delay is varied as shown in <a href="#">figure 6 on page 21</a> . Otherwise the jump delay is constant. To disable the Variable Jump Delay mode, set the JumpLengthLimit to 0.		
Parameter	Allowed Values	Description																			
varpoly	> 0 = 0	Enables the variable polygon delay mode. See <a href="#">page 24</a> . Disables the variable polygon delay mode. (This is the default setting.)																			
directmove3d	> 0 = 0	This parameter effects only 3D-applications. The x-, y- and z-values are changed directly (linearly) to their end values during a jump. 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 <a href="#">figure 9 on page 24</a> . <b>Note:</b> 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 <a href="#">figure 6 on page 21</a> .																			
JumpLengthLimit	0 ... 32500	Jump length limit in <i>bits</i> . If the jump vector is <i>shorter</i> than this value, the jump delay is varied as shown in <a href="#">figure 6 on page 21</a> . 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	<a href="#">set_scanner_delays</a> , <a href="#">load_varpolydelay</a>																				

List Command	set_end_of_list
Function	closes the currently open list
Use	for PC and standalone operation
Integration	Pascal: procedure set_end_of_list;
	C: void set_end_of_list(void);
	Basic: sub set_end_of_list()
Comments	<ul style="list-style-type: none"> <li>A list can hold up to 500000 commands.</li> <li>Also see <a href="#">chapter 5.4 "Structured Programming"</a>, <a href="#">page 45</a>.</li> </ul>
References	<a href="#">set_start_list</a>

Ctrl Command	set_extstartpos
Function	defines the start position (list buffer address) of the list to be executed by the next external start signal
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANAlone (see <a href="#">page 14</a> ).
Parameters	position list pointer to the start address of the list [0 ... 999999]
Integration	Pascal: procedure set_extstartpos(position: longint);
	C: void set_extstartpos(long position);
	Basic: sub set_extstartpos(ByVal position&)
Comments	<ul style="list-style-type: none"> <li>The <b>set_extstartpos</b> command can be used, for example, while a list is being processed to "link" the currently running list to the next list.</li> <li>set_extstartpos_list(0) selects list 1, set_extstartpos_list(500000) selects list 2. However, any other position in the list buffer can be specified as well.</li> <li>The specified start address will be used for all subsequent external starts until a new address is specified either by <b>set_extstartpos</b>, <b>set_extstartpos_list</b> or <b>select_list</b>.</li> </ul>
References	<b>set_extstartpos_list</b> , <b>select_list</b> , <b>set_control_mode</b> , <b>simulate_ext_start</b> (see supplementary manual <b>Processing-On-The-Fly Software</b> )

List Command	set_extstartpos_list
Function	same as <b>set_extstartpos</b> , but a list command
Use	for PC or standalone operation
Parameters	position list pointer to the start address of the list [0 ... 999999]
Integration	Pascal: procedure set_extstartpos_list(position: longint);
	C: void set_extstartpos_list(long position);
	Basic: sub set_extstartpos_list(ByVal position&)
Comments	<ul style="list-style-type: none"> <li>This command is a list command. When used within a list, the command "links" the current list to the next list.</li> </ul>

Ctrl Command	set_firstpulse_killer
Function	defines the length of the FirstPulseKiller signal for a YAG laser
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANAlone (see <a href="#">page 14</a> ).
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	<ul style="list-style-type: none"> <li>The time base for the FirstPulseKiller signal is always 8 MHz (i.e. 1 bit equals 1/8 µs).</li> <li>In CO<sub>2</sub> mode, the command <b>set_firstpulse_killer</b> has no effect.</li> <li>The laser control mode has to be set via the command <b>set_laser_mode</b> (<a href="#">page 107</a>). Please refer to <a href="#">chapter 4.6 "Laser Control"</a>, <a href="#">page 35</a> for details.</li> <li>To set the Q-Switch pulse width and period, use the list command <b>set_laser_timing</b> (<a href="#">page 108</a>).</li> </ul>
References	<b>set_laser_mode</b> , <b>set_laser_timing</b>



List Command	<b>set_firstpulse_killer_list</b>
Function	same as <b>set_firstpulse_killer</b> (see above), but a list command
Use	for PC or standalone operation
Integration	Pascal: <code>procedure set_firstpulse_killer_list(fpk: word);</code>
	C: <code>void set_firstpulse_killer_list(unsigned short fpk);</code>
	Basic: <code>sub set_firstpulse_killer_list(ByVal fpk%)</code>

Ctrl Command	<b>set_input_pointer</b>
Function	sets the list input pointer to the specified address in the RTC® SCANalone list buffer. The next list command will be stored at this address. Also see <a href="#">chapter 5.4 "Structured Programming", page 45</a> .
Use	for PC operation only
Parameter	pointer list input pointer [0 ... 999999] as signed 32-bit value
Integration	Pascal: <code>procedure set_input_pointer(pointer: longint);</code>
	C: <code>void set_input_pointer(long pointer);</code>
	Basic: <code>sub set_input_pointer(ByVal pointer&amp;)</code>
Comments	<ul style="list-style-type: none"> <li>• This command can be used alternatively instead of the <b>set_start_list</b> commands.</li> <li>• 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.</li> </ul>
References	<a href="#">execute_at_pointer</a> , <a href="#">get_input_pointer</a>

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: $((IOvalue \text{ AND } mask\_1) = mask\_1) \text{ AND } (((\text{not } IOvalue) \text{ AND } mask\_0) = mask\_0)$ (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).
Use	for PC or standalone operation
Parameters	mask_1,      unsigned 16-bit masks mask_0, mask_set
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	<ul style="list-style-type: none"> <li>The command affects only those bits of the output port that are set (=1) in the parameter mask_set.</li> </ul>
Examples (Pascal)	<ul style="list-style-type: none"> <li>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 (DIGITAL_IN1...3) of the input port are LOW:  <code>set_io_cond_list(\$0001, \$000E, \$0010)</code></li> <li>always set bit #15 of the output port (and leave the other bits unchanged):  <code>set_io_cond_list(0, 0, \$8000)</code></li> </ul>
References	<a href="#">clear_io_cond_list</a> , <a href="#">get_io_status</a>

List Command	set_jump_speed
Function	defines the jump speed for the following vector commands
Use	for PC or standalone operation
Parameter	jump_speed      jump speed in <i>bits per ms</i> (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	<ul style="list-style-type: none"> <li>The command is written directly into the list.</li> <li>By default a jump speed of 100 <i>bits per ms</i> is preset.</li> <li>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.</li> <li>To obtain the actual jump speed <math>v</math> in the image plane (in <i>meters per second</i>), the specified speed value (in <i>bits per ms</i>) must be divided by the calibration factor <math>K</math> of the correction file (in <i>bits per mm</i>):  <math display="block">v_{jump} = \text{jump\_speed} / K</math> </li> </ul>
References	<a href="#">jump_abs</a> , <a href="#">jump_rel</a> , <a href="#">set_mark_speed</a>

List Command	set_laser_delays
Function	sets the LaserOn delay and the LaserOff delay. 1 bit equals 1 $\mu$ s.
Use	for PC or standalone operation
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	<ul style="list-style-type: none"> <li>See the section "Laser Delays" on page 19, for details.</li> <li>The LaserOff delay must always be longer than the LaserOn delay. See "Limits For The Delays" on page 27.</li> <li>If the LaserOn delay is negative, the total marking time is extended.</li> </ul>
References	set_scanner_delays

Ctrl Command	set_laser_mode
Function	selects the laser control mode of the RTC® SCANAlone
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANAlone (see page 14).
Parameter	mode = 0: CO <sub>2</sub> 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	<ul style="list-style-type: none"> <li>The available laser signals depend on the selected laser control mode. Please refer to chapter 4.6 "Laser Control", page 35 for a detailed description.</li> <li>The command should be used only once at the program start. Also see chapter 9.3 "Initializing the RTC® SCANAlone", page 67.</li> </ul>
References	set_laser_timing, set_firstpulse_killer

List Command	<b>set_laser_timing</b>
Function	defines the output period and the pulse widths for the laser signals LASER1 and LASER2
Use	for PC and standalone operation
Parameters	<b>half_period</b> half of the output period in <i>bits</i> . 1 bit equals 1/8 $\mu$ s or 1 $\mu$ s, depending on the selected time base. Allowed range: [2 ... 65500]
	<b>pulse_width1, pulse_width2</b> Pulse widths of the laser signals LASER1 and LASER2 in <i>bits</i> . 1 bit equals 1/8 $\mu$ s or 1 $\mu$ s, depending on the selected time base. Allowed range: [2 ... 65500]
	<b>time_base</b> = 0: sets the time base to 1 MHz (1 bit equals 1 $\mu$ s) $\neq$ 0: sets the time base to 8 MHz (1 bit equals 1/8 $\mu$ s)
Integration	<b>Pascal:</b> procedure set_laser_timing(half_period, pulse_width1, pulse_width2, time_base: word);
	<b>C:</b> void set_laser_timing(unsigned short half_period, unsigned short pulse_width1, unsigned short pulse_width2, unsigned short time_base);
	<b>Basic:</b> sub set_laser_timing(ByVal half_period%, ByVal pulse_width1%, ByVal pulse_width2%, ByVal time_base%)
Comments	<ul style="list-style-type: none"> <li>• The time base setting applies only for the parameters of this command.</li> <li>• 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.</li> <li>• Please refer to <a href="#">chapter 4.6 "Laser Control", page 35</a>, for details.</li> </ul>
References	<a href="#">set_laser_mode</a> , <a href="#">set_firstpulse_killer</a> , <a href="#">set_standby</a> , <a href="#">set_standby_list</a>

