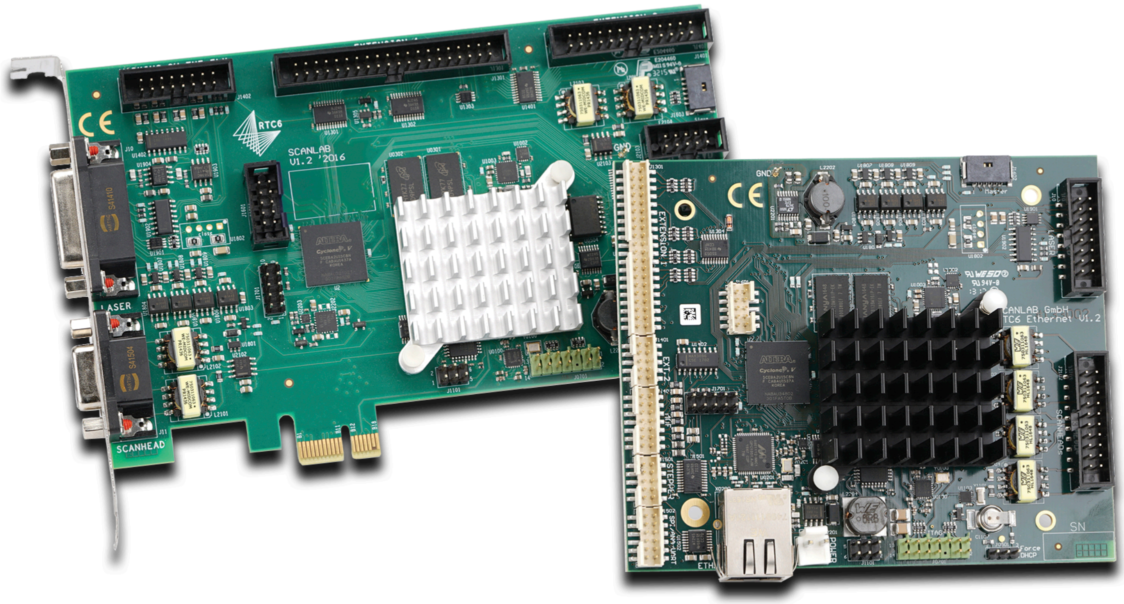
Installation and Operation

RTC6 PCIe Board
RTC6 Ethernet Board

Real Time Control of Scan Systems and Lasers
RTC6 Software Package 1.18.0



RTC6 Manual
Doc. Rev. 1.1.1 en-US

2024-06-13

SCANLAB GmbH
Siemensstr. 2a
82178 Puchheim
Germany

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

info@scanlab.de
www.scanlab.de

Contents

1 About this Manual

This manual describes the available SCANLAB RTC6 boards and their usage for synchronous control of scan systems, lasers and peripheral equipment.

The main chapters of this manual use the RTC6 PCIe Board (= "RTC6 PCI Express Board") to exemplify. For a product overview, see Chapter 2 "Product Overview", page 32.

Other RTC6 board variants are described in the Appendix:

- RTC6 Ethernet Board, see Chapter 16 "Appendix A: RTC6 Ethernet Board", page 955

The manual is a part of the product. Read these instructions carefully before you proceed with installing and operating.

In particular, observe all safety guidelines in this manual. If there are any questions regarding the contents of this manual, contact SCANLAB, see Chapter 1.1 "Manufacturer".

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

This manual refers to:

- RTC6 Software Package 1.18.0

DLL file for 32 bit user programs[a]	RTC6DLL.dll	DLL 646[b]
DLL file for 64 bit user programs[a]	RTC6DLLx64.dll	
Program file for the PCIe-DSP	RTC6OUT.out	OUT 649[b]
Program file for the Eth-DSP	RTC6ETH.out	ETH 649[b]
Firmware file for the FPGA	RTC6RBF.rbf	RBF 641[b]
Auxiliary file	RTC6DAT.dat	DAT 604[b]

(a) Software for laser–scan processes, which controls RTC6 boards based on this RTC6 DLL file is consistently denoted as "user program" in this manual.

(b) Abbreviated version in this manual for the Versions.

The version numbers of the supplied RTC6 DLL and RTC6 files are indicated in the names of the corresponding `zip` files, see Section "Folder RTC6 Files", page 34.

To identify the version numbers of your files after installation, use **get_dll_version**, **get_hex_version** and **get_rtc_version**.

1.1 Manufacturer

SCANLAB GmbH
Siemensstr. 2a
82178 Puchheim
Germany
Tel. +49 (89) 800 746-0
Fax +49 (89) 800 746-199
info@scanlab.de
www.scanlab.de

1.2 Related Documents

- "excelli*SCAN* Scan Heads – Functional Principle of *SCAN*ahead Servo Control and Operation by RTC6 Boards" Manual
- "Calibrating a 3-Axis Laser Scan System" Manual
- StreamParser-DLL –
  Application Programming Interface Manual

1.3 Glossary and Abbreviations

[*]mark[*] Command	All commands with “mark” as part of their names. See Section “Mark Commands”, page 140.
[*]para[*] Command	All commands with “para” as part of their names. See Section ”[*]Para[*] Commands”, page 142.
[*]scanahead[*] Command	All commands with “scanahead” as part of their names. Dedicated for controlling SCANahead Systems. See Chapter 8.13 ”SCANahead Functionality”, page 291.
3D Image Field	Synonym: working volume (process volume). See Chapter 7.3.4 ”3D Image Field”, page 175.
“Arc” Command	Umbrella term for Arc Commands and Ellipse Commands.
Answer Telegram	In context with Remote Interface Mode: telegram *from* RTC6 Ethernet Board, see `telegrams.h`. Consists of header and payload.
API	Abbreviation of Application Programming Interface. Program part which is available for other programs for connecting to the system.
BCD	Binary Coded Decimal.
BIOS	Basic Input/Output System. Is permanently stored in the Flash Memory of the RTC6 board. See also `RTC6conf.exe`.
BUSY Pin	Pin with BUSY OUT signal (= BUSY list execution status). • RTC6 PCIe Board: – LASER Connector, page 75 – EXTENSION 1 Socket Connector, page 81 • RTC6 Ethernet Board: – LASER Socket Connector, page 961 – EXTENSION 1 Socket Connector, page 970
CalibrationLibrary	Refer to www.scanlab.de.
Command Telegram	In context with Remote Interface Mode: telegram *to* RTC6 Ethernet Board, see `telegrams.h`. Consists of header and payload.
Data Streaming	See StreamParser-DLL – Application Programming Interface Manual, Brief information Data Streaming.
DSP	Digital signal processor on the RTC6 board.
Dynamic Focusing Unit	This includes, for example, the following SCANLAB products: vario*SCAN*, vario*SCAN*$_{de}$, vario*SCAN* FC and vario*SCAN* FLEX, excelli*SHIFT*.
Ethernet Link Loss	The cable connection to the RTC6 Ethernet Board has been disconnected.
Flash Memory	Non-volatile memory on the RTC6 board that replaces the EEPROM of the RTC5 board.

FPGA	Field programable gate array on the RTC6 board.
Hardware Reset	New start after powering the RTC6 board. Synonym: “power up”, “power cycle“.
Hard Jump	Direct output to a specified position. Decomposition into Microsteps into a single 10 µs clock cycle.
High-Bandwidth Return Channel Multiplexing	Functionality not yet released.
i*DRIVE* Scan Systems	In this manual, the term subsumes, for example, the following SCANLAB products: intelli*SCAN*, intelli*SCAN*$_{de}$, intelli*SCAN*$_{se}$, intelli*DRILL*, intelli*cube*, intelli*WELD*, vario*SCAN*$_{de}$, power*SCAN* II 50i, excelli*SCAN*.
Image Field	Synonym: Working Field.
intelli*SCAN*	In this manual, the term subsumes, for example, the following SCANLAB products: intelli*SCAN*, intelli*SCAN*$_{de}$, intelli*SCAN*$_{se}$, intelli*DRILL*, intelli*cube*, intelli*WELD*, power*SCAN* II 50i.
Jump Command	Serves to move the scan system axes to a new position while the laser is *off*. See also Section ”Jump Commands”, page 140. See 2D Jump Commands, page 330 and 3D Jump Commands, page 330.
Laser Control Signals	LASER1 signal, LASER2 signal, LASERON signal. See, for example, Chapter 7.4.1 ”Enabling, Activating and Switching Laser Control Signals”, page 189.
LSB	Least Significant Bit.
Mark Command	Serves to perform marking motions while the laser is switched *on*. Examples: mark, arc and ellipse. See 2D Mark Commands, page 330, and 3D Mark Commands(1), page 330.
McBSP	Multi channel Buffered Serial Port.
Microstep	See Chapter 7.1.2 ”Microstepping”, page 143.
Microvector	The term refers to micro_vector[*] Commands, see Chapter 8.8 ”micro_vector[*] Commands”, page 282.
micro_vector[*] Command	All commands with “micro_vector” as part of their names. See Chapter 8.8 ”micro_vector[*] Commands”, page 282.
MSB	Most Significant Bit.
NAND Memory	Non-volatile memory on the RTC6 Ethernet Board. Synonym: NAND flash.

NULL	Means on the one hand the number 0, on the other hand a pointer with the value 0. The spelling for this is different in the different programming languages.
OIE	Open Interface Extension. SCANLAB product (hardware). See SCANLAB website.
PCB	Printed Circuit Board.
PID	Proportional–Integral–Derivative. Category of controller.
Pixel Mode	Brief for "Pixel Output Mode". See Chapter 8.7 "Pixel Output Mode", page 272.
Polyline	A direct sequence of [*]mark[*] Commands or "Arc" Commands. The marking is continuous. Synonym: polygonal chain, polygonal line, polygonal traversal.
PosAck	"Position Acknowledge", see PosAck Signal.
PosAck Signal	• As of scan system firmware ≥ 2001. • PosAck Signal-capable SCANLAB scan systems, for example, intelliSCAN • Refers to the meaning of: – Bit #12, Bit #4, Bit #11, Bit #3 of the XY2-100 status word transferred with 16-bit protocol – Bit #16, Bit #8, Bit #15, Bit #7 of the XY2-100 status word transferred with 20-bit protocol • These bits are set, if the position errors of x axis and y axis are in the allowed range – For more information, refer to the corresponding scan head manuals. • Compare to TrAck Signal.
Preprocessing Servo Control	Servo control (of newer generation) in Preprocessing Systems = with Tracking Error and with preprocessing.
Preprocessing System	SCANLAB scan system with Preprocessing Servo Control. Compare to Tracking Error System, SCANahead System.
Processing-on-the-fly Session	A section of a list that uses external inputs (encoder pulses, McBSP transmissions) to correct moving workpiece positions.
RCM	Return Channel Multiplexing
Remote Command	Umbrella term for Remote Control Commands and Remote List Commands. See Chapter 16.9 "Remote Interface Mode", page 997.
Remote Control Command	`telegrams.h` > enum REMOTE_ID > enumeration constants R_DC_* ≤ 255. See Chapter 16.9 "Remote Interface Mode", page 997.

Remote List Command	`telegrams.h` > enum REMOTE_ID > enumeration constants R_LC_* ≥ 256. See Chapter 16.9 "Remote Interface Mode", page 997.
Reset of the RTC6 board	Synonym: Software Reset.
StreamParser-DLL	See StreamParser-DLL – Application Programming Interface Manual.
*SCAN*ahead Servo Control	Equipment feature of certain SCANLAB scan systems (= "SCANahead Systems"), essentially based on an ISB1 as a servo board for the 2 galvanometer scanner with corresponding firmware and parameterization. Refer to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.
*SCAN*ahead System	SCANLAB scan system with SCANahead Servo Control, for example, excelli*SCAN* series. Compare to Tracking Error System, Preprocessing System. Only SCANahead Systems transfer the HEAD BUSY list execution status (control command **get_status**, Bit #23 = 1), see also Chapter 6.4.3 "List Execution Status", page 111.
Scanner Delay	See Chapter 7.2.2 "Scanner Delays", page 150.
set_trigger[*]	Read: "**set_trigger**/**set_trigger4**/**set_trigger8**".
Software Reset	Restart after **load_program_file**. This does not reset everything, for example, loaded correction tables are retained. Synonym: Reset of the RTC6 board.
SPI	Serial Peripheral Interface.
Standalone Operation Mode	See Chapter 16.7 "Standalone Functionality", page 988.
TrAck	"Trajectory Acknowledge", see TrAck Signal.

TrAck Signal	• As of scan system firmware ≥ 5102 + ≥ 5112. • TrAck Signal-capable SCANLAB scan systems, for example, SCANahead Systems • Refers to the meaning of: – Bit #12, Bit #4, Bit #11, Bit #3 of the XY2-100 status word transferred with 16-bit protocol – Bit #16, Bit #8, Bit #15, Bit #7 of the XY2-100 status word transferred with 20-bit protocol • These bits are set, if the Trajectory Errors of x axis and y axis are in the allowed range – Refer to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual – For more information, refer to the corresponding scan head manuals • Compare to PosAck Signal.
Trajectory	In this manual: curve with 10 μs parameterization.
Trajectory Error	Deviation of the actual Trajectory from the set Trajectory.
Tracking Error	Time difference between the planned and actual reaching of a certain mirror position at constant speed.
Tracking Error Servo Control	The "conventional" servo control of Tracking Error Systems.
Tracking Error System	SCANLAB scan system with conventional control (1st generation) = with Tracking Error and without preprocessing. For example, intelli*SCAN* III. Compare to SCANahead System, Preprocessing System.
Vector Command	Umbrella term for Jump Commands and Mark Commands.
Working Field	Synonym: Image Field.

2 Product Overview

2.1 Labeling

The serial number of the RTC6 PCIe Board is printed on a label attached to the board.

The article number and configuration of the board are described in the packaging list, see Chapter 2.6 "Options", page 39, and Chapter 2.7 "Jumper Settings and Type Designations", page 41.

2.2 Unpacking Instructions and Typical Scope of Delivery

(1) Carefully remove the RTC6 PCIe Board from the package.

(2) Keep the packaging, including the antistatic bag the RTC6 PCIe Board is delivered in, so that in case of repair the board can be properly repackaged and returned to SCANLAB.

(3) Also remove all other articles from the package. Check that all parts have been delivered. Refer to the corresponding packaging list.
The scope of delivery typically includes an RTC6 PCIe Board and a data CD (with the RTC6 Software Package, see below, and this manual). Possibly additional components are also contained, see Chapter 2.8 "Accessories for the RTC6 PCIe Board", page 45.

2.3 Delivered RTC6 Software Package

- See also Chapter 19 "Appendix D: RTC6 Software Packages – History", page 1055.

The delivered RTC6 Software Package contains the RTC6 board driver(1) for the 32-bit and 64-bit versions of the operating systems Microsoft Windows 10, 8, 7.

Windows XP and Windows Vista are not supported!

Observe the safety notice on legacy RTC board drivers in Chapter 2.5.2 "Software", page 38.

The data CD contains all files as unzipped versions. The complete RTC6 Software Package is also delivered zipped for easy identification and management of different software versions:

- `RTC6_Software_Package_Rev.n.n.n_<Datum>.zip`

The content of the RTC6 Software Package is as follows:

- `Readme.txt`
  Description in English
- `Liesmich.txt`
  Description in German
- `IMPORTANT_RELEASE_NOTES.txt`
  Bilingual English and German
- `RTC6_Release_Notes_n.n.n.pdf`
  Mentions known restrictions.
  Bilingual English and German
- Folder Correction Files
- Folder CorrectionFileConverter
- Folder Demo Files
- Folder HPGL
- Folder iSCANCfg
- Folder RTC6 Driver
- Folder RTC6 Files
- Folder RTC6 Tools

(1) The RTC6 board driver is not required for RTC6 Ethernet Boards.

Folder Correction Files

- Cor_1to1.ct5
  1to1 correction file[1][2]

Folder CorrectionFileConverter

- CorrectionFileConverter.exe
  This Win32-based program converts RTC4 correction files *.ctb to RTC5/RTC6 correction files *.ct5 and vice versa. The corresponding manual is supplied in English and German.

Folder Demo Files

Subfolder Remote Interface

See Chapter 16.9 "Remote Interface Mode", page 997.

- example_1.cpp

Folder HPGL

- Hpgl.exe is a Win32-based HPGL-to-RTC6 converter. See also Chapter 5.7 "Functionality Test", page 95.
- The folder contains some *.plt files (Hewlett Packard HPGL format (vector graphic plotter files) for test purposes.
- The folder contains RTC6 files and the RTC6DLL.dll because Hpgl.exe needs them in the same folder.

Folder iSCANCfg

- iSCANcfg.exe is a Win32-based diagnosis and configuration program for iDRIVE Scan Systems. RTC4, RTC5 and RTC6 (PCI/PCIe as well as Ethernet variants) are supported.
- Needs RTC6 files and the RTC6DLL.dll in the same folder and in addition RtcHalDLL.dll. The corresponding manual is supplied in English (Manual_iSCANcfg_v1-7.pdf) and German (Handbuch_iSCANcfg_v1-7.pdf).

Folder RTC6 Driver

(RTC6 board driver for Windows)

- RTC6DRV.sys, RTC6DRVx64.sys, RTC6DRVx86.sys
  RTC6 board driver files
- RTC6DRV.inf
  Installation file (setup information)
- RTC6DRVx64.cat, RTC6DRVx86.cat
  Security catalog files
- amd64/WdfCoInstaller01009.dll,
  x86/WdfCoInstaller01009.dll
  Installation assistant help files
- AfterInstallation/ScanlabClassChecker.cmd,
  Security installation script with description in
  ReadMe_ScanlabClassChecker.pdf

(1) Additional correction files (D2_XXX.CT5, D3_YYY.CT5) and *_ReadMe.txt file(s) are not part of the RTC6 Software Package.

(2) See also Section "1to1 Correction Tables", page 182.

Folder RTC6 Files

- `RTC6_RevisionHistory_<SW-V>.pdf`
  Description of RTC6 Software Package changes in English
- `RTC6_Aenderungshistorie_<SW-V>.pdf`
  Description of RTC6 Software Package changes in German
- Subfolder Linux
- Subfolder Program Files
- Subfolder Remote Interface
- Subfolder Windows

Subfolder Linux

- For Linux developers only:
  Debian packages with shared libraries for controlling RTC6 Ethernet Boards
  - `readme.txt`
    Package contents and installation instructions
  - Subfolder debian-bullseye
    Package for Debian Bullseye
  - Subfolder debian-buster
    Package for Debian Buster
  - Subfolder debian-bookworm
    Package for Debian Bookworm

Subfolder Program Files

- RTC6 files
  - `RTC6OUT.out`
    Program file for the PCIe-DSP
  - `RTC6ETH.out`
    Program file for the Eth-DSP
  - `RTC6RBF.rbf`
    Firmware file for the FPGA
  - `RTC6DAT.dat`
    Binary auxiliary file
- Differing versions of RTC6 files and RTC6 DLL cannot be arbitrarily combined with another. The following `zip` files (each including a text file with version/compatibility information) are provided to make it easier to identify the versions:
  - `RTC6OUT_<current OUT version number>.zip`
    (includes `RTC6OUT.out` and `RTC6ETH.out`)
  - `RTC6RBF_<current RBF version number>.zip`
  - `RTC6DAT_<current DAT version number>.zip`

Subfolder Remote Interface

See Chapter 16.9 "Remote Interface Mode", page 997.

- `rtc6_rif_wrapper.cpp`
  Source file for C++ Remote Interface Wrapper
- `rtc6_rif_wrapper.h`
  Header file for C++ Remote Interface Wrapper
- `telegrams.h`
  Header file for `rtc6_rif_wrapper.h`. Defines the Command Telegram structure as well as Remote Command IDs

Subfolder Windows

- RTC6 DLL
  - `RTC6DLL.dll`
    Win32-based RTC6 dynamic link library
  - `RTC6DLLx64.dll`
    Win64-based RTC6 dynamic link library
- Utility files for C, C++ and C#
  - `RTC6expl.c`
    C functions for RTC6 DLL handling for explicit linking
  - `RTC6expl.h`
    C function prototypes of the RTC6 for explicit linking of the RTC6 DLL
  - `RTC6DLL.lib`
    Visual C++ import libraries for implicit linking of the RTC6 DLL for Win32-based applications
  - `RTC6DLLx64.lib`
    Visual C++ import libraries for implicit linking of the RTC6 DLL for Win64-based applications
  - `RTC6impl.h`
    C function prototypes of the RTC6 for implicit linking of the RTC6 DLL
  - `RTC6impl.hpp`
    C++ function prototypes of the RTC6 for implicit linking of the RTC6 DLL
  - `RTC6Wrap.cs`
    Import declarations of the wrapper class for implicit linking in C#
- Utility file for Delphi
  - `RTC6Import.pas`
    Import declarations for Delphi
- Differing versions of RTC6 files and RTC6 DLL cannot be arbitrarily combined with another. The following `zip` file (including a text file with version/compatibility information) is provided to make it easier to identify the versions:
  - `RTC6DLL_<current DLL version number>.zip`
    (includes DLLs and related utility files)

Folder RTC6 Tools

- `RTC6conf.exe`
  "RTC6 Configuration Tool"
  with the following main features
  - for RTC6 PCIe Boards and RTC6 Ethernet Boards: viewing board information (serial number, enabled options, BIOS version), enabling options (requires an auxiliary file which is to be purchased from SCANLAB), upgrading and downgrading BIOS
  - for RTC6 Ethernet Boards additionally: searching boards in (configurable) network segments (subnets), show and change network settings for static IP configuration.
- `RTC6conf.pdf`
  Description of the use of `RTC6conf.exe`
- `RTC6BIOSOUT_23.out`
  DSP program file for RTC6 PCIe Board BIOS (version `0x23`)
- `RTC6BIOSETH_42.out`
  DSP program file for RTC6 Ethernet Board BIOS (version `0x42`)(1)
- The RTC6 files and RTC6 DLL of this RTC6 Software Package
  - `RTC6DAT.dat`
  - `RTC6DLL.dll`
  - `RTC6ETH.out`
  - `RTC6OUT.out`
  - `RTC6RBF.rbf`
- `SleepMode.cmd`
  Script to deactivate *all* Windows sleep and hibernate modes.
- `ReadMe_SleepMode.pdf`
  Description of the use of `SleepMode.cmd`

(1) See also Chapter 16.7.1 "Upgrading BIOS-ETH", page 989.

2.4 Intended Use

The SCANLAB RTC6 PCIe Board and its associated RTC6 Software Package is intended for synchronous real-time control of scan systems, lasers and peripheral equipment by a Windows PC with a PCIe bus interface.

RTC6 boards are available in different hardware variants, Chapter 1 "About this Manual", page 25.

The delivered RTC6 DLL provides an extensive command set for control. This allows a quick and flexible software development for laser–scan processes.

The RTC6 PCIe Board is equipped with a fast digital signal processor (DSP). During execution of control commands it also handles more complex signal processing, such as simultaneous control of two scan systems or coordinate transformations.

Moreover, you can store controlling commands (= list commands) on the RTC6 PCIe Board and start their execution at a later point in time. Command execution by the RTC6 PCIe Board can then take place independently of the host PC. This makes it possible to meet the stringent demands of real-time control for scan systems, lasers and peripheral equipment, even if the PC must simultaneously respond to other tasks such as machine control and network communication.

The interface to the scan system, together with the associated software commands, allows bidirectional communication with the scan system, thereby providing both control and monitoring capabilities for the scan system.

With the RTC6 PCIe Board, commonly used laser types can be controlled. To control lasers, the supplies interfaces that output Laser Control Signals and are software-configurable for each application's requirements.

Users can choose among different laser modes and set the signal parameters (for example, the signal level – active-HIGH or active-LOW) or the output frequency to a suitable value.

For controlling peripheral equipment and incorporating external control signals, the RTC6 PCIe Board provides a range of interfaces (for example, a 16-Bit Digital Input Port, a 16-Bit Digital Output Port, 12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2 and an RS-232 interface, see Chapter 4 "RTC6 PCIe Board – Layout and Interfaces", page 66) and associated software commands.

As many RTC6 PCIe Board as the PCIe bus permits can be operated simultaneously in a single PC. The total number of RTC6 PCIe Boards and RTC6 Ethernet Boards must not exceed 255.

Moreover, the RTC6 DLL allows multi-threading as well as multi-processing; therefore several user programs can be used simultaneously.

No board can be simultaneously used by multiple applications. Multiple threads of *one* user program can use the same board, but can not send commands to it at the same time. The RTC6 DLL serializes these accesses automatically.

The RTC6 PCIe Board is available in various configurations, see Chapter 2.6 "Options", page 39, and Chapter 2.7 "Jumper Settings and Type Designations", page 41.

The RTC6 PCIe Board interfaces are described on Chapter 4 "RTC6 PCIe Board – Layout and Interfaces", page 66, installation and start-up on Chapter 5 "Installation and Start-Up", page 91, and programming on Chapter 6 "Developing RTC6-User Programs", page 96. Individual command descriptions are listed beginning with Chapter 10 "RTC6 Commands", page 324.