List Command	<b>set_list_jump</b>
Function	defines a jump to the specified list buffer address
use	for PC or standalone operation
Parameter	<b>address</b> jump address [0 ... 999999] as signed 32-bit value
Integration	<b>Pascal:</b> procedure set_list_jump(address: longint);
	<b>C:</b> void set_list_jump(long address);
	<b>Basic:</b> sub set_list_jump(ByVal address&)
Comments	<ul style="list-style-type: none"> <li>• See <a href="#">chapter 5.4 "Structured Programming", page 45</a>.</li> </ul>
References	<a href="#">list_call</a>

List Command	set_mark_speed
Function	defines the marking speed for the subsequent vector and arc commands
Use	for PC or standalone operation
Parameter	mark_speed      marking speed in <i>bits per ms</i> (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	<ul style="list-style-type: none"> <li>The command is written directly into the list.</li> <li>By default a marking speed of 100 <i>bits per ms</i> is preset.</li> <li>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.</li> <li>To obtain the actual marking speed <math>v</math> in the image plane (in <i>meters per second</i>), the specified speed value (in <i>bits per ms</i>) must be divided by the calibration factor <math>K</math> of the correction file (in <i>bits per mm</i>): <math display="block">v_{mark} = \text{mark\_speed} / K</math> </li> </ul>
References	<a href="#">mark_abs</a> , <a href="#">mark_rel</a> , <a href="#">set_jump_speed</a>

Ctrl Command	set_matrix
Function	defines a (2 x 2) transformation matrix which will be used for all subsequent vector outputs.
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
Parameters	m11, m12,      Matrix coefficients as 64-bit IEEE floating point values m21, m22      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	<ul style="list-style-type: none"> <li>See <a href="#">chapter 5.1 "Coordinate Transformations"</a>, <a href="#">page 42</a>.</li> </ul>
References	<a href="#">set_matrix_list</a> , <a href="#">set_offset</a> , <a href="#">set_offset_list</a>

List Command	<b>set_matrix_list</b>
Function	sets <i>one</i> of the four coefficients of the (2 x 2) transformation matrix during execution of a list.
Use	for PC or standalone operation
Parameters	<div> <div>i, j</div> <div>Row index and column index [1 or 2] of the matrix coefficient to be changed</div> </div> <div> <div>m_ij</div> <div>Matrix coefficient as a 64-bit IEEE floating point value Allowed range: [-100 ... 100]</div> </div>
Integration	<div>Pascal: <code>procedure set_matrix_list(i, j: word; m_ij: double);</code></div> <div>C: <code>void set_matrix_list(unsigned short i, unsigned short j, double m_ij);</code></div> <div>Basic: <code>sub set_matrix_list(ByVal i%, ByVal j%, ByVal m_ij#)</code></div>
Comments	<ul style="list-style-type: none"> <li>The command <b>set_matrix_list</b> only allows changing <i>one</i> of the four coefficients at a time. To change several coefficients during execution of a list, the command has to be called repeatedly.</li> <li>See <a href="#">chapter 5.1 "Coordinate Transformations"</a>, page 42.</li> </ul>
References	<b>set_matrix</b> , <b>set_offset</b> , <b>set_offset_list</b>

Ctrl Command	<b>set_max_counts</b>
Function	defines the maximum number of external list starts
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
Parameter	<div> <div>max_counts</div> <div>maximum number of external list starts as a signed 32-bit value. Allowed range: <math>0 \leq \text{max\_counts} \leq 147483647</math></div> </div>
Integration	<div>Pascal: <code>procedure set_max_counts(max_counts: longint);</code></div> <div>C: <code>void set_max_counts(long max_counts);</code></div> <div>Basic: <code>sub set_max_counts(ByVal max_counts&amp;)</code></div>
Comments	<ul style="list-style-type: none"> <li>If the parameter <i>max_counts</i> is set to 0, the maximum number of external start signals is unlimited.</li> <li>After a reset of the RTC® SCANalone the parameter <i>max_counts</i> is set to 0.</li> <li>The counter can be read with the command <b>get_counts</b>.</li> <li>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.</li> <li>To reset the counter, call the command <b>set_control_mode</b> (see <a href="#">page 102</a>).</li> </ul>
References	<b>get_counts</b> , <b>set_control_mode</b>

Ctrl Command	set_offset
Function	defines an offset in the X and Y directions which will be added to all subsequent vector outputs.
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
Parameters	x_offset, offset in <i>bits</i> y_offset 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	<ul style="list-style-type: none"> <li>See <a href="#">chapter 5.1 "Coordinate Transformations"</a>, <a href="#">page 42</a>.</li> </ul>
References	<a href="#">set_matrix</a> , <a href="#">set_offset_list</a>

List Command	set_offset_list
Function	same as <a href="#">set_offset</a> (see above), but a list command
Use	for PC or standalone operation
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	<ul style="list-style-type: none"> <li>This command allows definition of an offset within a list.</li> <li>See <a href="#">chapter 5.1 "Coordinate Transformations"</a>, <a href="#">page 42</a>.</li> </ul>
References	<a href="#">set_matrix_list</a>

Ctrl Command	set_piso_control
Function	changes the delay which is inserted before the RTC® SCANalone reads the Status Signal Word from the scan head
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> ).
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	<ul style="list-style-type: none"> <li>This command provides adjustment of the timing of the (bi-directional) communication between the scan head and the RTC® SCANalone to reflect the length of the data cable.</li> <li>The command <i>should</i> 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 <a href="#">"Data Cable"</a>, <a href="#">page 56</a>.</li> <li>The maximum length for the data cable is 75 m.</li> </ul>
References	<a href="#">get_head_status</a>

List Command	set_pixel
Function	defines the output parameters (laser pulse width and ANALOG OUT2 value) for each individual pixel in an image line (also see <a href="#">chapter 5.5 "Scanning Raster Images (Bitmaps)"</a> , <a href="#">page 47</a> ).
Use	for PC and standalone operation
Parameters	pulse_width    laser output pulse width (unsigned 16-bit value). <b>1 bit equals 1/8 µs.</b>
	da_value       output value for the ANALOG OUT2 port of the RTC® SCANalone (10-bit resolution) (unsigned 16-bit value; the upper 6 bits are ignored)
	ad_channel     unsigned 16-bit dummy value (e.g. 0)
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	<ul style="list-style-type: none"> <li>Each image line must start with the command <b>set_pixel_line</b>.</li> <li>Each pixel in the image line is defined by one <b>set_pixel</b> command. The <b>set_pixel</b> commands must follow immediately after the command <b>set_pixel_line</b>. No other commands must be written into the list until the image line is completed.</li> </ul>
References	<b>set_pixel_line</b>



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 <a href="#">chapter 5.5 "Scanning Raster Images (Bitmaps)", page 47</a> .
Use	for PC or standalone operation
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 <a href="#">chapter 5.5, page 47</a> .
	pixel_period   pixel output period (unsigned 16-bit value). <b>1 bit equals 10 µs.</b>
	dx, dy         distance in the X and Y directions between adjacent pixels in <i>bits</i> (64-bit IEEE floating point values)
Integration	Pascal:     procedure set_pixel_line(pixel_mode: 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	<ul style="list-style-type: none"> <li>Each image line must start with the command <b>set_pixel_line</b>. This command should be preceded by a jump command to the start point of the image line.</li> <li>The command <b>set_pixel_line</b> turns on the pixel output mode of the RTC® SCANalone. The individual pixels of the image line are defined by <b>set_pixel</b> commands. The <b>set_pixel</b> commands must follow immediately after the command <b>set_pixel_line</b>.</li> <li>The first subsequent command in the list which is <i>not</i> a <b>set_pixel</b> command turns off the pixel output mode.</li> <li>Each <b>set_pixel_line</b> command requires <b>two</b> list entries in the list buffer.</li> </ul>
References	<a href="#">set_pixel</a>

List Command	set_scanner_delays
Function	sets the scanner delays
Use	for PC or standalone operation
Parameters	jump_delay,     unsigned 16-bit values. Allowed range: [0...32767]; <b>1 bit equals 10 µs.</b> mark_delay, polygon_delay
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	<ul style="list-style-type: none"> <li>See <a href="#">"Scanner Delays" on page 20</a></li> <li>and <a href="#">"Limits For The Delays" on page 27</a>.</li> </ul>
References	<a href="#">set_delay_mode</a> , <a href="#">set_laser_delays</a>

Ctrl Command	set_serial
Function	sets the serial number that will be marked via the next call of <b>mark_serial</b> .
Use	for PC operation only; the command can also be used for standalone initialization of the RTC <sup>®</sup> SCANalone (see <a href="#">page 14</a> ).
Parameters	no            serial number as signed 32-bit value (the sign is ignored)
Integration	Pascal:    procedure set_serial(no: longint);
	C:           void set_serial(long no);
	Basic:     sub set_serial(ByVal no&)
Comments	<ul style="list-style-type: none"> <li>After the RTC<sup>®</sup> SCANalone Board boots with an installed MMC, the serial number defined via <b>set_serial</b> will be read from the MMC and applied. If the MMC's content is not updated following a power interruption, then already-marked serial numbers will be marked again</li> </ul>
References	<b>mark_serial</b>

Ctrl Command	set_softstart_level		
Function	sets the softstart values.		
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> )		
Parameters	All parameters must be unsigned 16-bit values.		
	Parameter	Allowed Values	Description
	index	0 ... <i>number</i>	index number of the softstart table value
	level	0 ... 1023	for softstart mode = 1, 2, 11 and 12 (see <a href="#">set_softstart_mode</a> ). Value 0 corresponds to an analog voltage value of 0 V; value 1023 corresponds to 2.56 V (for jumper X1 position 1-2) or 10 V (for jumper X1 position 2-3).
		2 ... 2 <sup>16</sup> -1	for softstart mode = 3 and 13 (see <a href="#">set_softstart_mode</a> ). 1 bit corresponds to a pulse width of 1 μs or 1/8 μs depending on the time base which can be specified with the command <a href="#">set_laser_timing</a> (also see <a href="#">page 35</a> ).
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	<ul style="list-style-type: none"><li>• The <a href="#">set_softstart_level</a> command defines the level value for the pulse number index. The command must be called separately for each individual level value.</li><li>• index must be in the range [0...1000], otherwise the <a href="#">set_softstart_level</a> command will not be executed.</li><li>• If index &gt; number (number is defined via <a href="#">set_softstart_mode</a>) then <a href="#">set_softstart_level</a> will be executed but the defined value will not be used during softstart.</li><li>• Please be sure to set as many softstart values as you defined by <a href="#">set_softstart_mode</a> (unspecified softstart values have undefined contents).</li><li>• The softstart mode is enabled by the command <a href="#">set_softstart_mode</a>.</li><li>• For softstart modes 1, 2, 11 or 12, level is restricted to 10 bits; higher bits will be ignored.</li><li>• If Softstart mode = 3 or 13 and the specified softstart pulse length is larger than the laser signal period duration specified via <a href="#">set_laser_timing</a> (2 · half_period), then the laser will remain continuously on.</li><li>• Particularly if softstart values are to be output as analog voltage values (softstart mode = 1, 2, 11 or 12), the <a href="#">set_softstart_mode</a> command must be called <i>before</i> first-time use of the <a href="#">set_softstart_level</a> command; otherwise the specified value will not be correctly converted to an analog value.</li><li>• Also observe the note in section "<a href="#">Softstart Mode</a>" on <a href="#">page 37</a>.</li></ul>		
References	<a href="#">set_softstart_mode</a>		

Ctrl Command	set_softstart_mode																								
Function	switches softstart mode on or off.																								
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> )																								
Parameters	All parameters are unsigned 16-bit values.																								
	<table><thead><tr><th>Parameter</th><th>Allowed Values</th><th>Description</th></tr></thead><tbody><tr><td rowspan="6">mode</td><td>= 0</td><td>disables the softstart mode, but does not remove previously loaded softstart values.</td></tr><tr><td>= 1, 11</td><td>the softstart values defined via <a href="#">set_softstart_level</a> will be output as analog voltage values at the ANALOG OUT1 port.</td></tr><tr><td>= 2, 12</td><td>the softstart values defined via <a href="#">set_softstart_level</a> will be output as analog voltage values at the ANALOG OUT2 port.</td></tr><tr><td>= 3, 13</td><td>the softstart values defined via <a href="#">set_softstart_level</a> specify the pulsewidths of the first number+1 laser signal pulses at the LASER1 port.</td></tr><tr><td>= 1 ... 3</td><td>The change from one Level to the next occurs with the trailing edge of the laser pulse.</td></tr><tr><td>= 11 ... 13</td><td>The change from one Level to the next occurs with the leading edge of the laser pulse.</td></tr><tr><td>number</td><td>1 ... 1000</td><td>(number+1) is the number of softstart values to be transmitted.</td></tr><tr><td>restartdelay</td><td>0 ... 65534 (1 bit equals 10 µs)</td><td>defines the time period for which the laser must have been switched off before softstart will be activated at the next switch-on.</td></tr></tbody></table>	Parameter	Allowed Values	Description	mode	= 0	disables the softstart mode, but does not remove previously loaded softstart values.	= 1, 11	the softstart values defined via <a href="#">set_softstart_level</a> will be output as analog voltage values at the ANALOG OUT1 port.	= 2, 12	the softstart values defined via <a href="#">set_softstart_level</a> will be output as analog voltage values at the ANALOG OUT2 port.	= 3, 13	the softstart values defined via <a href="#">set_softstart_level</a> specify the pulsewidths of the first number+1 laser signal pulses at the LASER1 port.	= 1 ... 3	The change from one Level to the next occurs with the trailing edge of the laser pulse.	= 11 ... 13	The change from one Level to the next occurs with the leading 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 time period for which the laser must have been switched off before softstart will be activated at the next switch-on.		
Parameter	Allowed Values	Description																							
mode	= 0	disables the softstart mode, but does not remove previously loaded softstart values.																							
	= 1, 11	the softstart values defined via <a href="#">set_softstart_level</a> will be output as analog voltage values at the ANALOG OUT1 port.																							
	= 2, 12	the softstart values defined via <a href="#">set_softstart_level</a> will be output as analog voltage values at the ANALOG OUT2 port.																							
	= 3, 13	the softstart values defined via <a href="#">set_softstart_level</a> specify the pulsewidths of the first number+1 laser signal pulses at the LASER1 port.																							
	= 1 ... 3	The change from one Level to the next occurs with the trailing edge of the laser pulse.																							
	= 11 ... 13	The change from one Level to the next occurs with the leading 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 time period for which the laser must have been switched off before softstart will be activated at the next switch-on.																							
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	<ul style="list-style-type: none"><li>• The <a href="#">set_softstart_mode</a> command must be called <i>before</i> first-time use of the <a href="#">set_softstart_level</a> command, particularly if softstart values are to be output as analog voltage values.</li><li>• The individual softstart values level(0) ... level(number) must be provided individually via separate calls of <a href="#">set_softstart_level</a>. Indices for which no level value was supplied are undefined.</li><li>• When softstart mode is switched off, the already transmitted values are not deleted and can be further used after a renewed switch-on. Here, you shouldn't unintentionally switch between analog voltage values (Mode = 1, 2, 11 or 12) and pulsewidths (Mode = 3 or 13). In contrast, you can switch at any time between ANALOG OUT1 (Mode = 1 or 11) and ANALOG OUT2 (Mode = 2 or 12).</li><li>• The pulse width or power of the first laser pulse is determined by level(1) if the values are assigned with the <i>leading</i> edge of the laser pulse selected, or by level(0) if the <i>trailing</i> edge of the laser pulse is selected.</li><li>• Observe the notes in the section "<a href="#">Softstart Mode</a>" on <a href="#">page 37</a>.</li></ul>																								
References	<a href="#">set_softstart_level</a>																								

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.
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANAlone (see <a href="#">page 14</a> )
Parameters	half_period <i>half</i> of the stand-by output period in <i>bits</i> . <b>1 bit equals 1/8 µs</b> . Allowed range: [0 ... 65500]
	pulse_width     Pulse width of the stand-by pulses in <i>bits</i> . <b>1 bit equals 1/8 µs</b> . Allowed range: [0 ... half_period] * <ul style="list-style-type: none"> <li>If a value larger than half_period is specified, the RTC® SCANAlone driver sets the pulse_width to half_period.</li> </ul>
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	<ul style="list-style-type: none"> <li>The time base for the stand-by pulses is always 8 MHz (i.e. 1 bit equals 1/8 µs).</li> <li>The stand-by pulses are available in <b>all</b> laser modes (YAG 1/2/3, CO<sub>2</sub> and laser mode 4). They can be turned off by setting the stand-by pulse width to zero (default).</li> <li>The laser control mode has to be set with the command <a href="#">set_laser_mode</a> (<a href="#">page 107</a>). Please refer to <a href="#">chapter 4.6 "Laser Control"</a>, <a href="#">page 35</a>, for details.</li> <li>To set the active output period and pulse width for the two laser signals, use the list command <a href="#">set_laser_timing</a> (see <a href="#">page 108</a>).</li> </ul>
References	<a href="#">set_standby_list</a> , <a href="#">set_laser_mode</a> , <a href="#">set_laser_timing</a>