The technical specifications of the RTC6 PCIe Board are summarized on Chapter 15 "Technical Specifications – RTC6 PCIe Board", page 950.

Caution!

- Do not operate the RTC6 PCIe Board outside of the PC.
- The RTC6 PCIe Board is intended only for industrial usage. It is designed to be incorporated in machines (normally laser systems). It does *not* meet all criteria of ready-to-use products. Do not operate the RTC6 PCIe Board unless it is incorporated in a machine which itself complies with the regulations of all applicable standards and directives (of the country concerned). It is *not* suitable to be used as toy, in household or under unfavourable environment conditions (for example, in the open). Appropriate precautions to avoid such unforeseeable misapplications must be taken by users.
- Installation and operation must only be performed by trained specialists, among other things knowledgeable in the safe and proper use of electrical devices. Only carry out installation and maintenance work when supply voltages and lasers have been switched off.
- The RTC6 PCIe Board is a class A product. In a domestic environment this product may cause radio interferences in which case the user may be required to take adequate measures.

2.5 System Requirements

2.5.1 Hardware

The RTC6 PCIe Board requires a Windows PC with a PCIe bus interface and at least one free PCIe slot.

RTC6 PCIe Boards intended for synchronized master/slave operation should (recommended) be installed in adjacent PCIe slots.

2.5.2 Software

To operate the RTC6 PCIe Board, RTC6 board driver and RTC6 DLL files for Microsoft Windows operating systems must be used. These are included in the scope of delivery (RTC6 Software Package). For the supported Windows versions, see Chapter 2.3 "Delivered RTC6 Software Package", page 32.

The RTC6 board driver supports the plug and play capability of the RTC6 PCIe Board as well as the simultaneous operation of up to 255 RTC6 PCIe Boards. The RTC6 DLL files contain the command set to control laser scan systems.

Notice!

- The RTC6 PCIe Board does not support power-saving modes, that switch off power to the PCIe bus. Accordingly, you must disable standby or sleep modes of the operating system. See also following Section "Notes", page 38.

Notice!

- If on your PC an WDM technology-based RTC3/4/5 board driver
  - is yet installed or
  - was installed and has been removed or
  - you are not sure in this regard:
  (1) Install the RTC6 board driver.
  (2) Run `ScanlabClassChecker.cmd` as Administrator (part of the delivered RTC6 Software Package; see there also the background information in `ReadMe_ScanlabClassChecker.pdf`).
  Step 2 can be skipped on brand new PCs on which an RTC board driver never has been installed.

Notes

- RTC6 Software Packages only contain WDF(1) drivers (version 6.1.7600.16385).
  A WDM(2) driver (outdated today) is not contained.
- If **init_rtc6_dll** is called, then the RTC6 board driver prevents automatic activation of standby or sleep modes (continuously until the next system restart).
  This enables RTC6 PCIe Boards to continue processing lists autonomously even when the initiating user program has already been terminated.

  However, standby or sleep modes cannot be prevented if triggered manually or by discharged batteries. Afterward, loaded lists and other settings of the RTC6 PCIe Board are lost. After a wake-up, the RTC6 PCIe Board might no longer be correctly addressable. Users themselves must prevent standby or sleep modes prior to the first call of **init_rtc6_dll**. Before calling **init_rtc6_dll** for the first time, make sure that standby or sleep modes of the operating system are deactivated.

  The script `SleepMode.cmd` which is contained in the RTC6 Software Package (under RTC6 Tools) deactivates *all* sleep and hibernation modes, see `ReadMe_SleepMode.pdf`.

(1) Windows Driver Framework. Current technology.

(2) Windows Driver Model. Legacy technology.

2.6 Options

RTC6 PCIe Boards can be equipped with different options (features). For new boards, the options ordered are enabled/installed at the SCANLAB factory on delivery. For information on retrofitting already delivered boards, see Section "Notes", page 40.

To query which options are actually enabled on a certain board, **get_rtc_version** is used.

The following options are available for RTC6 PCIe Boards:

- **Option Processing-on-the-fly**
  Allows that a Processing-on-the-fly correction can be activated, see Chapter 8.6 "Processing-on-the-fly", page 251.
- **Option "3D"**
  – Controlling a 3-Axis Scan System
  Allows that an RTC6 PCIe Board can synchronously control a third axis (z axis, for example, a vario*SCAN* as a dynamic focus unit) along with the scan system's x axis and y axis (usually two galvanometer scanners) by its two scan head connectors, see Chapter 8.5.2 "3D Scan Systems", page 246.
- **Option "Second Scan Head Control"**
  – Controlling Two Scan Systems Simultaneously
  Allows that an RTC6 PCIe Board can simultaneously control two xy-scan systems via its two scan head connectors, see also Chapter 4.5.1 "Scan Head Connectors and Transfer Protocol", page 69 and Chapter 8.5 "Controlling 2D Scan Systems and 3D Scan Systems", page 245.
- **Option "SCANa"**
  – Controlling SCANahead Systems
  Allows that SCANahead Systems (for example, excelli*SCAN* series) can (also) be controlled. Refer also to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.
- **Option "UFPM"**
  – Pixel Output Modes with pixel output frequencies more than 800 kHz(1)
  Allows that pixel output frequencies of > 800 kHz…3,2 MHz can be achieved in certain Pixel Output Modes, see also **set_pixel_line**.
- **Option "syncA"**
  – Support of the sync*AXIS* control Software
  Allows that SCANLAB sync*AXIS* control software can be used (board and software are part of SCANLAB's scope of delivery for XL *SCAN* systems).
- **Option "DC/DC Converter"**
  RTC6 PCIe Boards with Option "DC/DC Converter" are equipped with an extra DC/DC converter (optoelectronic coupler) ex works.

Notice!

- On RTC6 PCIe Boards with Option "DC/DC Converter", GND2 and the Laser Control Signals LASERON, LASER1 and LASER2 at the LASER Connector and EXTENSION 2 Socket Connector are galvanically decoupled from GND.(a)
- On RTC6 PCIe Boards without Option "DC/DC Converter", GND2 are GND identical.

(a) RTC6 PCIe Boards: GND is the PC ground.
RTC6 Ethernet Boards: GND is the ground at the POWER connector.

See also:

– Section "Laser Control Signals", page 76
– Section "Laser Control Signals", page 84

(1) UFPM, Ultra Fast Pixel Mode.

Notes

- Each RTC6 PCIe Board article number indicates which of the options above are enabled and/or installed. These are stated in the packing list. The naming there is as follows:
  - "Fly" for Option Processing-on-the-fly
  - "3D" for Option "3D"
  - "SSHC" for Option "Second Scan Head Control"
  - "SCANa" for Option "SCANa"
  - "UFPM" for Option "UFPM"
  - "syncA" for Option "syncA"
  - "DCDC" for Option "DC/DC Converter"
- To query which options are actually enabled on a board, **get_rtc_version** is used.
- If you need one or more options which are not activated out of the factory on your RTC6 PCIe Board: order an corresponding license file from SCANLAB, specifying the serial number and the options you want. Then, to apply the upgrade to the board, specify it in `RTC6conf.exe`.
- For retrofitting your RTC6 PCIe Board with the extra DC/DC converter (optoelectronic coupler) you need to send it to SCANLAB.
- For optical data transfer between the RTC6 PCIe Board and the scan system, solely a suitable data cable is required, see Chapter 4.5.3 "Data Cables (Accessories)", page 73. Neither the RTC6 PCIe Board side nor the scan system side requires a special optical data interface for that.
- With software package version ≤ 1.4.4, the "Processing-on-the-fly" functionality cannot be used for SCANahead Systems.

2.7 Jumper Settings and Type Designations

SCANLAB ships RTC6 PCIe Boards in various jumper configurations. Jumpers are connections which are either open or closed. The *factory* solder jumper configuration of an RTC6 PCIe Board can be identified by its article number.

In addition, a three-digit type code scheme is used, for example,

- "RTC6 PCI-Express TYPE 000"
  - No signals (that is, all solder jumpers are open)
- "RTC6 PCI-Express TYPE 124"
  - 5 V output signal level at the EXTENSION 1 Socket Connector
  - DATA7 at pin 15 of the EXTENSION 2 Socket Connector
  - LATCH at pin 17 of the EXTENSION 2 Socket Connector

Digit 1	Refers to the output signal level at the EXTENSION 1 Socket Connector(1).
	=0: No signals. =1: 5 V. =2: 3,3 V.
Digit 2	Refers to pin 15 of the EXTENSION 2 Socket Connector(3). =0: No signals. =1: +5 V. =2: DATA7. =3: GROUND.
Digit 3	Refers to pin 17 of the EXTENSION 2 Socket Connector(2). =0: No signals. =1: +5 V. =2: DATA7. =3: GROUND. =4: LATCH.

Trained users can reconfigure solder jumpers by using a soldering iron. The assignment of a desired signal is done by closing or opening (applying or removing solder or zero-ohm resistor) of corresponding solder jumpers as described in the following(1)(2)(3).

Notice!

- Only configure allowed jumper settings. Otherwise, the board gets damaged!

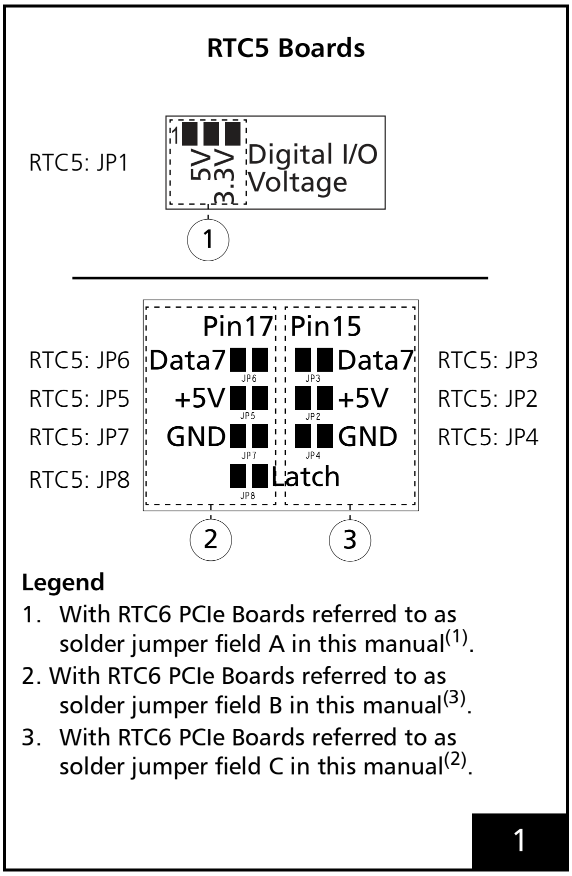
(1) For the solder jumper setting, see Chapter 2.7.1 "Solder Jumper Field A – Configuring Output Signal Level at EXTENSION 1 Socket Connector", page 42.

(2) For the solder jumper setting, see Chapter 2.7.2 "Solder Jumper Field B – Configuring Pin (17) of EXTENSION 2 Socket Connector", page 43.

(3) For the solder jumper setting, see Chapter 2.7.3 "Solder Jumper Field C – Configuring Pin (15) of EXTENSION 2 Socket Connector", page 44.

Notes for RTC5 Users

- In contrast to RTC5 boards, jumper numbers are no longer imprinted on RTC6 PCIe Boards. Therefore, RTC5 jumper designations cannot longer be used in this RTC6 Manual. The relation of the designations establishes Figure 1.



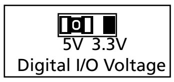
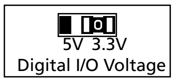
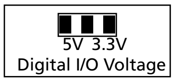
Jumper designations with RTC5 boards and RTC6 PCIe Boards.
Figure shows RTC5 board.

2.7.1 Solder Jumper Field A – Configuring Output Signal Level at EXTENSION 1 Socket Connector

The solder jumper field A is located on the lower side of the RTC6 PCIe Board, see Figure 6.

It is used to set the level (5 V or 3.3 V) of all output signals at the EXTENSION 1 Socket Connector, see the following table.

See also Section "Configuring the Output Signal Level", page 81.

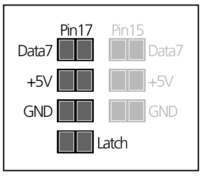
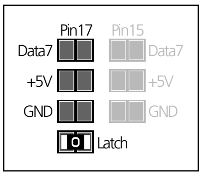
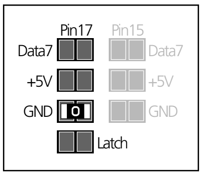
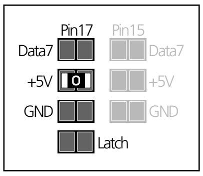
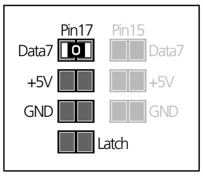
Allowed jumper setting	EXTENSION 1 Socket Connector
closed* open 	Output signal level 5 V.
open closed* 	Output signal level 3.3 V.
open open 	No signal output.
*Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!	

2.7.2 Solder Jumper Field B – Configuring Pin (17) of EXTENSION 2 Socket Connector

The solder jumper field B is located on the lower side of the RTC6 PCIe Board, see Figure 6.

It serves to configure the signal at pin (17) of the EXTENSION 2 Socket Connector, see the following table.

See also “Configuration by Solder Jumpers”, page 83.

Allowed jumper setting		Output at the EXTENSION 2 Socket Connector pin (17)
open open open open		No signal.
open open open closed*		LATCH signal.
open open closed open		GROUND (low level).
open closed open open		+5 V (high level).
closed* open open open		DATA7[a].

*Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!

(a) Synonym: Data Bit #7. MSB of the 8-bit output value.

Notes

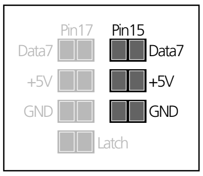
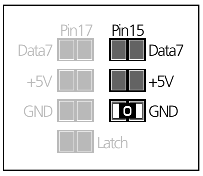
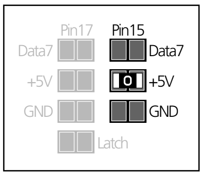
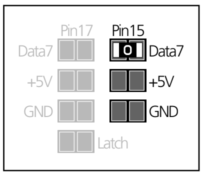
- Configurations of solder jumper field B and solder jumper field C are independent from each other.
- On RTC6 Ethernet Boards, the printed label of solder jumper field B is ’Pin17’. This has been chosen deliberately in order to keep consistency with already existing RTC boards. Even though with RTC6 Ethernet Boards pin (09) of the EXT. 2 socket connector, see Chapter 16.2.12 “EXT. 2 Socket Connector”, page 971, is actually configured.

2.7.3 Solder Jumper Field C – Configuring Pin (15) of EXTENSION 2 Socket Connector

The solder jumper field C is located on the lower side of the RTC6 PCIe Board, see Figure 6.

It serves to configure the signal at pin (15) of the EXTENSION 2 Socket Connector, see the following table.

See also "Configuration by Solder Jumpers", page 83.

Allowed jumper setting	Output at the EXTENSION 2 Socket Connector pin (15)
 open open open	No signal.
 open open closed*	GROUND (low level).
 open closed* open	+5 V (high level).
 closed* open open	DATA 7[a].
*Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!	

(a) Synonym: Data Bit #7. MSB of the 8-bit output value.

Notes

- Configurations of solder jumper field C and solder jumper field B are independent from each other.
- On RTC6 Ethernet Boards, the printed label of solder jumper field B is 'Pin15'. This has been chosen deliberately in order to keep consistency with already existing RTC boards. Even though with RTC6 Ethernet Boards pin (08) of the EXT. 2 socket connector, see Chapter 16.2.12 "EXT. 2 Socket Connector", page 971, is actually configured.

2.8 Accessories for the RTC6 PCIe Board

Only hardware extensions from SCANLAB should be used in combination with the RTC6 PCIe Board. In addition to the RTC6 PCIe Board and its software package, the following accessories can be obtained:

- XY2-100 Converter, page 45
- Laser Adapter, page 45
- Data Cables, page 45
- Slot Cover with 9-pin D-SUB Connector for 2. SCANHEAD Socket Connector, page 46
- Slot Cover with 15-pin D-SUB Connector for MARKING ON THE FLY Socket Connector, page 46
- Slot Cover with 15-pin D-SUB Connector and 9-pin D-SUB Connector, page 47

Notes

- On the UFPM Extension Board for the RTC6 PCIe Board(1), see Chapter 17 "Appendix B: UFPM Extension Board", page 1044.
- The following component can be used with RTC6 PCIe Boards:
  - "RTC5/6 varioSCAN 40 FLEX Extension" Extension Board (#0128683)(2)
- The following components *cannot* be used with RTC6 PCIe Boards:
  - EXT1 extension board for the RTC5 (#0123804)(3)
  - ADC add-on board (#0121126)
  - "RTC STEP MOTOR EXTENSION" extension board (#0112097)
  - I/O extension board for RTC3 and RTC4, (#0108285) (#0121721)

(1) Recommended for users with ANALOG controlled lasers, who want to achieve pixel output frequencies > 100 kHz, see **set_pixel_line**.

(2) Can also be used with RTC6 Ethernet Boards. Requires Y cable #0116050 (see Figure 81) and flat ribbon cable #0105446. In addition, the customer has to supply a 1:1 cable (plug 1: identical in construction to Würth 61201023021, 10-pin, female; plug 2: see under Notes, page 968).

(3) Developed for RTC5. Cannot be used with RTC6 PCIe Boards due to mechanical reasons (collision with cooling element; replacement: UFPM Extension Board, page 1044).

2.8.1 XY2-100 Converter

The XY2-100 Converter (Accessory) allows the RTC6 PCIe Board to control scan systems which are equipped with a conventional XY2-100 interface.

2.8.2 Laser Adapter

The SCANLAB laser adapter is plugged into the 15-pin LASER Connector of the RTC6 PCIe Board. Then a 9-pin female D-SUB connector of this adapter provides the same signals and pin-out as the 9-pin LASER connector of the RTC4/RTC5. See also Section "Laser Adapter (Accessory)", page 79.

2.8.3 Data Cables

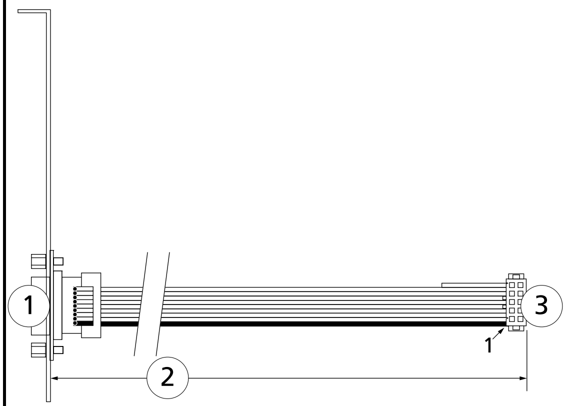
To connect the RTC6 PCIe Board to scan systems, SCANLAB offers appropriate cables in a variety of lengths – either conventional cables for electrical data transfer or polymer optical fiber cables or optical data transfer. See also Chapter 4.5.3 "Data Cables (Accessories)", page 73).

2.8.4 Slot Cover with 9-pin D-SUB Connector for 2. SCANHEAD Socket Connector

For connecting a second scan system or a z axis to the 2. SCANHEAD Socket Connector a slot cover is available, see Figure 2:

- #0115132

Its 9-pin D-SUB connector has the same pin-out as the SCANHEAD Connector, see Figure 10.
See also Section "Second Scan Head Slot Cover (Accessory)", page 70.



Legend

1. 9-pin D-SUB connector (female). pin-out as Figure 10.
2. l = 225 mm.
3. To 2. SCANHEAD Socket Connector.

2

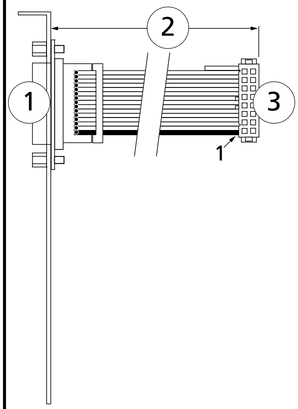
Second Scan Head Slot Cover (Accessory) #0115132.

2.8.5 Slot Cover with 15-pin D-SUB Connector for MARKING ON THE FLY Socket Connector

For using the input ports and signals of the MARKING ON THE FLY Socket Connector, a slot cover is available, see Figure 3:

- #0109272

The pin-out of its 15-pin D-SUB connector (female) is shown in Figure 26.
See also Section "MARKING ON THE FLY Slot Cover (Accessory)", page 86.



Legend

1. 15-pin D-SUB connector (female). pin-out as Figure 26.
2. l = 105 mm.
3. To MARKING ON THE FLY Socket Connector.

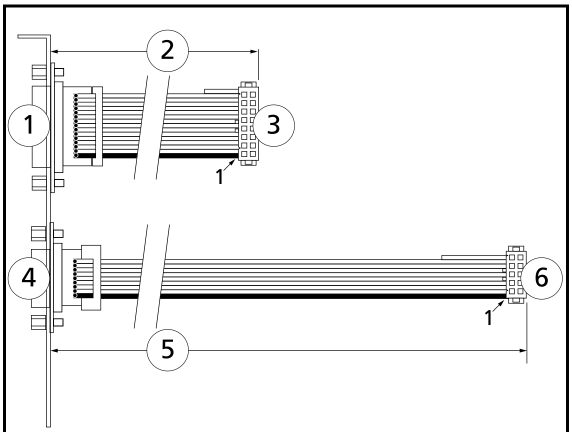
3

MARKING ON THE FLY Slot Cover (Accessory) #0109272.

2.8.6 Slot Cover with 15-pin D-SUB Connector and 9-pin D-SUB Connector

The connectors of #0115132 and #0109272 together on a single bracket (for the same purpose as these) features bracket, see Figure 4:

- #0130209



Legend

1. 15-pin D-SUB connector (female). pin-out as Figure 26.
2. l = 105 mm.
3. To MARKING ON THE FLY Socket Connector.
4. 9-pin D-SUB connector (female). pin-out as Figure 10.
5. l = 225 mm.
6. To 2. SCANHEAD Socket Connector.

4

Slot cover #0130209.

2.8.7 Extension Board "RTC5/6 varioSCAN FLEX Extension"

For RTC5/6 Boards available is(1):

- "RTC5/6 varioSCAN 40 FLEX Extension" Extension Board (#0128683)

It has been specially developed to control analog and digital vario*SCAN* 40 FLEX devices by SCANLAB laserDESK software. A position change of the vario*SCAN* 40 FLEX focusing optics is caused by the step motor which changes the working distance of the 3D scan system in the end.

For further information, refer to the pertaining manual "Installation and Operation RTC5/6 vario*SCAN* FLEX Extension for RTC5 and RTC6 Control Boards".

Notes

- "RTC5/6 varioSCAN 40 FLEX Extension" Extension Board (#0128683) does *not* support **move_to**(2).

(1) In contrast to "RTC STEP MOTOR EXTENSION" extension board (#0112097) – which can also be used with RTC5 Boards – it has the advantage that the EXTENSION 1 Socket Connector remains unoccupied.

(2) move_to has been introduced for the "RTC STEP MOTOR EXTENSION" extension board (#0112097).

2.9 Supplementary Software

To facilitate customizing RTC correction files basing on data of your own test measurements, SCANLAB offers the correXion pro software with accompanying manual, see also Section "Image Field Correction Algorithm", page 179.

By using the SCANLAB laserDESK software, own laser marking and material-processing programs ("laser jobs") can be created and executed without software development. Many of the RTC6 functions are supported.[1]
For further information on laserDESK, refer to the SCANLAB homepage.

(1) See also Notice!, page 75.

2.10 Notes for RTC4 Users

This chapter provides an overview of key changes introduced with RTC6 PCIe Boards for comparable functions of RTC4 boards.

For example, Chapter 2.10.2 "Porting RTC4 Source Code to the RTC6 PCIe Board", page 50 discusses a possible approach to porting RTC4 programs to run on the RTC6 PCIe Board.

The individual Command Descriptions, page 342 in particular note changes in row "RTC4→RTC6".

2.10.1 Hardware Changes

When migrating from the RTC4 to the RTC6 PCIe Board, you need to consider the following hardware changes for correct cabling of the system components.

Controlling Scan Systems

- To control scan systems with XY2-100 interface or XY2-100 Enhanced interface, you additionally need the XY2-100 Converter (Accessory).
- The RTC6 PCIe Board *cannot* control scan systems with XY2-100-O interfaces (for optical data transfer).
- To control a scan system with the SL2-100 interface, you need a different data cable (available from SCANLAB), see Chapter 4.5.3 "Data Cables (Accessories)", page 73.
- To use the Connector for Second Scan Head, you need a different adapter cable (including slot cover available from SCANLAB), see Section "Second Scan Head Slot Cover (Accessory)", page 70.
- For controlling *one* 3-axis scan system, both scan head connectors must be used, see Chapter 8.5.2 "3D Scan Systems", page 246.
- Simultaneous control of two 3-axis scan systems requires *two* RTC6 PCIe Boards, see Chapter 8.5.2 "3D Scan Systems", page 246.