List Command	set_standby_list
Function	same as <a href="#">set_standby</a> (see above), but a list command
Use	for PC or standalone operation
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 buffer, or half of it, for writing a list (list 1 or list 2). All subsequent list commands are stored in the corresponding list.
Use	for PC operation only
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	<ul style="list-style-type: none"> <li>The commands <a href="#">set_start_list_1</a> and <a href="#">set_start_list_2</a> (with no parameter) can be used alternatively.</li> </ul>
References	<a href="#">execute_list</a> , <a href="#">read_status</a>

List Command	<b>set_trigger</b>
Function	starts measurement and storage of the specified signals
Use	for PC operation only
Parameters	<p>sample-period      measurement period <math>[(1..65535) \cdot 10 \mu s]</math> as an unsigned 16-bit value</p> <p>signal1, signal2      The following signals (each represented by an unsigned 16-bit value) are specifiable for measurement channels 1 and 2:</p> <ul style="list-style-type: none"> <li>= 0:      LASERON signal (1 = laser on, 0 = laser off)</li> <li>= 1:      StatusAX (X-axis status channel of head A)</li> <li>= 2:      StatusAY (Y-axis status channel of head A)</li> <li>= 3:      StatusAZ (Z-axis status channel of head A)</li> <li>= 4:      StatusBX (X-axis status channel of head B)</li> <li>= 5:      StatusBY (Y-axis status channel of head B)</li> <li>= 6:      StatusBZ (Z-axis status channel of head B)</li> <li>= 7:      SampleX (X-axis cartesian output value)</li> <li>= 8:      SampleY (Y-axis cartesian output value)</li> <li>= 9:      SampleZ (Z-axis cartesian output value)</li> <li>= 10:      SampleAX_Corr (X-axis output value of head A)</li> <li>= 11:      SampleAY_Corr (Y-axis output value of head A)</li> <li>= 12:      SampleAZ_Corr (Z-axis output value of head A)</li> <li>= 13:      SampleBX_Corr (X-axis output value of head B)</li> <li>= 14:      SampleBY_Corr (Y-axis output value of head B)</li> <li>= 15:      SampleBZ_Corr (Z-axis output value of head B)</li> </ul>
Integration	<p>Pascal:      procedure set_trigger(sampleperiod, signal1, signal2: word);</p> <p>C:              void set_trigger(unsigned short sampleperiod, unsigned short signal1, unsigned short signal2);</p> <p>Basic:        sub set_trigger(ByVal sampleperiod As Integer, ByVal signal1 As Integer, ByVal signal2 As Integer)</p>
Comments	<ul style="list-style-type: none"> <li>The <b>set_trigger</b> command will always cause exactly 32768 values per measurement channel to be measured and stored onto the RTC® SCANalone. Afterward, <b>get_waveform</b> can be used to transfer the measured values to the PC for evaluation.</li> <li>The current status of a measurement session can be queried via the <b>measurement_status</b> command.</li> <li>Once started, a measurement session can only be canceled by again calling the <b>set_trigger</b> command. Previously measured values are thereby lost.</li> <li>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.</li> <li>For scan systems with only one status channel, the status signals are only readable if <math>signal1,2 = 1</math> or <math>signal1,2 = 4</math>.</li> <li>Coordinate transformations defined by <b>set_matrix</b>, <b>set_matrix_list</b>, <b>set_offset</b> or <b>set_offset_list</b> are already reflected in the SampleX, SampleY and SampleZ cartesian output values.</li> <li>The SampleAX_Corr..SampleBZ_Corr output values are the (digital) output values actually transmitted from the RTC® SCANalone to the scan system. The RTC® SCANalone computes these values while taking into account the SampleX, SampleY and SampleZ output values as well as the selected correction file.</li> </ul>
References	<b>get_waveform</b> , <b>measurement_status</b>

List Command	<b>set_wait</b>
Function	sets a wait marker (break point) in the list
Use	for PC operation only
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	<ul style="list-style-type: none"> <li>• Processing of a list will be interrupted at each wait marker. The laser will be turned off.</li> <li>• If processing has been stopped at a wait marker, the command <b>get_wait_status</b> returns the number of this marker.</li> <li>• Processing of the list can be resumed by calling the command <b>release_wait</b>.</li> </ul>
References	<b>get_wait_status</b> , <b>release_wait</b>

List Command	<b>set_wobble</b>
Function	defines a circular movement of the output position which is added to the regular marking movement
Use	for PC and standalone operation
Parameters	amplitude    amplitude of the circular wobble movement in <i>bits</i> (unsigned 16-bit value) Allowed range: [1 ... 5000]
	frequency    frequency of the wobble movement in <i>Hz</i> (circles per second, 64-bit IEEE floating point value) Allowed range: [1 ... 6000]
Integration	Pascal:    procedure set_wobble(amplitude: word; frequency: double);
	C:          void set_wobble(unsigned short amplitude, double frequency);
	Basic:     sub set_wobble(ByVal amplitude%, ByVal frequency#)
Comments	<ul style="list-style-type: none"> <li>• Calling the command set_wobble sets the wobble phase to a defined starting value (which is independent of the direction of the marking vector).</li> <li>• The wobble function can be used for marking lines with variable line width.</li> <li>• A circular movement is added to the linear marking movement, resulting in a spiral movement of the laser focus in the image field.</li> <li>• 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.</li> <li>• The wobble function is terminated by setting the amplitude to zero.</li> </ul>
References	<b>set_mark_speed</b>

Ctrl Command	start_loop
Function	starts an alternately repeating output of the two lists
Use	for PC operation only
Integration	Pascal:    procedure start_loop;
	C:           void start_loop(void);
	Basic:       sub start_loop()
Comments	<ul style="list-style-type: none"> <li>• The command <b>start_loop</b> can only be called if the two lists are loaded and closed, and if one of the lists is executing at the moment. See "<b>Repeating Output</b>" on page 15.</li> <li>• Both lists are alternately repeated until execution is terminated by calling the command <b>quit_loop</b>.</li> </ul>
References	<b>quit_loop</b>

Ctrl Command	stop_execution
Function	stops execution of the list and turns off the laser immediately
Use	for PC operation only
Integration	Pascal:    procedure stop_execution;
	C:           void stop_execution(void);
	Basic:       sub stop_execution()
Comments	<ul style="list-style-type: none"> <li>• The mirrors will stay in the current position.</li> <li>• The external START inputs are disabled by default. The Processing-on-the-fly correction is turned off (Processing-on-the-fly option only).</li> <li>• A list that was interrupted with <b>stop_execution</b> <i>cannot</i> be resumed. Use the command <b>stop_list</b> if execution is to be temporarily stopped and resumed later.</li> </ul>
References	<b>get_startstop_info</b>

Ctrl Command	stop_list
Function	pauses execution of the list and turns the laser off
	for PC operation only (if, when storing to the MMC, the setting for the <b>stop_list</b> command is also stored, then stored lists can no longer be started in standalone operation)
Integration	Pascal:    procedure stop_list;
	C:           void stop_list(void);
	Basic:       sub stop_list()
Comments	<ul style="list-style-type: none"> <li>• The command <b>restart_list</b> <i>has to be used</i> to resume execution of the list.</li> </ul>
References	<b>restart_list, stop_execution</b>



Ctrl Command	<b>store_on_mmc</b>						
Function	stores the current lists, correction files and settings onto the MMC						
Use	for PC operation only						
Result	signed 16-bit error code: <table> <tr> <th>value</th><th>description</th></tr> <tr> <td>0</td><td>success</td></tr> <tr> <td>≠0</td><td>an error occurred while storing to the MMC</td></tr> </table>	value	description	0	success	≠0	an error occurred while storing to the MMC
value	description						
0	success						
≠0	an error occurred while storing to the MMC						
Integration	Pascal:    function store_on_mmc: smallint; C:           short store_on_mmc(void); Basic:       function store_on_mmc()%						
Comments	<ul style="list-style-type: none"> <li>The <b>clear_list</b> command should be called before an application is loaded from the PC into the RTC® SCANalone's list buffer and subsequently stored onto the MMC via <b>store_on_mmc</b>. This command sequence prevents unneeded list entries (e.g. from prior test runs that might still remain in the list buffer) from being stored onto the MMC.</li> <li>Storing the complete list memory (with 1 000 000 list positions) onto the MMC can take up to 2.2 minutes (booting and loading the MMC content into the list buffers after an RTC® SCANalone reset can take up to 1.3 minutes).</li> </ul>						
References	<b>clear_list</b>						

List Command	<b>time_fix</b>
Function	stores the current time and data of the RTC® SCANalone clock/calender.
Use	for PC and standalone operation
Integration	Pascal:    procedure time_fix; C:           void time_fix(void); Basic:       sub time_fix()
Comments	<ul style="list-style-type: none"> <li>The <b>time_fix</b> command must be called before the current time and date can be marked via <b>mark_time</b> and <b>mark_date</b> (also see <a href="#">page 51</a>).</li> <li>Calibration of the RTC® SCANalone's calender and 24-hour clock is achieved by comparing with the PC time via the <b>time_update</b> command.</li> </ul>
References	<b>mark_date, mark_time, time_update</b>