Controlling the Laser

- The LASER Connector at the RTC6 PCIe Board slot cover has 15 pins (RTC4: 9 pins). Its pin-out is not jumper-configurable.
  – The RTC6 PCIe Board does not require jumpers **X6** and **X7** of the RTC4 because all signals are available at the RTC6 PCIe Board LASER Connector.
  – The voltage range of the analog outputs is always 0…10 V (0 V…2.50 V is no longer supported; the jumper **X3** of the RTC4 does not exist on the RTC6 PCIe Board).
- If you want to connect a laser to the RTC6 PCIe Board by the same (9-pin) cable that you previously used with the RTC4, then you need an adapter with a 9-pin female D-SUB connector (available from SCANLAB).
  – For use of the SCANLAB laser adapter, see Section "Laser Adapter (Accessory)", page 79, two jumpers are provided for configuring the pin-outs (**JP1** corresponds to jumper **X7** of the RTC4, **JP2** corresponds to jumper **X6** of the RTC4).
- The signal levels of the Laser Control Signals are no longer determined by configuring jumpers. Instead, they can/must be defined by an RTC6 command, see **set_laser_control**.
  – Jumper **X10** of the RTC4 does not exist on the RTC6 PCIe Board.

EXTENSION 1 Socket Connector

- The RTC6 PCIe Board EXTENSION 1 Socket Connector is – except for the additionally provided signals at pin (33)…(35) – identical to the EXTENSION 1 socket connector of the RTC4, see Figure 22.
- With the RTC6 PCIe Board, the level of all output signals at the EXTENSION 1 Socket Connector can be configured for 5 V or 3.3 V by a jumper, see Section "Configuring the Output Signal Level", page 81.

EXTENSION 2 Socket Connector

- The RTC4 EXTENSION 2 connector does not exist on the RTC6 PCIe Board. Therefore, the "I/O extension board for RTC3 and RTC4" (#0108285, #0121721) cannot be attached to the RTC6 PCIe Board.
- The RTC6 PCIe Board EXTENSION 2 Socket Connector is – except for the LATCH signal at pin (17) which can be optionally set – identical to the LASER EXTENSION socket connector of the RTC4, see Figure 24.
  - For configuring pin-outs the RTC6 PCIe Board provides the solder jumper field B and C, see also Section "Configuration by Solder Jumpers", page 83.
    These correspond to RTC4 jumpers **X8** and **X9**.

MARKING ON THE FLY Socket Connector

- The RTC6 MARKING ON THE FLY Socket Connector and the RTC4 MARKING ON THE FLY socket connector are identical, see Figure 25. Observe the safety notice on encoder counter direction, page 59.

Other Hardware Interfaces

- The following interfaces only exist on the RTC6 PCIe Board:
  - Master Socket Connector, Slave Socket Connector, page 68
  - RS232 Socket Connector, page 86
  - McBSP/ANALOG Socket Connector, page 87
  - STEPPER MOTOR Socket Connector, page 90

2.10.2 Porting RTC4 Source Code to the RTC6 PCIe Board

User programs written for the RTC4 can only run on the RTC6 PCIe Boards after suitable code revision. This applies even when the actual program flow remains unchanged.

Changed Initialization

The program's initialization section should be revised at least as follows:

- At the beginning of the program, an **init_rtc6_dll** must be inserted for initializing the RTC6 DLL and RTC6 board management, see Chapter 6.2.3 "Initializing the RTC6 DLL and RTC6 Board Management", page 100.
- The files for initializing the board by **load_program_file** are different than with the RTC4, see command description.
- Scan system initialization by **load_correction_file** and **select_cor_table** utilizes different correction files (with file extension `*.ct5`), see Chapter 7.3.5 "Image Field Correction and Correction Tables", page 178.
- For laser control initialization, **set_laser_control** must be additionally inserted, see Chapter 7.4 "Laser Control", page 189.

Command Changes

All unsupported RTC4 commands must be removed or replaced. See also Chapter 10.4 "Unsupported RTC4 Commands", page 941.

Changed or enhanced RTC4 commands must be handled differently in the program (for example, by modifying supplied parameter values or evaluating returned values differently). Changes to still supported commands are listed in the individual Command Descriptions, page 342, row "RTC4→RTC6".

RTC4 commands that need to be replaced or checked are:

• **auto_cal**	changed
• **auto_change_pos**	changed
• **control_command**	changed
• **dsp_start**	not supported
• **get_head_status**	changed
• **get_hi_data**	changed
• **get_list_space**	changed
• **get_marking_info**	changed
• **get_rtc_version**	changed
• **get_startstop_info**	changed
• **get_status**	changed
• **get_value**	changed
• **get_waveform**	changed
• **get_xy_pos**	not supported
• **get_xyz_pos**	not supported
• **goto_xy**	changed
• **goto_xyz**	changed
• **list_jump_cond**	changed
• **list_nop**	changed
• **load_correction_file**	changed
• **load_program_file**	changed
• **read_pixel_ad**	not supported
• **read_status**	changed
• **rtc4_count_cards**	not supported
• **select_cor_table**	changed
• **select_list**	not supported
• **select_rtc**	changed
• **set_control_mode**	enhanced
• **set_control_mode_list**	enhanced
• **set_laser_mode**	enhanced
• **set_laser_timing**	changed
• **set_list_mode**	not supported
• **set_matrix**	changed
• **set_matrix_list**	changed
• **set_offset**	changed
• **set_offset_list**	changed
• **set_piso_control**	not supported
• **set_pixel**	changed
• **set_pixel_line**	changed
• **set_softstart_level**	changed
• **set_softstart_mode**	changed
• **set_trigger**	enhanced
• **set_wobbel**	changed
• **set_wobbel_xy**	not supported
• **timed_jump_abs**	changed
• **timed_jump_rel**	changed
• **timed_mark_abs**	changed
• **timed_mark_rel**	changed
• **z_out**	not supported
• **z_out_list**	not supported

Increased Parameter Resolution

When switching from the RTC4 to the RTC6 PCIe Board – even for some commands not mentioned above – take note that the resolution has been increased for several parameters. Examples:

- For commands such as **mark_abs** or **jump_rel**, the real-image-field x coordinate values and y coordinate values are specified with 20-bit resolution for the RTC6 PCIe Board (whereas with 16-bit resolution for the RTC4), see also Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 172. Moreover, for Processing-on-the-fly applications, an extended, 29-bit value range is available with the RTC6 PCIe Board, see also Chapter 7.3.3 "Virtual Image Field", page 174.
- For commands such as **write_da_x**, analog output values are specified with 12-bit resolution for the RTC6 PCIe Board (whereas with 10-bit resolution for the RTC4).
- For **set_laser_timing**, output period and pulse length are specified with 1/64 µs resolution for the RTC6 PCIe Board (whereas with 1/8 µs or 1 µs resolution for the RTC4).
- For **set_laser_delays**, the Laser Delays are specified with 1/64 µs resolution for the RTC6 PCIe Board (whereas with 1 µs resolution for the RTC4).

Alternatively, the RTC6 DLL can be set to the RTC4 Compatibility Mode by **set_rtc4_mode**. Then the RTC6 DLL automatically converts parameter values so that many RTC4 command sequences can run unchanged on the RTC6 PCIe Board.

RTC4 Compatibility Mode affects the following RTC4 commands (descriptions of the respective commands include relevant information under the heading "RTC4→RTC6"):

- **arc_abs**
- **arc_rel**
- **fly_return**
- **get_z_distance**
- **goto_xy** (changed)
- **goto_xyz** (changed)
- **home_position**
- **jump_abs**
- **jump_abs_3d**
- **jump_rel**
- **jump_rel_3d**
- **mark_abs**
- **mark_abs_3d**
- **mark_rel**
- **mark_rel_3d**
- **set_delay_mode**
- **set_ext_start_delay**
- **set_ext_start_delay_list**
- **set_firstpulse_killer**
- **set_firstpulse_killer_list**
- **set_fly_x**
- **set_fly_y**
- **set_jump_speed**
- **set_laser_delays**
- **set_mark_speed**
- **set_pixel** (changed)
- **set_pixel_line** (changed)
- **set_rot_center**
- **set_softstart_level**
- **set_standby**
- **set_standby_list**
- **simulate_ext_start**
- **timed_jump_abs** (changed)
- **timed_jump_rel** (changed)
- **timed_mark_abs** (changed)
- **timed_mark_rel** (changed)
- **write_da_1**
- **write_da_1_list**
- **write_da_2**
- **write_da_2_list**
- **write_da_x**
- **write_da_x_list**

The previously mentioned revision of initialization and checking of unsupported or changed RTC4 commands needs to be carried out regardless of whether the program is to be executed in RTC6 Standard Mode or RTC4 Compatibility Mode.

Changed Timing Behavior

The following RTC4 list commands are processed by the RTC6 PCIe Board as “short list commands”. This can result in a changed timing behavior during user program execution, see Section “Normal List Command, Short List Command, Variable List Command and Multiple List Command”, page 335.

- **clear_io_cond_list**
- **list_call**
- **list_call_cond**
- **list_jump_cond** (changed)
- **list_return**
- **save_and_restart_timer**
- **set_extstartpos_list**
- **set_firstpulse_killer_list**
- **set_io_cond_list**
- **set_jump_speed**
- **set_laser_delays**
- **set_laser_timing** (changed)
- **set_list_jump**
- **set_mark_speed**
- **set_scanner_delays**
- **set_standby_list**
- **set_trigger** (enhanced)
- **set_wobbel** (changed)
- **write_8bit_port_list**
- **write_da_1_list**
- **write_da_2_list**
- **write_da_x_list**
- **write_io_port_list**

Likewise, automatic delay adjustments can produce a changed timing behavior, see Section "Automatic Delay Adjustments", page 158.

2.10.3 New and Changed Functionality

Interface to the PC

- The RTC6 board driver supports simultaneous control of any number of RTC6 PCIe Boards in a single PC, see Chapter 6.6 "Using Several RTC6 PCIe Boards in One PC", page 126.
- Connectors and commands for master/slave synchronization of several RTC6 PCIe Boards, see Chapter 4.4 "Master Socket Connector, Slave Socket Connector", page 68.

Controlling Scan Systems

- New interface to the scan system, see Chapter 4.5.1 "Scan Head Connectors and Transfer Protocol", page 69:
  - 9-pin female D-SUB connector at the RTC6 PCIe Board slot cover and 10-pin socket connector
  - SL2-100 protocol
  - 2 data channels each for both scan head connectors
  - Galvanically isolated signals
  - 20-bit positioning resolution, see Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 172
  - Enhanced status return from the scan system, see Chapter 7.3.7 "Status Monitoring and Diagnostics", page 188
  - Control channels and status channels for enhanced data transfer with iDRIVE Scan Systems
- An XY2-100 Converter (Accessory) is available for data transfer according to the XY2-100 protocol.
  - 25-pin female D-SUB connector
  - 16-bit positioning resolution
  - Status return according XY2-100 protocol or XY2-100 Enhanced protocol
  - Transfer synchronization is configurable for long data cables by solder jumpers in the XY2-100 Converter (Accessory)

  The RTC6 PCIe Board provides power for the XY2-100 Converter (Accessory).
- For optical data transfer between the RTC6 PCIe Board and scan systems, *no* special variant of the RTC6 PCIe Board (with XY2-100-O interface) is required. Optical data transfer can be realized by a SCANLAB data cable with electrical-to-optical conversion in its D-SUB connector housing, see Chapter 4.5.3 "Data Cables (Accessories)", page 73.
- For controlling a 3-axis scan system, see Chapter 8.5.2 "3D Scan Systems", page 246:
  - Both scan head connectors must be used
  - Simultaneous control of two 3-axis scan systems requires 2 RTC5 PCI Boards
- Image Field correction
  - New correction files are needed, see Chapter 7.3.5 "Image Field Correction and Correction Tables", page 178:
    - File name extension "`.ct5`"
    - Correction with higher resolution
    - Queryable information in the correction file header
    - For the RTC6 PCIe Board, RTC4 correction files (`.ctb`) need to be newly calculated or converted by `CorrectionFileConverter.exe` which is part of the RTC6 Software Package.
  - Up to 8 correction files can be loaded to the RTC6 PCIe Board
  - Enhanced 3D Image Field correction by stretch correction tables, see Section "Enhanced 3D Correction", page 249
- Coordinate transformations, see Chapter 8.2 "Coordinate Transformations", page 233:
  - The correction file is no longer transformed (rotation, shift, extension) at download
  - Matrix transformations are only applied after Microstepping – this may cause the mark speed to change
  - "Local Online Positioning", see Chapter 8.3.1 ""Local Online Positioning"", page 237

- Coordinate transformations in the virtual Image Field (incl. "Global Online Positioning"), see Chapter 7.3.3 "Virtual Image Field", page 174
  - 29-bit position coordinates (virtual Image Field): objects larger than the real Image Field are possible
- Position monitoring of iDRIVE Scan Systems by backward transformation of actual position values, see Chapter 8.1.3 "Monitoring the Positioning", page 223
- Automatic self-calibration, see Chapter 8.10 "Automatic Self-Calibration", page 284:
  - Optimization of previous functions
  - ASC hardware check
- Jump Mode, see Chapter 8.1.5 "Jump Mode", page 226
- Cycle synchronization, see Chapter 7.4.10 "Synchronization of the RTC6 Clock Cycle and an External Clock Signal", page 216

Controlling the Laser

- The signal levels of the Laser Control Signals are no longer determined by a jumper configuration. Instead, they are software-configured, see **set_laser_control**
- LASER Connector with all Laser Control Signals at the RTC6 PCIe Board slot cover, see Chapter 4.6.1 "LASER Connector", page 75, 9-pin female D-SUB connector only by the SCANLAB laser adapter, see Section "Laser Adapter (Accessory)", page 79
- LASER Connector configurable by software command, see Chapter 7.4.2 "Configuring the LASER Connector", page 192
- Laser Control Signals with 15 ns resolution and 20 mA output current
- Standby signals in a YAG Mode, see Chapter 7.4.4 "YAG Mode 1, YAG Mode 2, YAG Mode 3, YAG Mode 5", page 195
- YAG Mode 5: Time between FirstPulseKiller signal and first laser pulse in YAG Mode is freely programmable, see Chapter 7.4.4 "YAG Mode 1, YAG Mode 2, YAG Mode 3, YAG Mode 5", page 195
- Laser Mode 6: LASERON signal synchronized with a continuously-running LASER1 signal, see Chapter 7.4.6 "Laser Mode 6", page 200
- Pulse Picking Laser Mode, see Chapter 7.4.8 "Pulse Picking Laser Mode", page 203
- Laser pulse period, pulse length or analog output are also programmable within a Polyline between two vectors – where the laser remains on(1), see Short List Command, page 335
- Commands for Position-Dependent Laser Control, Speed-Dependent Laser Control, Vector-Defined Laser Control and Encoder-Speed-Dependent Laser Control, see Chapter 7.4.9 ""Automatic Laser Control"", page 205

(1) Is switched off with the RTC4.

Interfaces for Peripheral Equipment

- 16-Bit Digital Output Port, see Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81, and Chapter 9.1.1 "16-Bit Digital Output Port", page 303:
  - Level of output signals selectable by a jumper (3.3 V or 5 V)
  - LATCH signal for synchronization of data transmission
- 8-Bit Digital Output Port, see Section "8-Bit Digital Output Port", page 84 and Chapter 9.1.2 "8-Bit Digital Output Port", page 304:
  - Provided at the EXTENSION 2 Socket Connector (on the RTC4, this socket connector is named "LASER EXTENSION")
  - LATCH signal for synchronization of data transmission
  - Adjustable "stop output value"
- Analog output ports, see Section "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 77, and Chapter 9.1.4 "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 305:
  - 12 bit resolution
  - 0…10 V (0 V…2.50 V no longer available)
  - Adjustable "stop output value"
- 16-Bit Digital Input Port, see Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81, and Chapter 9.2.1 "16-Bit Digital Input Port", page 310.
  - SYNC signal for synchronization of data transmission
- Programmable debouncing of external start signals, see **bounce_supp** and Section "External Start", page 312
- Regular (periodic) External Starts, see Section "Regular (Periodic) External Starts", page 315
- New interfaces:
  - 2-Bit Digital Input Port and 2-Bit Digital Output Port at the LASER Connector, see Section "2-Bit Digital Input Port", page 77 and Section "2-Bit Digital Output Port", page 77
  - RS-232 interface by an 10-pin socket connector, see Chapter 4.6.5 "RS232 Socket Connector", page 86
  - Stepper motor signals for 2 motors by an on-board 10-pin socket connector, see Chapter 4.6.7 "STEPPER MOTOR Socket Connector", page 90
  - McBSP and ANALOG IN by a 10-pin socket connector, see Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87
- For the RTC6 PCIe Board, there is no longer an IO extension board. Therefore, it does not have a socket connector for installing such a board (on the RTC4, this socket connector is named "EXTENSION 2"; the RTC6 PCIe Board EXTENSION 2 Socket Connector corresponds to the RTC4 "LASER EXTENSION" socket connector).

General Programming

- Utility files for C, C++, C# and Delphi, see Chapter 6.2.2 "Importing Commands", page 98, but no longer utility files for Basic
- Commands for changing access rights to RTC6 PCIe Boards, see Chapter 6.7 "Usage of RTC6 PCI Express Boards by Several User Programs", page 131
- Improved and extended list handling, see Chapter 6.4 "List Handling", page 108
  - RTC6 List Memory with 8,388,608 storage positions
  - RTC6 List Memory free configurable (in 2 RTC6 List Memory areas and 1 protected RTC6 List Memory area)
  - Defining protected subroutines
  - Loading lists with protection
  - Loops in lists and subroutines
  - RTC4-Circular queue mode is not available (but can be coded alternatively)
  - List Status
  - List Execution Status
- "Short" list commands (for example, to change speed, analog output, I/O port, etc.) can be executed without time losses (multiple "short" list commands can be executed within one 10 μs clock cycle, see Section "Normal List Command, Short List Command, Variable List Command and Multiple List Command", page 335
- Functions for error handling and download verification, see Chapter 6.8 "Error Handling", page 133

Laser Marking

- Vectors and arcs
  - Commands for marking ellipses, see Section "Ellipse Commands", page 141
  - Commands for marking helices, see Chapter "3D Commands", page 247
  - Timed arc commands, timed 3D vector commands, see Chapter 8.9 "Timed Commands", page 283
  - Para-Mark Commands and Para-Jump Commands for Vector-Defined Laser Control
- Characters and texts
  - Defining character sets and text strings, see Section "Defining Indexed Character Sets", page 121
  - Commands for marking individual characters and for marking texts (with selectable character set), see Section "Calling Indexed Characters", page 122
  - Commands for marking dates, times and serial numbers (with selectable character set and selectable serial-number-set), see Chapter 7.5 "Marking Dates, Times and Serial Numbers", page 219
- micro_vector[*] Commands (direct position output without Microstepping), see Chapter 8.8 "micro_vector[*] Commands", page 282.

Special Functions

- Synchronization of scan system and laser control
  - Sky Writing, see Chapter 7.2.4 "Sky Writing", page 163
- Pixel Output Mode (marking of bitmaps), see Chapter 8.7 "Pixel Output Mode", page 272:
  - Pixel output frequencies up to 800 kHz, beyond that with Option "UFPM" up to 3.2 MHz
  - RTC4-Pixel Mode 0 with 10 µs clock synchronous output is no longer supported
  - 15 ns resolution
  - 0…100% laser pulse length
  - Pixel-Mode 0 not supported
  - Reading of analog voltages is not supported
  - 3D pixel lines with **set_pixel_line_3d**
- Commands for conditional execution of any list command, see Chapter 9.3.2 "Execution of Conditional Commands", page 317
- Possible wobbel motion shapes include not only circles, but also ellipses, horizontal figure-of-8s, vertical figure-of-8s, and "freely definable wobbel shapes". Options for the orientation of the wobbel shapes are: stationary in space, continuously and automatically adjusted to the current direction of motion, or any other freely assigned motion direction. With "Freely Definable Wobbel Shapes", also the laser power can be varied, see Chapter 8.4 "Wobbel Mode", page 241
- Camming functionality, see Chapter 8.11 "Camming", page 288
- Enhanced signal recording, see **set_trigger[*]**
- Processing-on-the-fly, see Chapter 8.6 "Processing-on-the-fly", page 251
  - 2 encoder input ports (RS-422) with 32-bit counter for Processing-on-the-fly correction with encoder signals on 2 axes, see Chapter 9.3.3 "Synchronization by Encoder Signals", page 320; alternatively: Processing-on-the-fly correction with McBSP signals, see Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 322
  - 29-bit coordinates (virtual Image Field): objects larger than the real Image Field are possible, see Chapter 7.3.3 "Virtual Image Field", page 174, and Chapter 8.6.6 "Virtual Image Field with Processing-on-the-fly", page 261
  - Up to 8 objects within the Processing-on-the-fly track delay (between trigger and marking position), see Section "External Start", page 312
  - Accurate "External Start":
    If accordingly configured by **set_control_mode**, the encoder counter can be reset by external start signals for synchronizing a Processing-on-the-fly process. The reset occurs fully simultaneously (without 10 µs jitter) with the external start signal.
  - Compensation of 2D motions (xy-positioning stage)
  - Encoder-based Processing-on-the-fly correction for the z axis ("FlyZ correction"), see Chapter 8.6.11 "Processing-on-the-fly Correction for the z Axis", page 266

Notice!

- The encoder counter direction of RTC4 boards (includes RTC4 Ethernet Board and RTC4 *SCAN*alone Board) is opposite to that of RTC5 boards and RTC6 boards.
  Therefore, when changing RTC board you need to either adapt the cabling or your user program accordingly.

2.11 Notes for RTC5 Users

RTC6 PCIe Board can control:

- Scan systems with SL2-100 interface
- Scan systems with XY2-100 interface (in conjunction with the XY2-100 Converter (Accessory))

The following is an overview of key differences between RTC6 PCIe Boards and RTC5 boards.

Also described are possible approaches for migrating RTC5 program code for use with RTC6 PCIe Boards.

Notes

- Refer also to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.

2.11.1 RTC6 PCIe Boards vs. RTC5 Boards

Compared to the RTC5 board family, the RTC6 PCIe Board uses newer and more powerful hardware components. This applies in particular to the DSP, the FPGA and the main memory.

Existing user programs only require a few changes, see Chapter 2.12.2 "Adapting RTC5 Source Code for the RTC6 PCIe Board", page 62.

Functionality Changes

- The RTC6 PCIe Board executes short vectors[1] faster than RTC5 boards, see **set_dsp_mode**.
- The RTC6 List Memory features 8,388,608 (= 2^{23}) list positions. List 1 and List 2 are each initialized with 4,194,304 (= 2^{22}) list positions.
- The Pixel Output Mode is significantly enhanced, see **set_pixel_line**.
  - RTC6 PCIe Boards without Option "UFPM" can achieve pixel output frequencies of up to 800 kHz out-of-the-box.
  - RTC6 PCIe Boards, with Option "UFPM", even up to 3.2 MHz.
  - With **set_pixel_line** it is now possible to digitally output laser power control values. This can be done either at the 8-Bit Digital Output Port (EXTENSION 2 Socket Connector) or 16-Bit Digital Output Port (EXTENSION 1 Socket Connector).
- Extended measurement-value recording with the RTC6 PCIe Board: 4 (8) recording channels, each storing 8,388,608 (4,194,304) data values, see **set_trigger4** (**set_trigger8**). Practically any amount of data can be recorded (ring buffer).

(1) "dashed lines"

- 8 3D correction tables can be used on the RTC6 PCIe Board simultaneously with 8 recording channels (see **set_trigger[*]**) and the complete RTC6 List Memory. With **number_of_correction_tables**, a user-defined limit to less than 8 can be set to check for incorrect user input.
- The virtual Image Field of the RTC6 PCIe Board is now ±28 bit = 29 bit.
- You can set an appropriate Tracking Error compensation value for 3D systems with different Tracking Error behavior between the xy axes and z axis, see **set_timelag_compensation**.
- The output synchronization of the RTC5 is replaced by a cycle synchronization with the RTC6 PCIe Board, see Chapter 7.4.10 "Synchronization of the RTC6 Clock Cycle and an External Clock Signal", page 216.

Parameter Changes

– Laser Delays (parameter `LaserOnDelay` and `LaserOffDelay` of **set_laser_delays**) as well as `LaserOnShift` with Sky Writing commands have a higher resolution of 1/64 µs.
For downward compatibility with RTC5 boards, see step 3, page 63.
– In fact the parameter `Timelag` of **set_sky_writing_para** is specified in 64-bit IEEE floating point format ("double"; 1.0 = 1 µs) but on the RTC6 PCIe Board it is actually executed with a resolution of 1/64 µs resolution.

2.11.2 RTC6 PCIe Boards and SCANahead Systems

- See Chapter 8.13 "SCANahead Functionality", page 291:
  – Only the RTC6 PCIe Board allows usage of SCANahead Servo Control (for example, excelli*SCAN* series).
  – If you configure the RTC6 PCIe Board by **set_scanahead_params** for operation with an SCANahead System, the laser control then automatically takes the `PreviewTime` of the SCANahead Servo Control into account. This ensures fuller usage of dynamics and precision.
  – **activate_scanahead_autodelays** lets you simply and quickly place the SCANahead System into operation. Neither Laser Delays nor Scanner Delays need to be determined or set.
- Refer to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.

2.11.3 Restrictions with RTC6 PCIe Boards

Availability of Technical Variants

RTC6 boards are available as RTC6 PCIe Boards and RTC6 Ethernet Boards.

Peripheral Interfaces

The ADC add-on card (#0121126, optional accessory for the RTC5 PCI board) cannot be used with the RTC6 PCIe Board.

The RTC6 PCIe Board supplies the two 10 V analog input ports ANALOG IN0 and ANALOG IN1 at the McBSP/ANALOG Socket Connector, page 87 directly.

Functionality

With RTC6 PCIe Boards, the McBSP/ANALOG Socket Connector, page 87 cannot be used in SPI mode.

Known restrictions are mentioned in `RTC6_Release_Notes_n.n.n.pdf`.

2.12 Source Code for RTC6 User Programs

2.12.1 Creating New RTC6 Source Code

To create new source code for RTC6 user programs, proceed as described in Chapter 6.2 "Initialization and Program Start-Up", page 98 while observing Chapter 2.12.2 "Adapting RTC5 Source Code for the RTC6 PCIe Board", page 62.

2.12.2 Adapting RTC5 Source Code for the RTC6 PCIe Board

User programs written for the RTC5 only become usable on the RTC6 PCIe Board, if you modify the program code as described in the following.

This is true even if program flow remains unchanged.

Mandatory Steps

(1) Ensure the following prerequisites are met:

- RTC6 PCIe Board is installed
- RTC6 board driver is installed for your operating system
- The following files are present:
  `RTC6DLL.dll` or `RTC6DLLx64.dll`,
  `RTC6OUT.out` (with RTC6 PCIe Boards) or
  `RTC6ETH.out` (with RTC6 Ethernet Boards),
  `RTC6RBF.rbf`,
  `RTC6DAT.dat`,
  `*.CT5` (correction file(s))
  Note, `RTC6OUT.out`/`RTC6ETH.out`, `RTC6RBF.rbf`, `RTC6DAT.dat` must all be in the same folder.

– The import declarations were imported in your software and the calling convention `std_call` has been applied, see Chapter 6.2 "Initialization and Program Start-Up", page 98.

(2) Take into account these command name changes of the RTC6 command set:

– Do *not* generally change `_rtc5_` to `_rtc6_`!

– Change:

`init_rtc5_dll` to **`init_rtc6_dll`**

`free_rtc5_dll` to **`free_rtc6_dll`**

`rtc5_count_cards` to **`rtc6_count_cards`**

– Notice: Do *not* generally change **`set_rtc5_mode`** to **`set_rtc6_mode`** – here see step 3!

(3) **set_rtc6_mode** is set automatically at program start. Then an expanded parameter resolution is available for

– z coordinates of 20 bits

– Laser Delays of 1/64 µs

In contrast, the control command **set_rtc5_mode** ensures downward compatibility of the parameters ("RTC5 Compatibility Mode").

– Laser Delays:

In RTC5 Compatibility Mode, the RTC6 board multiplies the values specified for `LaserOnDelay` and `LaserOffDelay` (of **set_laser_delays**) automatically by 32. This also applies to the values specified for `LaserOnShift` (of **set_sky_writing**) and `LaserOnShift` (of **set_sky_writing_para**).

– z coordinates:

In RTC5 Compatibility Mode, the RTC6 board multiplies the z coordinates and defocus values (for example, `Shift` of **set_defocus**) automatically by 16.

For further modifications and optimizations of your code, see the following chapter.

Optional Steps and Notes on Further Modification and Optimization

- Commands which have been removed from the RTC6 command set or are no longer supported must be removed or replaced in the program code, see Chapter 2.13.2 "Removed from RTC6 Command Set", page 64.
- Check if command changes might affect your user program, see Chapter 2.13.1 "Changes to the RTC6 Command Set", page 64, and modify your program code accordingly.
- Take into account the changed behavior of **load_program_file**.
  Notice: **load_program_file** now stops lists without warning. The single-board command can even be used again after a version conflict.

  – **Note on enhanced parameter resolution**
  Take into account the enhanced parameter resolution of z coordinates when using **set_rtc6_mode**. See also step 3, page 63.

  – **Note on changed time behavior**
  The replacement of the automatic delay adjustment can result in faster time behavior for short vectors.
  By `set_dsp_mode( 2 )`, the original RTC5 time behavior can be restored, see **set_dsp_mode**.

2.13 Changes to the RTC6 Command Set

The most RTC5 commands are covered in the RTC6 command set. The following chapter describes the few differences:

- *Changed* commands, see Chapter 2.13.1 "Changes to the RTC6 Command Set", page 64
- *Removed* commands, see Chapter 2.13.2 "Removed from RTC6 Command Set", page 64
  list commands list command

2.13.1 Changes to the RTC6 Command Set

Meanings:

con	control command
nor	Normal List Command
ds	Delayed Short List Command

* Note: RTC6 command with extended functionality.

2.13.2 Removed from RTC6 Command Set

3 Safety During Installation and Operation

Read these operating instructions completely before you proceed with installing and operating the RTC6 PCIe Board.

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

The following conventions apply to safety notices in this manual:

- ⚠ Safety notices which draw attention to severely injuries or even death are identified by the hazard symbol and the signal word “Warning!“.
- ⚠ Safety notices which draw attention to a health hazard are identified by the hazard symbol and the signal word “Caution!“.
- Safety notices that recommend proper use of the device or warn against possible damage to property are (without hazard sign) only identified by the signal word “Notice!“.

3.1 Safe Operation

Notice!

- Carefully check your user program before running it. Programming errors can cause a break down of the system. In this case neither the laser nor the scan system can be controlled.
- Protect the board from humidity, dust, corrosive vapors and mechanical stress.

Notice!

- For storage and operation of the board, avoid electromagnetic fields and static electricity. These can damage the electronics on the board. For storage, always use the antistatic bag the board 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.

3.2 Laser Safety

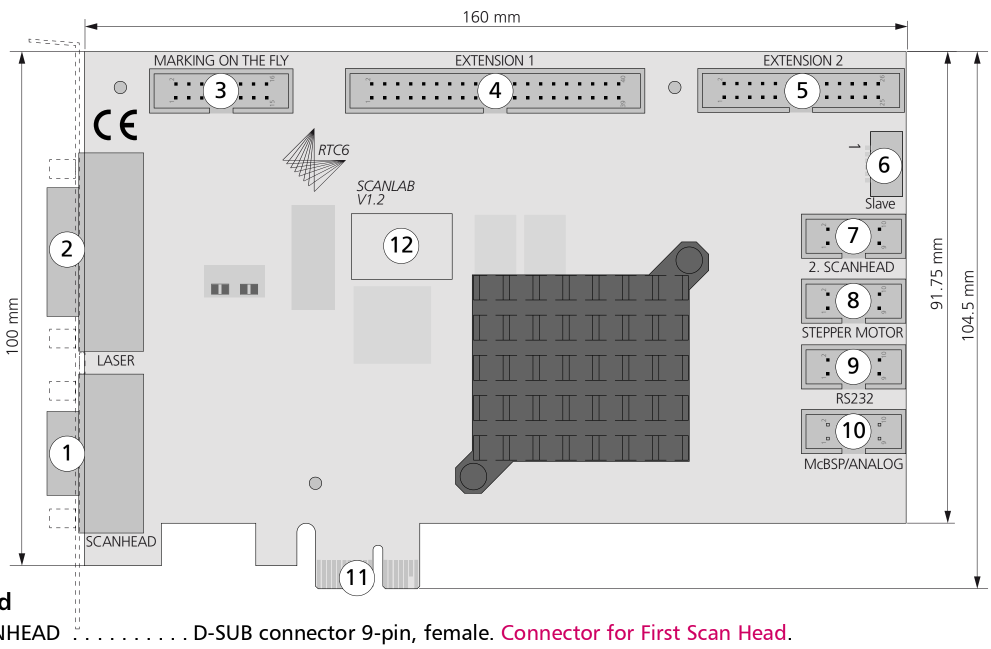
The RTC6 PCIe Board is intended for controlling scan systems and lasers. Therefore all relevant laser safety directives must be known and applied before installation and operation. The customer is solely responsible for ensuring the laser safety of the entire system.

⚠ **Caution!**

- 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.
- Observe all laser safety instructions as described in your scan system manual, chapter “Safety during Installation and Operation“.
- *Always turn on the PC and the power supply for the scan head first before turning on the laser. Otherwise, there is the danger of uncontrolled deflection of the laser beam.*
  SCANLAB recommends the use of a shutter to prevent uncontrolled emission of laser radiation.

4 RTC6 PCIe Board – Layout and Interfaces

4.1 Layout – Upper Side



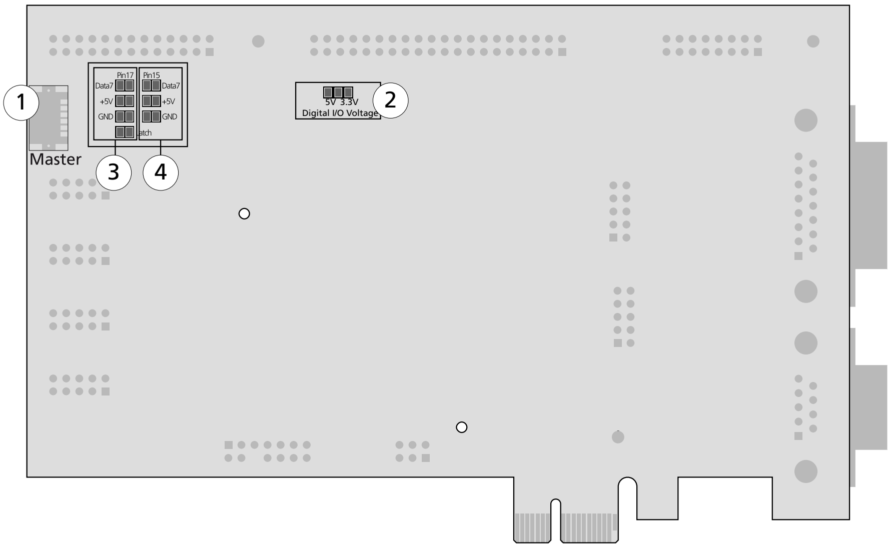
Legend

1. SCANHEAD . . . . D-SUB connector 9-pin, female. Connector for First Scan Head. Pin-out see Figure 10.
2. LASER . . . . D-SUB connector 15-pin, female. Connector for the laser incl. digital and analog output ports. Pin-out see Figure 17.
3. MARKING ON THE FLY . . 16-pin socket connector. With encoder input ports for Processing-on-the-fly applications. Pin-out see Figure 25.
4. EXTENSION 1 . . . . 40-pin socket connector. With a 16-Bit Digital Input Port and a 16-Bit Digital Output Port. Pin-out see Figure 22.
5. EXTENSION 2 . . . . 26-pin socket connector. With an 8-Bit Digital Output Port, for example, for alternative laser control. Pin-out see Figure 24.
6. Slave . . . . 6-pin socket connector. To connect with another RTC6 PCIe Board for clock synchronization. For details, see Chapter 4.4 "Master Socket Connector, Slave Socket Connector", page 68.
7. 2. SCANHEAD . . . . 10-pin socket connector. Connector for Second Scan Head. Pin-out see Figure 11.
8. STEPPER MOTOR . . . . . . 10-pin socket connector. For control of 1…2 stepper motors. Pin-out see Figure 30.
9. RS232 . . . . 10-pin socket connector. For controlling external devices by RS-232 interface. Pin-out see Figure 27.
10. McBSP/ANALOG. . . . . . 10-pin socket connector. Serial extension interface. Pin-out see Figure 28.
11. PCIe-x-1 Plug . . . . Interface to the PC. See Chapter 4.3 "Interface to the PC", page 68.
12. Label with serial number.

5

RTC6 PCIe Board: upper side.

4.2 Layout – Lower Side



Legend

1. Master . . . . 6-pin socket connector. To connect with another RTC6 PCIe Board for clock synchronization. For details, see Chapter 4.4 "Master Socket Connector, Slave Socket Connector", page 68.
2. Solder jumper field A . . . To configure the level of the output signals at the EXTENSION 1 Socket Connector. For details, see "Solder Jumper Field A – Configuring Output Signal Level at EXTENSION 1 Socket Connector", page 42.
3. Solder jumper field B . . . To configure the signal at EXTENSION 2 Socket Connector pin (17). For details, see "Solder Jumper Field B – Configuring Pin (17) of EXTENSION 2 Socket Connector", page 43.
4. Solder jumper field C . . . To configure the signal at EXTENSION 2 Socket Connector pin (15). For details, see "Solder Jumper Field C – Configuring Pin (15) of EXTENSION 2 Socket Connector", page 44.

6

RTC6 PCIe Board: lower side.

4.3 Interface to the PC

The interface to the PC is the PCIe-x-1 connector of the RTC6 PCIe Board, see Figure 5.

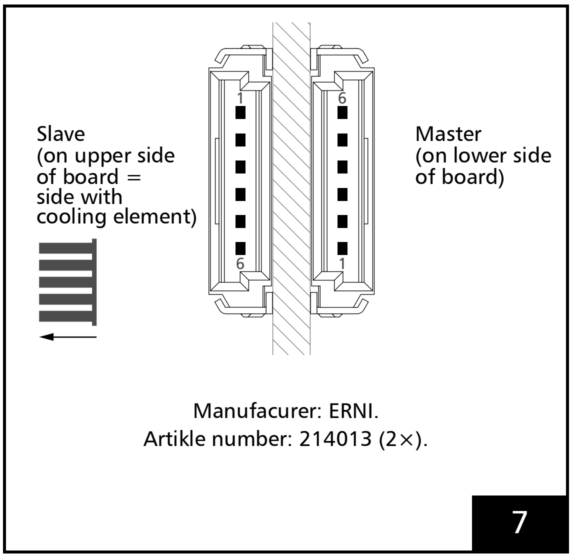
The RTC6 PCIe Board can be installed into any Windows PC with a PCI-Express bus interface and at least one free PCI-Express slot.

Notice!

- The RTC6 PCIe Board does not support power-saving modes that switch off power to the PCIe bus. Accordingly, you must disable standby or sleep modes of the operating system. See also Section "Notes", page 38.

4.4 Master Socket Connector, Slave Socket Connector

The Slave socket connector as well as the Master socket connector have 6 pins, see Figure 7. The Slave socket connector is located on the upper side of the RTC6 PCIe Board, see Figure 5. The Master socket connector is located on the lower side, see Figure 6[1].



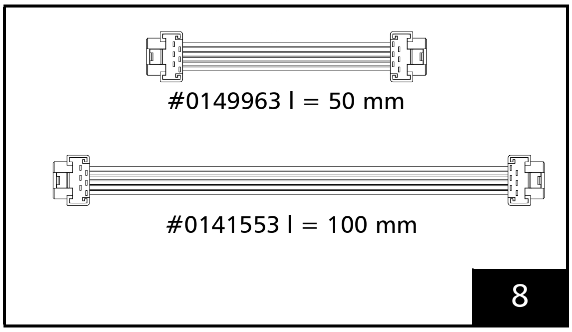
Master socket connector and Slave socket connector. The pitch of the pins is 1.27 mm.

Purpose of the both socket connectors is to make a clock cycle synchronization of several RTC6 Ethernet Boards possible. Then they must be connected pairwise with each other by the Master and Slave socket connectors. Always connect a Master connector of a board to the Slave connector of another board by a suitable cable (available from SCANLAB, see Figure 8). The necessary information for assembling your own cables is shown in Figure 9.

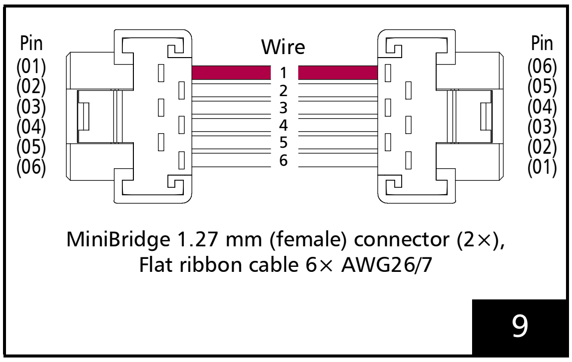
Interconnected RTC6 PCIe Boards should (recommended) be plugged into adjacent PCIe slots.

Important: In order to use the master/slave functionality, see prerequisites in Chapter 6.6.3 "Master/Slave Operation", page 127.

See also Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 311.



SCANLAB Master/Slave connecting cable.



Cable to connect the Master socket connector and Slave socket connector: requirements. Keep length as short as possible!

Notes

- With **master_slave_config** the master/slave interface properties can be configured individually for each RTC6 board.

(1) Vice versa with RTC6 Ethernet Boards.

4.5 Interfaces to Scan System

4.5.1 Scan Head Connectors and Transfer Protocol

The SCANHEAD Connector and the optionally activated 2. SCANHEAD Socket Connector are available for digitally controlling scan systems.

At those connectors, scan-system control values are transmitted and scan-system status signals received. Each scan head connector can transmit data for up to two axes. 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 transfer between the RTC6 PCIe Board and the scan system is in accordance with the SL2-100 protocol. The XY2-100 Converter (Accessory) is available for converting the signals (for data transmission according to the XY2-100 protocol).

If neither the Option "Second Scan Head Control" nor the Option "3D" is enabled, only the Connector for First Scan Head outputs signals for an xy scan system.

If the Option "Second Scan Head Control" is enabled, two xy scan systems can be simultaneously controlled by one RTC6 PCIe Board.

If the Option "3D" is enabled, then a 3-axis scan system can be controlled by Connector for First Scan Head and Connector for Second Scan Head (if the Connector for First Scan Head has been assigned a 3D correction table). Signals can then be outputted by the Connector for First Scan Head to an xy scan head – and by both channels of the Connector for Second Scan Head to the 3rd axis (z axis).

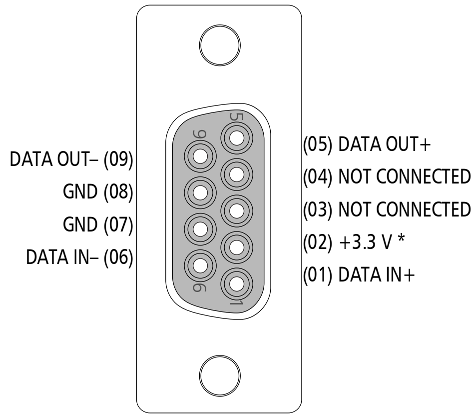
If *both* options (Option "Second Scan Head Control" *and* Option "3D") are enabled, the assignment of the correction tables determines which signals (xy or z) are to be outputted by which connector, see also Section "2D and 3D Correction Files", page 179.

If several RTC6 PCIe Boards with enabled Option "3D" are installed in a PC, then that many 3-axis systems can be simultaneously controlled.

SCANHEAD Connector

Connector for First Scan Head

The pin-out of the SCANHEAD Connector (D-SUB 9-pin female) is shown in Figure 10.



* DO NOT CONNECT

10

SCANHEAD Connector (9-pin D-SUB connector, female): pin-out.

At the differential DATA OUT output port, the control values for the scan system are outputted.

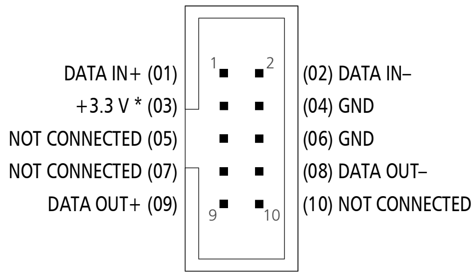
The differential DATA IN input port receives the status signals transmitted by the scan system.

Pin (02) supplies 3.3 V power for the XY2-100 Converter (Accessory) (or a Polymer Optical Fiber converter for optical data transmission). This voltage should not be used for other purposes.

2. SCANHEAD Socket Connector

Connector for Second Scan Head

The pin-out of the 2. SCANHEAD Socket Connector (10-pin socket connector) is shown in Figure 11.



* DO NOT CONNECT

Identical in construction to Würth 61201021621.

11

2. SCANHEAD Socket Connector: pin-out. The pitch of the pins is 2.54 mm. Signals for second scan head are only outputted with enabled Option "Second Scan Head Control".

Second Scan Head Slot Cover (Accessory)

SCANLAB recommends using an additional slot cover for connecting a second scan system or a z axis to the 2. SCANHEAD Socket Connector, see Chapter 2.8.4 "Slot Cover with 9-pin D-SUB Connector for 2. SCANHEAD Socket Connector", page 46.

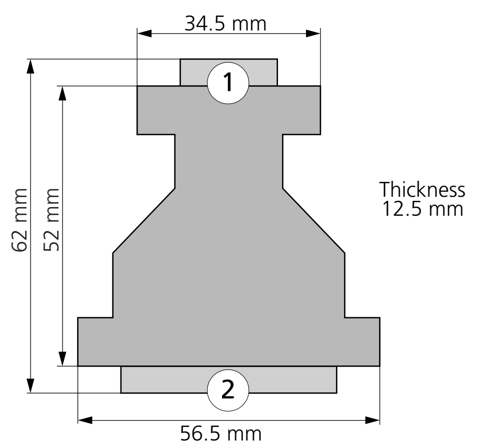
4.5.2 XY2-100 Converter (Accessory)

The SCANLAB XY2-100 Converter (Accessory) (#0125377) converts:

- SL2-100 protocol-compliant control signals (20 bit) to XY2-100 protocol-compliant control signals (16 bit)
- Scan system XY2-100 protocol-compliant status signals to SL2-100 protocol-compliant status signals, see also Chapter 7.3.7 ”Status Monitoring and Diagnostics“, page 188

When controlling scan systems, the XY2-100 Converter (Accessory) introduces a 10 µs signal propagation delay. To compensate for this, the LaserOn Delay and LaserOff Delay must be increased by 10 µs each, see **set_laser_delays**.

The dimensions are shown in Figure 12.



Legend

1. 9-pin D-SUB plug, male
2. 25-pin D-SUB connector, female

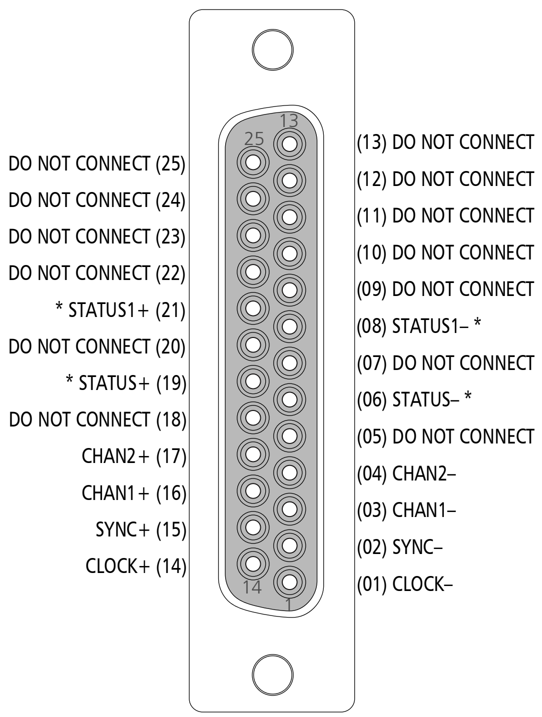
12

XY2-100 Converter (Accessory): dimensions.

The 9-pin D-SUB (male) plug of the XY2-100 Converter (Accessory) should be directly plugged into the SCANHEAD Connector of the RTC6 PCIe Board or, (by a short-as-possible 1:1 cable) connected to the corresponding 2. SCANHEAD Socket Connector pins of the RTC6 PCIe Board. For the pin-out, see Figure 10 and Figure 11.

The 25-pin D-SUB connector (female) of the XY2-100 Converter (Accessory) is compatible with *scan heads* that provide an XY2-100 standard interface.

The pin-out is shown in Figure 13.



13

XY2-100 Converter (Accessory): pin-out of the 25-pin D-SUB connector (female).

* For iDRIVE Scan Systems, the STATUS± channel is the status channel of axis 2 (x axis; this channel is then also called STATUS2±) and the STATUS1± channel is the status channel of axis 1 (y axis). For other scan systems, the STATUS1± iDRIVE Scan Systemschannel can not be used: “DO NOT CONNECT”.

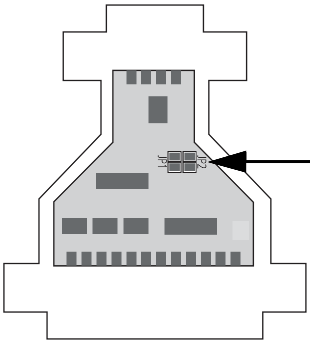
The data channels CHAN1 and CHAN2 transmit control values to the scan head. The SYNC and CLOCK channels transmit synchronization and clock signals to the scan system.