Ctrl Command	<b>time_update</b>
Function	sets the RTC® SCANalone's time (clock and calender) according to the PC's time
Use	for PC operation only
Integration	Pascal:    procedure time_update; C:           void time_update(void); Basic:       sub time_update()
Comments	<ul style="list-style-type: none"> <li>Use of the RTC® SCANalone's clock/calender requires a battery (see <a href="#">page 59</a>).</li> </ul>
References	<b>time_fix, mark_date, mark_time</b>

List Command	<b>timed_jump_abs</b>
Function	jump to the specified position in the image field using the specified duration
Use	for PC or standalone operation
Parameters	xval, yval      absolute coordinates of the vector end point in <i>bits</i> as signed 16-bit values.
	time      duration of the complete jump vector in <i>microseconds</i> 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	<ul style="list-style-type: none"> <li>• The parameter <i>time</i> will be rounded down to a multiple of 10 <math>\mu</math>s (within the allowed range).</li> <li>• A <i>timed</i> 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 <a href="#">chapter 5.6 "Timed Jump And Mark Commands"</a>, page 51.</li> <li>• <b>Note:</b> After a timed jump, a <i>jump delay</i> is inserted, just like after a normal jump. That means the total time for the jump is the specified time <i>plus</i> the jump delay.</li> <li>• Subsequent <b>jump_abs</b> / <b>jump_rel</b> commands (not timed) will be executed with the normal jump speed. (See <a href="#">set_jump_speed</a>, page 106.)</li> </ul>
References	<a href="#">set_scanner_delays</a> , <a href="#">timed_jump_rel</a>

List Command	<b>timed_jump_rel</b>
Function	jump to the specified position in the image field using the specified duration
Use	for PC or standalone operation
Parameters	dx, dy <i>relative</i> coordinates of the vector end point in <i>bits</i> as signed 16-bit values.
	time      duration of the complete jump vector in <i>microseconds</i> 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	<ul style="list-style-type: none"> <li>• The coordinates are relative to the current position.</li> <li>• The maximum value for the <i>absolute</i> coordinates is <math>\pm 32767</math> bits.</li> <li>• See <a href="#">timed_jump_abs</a>.</li> </ul>
References	<a href="#">set_scanner_delays</a> , <a href="#">timed_jump_abs</a>

List Command	timed_mark_abs
Function	marking vector to the specified position in the image field using the specified duration
Use	for PC or standalone operation
Parameters	xval, yval      absolute coordinates of the vector end point in <i>bits</i> as signed 16-bit values.
	time      duration of the complete mark vector in <i>microseconds</i> 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	<ul style="list-style-type: none"> <li>The parameter <i>time</i> will be rounded down to a multiple of 10 <math>\mu</math>s (within the allowed range).</li> <li>A <i>timed</i> 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 <a href="#">chapter 5.6 "Timed Jump And Mark Commands"</a>, page 51.</li> <li><b>Note:</b> After a timed mark command, a <i>mark delay</i> or a <i>polygon delay</i> is inserted, just like after a normal mark command. That means the total time for the command is the specified time <i>plus</i> the mark delay or polygon delay.</li> <li>Subsequent <b>mark_abs</b> / <b>mark_rel</b> commands (not timed) will be executed with the normal marking speed. (See <a href="#">set_mark_speed</a>, page 109.)</li> </ul>
References	<a href="#">set_scanner_delays</a> , <a href="#">timed_mark_rel</a>

List Command	timed_mark_rel
Function	marking vector to the specified position in the image field using the specified duration
Use	for PC or standalone operation
Parameters	dx, dy <i>relative</i> coordinates of the vector end point in <i>bits</i> as signed 16-bit values.
	time      duration of the complete mark vector in <i>microseconds</i> 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	<ul style="list-style-type: none"> <li>The coordinates are relative to the current position.</li> <li>The maximum value for the <i>absolute</i> coordinates is <math>\pm 32767</math> bits.</li> <li>See <a href="#">timed_mark_abs</a>.</li> </ul>
References	<a href="#">set_scanner_delays</a> , <a href="#">timed_mark_abs</a>

Ctrl Command	usb_status						
Function	verifies correct functioning of command transfer between PC and the RTC® SCANalone						
Use	for PC operation only						
Result	signed 32-bit error code: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>0</td><td>command transfer is working properly</td></tr> <tr> <td>≠0</td><td>command transfer between PC and RTC® SCANalone is not working properly</td></tr> </table>	Value	Description	0	command transfer is working properly	≠0	command transfer between PC and RTC® SCANalone is not working properly
Value	Description						
0	command transfer is working properly						
≠0	command transfer between PC and RTC® SCANalone is not working properly						
Integration	Pascal:    function usb_status: longint; C:           long usb_status(void); Basic:       function usb_status() &						
Comments	<ul style="list-style-type: none"> <li>If a non-zero error code is returned, then the USB connection between the PC and the RTC® SCANalone should be checked.</li> </ul>						

Ctrl Command	write_8bit_port
Function	writes an 8-bit value to the digital output port on the LASER EXTENSION connector
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> )
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	<ul style="list-style-type: none"> <li>Please refer to "Digital Output Port", <a href="#">page 55</a>, and to the section "LASER EXTENSION/Digital Output Port", <a href="#">page 63</a>.</li> <li>The upper 8 bits of the specified value are ignored.</li> </ul>
References	<a href="#">write_8bit_port_list</a>

List Command	write_8bit_port_list
Function	same as <a href="#">write_8bit_port</a> (see above), but a list command
Use	for PC and standalone operation
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 <a href="#">write_da_x</a>
Integration	Pascal:    procedure write_da_1(value: word); C:           void write_da_1(unsigned short value); Basic:       sub write_da_1(ByVal value%)

<b>List Command</b>	<b>write_da_1_list</b>
<b>Function</b>	see <b>write_da_x_list</b>
<b>Integration</b>	Pascal: <code>procedure write_da_1_list(value: word);</code>
	C: <code>void write_da_1_list(unsigned short value);</code>
	Basic: <code>sub write_da_1_list(ByteVal value%)</code>

<b>Ctrl Command</b>	<b>write_da_2</b>
<b>Function</b>	see <b>write_da_x</b>
<b>Integration</b>	Pascal: <code>procedure write_da_2(value: word);</code>
	C: <code>void write_da_2(unsigned short value);</code>
	Basic: <code>sub write_da_2(ByteVal value%)</code>

<b>List Command</b>	<b>write_da_2_list</b>
<b>Function</b>	see <b>write_da_x_list</b>
<b>Integration</b>	Pascal: <code>procedure write_da_2_list(value: word);</code>
	C: <code>void write_da_2_list(unsigned short value);</code>
	Basic: <code>sub write_da_2_list(ByteVal value%)</code>

<b>Ctrl Command</b>	<b>write_da_x</b>
<b>Function</b>	writes a 10-bit value to one of the analog output ports of the RTC <sup>®</sup> SCANalone
<b>Use</b>	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC <sup>®</sup> SCANalone (see <b>page 14</b> )
<b>Parameters</b>	x            [1, 2]    ANALOG OUT1 or ANALOG OUT2 port on the RTC <sup>®</sup> SCANalone board
	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.
<b>Integration</b>	Pascal: <code>procedure write_da_x(x, value: word);</code>
	C: <code>void write_da_x(unsigned short x, unsigned short value);</code>
	Basic: <code>sub write_da_x(ByteVal x%, ByteVal value%)</code>
<b>Comments</b>	<ul style="list-style-type: none"> <li>• For the ANALOG OUT1 / ANALOG OUT2 ports, the commands <b>write_da_1</b> / <b>write_da_2</b> can be used alternatively (without parameter x).</li> <li>• Note that the output range of the ANALOG OUT1 port can be either 0 V ... 2.56 V or 0 V ... 10 V (see <b>page 62</b>), whereas the output range of the ANALOG OUT2 port is always 0 V ... 10 V.</li> </ul>

<b>List Command</b>	<b>write_da_x_list</b>
<b>Function</b>	same as <b>write_da_x</b> (see above), but a list command
<b>Use</b>	for PC or standalone operation
<b>Integration</b>	Pascal: <code>procedure write_da_x_list(x, value: word);</code>
	C: <code>void write_da_x_list(unsigned short x, unsigned short value);</code>
	Basic: <code>sub write_da_x_list(ByteVal x%, ByteVal value%)</code>

Ctrl Command	<b>write_io_port</b>
Function	writes a value to the 16-bit digital output port on the "EXTENSION 1" connector
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see <a href="#">page 14</a> )
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	<ul style="list-style-type: none"> <li>Use the commands <a href="#">set_io_cond_list</a> and <a href="#">clear_io_cond_list</a> to set/clear individual bits of the 16-bit digital output port, depending on the state of the <i>input</i> port.</li> </ul>
References	<a href="#">write_io_port_list</a> , <a href="#">read_io_port</a>

List Command	<b>write_io_port_list</b>
Function	same as <a href="#">write_io_port</a> , but a list command
Use	for PC or standalone operation
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	<a href="#">write_io_port</a>

Ctrl Command	<b>z_out</b>
Function	sends a 16-bit value directly to the Z channel of the RTC® SCANalone (CHAN3; see <a href="#">figure 29</a> or <a href="#">figure 29 on page 56</a> )
Use	for PC operation only
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	<ul style="list-style-type: none"> <li>The output value is sent directly to the Z channel of the RTC® SCANalone.</li> <li>The Z output value must be in the range –32768 ... +32767 (signed 16-bit value).</li> <li><b>Note:</b> This command should only be used in 2D applications. In 3D applications which use the program file RTC4D3.HEX, the RTC® SCANalone will overwrite the Z output value every 10 µs.</li> </ul> <p>Also see the supplement manual "<a href="#">3D Software</a>".</p>
References	<a href="#">z_out_list</a>

List Command	z_out_list
Function	same as <b>z_out</b> , but a list command
Use	for PC and standalone operation
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® Commands

To achieve compatibility with command sets of the RTC® SCANalone PC interface boards, the RTC® SCANalone emulates some commands not listed in the previous section.