The STATUS channel (and, if appropriate, the STATUS1 channel) receives XY2-100-compliant status signals returned by the scan system.

If cables longer than 20 m (not recommended) are used for data transmission between the XY2-100 Converter (Accessory) and the scan system, then the synchronization of bidirectional communication between the scan system and the RTC6 PCIe Board should be configured.

This can be accomplished by two solder jumpers in the XY2-100 Converter (Accessory).

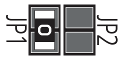
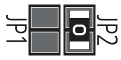
To do so, carefully open the housing of the XY2-100 Converter (Accessory) by its 4 clip latches[1]. The solder jumpers **JP1** and **JP2** are on the PCB, see arrow in Figure 14.



14

XY2-100 Converter (Accessory): positions of solder jumpers **JP1** and **JP2** on the PCB.

The following table shows the possible jumper settings and the corresponding cable lengths. Other jumper settings are not allowed. Cable lengths above 20 m are not recommended.

Jumper setting	Cable length*
JP1 closed **JP2** open (default configuration) 	0 m to 20 m
JP1 open **JP2** closed 	20 m to 40 m
* The cable length range mentioned here for the respective jumper configuration depends on the used cable type and may differ from the values mentioned above. Cable lengths over 20 m are generally not recommended (and require extensive checks by users prior to productive use).	

Notes

- Observe the note on **get_head_status**, page 222.

(1) For example, using a slotted screwdriver.

4.5.3 Data Cables (Accessories)

For transmission of the data signals between the RTC6 PCIe Board (or the XY2-100 Converter (Accessory)) and the scan system, appropriate cables are obtainable from SCANLAB. Data cables are generally not included in the scope of delivery.

For SCANLAB scan systems equipped with an SL2-100 interface (9-pin female D-SUB connector), data cables are available for electrical transmission, for optical transmission (only upon request) also optical fiber cables(1).

For SCANLAB scan systems equipped with a XY2-100 interface (25-pin female D-SUB connector) solely electrical cables are obtainable.

Scan systems equipped with an optical interface (XY2-100-O) cannot be controlled by the RTC6 PCIe Boards.

With self-constructions, SCANLAB recommends the following design for electrical data transmission:

- For SL2-100 protocol-compliant data transmission, see Figure 15, the cable should be fitted with 9-pin male D-SUB connectors at both ends. The two channels DATA IN± and DATA OUT± must consist of twisted cable pairs and be *cross-connected* at both D-SUB connectors (for example, so that the DATA OUT signal of the RTC6 PCIe Board flows to the DATA IN input of the scan system). The cable length should not exceed 25 m. SCANLAB recommends a cable impedance of 110 Ω, independent from the cable length.
- For XY2-100-compliant data transmission, see Figure 16, the cable must have identical 25-pin (male) D-SUB connectors at both ends. The five (or six) channels SYNC±, CHAN1±, CHAN2±, STATUS± (and STATUS1±) and CLOCK± must consist of twisted cable pairs. Together with an XY2-100 Converter (Accessory) with standard jumper configuration, the data cable should not be longer than 20 m. If a longer data cable is needed, then the solder jumper setting of the XY2-100 Converter (Accessory) need to be reconfigured.
- For XY2-100-compliant data transmission, the controller end of the data cable must be fitted with a ferrite identical in construction to Würth WE 74271132.

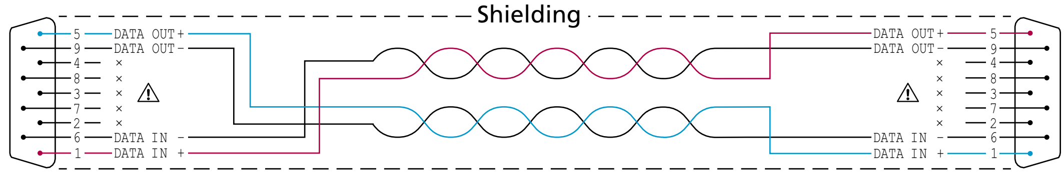
Independently from the data transmission protocol, the following applies in addition:

- The data cable must have coaxial copper braided shielding.
- The D-SUB connectors must have fully shielded metal housings.
- The electrical connection of the cable's braided shielding to the D-SUB housing *must not* be implemented as a wire. Instead, the cable's braided shielding should be *coaxially* connected to the D-SUB housing by shielded clamps.

Some *scan heads* have only a single connector to provide both the operating voltage and the data signals. For these *scan heads*, SCANLAB recommends implementing a cabling solution which uses a separate cable for data. The cable should have the properties as already described, see above.

(1) The optical fiber cables, too, are attached by 9-pin D-SUB connectors. Optical conversion (Polymer Optical Fiber conversion) for optical data transmission takes place inside the D-SUB connectors. The operating voltage for Polymer Optical Fiber conversion is supplied at the RTC6 PCIe Board scan head connectors and the digital interface of the scan system.

Cable for SL2-100 protocol-compliant data transmission



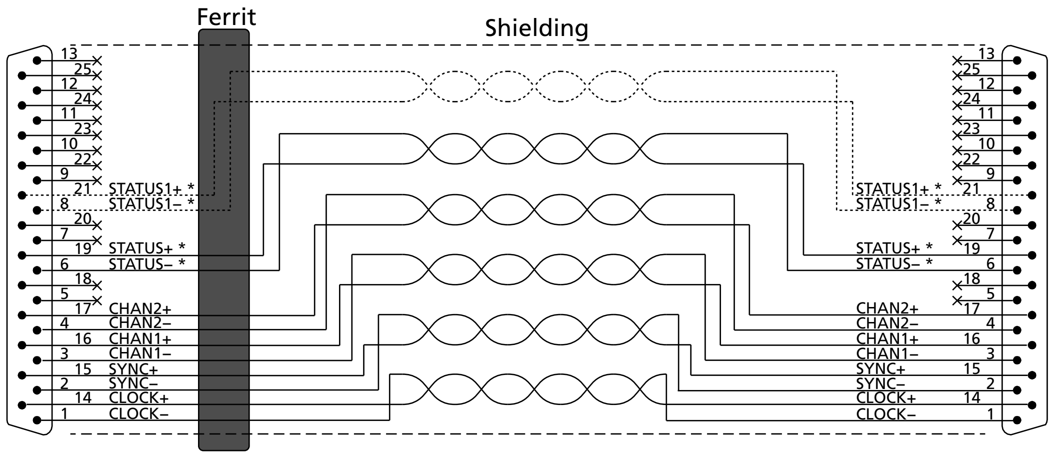
← RTC board
9-pin D-SUB connector, male

Scan system →
9-pin D-SUB connector, male

15

Cable for data transmission compliant to SL2-100 protocol: requirements and pin assignments: requirements and pin assignments.

Cable for XY2-100 protocol-compliant data transmission



To RTC board
25-pin D-SUB connector, male

To scan head
25-pin D-SUB connector, male

* For iDRIVE Scan Systems, the following applies:
The STATUS± channel is the status channel for axis 2 (x axis; this channel is also called STATUS2± there).
The STATUS1± channel is the status channel for axis 1 (y axis).
For other scan systems, the STATUS1± channel is not needed.

16

Cable for data transmission compliant to XY2-100 protocol: requirements and pin assignments.

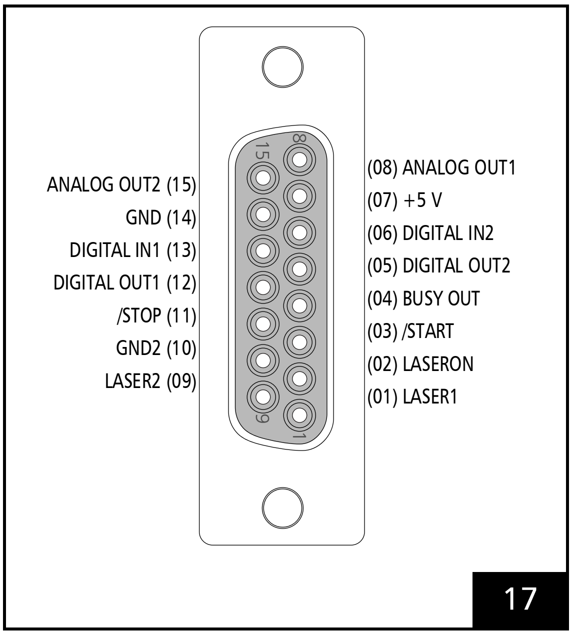
4.6 Interfaces for the Laser and Peripheral Equipment

4.6.1 LASER Connector

- For technical data see Chapter 15 "Technical Specifications – RTC6 PCIe Board", LASER Connector, page 951.

The LASER Connector is a 15-pin D-SUB connector (female). It is located on the slot cover of the RTC6 PCIe Board, see Figure 5.

The pin-out is shown in Figure 17.



LASER Connector (15-pin D-SUB connector, female): pin-out. On RTC6 PCIe Boards without Option "DC/DC Converter", GND2 are GND identical.

Notice!

- On RTC6 PCIe Boards with Option "DC/DC Converter", GND2 and the Laser Control Signals LASERON, LASER1 and LASER2 at the LASER Connector and EXTENSION 2 Socket Connector are galvanically decoupled from GND.[a]
- On RTC6 PCIe Boards without Option "DC/DC Converter", GND2 are GND identical.

(a) RTC6 PCIe Boards: GND is the PC ground. RTC6 Ethernet Boards: GND is the ground at the POWER connector.

Notice!

- If you want to use the RTC6 PCIe Board in conjunction with laserDESK, observe the following:
  – laserDESK uses additional pins on the EXTENSION 1 Socket Connector, for example, to control the laser.
  – Refer to the extended documentation for connecting the laser. Refer to laserDESK online help (alternatively available from SCANLAB or in laserDESK-zip, which can be downloaded from the SCANLAB website).

Laser Control Signals

Laser Control Signals are LASERON, LASER1 and LASER2. They are digital TTL level signals and are referenced to GND2.

Laser Control Signals LASER1 and LASER2 differ depending on the set laser mode, see following table and the timing diagrams in Chapter 7.4 "Laser Control", page 189.

	LASER1 pin (01)	**LASER2** pin (09)
CO_2 Mode	Modulation pulse 1, standby signal	Modulation pulse 2, standby signal
YAG Mode 1, YAG Mode 2, YAG Mode 3, YAG Mode 5	Q-Switch Signal	FirstPulseKiller signal
Laser Mode 4	Standby signal	FirstPulseKiller signal
Laser Mode 6	Standby signal	–

All Laser Control Signals can be set by **set_laser_control** to either active-LOW or active-HIGH logic. "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 TTL laser control signal level according to the specifications of your laser control. Observe the documentation of your laser.

Pin (01), (02) and (09) of the LASER Connector can be configured by **config_laser_signals** and **config_laser_signals_list**, see also Chapter 7.4.2 "Configuring the LASER Connector", page 192.

External Control Signals

The external control signals (input signals) are:

- /START
- /STOP

See also Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 311.

BUSY List Execution Status

The BUSY list execution status is available as the BUSY OUT signal (identical to BUSY OUT, page 953) at pin (04).

When the BUSY list execution status is set, the BUSY OUT signal is HIGH.

See also Chapter 6.4.3 "List Execution Status", page 111.

2-Bit Digital Input Port

The RTC6 PCIe Board provides a 2-Bit Digital Input Port:

- DIGITAL IN1
- DIGITAL IN2

For programming the input port see Chapter 9.2.2 "2-Bit Digital Input Port", page 310.

2-Bit Digital Output Port

The RTC6 PCIe Board provides a buffered 2-Bit Digital Output Port:

- DIGITAL OUT1
- DIGITAL OUT2

For programming the output port see Chapter 9.1.3 "2-Bit Digital Output Port", page 304.

12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2

The RTC6 PCIe Board provides:

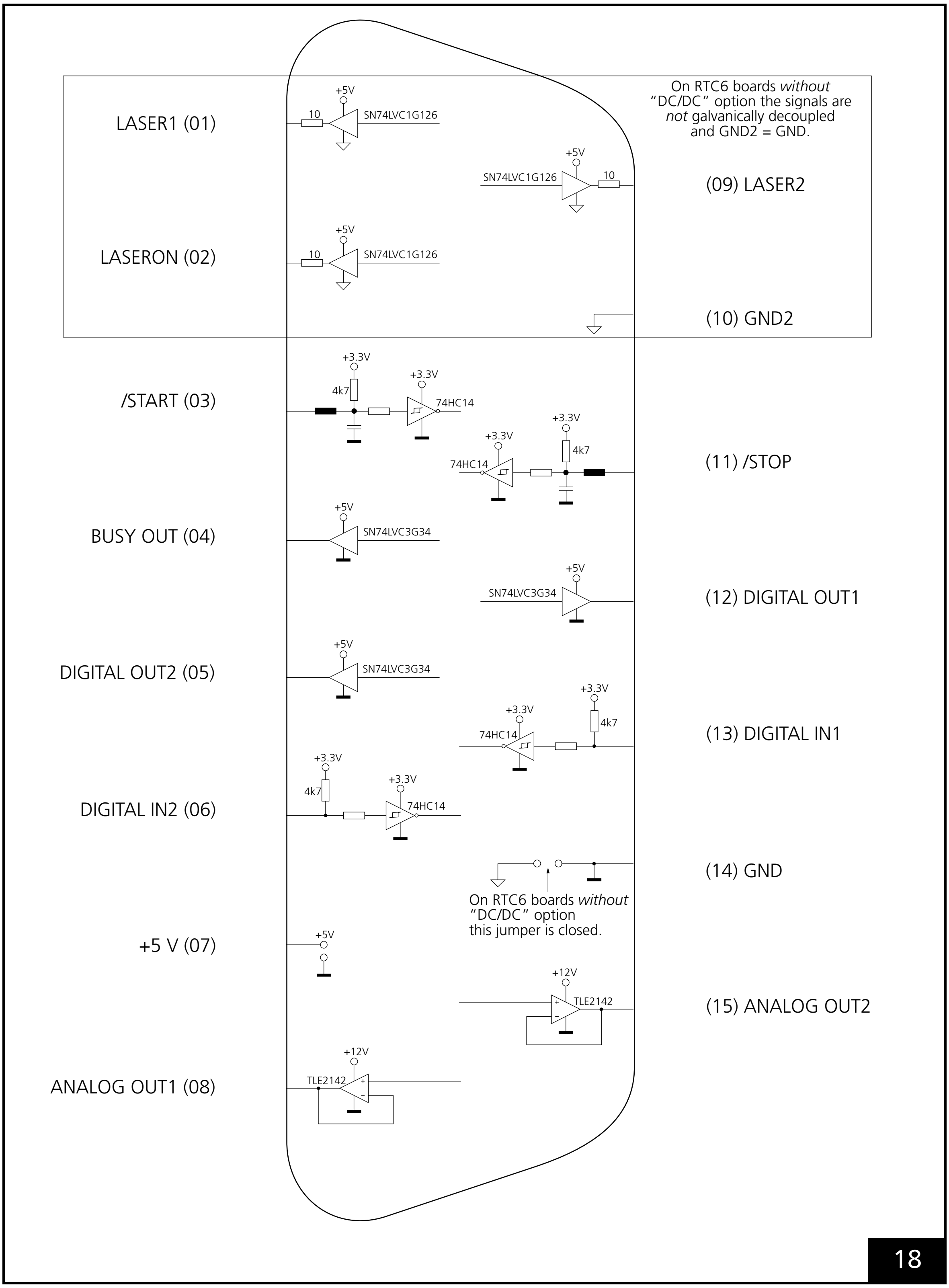
- 12-Bit Analog Output Port 1
  ANALOG OUT1
- 12-Bit Analog Output Port 2
  ANALOG OUT2(1)

For programming the output ports, see Chapter 9.1.4 "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 305.

Input and Output Wiring

The input and output wiring of the LASER Connector is shown in Figure 18.

(1) The signal of ANALOG OUT2 is also available by the MARKING ON THE FLY Socket Connector, see Section "Analog Output Port", page 85.



LASER Connector, see also Figure 17: input/output wiring.
See also Section "Option "DC/DC Converter"", page 39.

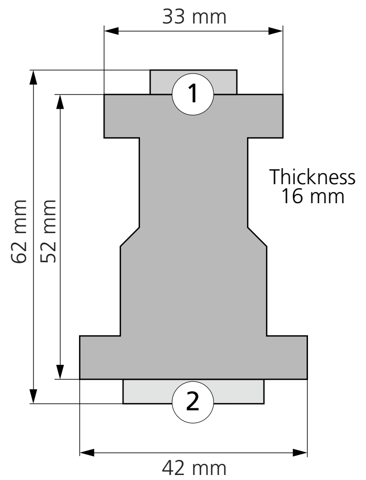
Laser Adapter (Accessory)

The SCANLAB laser adapter (accessory; #0114774) is plugged into the 15-pin LASER Connector of the RTC6 PCIe Board. Then its 9-pin D-SUB connector (female) provides the same signals and pin-out as the RTC4 9-pin LASER connector.

Notes

- The laser adapter can *not* be directly plugged into the RTC6 PCIe Board if an XY2-100 Converter (Accessory) is already plugged in.
- The BUSY OUT signal, 2-Bit Digital Input Port and 2-Bit Digital Output Port are *not* available at the laser adapter's 9-pin D-SUB connector.

The dimensions of the laser adapter is shown in Figure 19.



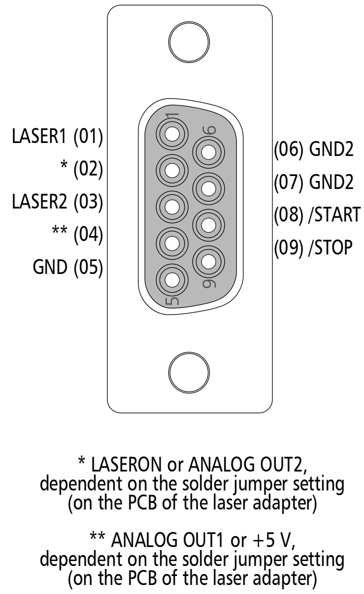
Legend

1. 9-pin D-SUB connector, female
2. 15-pin D-SUB plug, male

19

Laser adapter (accessory): dimensions.

The pin-out of the 9-pin D-SUB connector is shown in Figure 20.

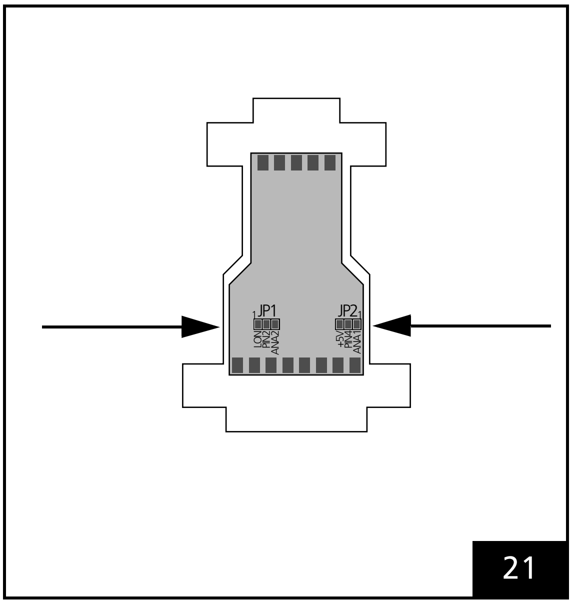


20

Laser adapter (accessory): pin-out of the 9-pin D-SUB connector, female.

The signals at pins (02) and (04) of the 9-pin D-SUB connector can be selected by two solder jumpers in the laser adapter. To do so, carefully open the laser adapter housing by its 4 clip latches (for example, using a screwdriver). The solder jumpers **JP1** and **JP2** are on the PCB of the laser adapter between the two D-SUB connectors.

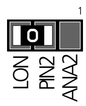
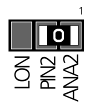
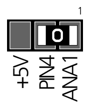
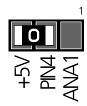
The positions of the solder jumpers on the PCB is shown in Figure 21.



21

Laser adapter (accessory): position of solder jumper **JP1** and **JP2** on the printed circuit board.

The following table shows the possible jumper settings. Other jumper settings are not allowed.

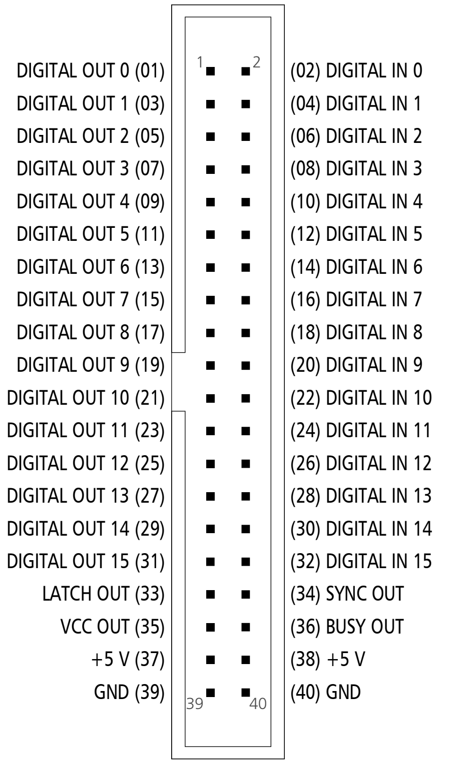
Solder jumper JP1: signal at pin (02)	Position 1-2  LASERON (default configuration)	Position 2-3  ANALOG OUT2
Solder jumper JP2: signal at pin (04)	Position 1-2  ANALOG OUT1 (default configuration)	Position 2-3  +5 V

4.6.2 EXTENSION 1 Socket Connector

- For technical data see Chapter 15 "Technical Specifications – RTC6 PCIe Board", EXTENSION 1 Socket Connector, page 952.

The EXTENSION 1 Socket Connector has 40 pins. It is located on the upper side of the RTC6 PCIe Board, see Figure 5.

The pin-out is shown in Figure 22.



Identical in construction to Würth 61204021621.

22

EXTENSION 1 Socket Connector: pin-out. The pitch of the pins is 2.54 mm.

Configuring the Output Signal Level

With the solder jumper field A on the lower side of the RTC6 PCIe Board, see Figure 6, the level of all output signals at the EXTENSION 1 Socket Connector can be configured for 5 V or 3.3 V, see Chapter 2.7.1 "Solder Jumper Field A – Configuring Output Signal Level at EXTENSION 1 Socket Connector", page 42:

- DIGITAL OUT 0…DIGITAL OUT 15
- LATCH OUT
- SYNC OUT
- BUSY OUT
- VCC_OUT

For monitoring purposes, the selected signal level is continuously outputted at pin (35): signal VCC_OUT.

16-Bit Digital Input Port and 16-Bit Digital Output Port

The EXTENSION 1 Socket Connector, see Figure 22, provides:

- 16-Bit Digital Input Port
  (DIGITAL IN 0…DIGITAL IN 15)
  protected (TTL level)
- 16-Bit Digital Output Port
  (DIGITAL OUT 0…DIGITAL OUT 15)
  buffered (TTL level)

This requires the output signals level to be configured by the jumper setting, see Section "Configuring the Output Signal Level", page 81.

For programming, see

- Chapter 9.1.1 "16-Bit Digital Output Port", page 303
- Chapter 9.2.1 "16-Bit Digital Input Port", page 310
- Chapter 9.3.2 "Execution of Conditional Commands", page 317

Synchronization of Data Acquisition

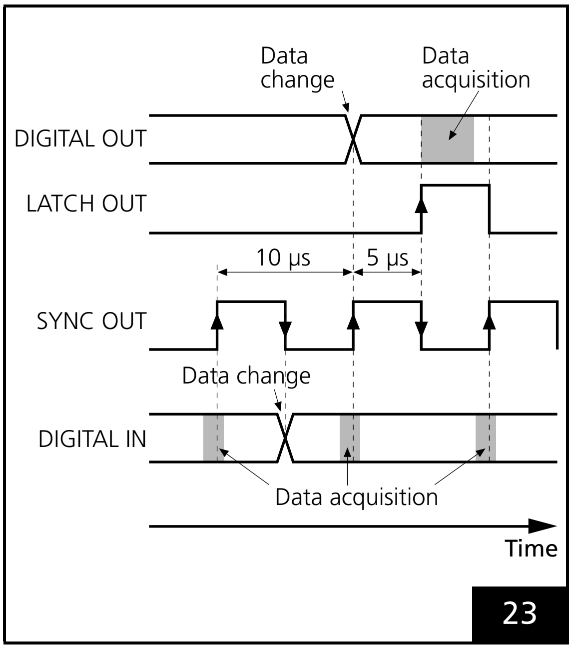
If several bits are simultaneously transferred as data words (and not independently from each other) by the 16-Bit Digital Output Port or the 16-Bit Digital Input Port, then the LATCH OUT signal or SYNC OUT signal should be used for synchronization of data acquisition.