Though these emulated commands can be used by the RTC® SCANalone, new application software should only use the commands described in the previous section.

A few RTC® commands are not supported by the RTC® SCANalone or shouldn't be used with it. Some of these RTC® commands can be replaced by RTC® SCANalone commands.

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

Ctrl Command	auto_cal
Support status	This RTC®3/RTC®4 command is not supported by the RTC® SCANalone.
Replaced by	<b>There is no equivalent RTC® SCANalone command.</b> automatic self-calibration is not supported by the RTC® SCANalone.

Ctrl Command	aut_change
Support status	The RTC® SCANalone emulates this RTC®2 command.
Replaced by	<b>auto_change</b> (see page 73)
Function	same as <b>auto_change</b> (see page 73)
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC® SCANalone (see page 14)
Integration	Pascal:     procedure aut_change;
	C:           void aut_change(void);
	Basic:      sub aut_change( )



Ctrl Command	<b>dsp_start</b>
Support status	This RTC <sup>®</sup> 3/RTC <sup>®</sup> 4 command is supported by the RTC <sup>®</sup> SCANalone, but should no longer be used.
Replaced by	<b>stop_execution</b> (see page 120)
Function	stops execution of a list and immediately switches off the laser
Use	for PC operation only
Integration	Pascal:     procedure dsp_start;
	C:           void dsp_start(void);
	Basic:      sub dsp_start()
Comments	<ul style="list-style-type: none"> <li>In PC operation, dsp_start works just like <b>stop_execution</b>. It does not produce a true reset of the RTC<sup>®</sup> SCANalone.</li> </ul>

List Command	<b>field_jump</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>home_position</b> (see page 84)

Ctrl Command	<b>get_hi_data</b>
Support status	This RTC <sup>®</sup> 3/RTC <sup>®</sup> 4 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>There is no equivalent RTC<sup>®</sup> SCANalone command.</b> Automatic self-calibration is not supported by the RTC <sup>®</sup> SCANalone.

Ctrl Command	<b>get_list_space</b>
Support status	This RTC <sup>®</sup> 3/RTC <sup>®</sup> 4 is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>There is no equivalent RTC<sup>®</sup> SCANalone command.</b> Circular queue mode is not supported by the RTC <sup>®</sup> SCANalone.

Ctrl Command	<b>get_rtc2_mode</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	The RTC <sup>®</sup> SCANalone has no need to query the mode, which is now set via the software command <b>set_laser_mode</b> (see page 107) instead of hardware.

Ctrl Command	<b>get_rtc2_version</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>get_rtc_version</b> (see page 79)

Ctrl Command	<b>get_version</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>get_hex_version</b> (see page 78)

List Command	<b>home_jump</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>home_position</b> (see page 84)

List Command	<b>laser_on</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>laser_on_list</b> (see page 86)

Ctrl Command	<b>load_cor</b> (load correction file)
Support status	The RTC <sup>®</sup> SCANalone emulates this RTC <sup>®</sup> 2 command.
Replaced by	<b>load_correction_file</b> (see page 90)
Function	loads the specified image field correction file into RTC <sup>®</sup> 3 memory. The file is used as correction file #1.
Use	for PC operation only; in PC operation, the command can also be used for standalone initialization of the RTC <sup>®</sup> SCANalone (see page 14)
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	<b>load_pro</b> (load program file)
Support status	The RTC <sup>®</sup> SCANalone emulates this RTC <sup>®</sup> 2 command. However, the command has no effect.
Replaced by	<b>load_program_file</b> (see page 131)
Function	The original function (loading a program file into the DSP) provided by this RTC <sup>®</sup> 2 command is not needed by the RTC <sup>®</sup> SCANalone.
Use	for PC operation only
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\$)%

<b>Ctrl Command</b>	<b>load_program_file</b>						
Support status	The RTC <sup>®</sup> SCANalone emulates this RTC <sup>®</sup> 3/RTC <sup>®</sup> 4 command. But the command only returns an error code and otherwise has no effects.						
Replaced by	not necessary, because the program file is permanently installed on the RTC <sup>®</sup> SCANalone						
Function	The original function (loading a program file into the DSP) of this RTC <sup>®</sup> 3/RTC <sup>®</sup> 4 command is not needed by the RTC <sup>®</sup> SCANalone.						
Use	for PC operation only						
Parameters	FileName    Name of the program file as a pointer to a null-terminated ANSI string						
Result	signed 16-bit error code: <table> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>0</td><td>OK</td></tr> <tr> <td>11</td><td>RTC<sup>®</sup> SCANalone not found</td></tr> </table>	Value	Description	0	OK	11	RTC <sup>®</sup> SCANalone not found
Value	Description						
0	OK						
11	RTC <sup>®</sup> SCANalone 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\$)%						

<b>List Command</b>	<b>pola_abs, polb_abs, polc_abs</b>
Support status	These RTC <sup>®</sup> 2 commands are not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>mark_abs</b> (see page 93)

<b>Ctrl Command</b>	<b>read_pixel_ad</b>
Support status	This RTC <sup>®</sup> 3/RTC <sup>®</sup> 4 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>There is no equivalent RTC<sup>®</sup> SCANalone command.</b> RTC <sup>®</sup> SCANalone boards cannot be used with I/O boards.

<b>Ctrl Command</b>	<b>rtc3_count_cards</b>
Support status	This RTC <sup>®</sup> 3 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>There is no equivalent RTC<sup>®</sup> SCANalone command.</b> RTC <sup>®</sup> SCANalone boards are not usable as multi-boards.

<b>Ctrl Command</b>	<b>rtc4_count_cards</b>
Support status	This RTC <sup>®</sup> 4 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>There is no equivalent RTC<sup>®</sup> SCANalone command.</b> RTC <sup>®</sup> SCANalone boards are not usable as multi-boards.

<b>Ctrl Command</b>	<b>select_rtc</b>
Support status	This RTC <sup>®</sup> 3/RTC <sup>®</sup> 4 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>There is no equivalent RTC<sup>®</sup> SCANalone command.</b> RTC <sup>®</sup> SCANalone boards are not usable as multi-boards.

Ctrl Command	<b>set_base</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>There is no equivalent command for the RTC<sup>®</sup> SCANalone.</b>

Ctrl Command	<b>set_co2_standby</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>set_standby</b> (see page 117)

List Command	<b>set_co2_standby_list</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>set_standby</b> (see page 117)

List Command	<b>set_delays</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>set_laser_delays</b> (page 107) and <b>set_scanner_delays</b> (page 113)

Ctrl Command	<b>set_gain</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>set_matrix</b> (page 109) and <b>set_offset</b> (page 111) or <b>load_correction_file</b> (page 90)

Ctrl Command	<b>set_list_mode</b>
Support status	This RTC <sup>®</sup> 3/RTC <sup>®</sup> 4 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>There is no equivalent command for the RTC<sup>®</sup> SCANalone.</b> The circular queue mode is not supported by the RTC <sup>®</sup> SCANalone.

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

Ctrl Command	<b>set_speed</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>set_jump_speed</b> (page 106) and <b>set_mark_speed</b> (page 109)

Ctrl Command	<b>set_yag_parameter</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>There is no equivalent command for the RTC<sup>®</sup> SCANalone.</b>

Ctrl Command	<b>write_da</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>write_da_1</b> (page 124)

List Command	<b>write_da_list</b>
Support status	This RTC <sup>®</sup> 2 command is not supported by the RTC <sup>®</sup> SCANalone.
Replaced by	<b>write_da_1_list</b> (page 125)

## 11 Customer Service

### 11.1 Servicing and Repairs

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

### 11.2 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® SCANAlone board has been altered
- for damage resulting from improper packaging of a product returned to SCANLAB
- for consequential damages

### 11.3 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

### 11.4 Product Disposal

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



#### Caution!

- If a battery is installed in the RTC® SCANAlone board, the battery must be disposed of separately from the RTC® SCANAlone board as hazardous waste at the responsible department. Do not dispose of the battery together with the regular household garbage.

## 12 Technical Specifications

### Power supply

Requirements +7...+30 V DC, 10 Watt

### System Requirements

IBM-compatible PC with free USB 1.1 port

Operating system Microsoft  
WINDOWS Vista / XP /  
2000 and WINDOWS ME /  
98

### Dimensions

Length 170 mm

Height 100 mm

### Connectors, I/O Signals

#### Laser

Connector 10-pin header or 9-pin  
D-SUB socket

Maximum current load 10 mA  
of the laser signals

#### Scan Head

Connector 26-pin header or 25-pin  
D-SUB socket  
or ST sockets

Signals XY2-100 or XY2-100-O  
protocol

### Scan Head Control

Number of list buffers 2

Capacity of a single list 500000 commands

Position update period 10  $\mu$ s  
(microstep period)

Maximum range for –32768 to +32767  
the image field (16-bit signed)  
coordinates

**Digital Input Port** 16 bits

LOW level < 0.5 V

HIGH level 2.6 V ... 24 V

Input resistance > 10 k $\Omega$

The input signals are referenced to GND.

**Digital Output Ports** 8 bits, 16 bits, buffered

Maximum output  $\pm$ 8 mA  
current

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 X8 has been configured to replace the  
ANALOG OUT 1 signal with +5 V, then the maximum  
current load at pin (7) or pin (4) of the "LASER" con-  
nector is 100 mA.

## Index

### 123

10-bit analog inputs .....	58
4-pin header J6 .....	58
3-axis system .....	56, 60

### A

analog output ports .....	54, 62, 125
arc commands .....	13, 17
arc_abs (command) .....	72
arc_rel (command) .....	72
article number – see extra "Package Description" file	
auto_change (command) .....	73
auto_change_pos (command) .....	73
automatic list change .....	15, 73

### B

battery .....	59
beam dump .....	84
bitmap images .....	47, 112

### C

cable length	
and signal timing .....	111
calender, battery powered .....	59
calibration factor .....	32
calling convention for DLL .....	65
CFMP program .....	34
char*, definition of data type .....	71
clear_io_cond_list (command) .....	74
clear_list (command) .....	74
clock, battery powered .....	59
CO <sub>2</sub> laser control .....	35
commands	
control commands .....	14, 69
import declarations .....	65–66
list commands .....	14, 70
parameter data formats and ranges .....	71
commands:	72–127
arc_abs .....	72
arc_rel .....	72
auto_change .....	73
auto_change_pos .....	73
clear_io_cond_list .....	74
clear_list .....	74
disable_laser .....	75
enable_laser .....	75
execute_at_pointer .....	76
execute_list .....	76
get_counts .....	77
get_dll_version .....	77
get_head_status .....	78

get_hex_version .....	78
get_input_pointer .....	79
get_io_status .....	79
get_rtc_version .....	79
get_serial_number .....	80
get_startstop_info .....	80
get_status .....	81
get_time .....	81
get_value .....	82
get_wait_status .....	83
get_waveform .....	83
get_xy_pos .....	83
goto_xy .....	84
home_position .....	84
jump_abs .....	85
jump_rel .....	85
laser_on_list .....	86
laser_signal_off .....	86
laser_signal_off_list .....	86
laser_signal_on .....	87
laser_signal_on_list .....	87
list_call .....	87
list_call_cond .....	88
list_jump_cond .....	88
list_nop .....	89
list_return .....	89
load_correction_file .....	90
load_varpolydelay .....	92
long_delay .....	93
mark_abs .....	93
mark_date .....	94
mark_rel .....	94
mark_serial .....	95
mark_time .....	96
measurement_status .....	96
quit_loop .....	97
read_ad_x .....	97
read_io_port .....	97
read_status .....	98
release_wait .....	98
restart_list .....	99
save_and_restart_timer .....	99
select_cor_table .....	100
select_list .....	100
set_char_table .....	101
set_control_mode .....	102
set_control_mode_list .....	102
set_delay_mode .....	103
set_end_of_list .....	103
set_extstartpos .....	104
set_extstartpos_list .....	104
set_firstpulse_killer .....	104
set_firstpulse_killer_list .....	105
set_input_pointer .....	105



set_io_cond_list .....	106
set_jump_speed .....	106
set_laser_delays .....	107
set_laser_timing .....	108
set_list_jump .....	108
set_mark_speed .....	109
set_matrix .....	109
set_matrix_list .....	110
set_max_counts .....	110
set_offset .....	111
set_offset_list .....	111
set_piso_control .....	111
set_pixel .....	112
set_pixel_line .....	113
set_scanner_delays .....	113
set_serial .....	114
set_softstart_level .....	115
set_softstart_mode .....	116
set_standby .....	117
set_standby_list .....	117
set_start_list .....	117
set_trigger .....	118
set_wait .....	119
set_wobble .....	119
start_loop .....	120
stop_execution .....	120
stop_list .....	120
store_on_mmc .....	121
time_fix .....	121
time_update .....	121
timed_jump_abs .....	122
timed_jump_rel .....	122
timed_mark_abs .....	123
timed_mark_rel .....	123
usb_status .....	124
write_8bit_port .....	124
write_8bit_port_list .....	124
write_da_x .....	125
write_da_x_list .....	125
write_io_port .....	126
write_io_port_list .....	126
z_out .....	126
z_out_list .....	127
configuration – see extra "Package Description" file .....	
connectors, pin-outs .....	53–59
contacting SCANLAB .....	134
continuous data transfer to scan head .....	15
control commands .....	14, 69
control signals .....	56
external .....	102
coordinate transformations .....	42, 109, 110, 111
correction file .....	
CFMP and correXion program .....	34
correXion program .....	34
customer service .....	134

## D

data cable .....	56
adjusting signal timing for long cables .....	111
data types .....	71
delays .....	19–31
laser delays .....	19, 107
LaserOff delay .....	19
LaserOn delay .....	19
scanner delays .....	20–21, 113
jump delay .....	20
jump delay, variable .....	21
mark delay .....	22
polygon delay .....	23
polygon delay, variable .....	24–26
optimizing the delays .....	27–31
delivered product .....	6
demo program .....	67
diagnostics .....	41
digital 16-bit input and output .....	58
37-pole "EXTENSION 1" D-SUB socket .....	58
40-pin "EXTENSION 1" header .....	58
digital output port .....	55, 63, 124
dimensions .....	8, 9
dimensions of board .....	135
disable_laser (command) .....	75
disposal of product .....	134
DLL .....	
calling convention .....	65
import declarations .....	65–66
installation .....	65
double, definition of data type .....	71
driver .....	64

## E

edgelevel (variable polygon delay) .....	24, 103
electrical connections .....	53–59
enable_laser (command) .....	75
environmental conditions .....	12
examples, programming .....	46
execute_at_pointer (command) .....	76
execute_list (command) .....	76
external control signals .....	54, 102

## F

field distortion .....	33–34
FirstPulseKiller signal .....	36
functionality test .....	67

## G

get_counts (command) .....	77
get_dll_version (command) .....	77
get_head_status (command) .....	78
get_hex_version (command) .....	78
get_input_pointer (command) .....	79
get_io_status (command) .....	79

<b>get_rtc_version</b> (command) .....	79
<b>get_serial_number</b> (command) .....	80
<b>get_startstop_info</b> (command) .....	80
<b>get_status</b> (command) .....	81
<b>get_time</b> (command) .....	81
<b>get_value</b> (command) .....	82
<b>get_wait_status</b> (command) .....	83
<b>get_waveform</b> (command) .....	83
<b>get_xy_pos</b> (command) .....	83
<b>goto_xy</b> (command) .....	84
<b>guarantee</b> .....	134

## H

<b>home jump mode</b> .....	84
<b>home_position</b> (command) .....	84

## I

<b>IEEE floating point format</b> .....	71
<b>image field</b>	
calibration factor .....	32
coordinate range .....	32, 135
<b>image field distortion</b> .....	33–34
<b>importing commands</b> .....	65–66
<b>initialization</b> .....	67
<b>input resistance</b>	
digital input ports .....	135
<b>installation and start-up</b> .....	61–67
DLL .....	65
driver .....	64
jumper settings .....	61–63
<b>integer, definition of data type</b> .....	71

## J

<b>jump commands</b> .....	17, 85, 122
scan head and laser control .....	20
<b>jump delay</b> .....	20
optimizing .....	30
variable jump delay .....	21, 103
<b>jump speed</b> .....	17
<b>jump_abs</b> (command) .....	85
<b>jump_rel</b> (command) .....	85
<b>jumper settings</b> .....	61–63

## L

<b>lag</b> .....	19
<b>lamp current</b> .....	37, 93
<b>laser connection</b>	
10-pole "LASER" header .....	54
25-pole "LASER EXTENSION" D-SUB plug .....	55
26-pole "LASER EXTENSION" header .....	55
9-pole "LASER" D-SUB socket .....	54
<b>laser control</b> .....	35–40
– during jump command .....	20
– during mark command .....	22