The LATCH OUT signal (a 5 μs pulse, active-HIGH) is outputted at pin (33) as a trigger signal for acquiring the output values of the 16-Bit Digital Output Port. The RTC6 PCIe Board automatically generates the LATCH OUT signal when the output value at the 16-Bit Digital Output Port is written. The output value should be read out with the rising edge of the LATCH OUT signal. The rising edge occurs 5 μs after the value has been outputted at the 16-Bit Digital Output Port, see Figure 23.

To synchronize data acquisition at the 16-Bit Digital Input Port, a SYNC OUT signal (a square wave with 5 μs pulse length and 10 μs period) at pin (34) is continuously outputted.

Value changes at the 16-Bit Digital Input Port should be made with a falling edge of the SYNC OUT signal. The DSP of the RTC6 PCIe Board always accepts the currently provided value with the rising edge of the SYNC OUT signal, see Figure 23.

For technical data see Chapter 15 "Technical Specifications – RTC6 PCIe Board", EXTENSION 1 Socket Connector, page 952.



Synchronization of data acquisition by LATCH OUT signal or SYNC OUT signal.

BUSY List Execution Status

BUSY OUT signal at pin (36) and BUSY OUT signal at the LASER Connector[1] are identical, see Section "BUSY List Execution Status", page 76.

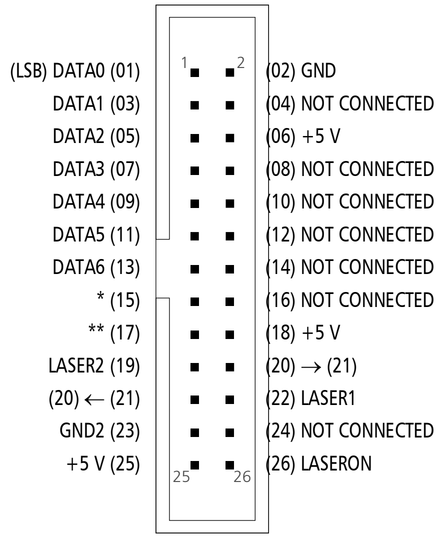
(1) RTC6 Ethernet Board: LASER Socket Connector.

4.6.3 EXTENSION 2 Socket Connector

- For technical data see Chapter 15 "Technical Specifications – RTC6 PCIe Board", EXTENSION 2 Socket Connector, page 952.

The EXTENSION 2 Socket Connector[(1)(2)] has 26 pins. It is located on the upper side of the RTC6 PCIe Board, see Figure 5.

The pin-out is shown in Figure 24.



* Depends on solder jumper position:
+5V or DATA7 or GND.

** Depends on solder jumper position:
+5V or DATA7 or GND or LATCH.

Identical in construction to Würth 61202621621.

24

EXTENSION 2 Socket Connector: pin-out. The pitch of the pins is 2.54 mm.

Notes

- Pin (15) and pin (17) are configurable by solder jumpers.

(1) On the RTC4, the functional corresponding socket connector is labeled LASER EXTENSION.

(2) RTC6 Ethernet Boards do not provide Laser Control Signals (LASER1 and LASER2) at their (10-pin) EXT. 2 socket connectors.

Configuration by Solder Jumpers

Pin (15) is configured by the solder jumper field C, see Chapter 2.7.3 "Solder Jumper Field C – Configuring Pin (15) of EXTENSION 2 Socket Connector", page 44.

Pin (17) is configured by solder jumper field B, see Chapter 2.7.2 "Solder Jumper Field B – Configuring Pin (17) of EXTENSION 2 Socket Connector", page 43.

Notes

- If the DATA7 bit is assigned to pin (15), then the full 8-bit output value is available at the output port (at the odd numbered pins of the EXTENSION 2 Socket Connector, pin (01)…pin (15)).
- If pin (15) is set to +5 V (HIGH level), then the output values have an offset of 128. That is, the output values are between 128…255.
- If pin (15) is set to GND (LOW level), then the possible output values are 0…127.
- The DATA7 bit can be used for other purposes by assigning it to pin (17).

Laser Control Signals

Laser Control Signals LASER1 and LASER2 are identical to the LASER Connector and depend on the set laser mode, see Section "Laser Control Signals", page 76.
The signals are referenced (as with the LASER Connector) to GND2.

Notice!

- On RTC6 PCIe Boards with Option "DC/DC Converter", GND2 and the Laser Control Signals LASERON, LASER1 and LASER2 at the LASER Connector and EXTENSION 2 Socket Connector are galvanically decoupled from GND.[a]
- On RTC6 PCIe Boards without Option "DC/DC Converter", GND2 are GND identical.

(a) RTC6 PCIe Boards: GND is the PC ground. RTC6 Ethernet Boards: GND is the ground at the POWER connector.

8-Bit Digital Output Port

The buffered 8-Bit Digital Output Port (TTL level) is intended for lasers with digital power control (for example, to control the lamp current of a YAG laser):

- DATA0…DATA7

Of course, it can be used for any other purpose as well. The output is in high-impedance mode until an initial value is assigned to it.

For programming the output see Chapter 9.1.2 "8-Bit Digital Output Port", page 304.

The MSB (DATA7) of the output value can be used for other purposes. To do this, it can be assigned to an other pin on the EXTENSION 2 Socket Connector (pin (15), pin (17), see above).

If the solder jumper is correspondingly set (see above), Pin (17) outputs a LATCH signal. The RTC6 PCIe Board automatically generates the LATCH signal (a 5 µs pulse, active-HIGH) when the value at the 8-Bit Digital Output Port is outputted.

If several bits are simultaneously transferred as a data word (and not independently from each other) by the 8-Bit Digital Output Port, then the LATCH signal should be used for synchronization of data transmission.

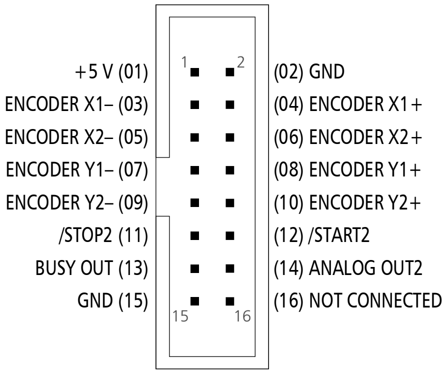
The value at the 8-Bit Digital Output Port should be read out with the rising edge of the LATCH signal, which is generated 5 µs after the value is written at the 8-Bit Digital Output Port, see also Figure 23.

4.6.4 MARKING ON THE FLY Socket Connector

- For technical data see Chapter 15 "Technical Specifications – RTC6 PCIe Board", MARKING ON THE FLY Socket Connector, page 953.

The MARKING ON THE FLY Socket Connector has 16 pins and is located on the upper side of the RTC6 PCIe Board, see Figure 5. It provides, among other things, pins for encoder inputs.

The pin-out is shown in Figure 25.



Identical in construction to Würth 61201621621.

25

MARKING ON THE FLY Socket Connector: pin-out. The pitch of the pins is 2.54 mm.

If the Option Processing-on-the-fly is enabled, laser material processing of moving workpieces (for example, on a moving conveyor belt or rotating disk) can be implemented, see Chapter 8.6 "Processing-on-the-fly", page 251.

Encoder Input Ports

The RTC6 PCIe Board provides 2 encoder input ports:

- ENCODER X
- ENCODER Y

ENCODER X and ENCODER Y are each designed for a pair (1, 2) of standardized differential input signals (RS-422). See also Section "Input Ports for External Encoder Signals", page 321.

External Control Signals

The external control signals (input signals) are:

- /START2
- /STOP2

See also Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 311.

Analog Output Port

The signal outputted at the 12-Bit Analog Output Port 2 ANALOG OUT2 (LASER Connector) is also outputted at pin (14), see also Section "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 77.

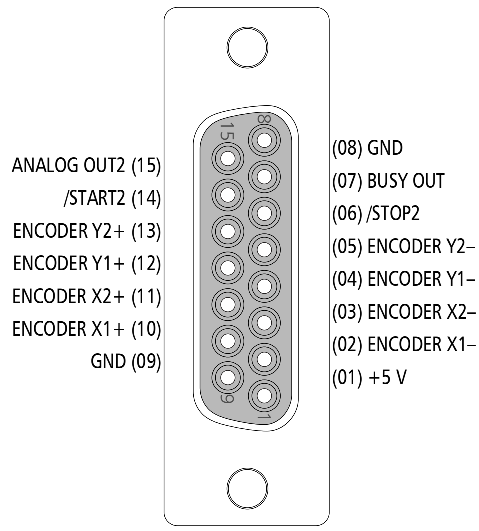
BUSY List Execution Status

The BUSY OUT signal at pin (13) and BUSY OUT signal at the LASER Connector are identical, see Section "BUSY List Execution Status", page 76.

MARKING ON THE FLY Slot Cover (Accessory)

SCANLAB recommends using an additional slot cover for using the input ports and signals of the MARKING ON THE FLY Socket Connector, see Chapter 2.8.5 "Slot Cover with 15-pin D-SUB Connector for MARKING ON THE FLY Socket Connector", page 46.

The pin-out is shown in Figure 26.



26

MARKING ON THE FLY Slot Cover (Accessory), #0109272: pin-out of the 15-pin female D-SUB connector.

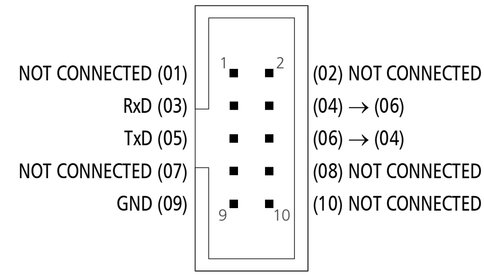
Alternatively, the Slot Cover with 15-pin D-SUB Connector and 9-pin D-SUB Connector, page 47 is also available.

4.6.5 RS232 Socket Connector

- For technical data see Chapter 15 "Technical Specifications – RTC6 PCIe Board", RS232 Socket Connector, page 953.

The RS232 Socket Connector has 10 pins and is located on the upper side of the RTC6 PCIe Board, see Figure 5.

The pin-out is shown in Figure 27.



Identical in construction to Würth 61201021621.

27

RS232 Socket Connector: pin-out. The pitch of the pins is 2.54 mm.

The RS232 Socket Connector is a hardware interface designed for bidirectional data exchange.

By **uart_config** (as of DLL 612), the RS-232 interface can be configured(1). **uart_config** returns the actual set Baud rate.

The baud rate (Default: 9600 baud) can be set with **uart_config** (to 160…12,800,000 Baud). Other parameters cannot be altered (data bits: 8, start bits: 1, stop bits: 1, parity: none).

For outputting data by the RS-232 interface, see Chapter 9.1.6 "RS-232 Interface", page 309.

For reading-in data by the RS-232 interface, see Chapter 9.2.3 "RS-232 Interface", page 310.

(1) Alternatively, the older **rs232_config** can be used which does not return a Baud rate. With $<$ DLL 612, the maximum Baud rate is limited to the range [300…115200].

4.6.6 McBSP/ANALOG Socket Connector

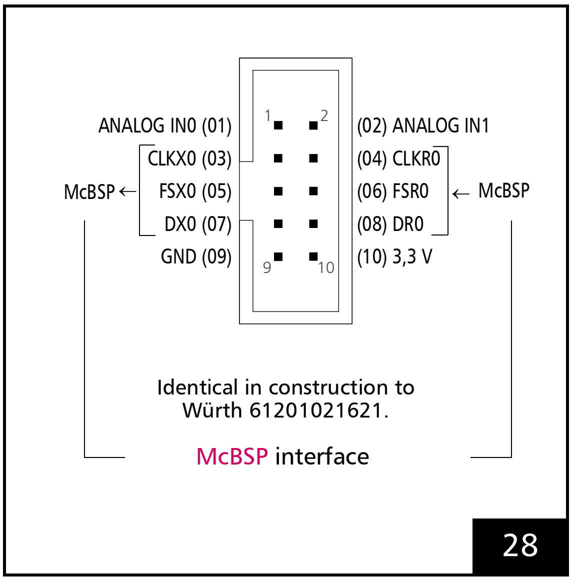
- For technical data see Chapter 15 "Technical Specifications – RTC6 PCIe Board", McBSP/ANALOG Socket Connector, page 954.

The McBSP/ANALOG Socket Connector[1] has 10 pins. It is located on the upper side of the RTC6 PCIe Board, see Figure 5.

The McBSP/ANALOG Socket Connector is a hardware interface that

- is designed for bidirectional data exchange (McBSP), see Section "McBSP Interface", page 87 and
- furthermore, provides two 10 V analog input ports: ANALOG IN0 at pin (01) and ANALOG IN1 at pin (02), see Section "Analog Input Ports", page 89.

The pin-out is shown in Figure 28.



McBSP/ANALOG Socket Connector: pin-out. The pitch of the pins is 2.54 mm.

McBSP Interface

The McBSP interface, see Figure 28, is initialized by **load_program_file** with:

- `XDelay` = `RDelay` = 1
- 8 MHz clock frequency

Intended for McBSP are pin (03), (05), (07) (outgoing signals) and pin (04), (06), (08) (incoming signals).

The following signals are continuously outputted after **load_program_file**:

- the clock signal at pin (03)
- every 10 μs a frame synchronization signal FSX at pin (05)
- a data signal DX at pin (07) (Default: SampleY|SampleX), see **set_mcbsp_out**

The McBSP interface can be integrated into applications for various purposes:

- As an alternative to encoder signals, the position of a moving workpiece can be directly integrated, see also Chapter "Correction via McBSP Interface", page 254. Prerequisite: Option Processing-on-the-fly.
- As an alternative to matrix and offset commands, coordinate transformations can be transmitted and integrated to align a workpiece with controllable timing (Online Positioning), see also Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 322.
- With **set_multi_mcbsp_in** you can transfer not only a 3D fly correction with position information but also laser power and other parameters, see also Chapter "Correction via McBSP Interface with Additional McBSP Input", page 255. Prerequisite: Option Processing-on-the-fly.
- Permanent data output in 10 μs cycles. With **set_mcbsp_out** and **set_mcbsp_out_ptr** it can be selected which data signals are outputted.
- OIE Output Mode, page 758 can be set at the McBSP interface by **set_mcbsp_out_oie_ctrl** and **set_mcbsp_out_oie_list**.

(1) With RTC5 boards referred to as SPI /I2C socket connector.

For data output using the McBSP interface, see Chapter 9.1.7 "McBSP Interface", page 309.

For data input using the McBSP interface, see Chapter 9.2.4 "McBSP Interface", page 310.

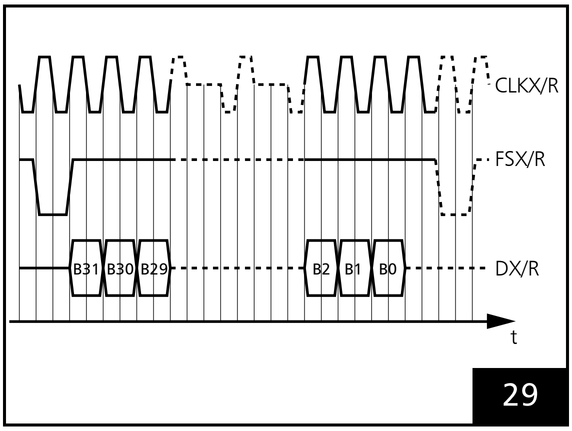
mcbsp_init allows setting a data delay for the McBSP data transmission independently for the transmitter and receiver. Possible values range from 0 to 2 (Default: 1).

All signals are referenced to GND[(1)].

RTC6 PCIe Board as Transmitter

The following specifications apply to CLKX0, FSX0 and DX0:

- Signal level 3.3 V TTL
- McBSP mode:
  – Single phase frame
  – Single element per frame
  – 32 bits per element
  – Data delay `XDelay` bits
- The timing diagram of the McBSP signals is shown in Figure 29. A frame synchronization signal (active-LOW) is generated upon the rising edge of the clock and held for one clock cycle (1 data bit). The 32 data bits are transmitted after `XDelay` clock cycles at the rising edges of the clock and in the sequence Bit #31…Bit #0. The transmission frequency is by default 8 MHz (4 μs per data word). Alternatively, a value between 4 MHz and 16 MHz can be set by **set_mcbsp_freq**.



Timing diagram of McBSP signals at 1 bit data delay.
Default: `XDelay` = `RDelay` = 1.

RTC6 PCIe Board as Receiver

The following specifications apply to CLKR0, FSR0, DR0:

- Signal level 3.3 V or 5 V TTL.
- McBSP mode:
  – Single phase frame
  – Single element per frame
  – 32 bits per element
  – Data delay `RDelay` bits

The timing diagram of the McBSP signals is shown in Figure 29. The frame synchronization signal (active-LOW) generated upon a rising edge (of an external clock signal) is detected upon the clock's next falling edge (trailing edge of the external clock pulse). The duration of the frame synchronization signal is irrelevant. After `RDelay` clock cycles the 32 data bits are detected with each additional falling edge of the clock signals in the sequence Bit #31…Bit #0, provided they are transmitted with rising edges.
When doing so, observe the following Notes:

– The McBSP interface always ignores the first frame synchronization signal after a **load_program_file** or **mcbsp_init**. Therefore, the data provided is not transmitted. If necessary, a dummy value should be initially sent to the interface. You can use **read_mcbsp** to check for successful transmission.
– The bit frequency (receiving frequency) is exclusively determined by the incoming clock pulses and has a maximum limit of 16 MHz.
– The last data bit (Bit #0) must be followed by transmission of at least one additional external clock cycle to ensure that the interface's DSP side acquires and buffers the data word. Simultaneously with this clock cycle, you can already initiate another new transfer by a frame synchronization signal.

(1) See footnote on page 39.

- After a **load_program_file**, the McBSP data is internally permanently acquired and buffered as soon as (new) data is transmitted. Newer data words overwrite older ones.
  – After **load_program_file** and/or after activation of Processing-on-the-fly correction by **set_fly_x_pos**, **set_fly_y_pos** or **set_fly_rot_pos**, the input values are stored to internal memory location 0.
  – After activation of an Online Positioning the input values are stored to internal memory locations 1 or 2
    - For a "Local Online Positioning" by **set_mcbsp_x**, **set_mcbsp_y** and/or **set_mcbsp_rot** or **set_mcbsp_matrix**, see Section "Configuring "Local Online Positioning"", page 238
    - For a "Global Online Positioning" by **set_mcbsp_global_x**, **set_mcbsp_global_y** and/or **set_mcbsp_global_rot** or **set_mcbsp_global_matrix**, see Section "Configuring "Global Online Positioning"", page 240
  – The most recent fully transferred values can be queried from a corresponding memory location by **read_mcbsp**.
  – After activation of Processing-on-the-fly correction by **set_mcbsp_in** or **set_mcbsp_in_list**, the input values are transferred to internal memory locations 0 and 3.
  – After activation of Processing-on-the-fly correction by **set_multi_mcbsp_in** or **set_multi_mcbsp_in_list**, the input values are transferred to internal memory locations 0 through 3.

Analog Input Ports

The McBSP/ANALOG Socket Connector (RTC6 Ethernet Boards: SPI/ANA/UART Socket Connector) provides two analog input ports(1):

- ANALOG IN0
- ANALOG IN1

After the first **read_analog_in** call, voltages that are applied there are:

- automatically converted endlessly (duration approx. 0.3 ms)(2)
- transferred to the DSP as 12-bit digital values

By the control command **read_analog_in**, the present values can be queried at any time.

For technical specifications, see Analog input ports, page 954.

(1) As with RTC5 PCI Express Boards (but not with RTC5 PCI Boards).

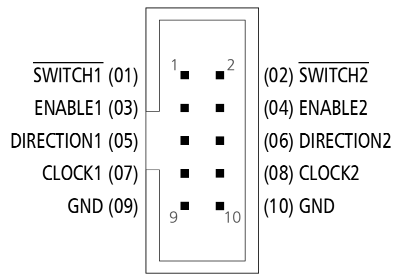
(2) Note the comment with **get_temperature**, page 471.

4.6.7 STEPPER MOTOR Socket Connector

- For technical data see Chapter 15 "Technical Specifications – RTC6 PCIe Board", STEPPER MOTOR Socket Connector, page 954.

The STEPPER MOTOR Socket Connector has 10 pins and is located on the upper side of the RTC6 PCIe Board, see Figure 5.

The pin-out is shown in Figure 30.



Identical in construction to Würth 61201021621.

30

STEPPER MOTOR Socket Connector: pin-out. The pitch of the pins is 2.54 mm.

At the STEPPER MOTOR Socket Connector, signals for controlling 1…2 stepper motors can be outputted, see Chapter 9.1.5 "Controlling Stepper Motors", page 306.

For each step that is to be carried-out the RTC6 PCIe Board generates an active-HIGH 5 µs pulse as CLOCK signal. The CLOCK signal is permanently LOW between pulses.

With ENABLE signals, a user program can, for example, switch the motor current on and off.

The DIRECTION signals can set the direction and each CLOCK pulse can be used to execute a single step.

SWITCH1 input port and SWITCH2 input port serve to connect end switches. The RTC6 PCIe Board detects an end switch position by a LOW level (active-LOW).

5 Installation and Start-Up

Installation of the RTC6 PCIe Board consists of the following steps:

(1) Checking the RTC6 PCIe Board jumpers, see Chapter 5.1 "Checking Jumper Settings", page 91

(2) Hardware installation: Installing the RTC6 PCIe Board, see Chapter 5.2 "Installing the RTC6 PCIe Board", page 91

(3) Installing the RTC6 board driver, see Chapter 5.3 "Installing the RTC6 Board Driver", page 92(1)

(4) Installing the RTC6 software, see Chapter 5.4 "Installing the RTC6 Software", page 93

(5) When turning on the laser system, observe the safe start-up sequence, see Chapter 5.6 "Safe Start-up and Shutdown Sequences", page 94

(6) For conducting a post-installation functionality test, for example, the included HPGL converter program can be used, see Chapter 5.7 "Functionality Test", page 95

5.1 Checking Jumper Settings

SCANLAB ships RTC6 PCIe Boards in various jumper configurations.

(1) Make sure that the to-be-installed RTC6 PCIe Board has the required jumper configuration. If you need to change the factory settings, proceed as described in Chapter 2.7 "Jumper Settings and Type Designations", page 41.

(2) Proceed with Chapter 5.2 "Installing the RTC6 PCIe Board", page 91.

5.2 Installing the RTC6 PCIe Board

Notice!

- Store the board in an electrostatically neutral environment using the supplied anti-static bag.
- Carry out the installation in an area that complies with Electro Static Discharge (ESD) directives. When installing the board, observe the instructions given in the instruction for use of the PC.
- Do not touch the contacts of the board.
- Protect the board from humidity, dust, corrosive vapors and mechanical stress.

Perform the following steps to install the RTC6 PCIe Board in a PC:

(1) Remove all data media from your PC.

(2) Shut down the operating system and switch off the PC. Disconnect the PC from the mains.

(3) Disconnect all peripherals (for example, monitor, mouse, keyboard, printer etc.) from the PC.

(4) Place the PC on a work table that complies with ESD directives.

(5) Remove the housing of the PC. Locate a free PCIe slot and remove the slot cover. A height from the slot connector's top of at least 105 mm is required.

(6) Remove the RTC6 PCIe Board from the antistatic bag. Do not touch the contacts of the board.

(1) Step 3 does not apply to RTC6 Ethernet Boards.

(7) If you want to use the signals on the RTC6 PCIe Board socket connectors, then attach the appropriate cables. SCANLAB recommends installing additional slot covers with suitable connectors. Then the signals are accessible even when the PC housing is closed. All RTC6 PCIe Board connectors and signals are described in Chapter 4 "RTC6 PCIe Board – Layout and Interfaces", page 66.

(8) Install the RTC6 PCIe Board into the PCIe slot. Observe the instructions in the manual of your PC(1).

(9) Close the PC housing.

(10) Place the PC at its operational location and connect all peripheral equipment.

(11) Connect the 9-pin D-SUB connector on the RTC6 PCIe Board – using the XY2-100 Converter (Accessory), if necessary – to the scan system by a data cable, see Chapter 4.5 "Interfaces to Scan System", page 69.

(12) Connect the 15-pin D-SUB connector on the RTC6 PCIe Board to the laser by a suitable interface, see Chapter 4.6 "Interfaces for the Laser and Peripheral Equipment", page 75.

(13) If necessary, connect one or more of the RTC6 PCIe Board socket connectors (using additional slot covers, see above) to the laser, a handling system or other supplementary equipment of the overall system.

(14) Check all connections and turn on the PC.

5.3 Installing the RTC6 Board Driver

Notice!

- If on your PC an WDM technology-based RTC3/4/5 board driver
  - is yet installed or
  - was installed and has been removed or
  - you are not sure in this regard:

    (1) Install the RTC6 board driver.

    (2) Run `ScanlabClassChecker.cmd` as Administrator (part of the delivered software package; see there also the background information in `ReadMe_ScanlabClassChecker.pdf`).
    Step 2 can be skipped on brand new PCs on which an RTC board driver never has been installed.