<b>commands</b> .....	108, 117
<b>lamp current</b> .....	37, 93
<b>timing (CO<sub>2</sub> mode)</b> .....	35
<b>timing (laser mode 4)</b> .....	40
<b>timing (YAG modes)</b> .....	36–39
<b>laser delays</b> .....	19, 107
<b>laser mode 4 laser control</b> .....	40
<b>laser safety</b> .....	12
<b>laser signals</b> .....	54, 55
TTL signal level .....	62
<b>laser_on_list</b> (command) .....	86
<b>laser_signal_off</b> (command) .....	86
<b>laser_signal_off_list</b> (command) .....	86
<b>laser_signal_on</b> (command) .....	87
<b>laser_signal_on_list</b> (command) .....	87
<b>LaserOff delay</b> .....	19
optimizing .....	29
<b>LaserOn delay</b> .....	19
optimizing .....	29
<b>LED, status signals</b> .....	59
<b>list buffers</b> .....	14
capacity (max. number of commands) .....	14, 135
<b>list change, automatic</b> .....	15, 73
<b>list commands</b> .....	14, 70
<b>list_call</b> (command) .....	87
<b>list_call_cond</b> (command) .....	88
<b>list_jump_cond</b> (command) .....	88
<b>list_nop</b> (command) .....	89
<b>list_return</b> (command) .....	89
<b>load_correction_file</b> (command) .....	90
<b>load_varpolydelay</b> (command) .....	92
<b>long, definition of data type</b> .....	71
<b>long_delay</b> (command) .....	93
<b>longint, definition of data type</b> .....	71

## M

<b>manufacturer</b> .....	6
<b>mark commands</b> .....	17, 93, 94, 95, 123
scan head and laser control .....	22
<b>mark delay</b> .....	22
optimizing .....	30
<b>mark_abs</b> (command) .....	93
<b>mark_date</b> (command) .....	94
<b>mark_rel</b> (command) .....	94
<b>mark_rel</b> (command) .....	95
<b>mark_time</b> (command) .....	96
<b>marking speed</b> .....	17
<b>marking time</b>	
measuring via timer .....	18
<b>Marking-on-the-fly</b> .....	59
<b>measurement</b>	
status signals and output values .....	41
<b>measurement_status</b> (command) .....	96
<b>microsteps</b> .....	17
output interval .....	135
<b>MMC memory card</b> .....	53

multi media card .....	53
------------------------	----

## O

operating temperature .....	12
optical data transfer .....	57
options .....	60
10-bit analog inputs .....	58
optical data interface .....	60
optical data transfer .....	57
optoelectronic couplers .....	54, 60
Processing-on-the-fly .....	59, 60
second scan head connector .....	57, 60
third data channel .....	56, 60
optoelectronic couplers .....	54, 60
output current, maximum	
digital output ports .....	135
output interval for microsteps .....	17
output interval for the microsteps .....	135

## P

package contents – see extra "Package Description" file	
parameter types .....	71
PC operation .....	13, 14
PC, connection via USB interface .....	53
pchar, definition of data type .....	71
pin-outs of electrical connectors .....	53–59
pixel images .....	47, 112
polygon delay .....	23
optimizing .....	31
variable polygon delay .....	24–26, 103
power requirements .....	53
power supply	
10-pin "POWER" header .....	53
9-pin "POWER" D-SUB plug .....	53
requirements .....	135
principle of operation .....	13–41
Processing-on-the-fly .....	59
15-pole "MARKING ON THE FLY"-D-SUB socket .....	59
16-pin "MARKING ON THE FLY" header .....	59
program flow control .....	45
programming, examples .....	46

## Q

<b>quit_loop</b> (command) .....	97
----------------------------------	----

## R

raster images .....	47, 112
<b>read_ad_x</b> (command) .....	97
<b>read_io_port</b> (command) .....	97
<b>read_status</b> (command) .....	98
<b>release_wait</b> (command) .....	98
repairs .....	134

<b>restart_list</b> (command) .....	99
-------------------------------------	----

## S

safety	
during installation and operation .....	12
during start-up .....	12, 64
laser safety .....	12
sample program .....	67
<b>save_and_restart_timer</b> (command) .....	99
scan head connection	
"2.SCAN HEAD" header and D-SUB socket ...	57
25-pole "1.SCAN HEAD" D-SUB socket .....	56
26-pole "1.SCAN HEAD" header .....	56
data cable .....	56
optical data transfer .....	57
scan head control .....	17–21
– during jump command .....	20
– during mark command .....	22
SCANLAB, contacting .....	134
scanner delays .....	20–21, 113
<b>select_cor_table</b> (command) .....	100
<b>select_list</b> (command) .....	100
servicing .....	134
<b>set_char_table</b> (command) .....	101
<b>set_control_mode</b> (command) .....	102
<b>set_control_mode_list</b> (command) .....	102
<b>set_delay_mode</b> (command) .....	103
<b>set_end_of_list</b> (command) .....	103
<b>set_extstartpos</b> (command) .....	104
<b>set_extstartpos_list</b> (command) .....	104
<b>set_firstpulse_killer</b> (command) .....	104
<b>set_firstpulse_killer_list</b> (command) .....	105
<b>set_input_pointer</b> (command) .....	105
<b>set_io_cond_list</b> (command) .....	106
<b>set_jump_speed</b> (command) .....	106
<b>set_laser_delays</b> (command) .....	107
<b>set_laser_timing</b> (command) .....	108
<b>set_list_jump</b> (command) .....	108
<b>set_mark_speed</b> (command) .....	109
<b>set_matrix</b> (command) .....	109
<b>set_matrix_list</b> (command) .....	110
<b>set_max_counts</b> (command) .....	110
<b>set_offset</b> (command) .....	111
<b>set_offset_list</b> (command) .....	111
<b>set_piso_control</b> (command) .....	111
<b>set_pixel</b> (command) .....	112
<b>set_pixel_line</b> (command) .....	113
<b>set_scanner_delays</b> (command) .....	113
<b>set_serial</b> (command) .....	114
<b>set_softstart_level</b> (command) .....	115
<b>set_softstart_mode</b> (command) .....	116
<b>set_standby</b> (command) .....	117
<b>set_standby_list</b> (command) .....	117
<b>set_start_list</b> (command) .....	117
<b>set_trigger</b> (command) .....	118
<b>set_wait</b> (command) .....	119

<b>set_wobbel</b> (command) .....	119
short, definition of data type .....	71
shutter .....	12, 55
signal propagation time .....	111
smallint, definition of data type .....	71
softstart mode .....	37
software .....	
DLL calling convention .....	65
DLL installation .....	65
import declarations .....	65–66
initialization .....	67
software driver .....	64
standalone operation .....	13, 14
external start .....	16
stand-by pulses .....	36, 117
<b>start_loop</b> (command) .....	120
start-up .....	12, 64
status LEDs .....	59
status monitoring and diagnostics .....	41
status signals .....	56
step period .....	17, 135
<b>stop_execution</b> (command) .....	120
<b>stop_list</b> (command) .....	120
<b>store_on_mmc</b> (command) .....	121
string, definition of data type .....	71
structured programming .....	45
sub-list .....	45
synchronization of processing .....	16
system requirements .....	7, 135

## T

technical specifications .....	135
temperature .....	12
testing the RTC_SCANalone .....	67
third data channel .....	56, 60
<b>time_fix</b> (command) .....	121
<b>time_update</b> (command) .....	121
timed jump and mark commands .....	51
<b>timed_jump_abs</b> (command) .....	122
<b>timed_jump_rel</b> (command) .....	122
<b>timed_mark_abs</b> (command) .....	123
<b>timed_mark_rel</b> (command) .....	123
time-lag .....	19
timing, laser control .....	
CO <sub>2</sub> mode .....	35
laser mode 4 .....	40
YAG modes .....	36–39

## U

unsigned short, definition of data type .....	71
USB connector .....	53
<b>usb_status</b> (command) .....	124

## V

variable jump delay .....	21, 103
---------------------------	---------

variable polygon delay .....	24–26, 103
customizing .....	26
edgelevel .....	24, 103
vector commands .....	13, 17
version numbers .....	
firmware .....	6, 79
software .....	6, 77, 78

## W

wait commands .....	16
warranty .....	134
wobbel function .....	43, 119
word, definition of data type .....	71
<b>write_8bit_port</b> (command) .....	124
<b>write_8bit_port_list</b> (command) .....	124
<b>write_da_x</b> (command) .....	125
<b>write_da_x_list</b> (command) .....	125
<b>write_io_port</b> (command) .....	126
<b>write_io_port_list</b> (command) .....	126

## X

XY2-100 standard .....	13, 41, 78
XY2-100-O protocol .....	13, 41, 78

## Y

YAG laser control .....	36–39
lamp current .....	37, 93

## Z

<b>z_out</b> (command) .....	126
<b>z_out_list</b> (command) .....	127
z-channel .....	56, 60, 126

## Notes