To install the RTC6 board driver for Windows 10, 8, 7, proceed as follows:

- After the RTC6 PCIe Board has been installed, start the computer.

For Windows 7:

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

(2) In the "Add Hardware Wizard" specify the folder that includes the RTC6 board driver. During installation, the operating system automatically selects the appropriate driver file (32-bit or 64-bit).

(3) Run `ScanlabClassChecker.cmd` as Administrator (part of the delivered RTC6 Software Package; see there also the background information in `ReadMe_ScanlabClassChecker.pdf`).

(1) If several master/slave-synchronized RTC6 PCIe Boards are to be used in a PC, then the boards must be connected pairwise with each other by the Master and Slave connectors and installed in preferably (recommended) adjacent PCIe slots, see also Chapter 4.4 "Master Socket Connector, Slave Socket Connector", page 68.

For Windows 10, 8:

(1) Open the Device Manager to display the device tree. Look for the "PCI device" entry and update its driver by specifying the folder that includes the RTC6 board driver.

(2) Run `ScanlabClassChecker.cmd` as Administrator (part of the delivered software package; see there also the background information in `ReadMe_ScanlabClassChecker.pdf`).

> **Notice!**
>
> - The RTC6 PCIe Board does not support power-saving modes that switch off power to the PCIe bus. Accordingly, you must disable standby or sleep modes of the operating system. Particularly disable the Windows sleep mode option (some Windows operating systems enable this option by default). You can use the `SleepMode.cmd` script included in the RTC6 Software Package (under `RTC6 Tools`) to do this. See also Section "Notes", page 38.

> **Notice!**
>
> - On some PCs it may happen that an RTC6 PCIe Board is not recognized by the device manager in a certain PCIe slot. Use a different slot.

5.4 Installing the RTC6 Software

Additionally to installing the RTC6 board driver, the RTC6 software must be "installed", that is, copied or unzipped on the hard drive of the PC.

RTC6 software for RTC6 PCIe Boards are the following files, see also Section "Folder RTC6 Files", page 34:

- RTC6 DLL
  - `RTC6DLL.dll`
    Win32-based RTC6 dynamic link library
  - `RTC6DLLx64.dll`
    Win64-based RTC6 dynamic link library
- RTC6 files
  - `RTC6OUT.out`,
    Program file for the PCIe-DSP
  - `RTC6ETH.out`
    Program file for the Eth-DSP
  - `RTC6RBF.rbf`
    Firmware file for the FPGA
  - `RTC6DAT.dat`
    Binary auxiliary file

The files are included in the RTC6 Software Package. For easy identifying and archiving of different software versions, some of the files are also delivered zipped (the `zip` file names `RTC6<…>_<Version>.zip` include the version numbers).

To install the RTC6 software, follow these steps:

- Copy `RTC6DLL.dll` (or `RTC6DLLx64.dll`), `RTC6OUT.out` (with RTC6 Ethernet Boards `RTC6ETH.out`), `RTC6RBF.rbf` and `RTC6DAT.dat` (of the desired version) to your PC.
  Note that differing file versions cannot be arbitrarily combined with another (each `zip` file includes a text file with corresponding version information).

Notes

- Also provide the necessary correction file(s) (existing `*.ct5` correction files can still be used; do not overwrite customized correction files!).

5.5 Changing RTC6 PCIe Board BIOS

- For RTC6 Ethernet Board BIOS-ETH upgrade, see Chapter 16.7.1 "Upgrading BIOS-ETH", page 989.

Prerequisites

- RTC6conf.exe, page 35
- A DSP program file for RTC6 PCIe Board BIOS RTC6BIOSOUT_*.out (alternatively):
  - From the current RTC6 Software Package RTC6BIOSOUT_23.out, page 35 (for upgrade)
  - From an older RTC6 Software Package, see BIOS, page 1055 (for downgrade)

To change the RTC6 PCIe Board BIOS:

(1) Start RTC6conf.exe[(1)].

(2) Select the desired card.
Note that the current BIOS version is displayed.

(3) Click **FLASH BIOS** button.

(4) Specify the desired RTC6BIOSOUT_*.out, see column BIOS, page 1055.

(5) Shut down and restart the PC.

5.6 Safe Start-up and Shutdown Sequences

To assure safety during start-up, turn on the components of your laser system exactly in the following order:

(1) Turn on the PC containing the RTC6 PCIe Board and start up the control software (initialization of the board).

(2) Turn on the desired peripheral equipment.

(3) Turn on the power supply for the scan system.

(4) Turn on the laser.

When shutting down the laser system, turn off the components exactly in reverse order:

(1) Turn off the laser.

(2) Turn off the power supply for the scan system.

(3) Turn off the peripheral equipment.

(4) Close the control software and turn off the PC containing the RTC6 PCIe Board.

⚠ **Caution!**

- While the PC is turned on and off, short-term level variations at the output ports of the RTC6 PCIe Board, for instance short-term arbitrary variations of the Laser Control Signals can occur. Therefore always observe the above listed start-up and shutdown sequences. Otherwise, there is the risk that the laser unexpectedly turns on for a short term.

- Always turn on the scan system after the PC and control software and turn it off prior to the PC and control software. Otherwise, arbitrary scan motions of the scan system could occur. Always turn on the laser as the last component and turn off the laser as the first component. Otherwise, there is the risk that the laser beam might be deflected in an uncontrolled direction.

(1) See also RTC6conf.pdf.

5.7 Functionality Test

Caution!

- *Always turn on the PC and the power supply for the scan head first before turning on the laser. Otherwise, there is the danger of uncontrolled deflection of the laser beam.*
  SCANLAB recommends the use of a shutter to prevent uncontrolled emission of laser radiation.

The HPGL conversion program `Hpgl.exe` is supplied for testing control of the scan head, see also Section "Folder HPGL", page 33. This program lets you load graphics files in Hewlett Packard HPGL format (vector graphic plotter files `*.plt`) for transfer to the RTC6 PCIe Board.

(1) Copy `Hpgl.exe` and the supplied `*.plt` files to the same folder as the `RTC6DLL.dll`, the RTC6 files and correction file(s).

(2) Start `Hpgl.exe`.

(3) Choose **Options** > **Correction**, and then select a correction file.

(4) Choose **File** > **Load HPGL-File**, and then select a `*.plt`.

(5) To start output, choose **Mark** > **Start Marking**.

5.8 User Programs and Demo Programs

Notice!

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

The DLLs for RTC6 user programs (`RTC6DLL.dll`, `RTC6DLLx64.dll`) support the RTC6 PCIe Board under 32-bit and 64-bit Microsoft operating systems Windows 10, 8, 7. The DLLs provide all necessary functions for operating the RTC6 PCIe Board.

Programming of user programs is described in detail in Chapter 6 "Developing RTC6-User Programs", page 96. Chapter 6.2 "Initialization and Program Start-Up", page 98 shows how to import the functions of the RTC6 DLL into user programs, if they are written in Pascal, C, C++ or C#.

On a 64-bit operating system, the 64-bit variant of the RTC6 board driver supports function calls from Win64 user programs as well as from Win32 user programs.

Therefore, existing Win32 user programs for the RTC6 PCIe Board are able to execute even on 64-bit systems, if the included Win32-based file `RTC6DLL.dll` is used.

For Win64 user program, the Win64-based file `RTC6DLLx64.dll` is included in the software package. In case a user program utilizes implicit linking to the `RTC6DLLx64.dll`, it must be linked with the Win64-based import library `RTC6DLLx64.lib`.

To help software developers get started, some example codes are shown in:

- Chapter 6.2.5 "Example Code (C)", page 103
- Chapter 6.8.3 "Example Code (C)", page 136
- Chapter 7.1.4 "Example Code (C)", page 145

6 Developing RTC6-User Programs

6.1 RTC6 Software Concept Basics

6.1.1 Controlling Scan Systems and Lasers – An Introductory Example

The SCANLAB RTC6 PCIe Board and its related software are designed for controlling scan systems and lasers.

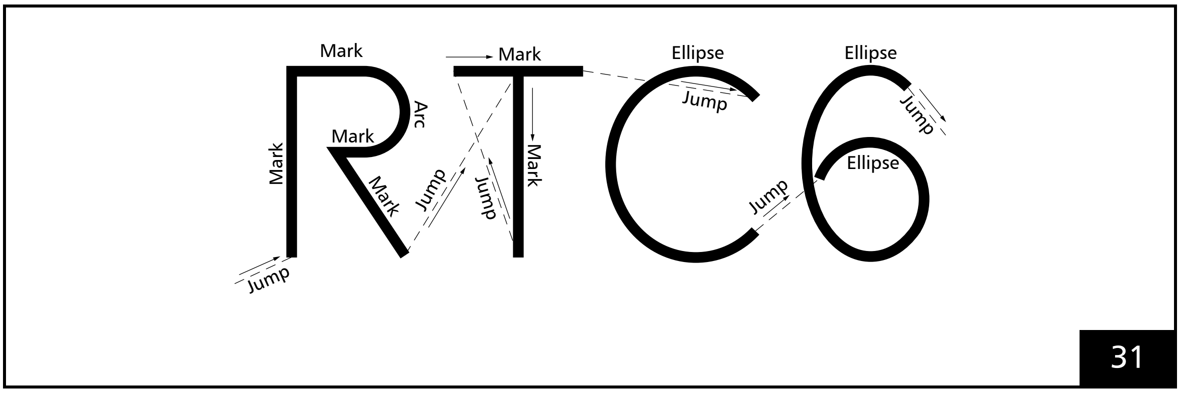
To illustrate the principle of operation, Figure 31 shows a simple laser marking sample(1). The lettering is made up of straight line segments (vectors) and arc segments. The RTC6 DLL provides a set of jump, mark, arc and ellipse commands for laser processing along such segments. Each of these commands describes one vector or arc. Additional commands are available for controlling the laser during the marking process.

The RTC6 PCIe Board processes the commands it receives and precisely transmits the required marking signals to the scan head (using a predefined 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.

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

The current status of a scan head can also be queried via the RTC6 PCIe Board using appropriate commands.

For laser control, the RTC6 PCIe Board provides various analog and digital signal output ports freely available.



A laser marking sample.

6.1.2 Control Commands and List Commands

The RTC6 command set consists of:

- Control commands
- List commands

Control commands are executed immediately. They are used, for instance, for initializing, controlling execution of lists, setting some general parameters, or for directly controlling the laser and scan head.

Before *list commands* can be sent to the RTC6 PCIe Board, a control command must define the input pointer to which subsequent list commands are transferred. This corresponds to opening a list, see Chapter 6.4.1 "Loading Lists", page 108.

List commands sent to the RTC6 PCIe Board afterwards are not executed immediately. They are stored in the RTC6 List Memory first. They are processed in real time only after the list has been started.

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 mark speed.

Some of the list commands are so called *short list commands*. These do not require the full 10 µs clock cycle for command execution. They are executed along with the next list command, one directly after the other within a single 10 µs clock cycle. A control command cannot be executed in between. The total list execution time is thereby reduced.

Short list commands include **list_jump_pos**, **list_call** etc. With a Polyline, for example, the laser power can be set individually for each vector by the short list command **write_da_x_list** without interrupting it (the laser remains on).

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 RTC6 commands are described in detail in Chapter 10 "RTC6 Commands".

6.2 Initialization and Program Start-Up

To use the commands and functions of the RTC6 PCIe Board in a user program:

- A complete installation must have been carried out – this includes the RTC6 hardware, RTC6 board driver and RTC6 software, see Chapter 5 "Installation and Start-Up", page 91
- The desired RTC6 DLL (Win32- or Win64-based) must be linked to the user program, see Chapter 6.2.1 "DLL Calling Convention", page 98
- The RTC6 DLL functions must be imported to the user program, see Chapter 6.2.2 "Importing Commands", page 98

At the beginning of each user program, commands must be called:

- that initialize the RTC6 DLL for the calling user program and assigns access rights for the installed RTC6 PCIe Boards, see Chapter 6.2.3 "Initializing the RTC6 DLL and RTC6 Board Management", page 100
- that initialize the installed RTC6 PCIe Boards, see Chapter 6.2.4 "Start of RTC6 PCIe Board Operation", page 101

Only afterward can the user program load the desired command lists into the RTC6 List Memory and start processing.

These steps are described individually in the following chapters. They are shown by an example program in Chapter 6.2.5 "Example Code (C)", page 103.

6.2.1 DLL Calling Convention

Link the user program to the `RTC6DLL.dll`/ `RTC6DLLx64.dll`(1).

The RTC6 DLL calling convention is `stdcall` for Win32 and `fastcall` for Win64.

The structure alignment is 4 byte for Win32 and 8 byte for Win64.

6.2.2 Importing Commands

To facilitate importing the commands of the RTC6 DLL into a C, C++, C# or Pascal user program, the RTC6 Software Package contains corresponding utility files.

Notice!

- Some RTC6 commands can only be *completely* executed with appropriate options, see also Chapter 2.6 "Options", page 39.
- **get_rtc_version** provides information about the options installed on your RTC6 PCIe Board.

(1) See Chapter 5.4 "Installing the RTC6 Software", page 93.

Pascal

Use the file RTC6Import.pas as a unit and call the RTC6 commands you need, like

```
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 RTC6impl.h and link with the (C) import library RTC6DLL.lib (for Win32 user programs or with RTC6DLLx64.lib for Win64 user programs) for building the executable.

Call the RTC6 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 RTC6expl.h. Before calling any RTC6 command, initialize the RTC6 DLL by calling the function RTC6open (which is defined in the file RTC6expl.c). When you have finished the RTC6 session, close the RTC6 DLL by calling the function RTC6close (also defined in the file RTC6expl.c).

For building the executable, link with the file RTC6expl.obj, which you can generate from the RTC6expl.c source code.

Call the RTC6 commands you need, like

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

for causing a jump to location 1000, 2500.

Differences between Implicit and Explicit Linking

	Implicit Linking	Explicit Linking
Necessary Files	RTC6impl.h or RTC6impl.hpp, RTC6DLL.lib or RTC6DLLx64.lib	RTC6expl.h, RTC6expl.c
Advantages	Easiest linking method.	Eliminates the need to link the user program with an import library.
Disadvantages	Users need to link the user program to a compiler-specific import library.	Users need to initialize (RTC6open) and close (RTC6close) the RTC6 DLL.

C++

If you want to use references instead of pointers for returning values to function parameters in C++ (for instance UINT&Pos instead of UINT*Pos), use the files RTC6impl.hpp instead of RTC6impl.h. Otherwise, the command import is realized as in C (see above).

C#

For implicit linking of the RTC6 DLL in C# the wrapper class is provided. It also supports the 'Any CPU' option for simultaneous usage with 32-bit and 64-bit programs.

6.2.3 Initializing the RTC6 DLL and RTC6 Board Management

As many RTC6 PCIe Boards as the PCIe bus allows can be operated simultaneously in one PC. However, the RTC6 board management can manage a maximum of 255 RTC6 PCIe Boards and RTC6 Ethernet Boards at the same time, see Chapter 16.5.3 "About the RTC6 Board Management", page 985.

The RTC6 DLL allows multi-processing[1] as well as multi--threading.

No RTC6 PCIe Board can be simultaneously used by several user programs. Access rights (even if temporary) to existing boards are assigned on an exclusive basis by the RTC6 DLL.

Multiple threads of *one* user program can use the same board, but can not send commands to it at the same time (the RTC6 DLL automatically serializes the command calls).

The RTC6 board management, see Chapter 16.5.3 "About the RTC6 Board Management", page 985, must be initiated by **init_rtc6_dll** at the beginning of each user program so that an RTC6 PCIe Board can be addressed at all. This even applies, if only *one* RTC6 PCIe Board is to be used by only *one* user program.

init_rtc6_dll does the following (for details, see the command description):

- Searches for all present RTC6 PCIe Boards
- Establishes the RTC6 board management
- Automatically assigns the user program access rights to the found boards (as long as access rights are not already assigned to another user program)
- Assigns RTC6 DLL-internal numbers for all found RTC6 Ethernet Boards (important for multi-board commands)
- Sets one board as the *active* board, which therefore, is the target for non-multi-board commands
- Does *not* search for RTC6 Ethernet Boards. These must be added separately, see Chapter 16 "Appendix A: RTC6 Ethernet Board", page 955.

acquire_rtc, **release_rtc**, **select_rtc** allow changing access rights and the active board at run-time.

Usage of several boards is described in Chapter 6.6 "Using Several RTC6 PCIe Boards in One PC", page 126; usage by several user programs is described in Chapter 6.7 "Usage of RTC6 PCI Express Boards by Several User Programs", page 131.

After RTC6 DLL initialization by **init_rtc6_dll**, the default DLL-operation mode is the RTC6 Standard Mode. If required, a different DLL-operation mode can be set (see **set_rtc4_mode**, **set_rtc5_mode** and **set_rtc6_mode**).

The RTC4 Compatibility Mode is provided so that user program written for the RTC4 can be processed by the RTC6 PCIe Board (for otherwise unchanged commands[2]) without modification. In the Command Descriptions, page 342, notes changes in this regard, see "RTC4→RTC6" rows.

See also Chapter 2.10.2 "Porting RTC4 Source Code to the RTC6 PCIe Board", page 50.

The similar applies to the RTC5 Compatibility Mode, see RTC5→RTC6 rows of the Command Descriptions, page 342. See also Chapter 2.12.2 "Adapting RTC5 Source Code for the RTC6 PCIe Board", page 62.

(1) Several user programs can run simultaneously.

(2) Commands having identical names with analogous parameterization and meaning.
Example: `mark_abs( X, Y )`.

6.2.4 Start of RTC6 PCIe Board Operation

At the beginning of every RTC6 user program – after initialization of the RTC6 DLL, see Chapter 6.2.3 "Initializing the RTC6 DLL and RTC6 Board Management", page 100 – it is recommended to carried out the following sequence of steps in order to start RTC6 PCIe Board operation.

(1) Initializing the Board, page 101

(2) Configuring the Board, page 101

(3) Initializing the Scan System Control, page 102

(4) Initializing the Laser Control, page 102

(5) Loading and Executing Lists, page 102

Initializing the Board

- Call **load_program_file**. For the individual actions, see Function, page 555.
  After execution of **load_program_file**, the position output is automatically set to the null position (0|0)[1] and laser control is *deactivated*[2][3].
  In order to be able to combine RTC6 files (`RTC6DLL.dll`/`RTC6DLLx64.dll`, `RTC6OUT.out`/`RTC6ETH.out`, `RTC6RBF.rbf`, `RTC6DAT.dat`), they must have certain file versions, see also Chapter 5.4 "Installing the RTC6 Software", page 93. **load_program_file** performs a version compatibility check. If a version error exists (error code 7 and **get_last_error** return code `RTC6_VERSION_MISMATCH`), the board is released by **release_rtc**. Thus, it is not available for further RTC6 commands other than those not requiring granted access rights.

RTC6 PCIe Boards can only be operated if all RTC6 files are available with a compatible combination of versions, see Chapter 5.4 "Installing the RTC6 Software", page 93.
If the version check fails and the board is not acquired by another user program, then **load_program_file** can also be used at any time to load a correct program version. After that, the board can be acquired by **select_rtc** or **acquire_rtc**.

If several RTC6 PCIe Boards are master/slave connected, see Chapter 6.6.3 "Master/Slave Operation", page 127.

Configuring the Board

- If necessary, configure the RTC6 List Memory by **config_list**.
  After **load_program_file**, see Figure 32:
  - RTC6 List Memory area "List 1" can store 4,194,304 list commands.
  - RTC6 List Memory area "List 2" can store 4,194,304 list commands
  - Protected RTC6 List Memory area "List 3" can store 0 list commands

(1) Center of the Image Field.

(2) There are no Laser Control Signals at the corresponding pins (LASERON, LASER1, LASER2). They are in high-impedance state.

(3) On the state of the various output ports, see page 556.

Initializing the Scan System Control

(1) Use **load_correction_file** to download the necessary correction file(s) to the RTC6 PCIe Board (you can load a correction table before or after **load_program_file** but you should load it *before* **select_cor_table**; you should at least load a 1:1 correction table).

See Chapter 8.5 "Controlling 2D Scan Systems and 3D Scan Systems", page 245, for information about using several different correction tables.

(2) Assign the previously loaded correction table(s) to the scan head control port(s) by **select_cor_table** [(1)]. This causes the intended Image Field correction to also be applied to the default position (0|0) previously set by **load_program_file** (in some circumstances, Image Field correction can even shift the null position).
After **load_program_file**, the default assignment is as follows:

- correction table #1 to Connector for First Scan Head
- no correction table to Connector for Second Scan Head

The desired Image Field correction becomes active only after a subsequent **select_cor_table**, **select_cor_table_list**, Jump Command or Mark Command.

(3) Define the Scanner Delay mode by **set_delay_mode** (among others, "Variable Polygon Delay" or constant Polygon Delay).

(4) If necessary[(2)], load a table for the "Variable Polygon Delay" by **load_varpolydelay**.

The remaining settings (Scanner Delays, jump speed and mark speed) are set by further control commands or list commands.

Initializing the Laser Control

(1) Set the laser mode by **set_laser_mode**.

(2) Set the polarity of the Laser Control Signals appropriate to your laser system by **set_laser_control**.

(3) Set the FirstPulseKiller signal length (only with a YAG Mode) by **set_firstpulse_killer**.

(4) Set the delay of the Q-Switch Signal (only with a YAG Mode, in particular for YAG Mode 5) by **set_qswitch_delay**.

(5) Set the stand-by pulses by **set_standby** (in particular for Laser Mode 4 and Laser Mode 6).

(6) Enable the Laser Control Signals (see **enable_laser**), if they have been suppressed by **set_laser_control**.

The remaining settings (such as laser timing or Laser Delays) are set by further control commands or list commands, for example, see Chapter 7.1.4 "Example Code (C)", page 145 or Section "Signals for "Laser Active" Operation", page 191.

Loading and Executing Lists

(1) Load the list(s).

(2) If necessary, enable the external start input by **set_control_mode**.

Notice!

- Carefully check your user 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.

(1) **load_correction_file** automatically calls **select_cor_table** with the last assigned table numbers after loading the correction table (if **load_correction_file** follows after **load_program_file**), see Section "Notes", page 181.

(2) Step 4 can be omitted, if a new `RTC6DAT.dat` is created by **create_dat_file** after loading the table. Then, this table is automatically loaded with the next **load_program_file**.

6.2.5 Example Code (C)

The following C source code for a console user program (environment: Win32) illustrates the programming fundamentals of RTC6 DLL and RTC6 PCIe Board initialization (for complete demo programs, see Chapter 11 "Demo Programs", page 944).

Necessary sources: `RTC6impl.h`, `RTC6DLL.lib` (for implicit linking) or `RTC6expl.h`, `RTC6expl.c` (for explicit linking) to link the RTC6 DLL to the program, see Chapter 6.2.1 "DLL Calling Convention", page 98. If the operating system does not find the `RTC6DLL.dll` on user program startup, it produces a corresponding error message and terminates the program.

```
// System header files
#include <windows.h>
#include <stdio.h>
#include <conio.h>

// RTC6 header file for implicitly linking to the RTC6DLL.dll (for building the executable, also link
// with the (Visual C++) import library RTC6DLL.lib):
#include "RTC6impl.h"
// Alternatively: RTC6 header file for explicitly linking to the RTC6DLL.dll
// (for building the executable, link with the file RTC6expl.obj, which you can generate
// from the RTC6expl.c source code):
//#include "RTC6expl.h"

void __cdecl main( void*, void* )
{

   // only for explicitly linking:
   // if ( RTC6open() ) // error detected, RTC6open returns 0 for no error
   // {
   // printf( "Error: RTC6DLL.dll not found\n" );
   // terminateDLL();
   // return;
   // }

   printf( "Initializing the DLL\n\n" );

   UINT ErrorCode;

   // Initializing the RTC6 DLL (the following command must be called as the first RTC6 command)
   ErrorCode = init_rtc6_dll();

   // Following init_rtc6_dll you should include a program code to catch an error during
   // initialization, for example, for the case the desired board is not detected, access is denied
   // or for another error (for example, a version mismatch).
   // See Chapter 6.8.3 "Example Code (C)", page 136.
```

```
   // Initializing the RTC6 PCIe Board:
   // - Selecting the board number 1 as the active board in this user program.
   // - If desired: Selecting the RTC4 Compatibility Mode as operation mode
   //   (Default: RTC6 Standard Mode)
   // - Optional: stopping any list running on RTC6 PCIe Board number 1 (if it has been used
   //   previously by another user program, a list might still be running). load_program_file does this
   //   automatically itself. Otherwise, load_correction_file cannot be executed before
   //   load_program_file.
   // - Calling load_program_file for initializing the board, loading the program file, etc.
   //   (here also a program code should be included to catch possible errors - for example, file or
   //   system errors - during initialization, see Chapter 6.8.3 "Example Code (C)", page 136).
   // - Clearing all previous error codes
   // - (stop_execution might have created an RTC6_TIMEOUT or RTC6_BUSY error).
   // - Configuring the RTC6 List Memory. Default: 4,194,304 for list 1 and list 2 each).
   (void) select_rtc( 1 );
   set_rtc4_mode();

   // Optional: stop_execution();

   ErrorCode = load_program_file( 0 ); // Path = 0: path of current working directory
   if ( ErrorCode )
   {
      printf( "Program file loading error: %d\n", ErrorCode );
      free_rtc6_dll();
      return;
   }
   reset_error( -1 );
   config_list( 4000, 4000 );

   // Following the above initialization code you can include the program code defining
   // the laser scan process. An example code is in Chapter 7.1.4 "Example Code (C)", page 145.

   // End of main program
   terminateDLL();
   return;
}

void terminateDLL()
{
   printf( "- Press any key to shut down \n" );
   while( !kbhit() );
   (void) getch();
   printf( "\n" );
   // Close the RTC6 DLL
   free_rtc6_dll();
   // only for explicitly linking:
   // RTC6close();
}
```

6.3 RTC6 List Memory

The RTC6 List Memory serves as intermediate storage for list commands.

Before list commands can be transferred to the RTC6 PCIe Board, a control command must define the input pointer to which subsequent list commands are transferred. This corresponds to opening a list, see Chapter 6.4.1 "Loading Lists", page 108.

By additional control commands, the processing of the transferred list commands can be started.

6.3.1 Lists and the Protected RTC6 List Memory Area

The RTC6 List Memory offers 8M = 8,388,608 = 2^{23} storage positions in total.

In general, it can be split into 3 areas. There sizes are freely configurable.

- 2 RTC6 List Memory areas, "List 1" and "List 2". In this manual, these are also simply designated as "lists".
- User-definable is in addition:
  a third "protected RTC6 List Memory area", "List 3".
  This is protected against unintended overwriting (by loading normal command lists).

"List 1" and "List 2"

In principal, both RTC6 List Memory areas "List 1" and "List 2" can be used in a manner identical to the two RTC6 List Memory areas of the RTC5 or RTC4 boards, for example, for continuous loading and processing of command lists.

RTC6 List Memory is even bigger. Furthermore, the size of each RTC6 List Memory area can be freely configured, see Chapter 6.3.2 "Configuring the RTC6 List Memory", page 106.

For list handling, see Chapter 6.4 "List Handling", page 108.

"List 3"– Protected RTC6 List Memory Area

"List 3" is intended for a protected storage of frequently needed list command sequences (as subroutines or character set definitions). It is protected against unintended overwriting (during loading of normal command lists).

There are principally two alternative ways to utilize this protection feature:

(1) Subroutines can be written to the upper positions of the RTC6 List Memory area. These subroutines can be subsequently assigned to the protected RTC6 List Memory area "List 3". Such subroutines are called – both initially in the RTC6 List Memory area as well as subsequently in the protected RTC6 List Memory area "List 3" – by **list_call**, specifying an absolute memory position.

(2) Special commands allow subroutines and character set definitions to be loaded directly in the protected RTC6 List Memory area "List 3" as indexed subroutines or definitions. They can then be called by providing the corresponding index. Indexed character set definitions can, for example, be used in conjunction with **mark_text** for directly marking text.

SCANLAB strongly recommends *not* intermixing usage of these two methods. Otherwise, unintended data loss by overwriting can occur even in the protected RTC6 List Memory area "List 3".

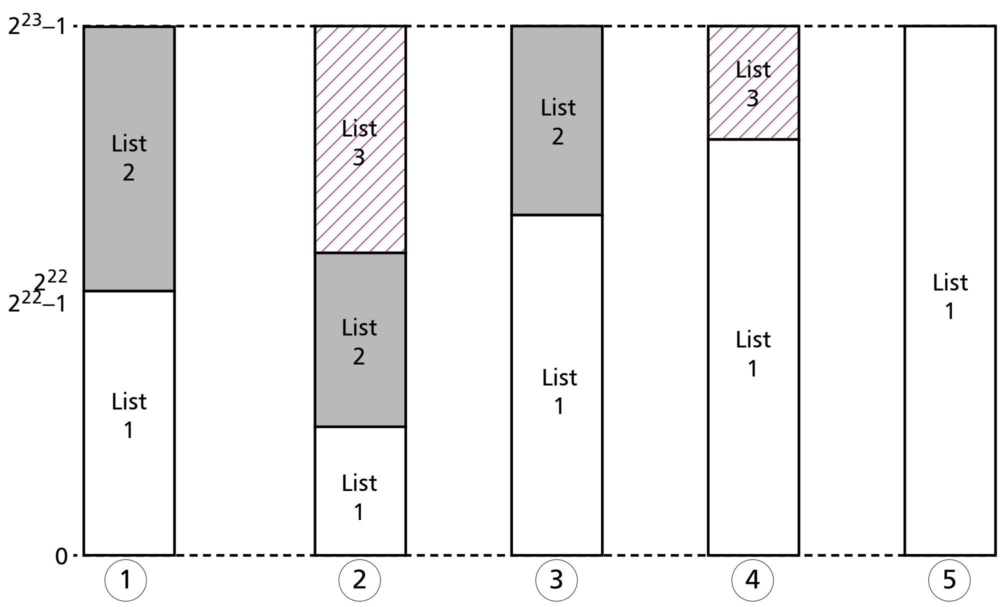
The RTC6 command set includes appropriate conversion commands so that users are not forced to continuously use only one of the two methods. The defining of subroutines and character sets, as well as their management and subsequent conversion options are detailed in Chapter 6.5 "Structured Programming", page 115.

6.3.2 Configuring the RTC6 List Memory

The RTC6 List Memory offers 8M = 8,388,608 = 2^{23} storage positions in total. After **load_program_file** the RTC6 List Memory areas “List 1” and “List 2” can each store 4,194,304 list commands. The protected RTC6 List Memory area “List 3” cannot store list commands, see Figure 32.

By **config_list**, the RTC6 List Memory areas can be can reconfigured.

If a user program only needs *one single* list, then the total RTC6 List Memory can be treated as a single RTC6 List Memory with a total capacity of 2^{23} storage positions, see Configuration (5) in Figure 32).



Legend

Left: numbers of the limiting memory positions of the two RTC6 List Memory areas “List 1” and “List 2”.

1 Default configuration (= without protected RTC6 List Memory area “List 3”). Corresponds to: `config_list( 4,194,304, 4,194,304 )`

2 `config_list( Mem1, Mem2 )` with `Mem1` > 0, `Mem2` ≤ 2^{23} – `Mem1`

3 `config_list( Mem1, -1 )` with `Mem1` > 0

4 `config_list( Mem1, 0 )` with `Mem1` > 0

5 `config_list( -1, x )` with `x` as desired

32

Examples of allowed RTC6 List Memory configurations.

Generally, during configuration are assigned:

- lower storage position numbers to “List 1”
- medium storage positions numbers to “List 2”
- upper storage position numbers to “List 3”

When configuring the RTC6 List Memory areas, the following must be observed:

- “List 1” must contain at least *one* storage position
- the total sum of storage positions for “List 1” and “List 2” must not exceed 2^{23}.

Other than that, the sizes of “List 1” and “List 2” can be set as desired. For example, “List 2” can be configured even with 0 storage positions, see configuration (4) and (5) in Figure 32.

With every configuration, remaining storage positions (not assigned to “List 1” or “List 2”) are automatically assigned the protected RTC6 List Memory area “List 3”.

The memory content is not altered by the configuration process. Therefore, repeating the call with differing parameters is nondestructive.

When altering the configuration, you should observe also the following:

- List boundaries should not be moved to within an eventual subroutine.
- The protection of a “List 3” range is removed, if this range is assigned to RTC6 List Memory area “List 1” or “List 2”.
- Valid jump addresses specified in Jump Commands might become invalid if the configuration is altered, see Chapter 6.5.3 “Jumps”, page 123.
- After the protected RTC6 List Memory area “List 3” has been made larger, defragmentation might be needed to make the newly assigned RTC6 List Memory area usable, see Section “Index Management and Defragmentation”, page 118.

6.4 List Handling

The two RTC6 List Memory areas “List 1” and “List 2” serve as intermediate storage for the continuous loading and processing of list commands.

6.4.1 Loading Lists

“List 1” and “List 2” are enabled to be filled with list commands by **set_start_list_pos**, **load_list** or other control commands (see below). An input pointer is thereby defined for the selected list. This input pointer specifies the memory position to which the subsequent list commands are transferred.

Lists are self-contained memory blocks for list commands. When in the process of list loading the list end is reached without setting the input pointer to another list, then the input pointer is automatically reset to the start of the current list, where loading continues.

An automatic change of the input pointer to another list never occurs, particularly not to the protected RTC6 List Memory area “List 3”.

In general, when list commands are loaded into storage positions, any list commands previously stored there are overwritten. This occurs even if they have not yet been processed or are currently being executed. User should make sure not to overwrite commands still needed by the user program (see below).

PCI transfer of the list commands into RTC6 List Memory is buffered to increase the speed for continuous downloads. The buffer is 16 commands in size.

Whenever the buffer is full or when the commands **set_end_of_list**, **list_return**, **set_input_pointer** (and related commands), **execute_list_pos** (and related commands), **auto_change**, **auto_change_pos**, **start_loop** or **release_rtc** are called, this automatically results in a flush. Thereby, the still buffered list commands are transferred to the RTC6 List Memory.

A flush can be initiated at any time by `set_input_pointer(get_input_pointer())`, even if the buffer is yet “incomplete”. This is only necessary in some circumstances when list commands should be processed and list input is not yet finished (for example, with an External Start).

“Unconditional” Loading

The input pointer is set:

- to the beginning of the selected list by
  - **set_start_list**
  - **set_start_list_1**
  - **set_start_list_2**
- to the specified address of the selected list by
  - **set_start_list_pos**
  - **set_input_pointer**

The next list command is written to this address regardless of the current status of the specified list, see Chapter 6.4.2 “List Status”, page 110.

If needed, the current positions of the input and output pointer can be queried by **get_input_pointer** or **get_list_pointer** and **get_status** or **get_out_pointer** - for example, to ensure that not-yet-processed list commands are not overwritten.

Loading with Protection

The loading process is initialized by **load_list**, which sets the input pointer to the specified address in the selected list (just like **set_start_list_pos**). However, this occurs only, if the selected list is not currently in use.

Alternatively, you can simply let the input pointer be set to a currently non-active or already processed list by **load_list** (the RTC6 PCIe Board automatically determines the corresponding appropriate list).

The return value of **load_list** reveals if, and in which list, the loading procedure has been successfully initialized.

Otherwise, the input pointer is set to an invalid position. Then, no further list commands can be input until the input pointer is correctly set back to a valid position again (for example, by repeating **load_list** with a positive result or **set_start_list_pos**).

This automatically prevents unintentional overwriting of commands that are still to be executed.

load_list is useful in scenarios such as alternating list changes, where you want to wait specifically for a list to be processed, see Section "Alternating List Changes", page 114.

Terminating Lists

A command list can be, but need not necessarily be, terminated by a **set_end_of_list**.

However, if an unterminated command list is executed and the output pointer thereby encounters the last possible position in the list, the output pointer automatically resets to the start of the list and processing continues there.

Automatic list changing after a list is processed can only occur, if the list has been terminated by **set_end_of_list**, see Chapter 6.4.6 "Changing Lists Automatically", page 113.

The loading of a **set_end_of_list** does *not* stop the loading procedure itself. Therefore, list commands immediately following a **set_end_of_list** are still loaded into the same list.

6.4.2 List Status

Dependent on the command input and output statuses, lists receive particular list status values (compare to Chapter 6.4.3 "List Execution Status", page 111).

By control command **read_status**, the current list status values can be queried – separately for both lists.

- LOAD list status

  The LOAD list status LOAD1 or LOAD2 indicates that the input pointer is currently in this list. In any case, the LOAD list status of the other list is *not* set.
- READY list status

  The READY list status READY1 or READY2 is set when a **set_end_of_list** is written into the list during the loading procedure. The READY list status is reset when the LOAD list status of the list is newly set.
- BUSY list status

  The BUSY list status BUSY1 or BUSY2 indicates that the output pointer is currently in this list after list execution (of "List 1" or "List 2") has been started. In any case, the BUSY list status of the other list are then *not* set. The BUSY list status of a list is reset when **set_end_of_list** is executed (or is alternating set, if automatic list changing has been previously activated). If a list is opened for loading while still being processed, its BUSY list status still remains set.
- USED list status

  The USED list status USED1 or USED2 is set when a **set_end_of_list** is reached during processing. The USED list status is reset when the LOAD list status of the list is set.

Notes

- If the list status is queried during processing of a subroutine in the protected RTC6 List Memory area "List 3", then the status is returned of the list "List 1" or "List 2" in which the output pointer most recently resided (typically from where the subroutine has been originally called).
- If list execution is interrupted (by **pause_list**, **stop_list** or **set_wait**), then the above-mentioned status values remain unchanged.
- If list execution is aborted (by **stop_execution** or an External Stop), the USED list status is set for both lists (as by initialization). See also `load_list( ListNo = 3 )`.
- If you want to explicitly set the USED list status for a list `ListNo` (for example, after abortion by **stop_execution** or an External Stop), then load a **set_end_of_list** to a free position `Pos` of this list by **set_start_list_pos**(`ListNo`, `Pos`) and execute by **execute_list_pos**(`ListNo`, `Pos`). If no other list has been active at this moment, then list `ListNo` has the USED list status afterward.
- When interpreting the status values read back by **read_status**, always take into account the programmed loading or execution processes of the lists (see command description).

6.4.3 List Execution Status

In addition to the list status values, see Chapter 6.4.2 "List Status", page 110, list execution status values are provided.

These can be queried by the control command **get_status**.

- BUSY list execution status
  The BUSY list execution status is set when:
  – the RTC6 PCIe Board currently processes one of the two lists
  – a list has been paused by the control commands **pause_list** or **stop_list**
  The BUSY list execution status is *not* set when a list has been paused by the list command **set_wait** (is set again by a subsequent **release_wait**).

  BUSY list execution status and INTERNAL-BUSY list execution status cannot be set at the same time.
- PAUSED list execution status
  The PAUSED list execution status is set when processing of the list has been paused by **pause_list**, **stop_list** or **set_wait**. It is reset by a subsequent **restart_list** or **release_wait**, see also Chapter 6.4.5 "Interrupting Lists for Synchronization of Processing", page 113.
- INTERNAL-BUSY list execution status
  The INTERNAL-BUSY list execution status is set when the RTC6 PCIe Board is busy with executing a control command, which needs more than 10 µs for executing a scan motion (for example, **goto_xy** or possibly **set_offset**) or while a home jump or home return is executed (with **set_wait**, **set_end_of_list** or **release_wait**, if the home jump mode has been previously activated by **home_position** or **home_position_xyz**).

  BUSY list execution status and INTERNAL-BUSY list execution status cannot be set at the same time.
- HEAD BUSY list execution status
  (control command **get_status**, Bit #23 = 1)
  Operating status for SCANahead Systems only.

  As prerequisite, **set_scanahead_params** must have been executed with valid parameters, that is, the RTC6 board is configured to control SCANahead Systems.

  The HEAD BUSY list execution status is set, when:
  – A list is being processed
  – A list has been finished, but scan system output is still in progress (temporal offset = `PreviewTime`)
  – The last effective `LaserOffDelay` of the marking has not yet expired, that is, as long as the laser is still switched on
  This allows waiting until the actual end of marking before an equipment controller closes a shutter or power is cut to the scan system, etc.

  In contrast, if the laser has been manually switched on by **laser_signal_on** after the list end but before expiration of `PreviewTime`, then its LaserOff is not be waited for.

Notes

- With the RTC6 PCIe Board, the BUSY list execution status is available as BUSY OUT signal at:
  – LASER Connector, see Figure 17
  – EXTENSION 1 Socket Connector, see Figure 22
  – MARKING ON THE FLY Socket Connector, see Figure 25
- Some control commands are ignored (not executed), when the BUSY list execution status and/or INTERNAL-BUSY list execution status are set (for example, **auto_cal**, **goto_xy**, **load_correction_file**) or – with set INTERNAL-BUSY list execution status – are only executed with delay after the INTERNAL-BUSY list execution status has been reset again (for example, **execute_list_pos**, **set_offset**).

6.4.4 Starting and Stopping Lists

List processing ("List 1" or "List 2") can be started by:

- The control command **execute_list**
- An external start signal, see Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 311

execute_at_pointer can be used to start output of a list at a specified address. If an external start signal is used, see Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 311, then **set_extstartpos_list** allows definition of a start address for the External Start.

The RTC6 PCIe Board starts the execution immediately. Even during 10 µs-clocked execution of the list commands, you can still send control commands to the RTC6 PCIe Board. These are immediately executed without hindering execution of the list.

This is useful, for example, for loading a second list while the first list is being processed (the PC and scan head then work in parallel). However, the second list can only be started after processing of the first list has been finished. During list processing, **execute_list** or an external start signal is ignored.

Execution of a list can also be stopped at any time, for example, for implementing an emergency shutdown. As soon **stop_execution** is called or an external stop signal is transferred to the RTC6 PCIe Board, the currently executed list is aborted and the Signals for "Laser Active" Operation are turned off (but not deactivated).

With **range_checking**, the processing of a list can also be terminated automatically (like with **stop_execution**).

If, during list processing, the list end is reached without encountering a **set_end_of_list**, then processing continues at the beginning of the current list. This is repeated until either **stop_execution** is called or an external stop signal is transferred to the RTC6 PCIe Board.

If during list processing a **set_end_of_list** is reached, then list execution stops – unless **auto_change**, **auto_change_pos** or **start_loop** has been previously called, a list change takes place, see Chapter 6.4.6 "Changing Lists Automatically", page 113. This list change occurs only upon reaching a **set_end_of_list**.

Notes

- Lists are not automatically started. Regardless of how many commands are loaded, a list must be started as described in order to be processed.
- To also enable starting and stopping of list execution by external signals, the RTC6 PCIe Board provides corresponding control input ports, see Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 311.
- **set_pause_list_cond** or **set_pause_list_not_cond** can be used to set a condition for the 16-bit input port so that a **pause_list** is executed instead of **stop_execution** when an External Stop is present. In The list execution can then be continued by **restart_list**.

6.4.5 Interrupting Lists for Synchronization of Processing

The list command **set_wait** makes it possible to insert numbered break points ("wait markers") into a list. Each break point is associated with a number greater than zero. When the RTC6 PCIe Board reaches a break point during list execution, see Chapter 6.5 "Structured Programming", page 115, output of the list is temporarily interrupted and the laser is switched off.

get_wait_status checks whether list processing is currently interrupted at a break point. If processing is interrupted, **get_wait_status** returns the number (wait_word) of the break point (otherwise the value zero).

Break points are provided for synchronization purposes. The user program should perform a handling routine for each break point. When that handling routine is finished, list processing can be resumed (at the list command that follows) by the control command **release_wait**.

By **set_wait** the PAUSED list execution status (queryable by **get_status**) is set and the BUSY list execution status is reset. The opposite occurs after a subsequent **release_wait**.

List execution can be interrupted at any desired point in time by the control command **pause_list** (or by the synonym **stop_list**) and resumed by **restart_list**.
By **pause_list**, Signals for "Laser Active" Operation are suppressed and the scan system remains in the most recently defined state – even if in the middle of Microstepping. After a subsequent **restart_list**, the scan system resumes the planned motions (of the current command) and the Laser Control Signals are released again (in general, an interrupted marking cannot be continued without a disruption in the marking result).

By **pause_list** the PAUSED list execution status (queryable by **get_status**) is set and is reset by **restart_list**. The BUSY list execution status is left unchanged by both commands.

6.4.6 Changing Lists Automatically

If the RTC6 List Memory is configured for two RTC6 List Memory areas "List 1" and "List 2", see Chapter 6.3.2 "Configuring the RTC6 List Memory", page 106, then a second list can be loaded while the first list is still being processed.

It typically takes substantially longer to process a list than to write it into the memory. Continuous processing of arbitrarily long lists is therefore also possible, if they are divided into command blocks.

Continuous command output, which requires switching between two lists, can be achieved by automatic list changing as described in the following sections.

The commands for automatic list changing only take effect when the next following **set_end_of_list** is executed. That is, automatic list changing after processing a list can only occur if that list has been finished with a **set_end_of_list**. Otherwise, processing resumes at the beginning of the same list.

If the RTC6 List Memory is configured for a RTC6 List Memory area ("List 2") to size 0, then all automatic list change commands lead to "List 1" to the specified position.

One-Time List Change

auto_change and **auto_change_pos** activate an automatic, one-time list change between "List 1" and "List 2". After processing of the current list (when **set_end_of_list** is reached), processing of the next list is thereby automatically started.

When using **auto_change** the next list is started at position 0; when using **auto_change_pos** the next list is started the specified start position (RTC6 List Memory address as an offset to the beginning of the list).

Alternating List Changes

Another way to achieve continuous command output is by alternatingly repeating output of the two lists.

To do so, **start_loop** must be called. This causes a continuous, automatic and alternatingly repeating processing of both lists, provided both lists are finished each with **set_end_of_list**.

The alternating processing repeats until **quit_loop** is called. **quit_loop** terminates continuous processing as soon as the current list is finished.

The currently non-active list can be newly reloaded even as the other list is processed. This allows continuous alternating output of two lists with not only fixed content, but also constantly new content.

Notes

- The commands for starting a one-time automatic list change and **start_loop** to start an alternating list change can be called at any point in time. However, they do not take effect until the next **set_end_of_list** is reached.
- When loading a list while another is being processed, make sure no still-needed commands are thereby overwritten. Useful here is **load_list**, which only starts loading a list if it is currently not in use or already has been processed, see Chapter 6.4.1 "Loading Lists", page 108.
- Moreover, the currently new list should have made a certain amount of loading progress before the list change occurs.
  The input pointer should always be adequately ahead of the output pointer (because the PCI transfer of the list commands is buffered, see Chapter 6.4.1 "Loading Lists", page 108, and so-called short list commands can be used, see Chapter 6.1.2 "Control Commands and List Commands", page 97).
  Otherwise, "old" commands might be unintentionally executed.
- The RTC6 PCIe Board does not support the RTC4 circular queue mode, see Chapter 6.5.4 "RTC4-Circular Queue Mode", page 124. However, this operating mode can also be effectively replaced using an alternating list change and **load_list** described above.

6.5 Structured Programming

The RTC6 command set supports structured programming and output of list commands by numerous ways to define subroutines and character sets, as well as list commands for controlling program flow.

6.5.1 Subroutines

- As list-command sequences, subroutines can principally be located in any part of the RTC6 List Memory.
- Preferably, subroutines should be written to a upper portion of the RTC6 List Memory area, see Section ""List 3"– Protected RTC6 List Memory Area", page 105.
- A list boundary should not run through a subroutine.
- A subroutine must be terminated with a **list_return**.
- It can be defined:
  - Non-Indexed Subroutines
  - Indexed Subroutines

Non-Indexed Subroutines

As with "normal" list-command sequences, non-indexed subroutines are loaded into a RTC6 List Memory area ("List 1" or "List 2") by list-loading commands (see Chapter 6.4.1 "Loading Lists", page 108). Each subroutine must be terminated with a **list_return**. It is called by **list_call** with a parameter specifying the absolute memory address.

After the subroutine (including the terminating **list_return** command) has been processed, it is continued with the command that follows the calling position.

Notes

- Non-indexed subroutines cannot be written directly to the protected RTC6 List Memory area "List 3". However, they can be subsequently protected, see also Section "Subsequent Protection and Conversion of Non-Indexed Subroutines", page 119. For the subroutine to be indexed for this purpose with **set_sub_pointer**, however, its start address must be known. Prior to loading a non-indexed subroutine into the RTC6 List Memory area "List 1" or "List 2", you should therefore always read out the start address by **get_input_pointer**.
- Make sure that there is no **list_return** in a normal list flow or in a body of a subroutine, for which there has not been a corresponding subroutine call. Otherwise, with nested subroutine calls the integrity of the nesting is destroyed. If there is no still active subroutine call, list processing is continued at the absolute position 0.
  If a subroutine begins directly after a **list_call**, **list_call_abs**, **list_call_repeat**, or **list_call_abs_repeat**, then the return address is automatically set from `Pos(list_call) + 1` to `Pos(list_return) + 1`. That is, the next processed command is the one which follows after **list_return** but not the command which follows after **list_call** (which would process the subroutine once again having an uncorrelated **list_return**).

Indexed Subroutines

load_sub assigns a desired index to a subroutine (which is defined by subsequent list commands), and loads it into the protected RTC6 List Memory area "List 3".

An indexed subroutine must be terminated by a **list_return** (otherwise it is not stored). It is called by **list_call** (with a parameter specifying the index).

A maximum of 1024 subroutines can be stored. The management of the indexed subroutines occurs on the RTC6 PCIe Board automatically.

For more information on index management, see Section "Index Management and Defragmentation", page 118.

The memory address of an indexed subroutine can be queried by **get_sub_pointer**. With it, the indexed subroutine can be called by specifying the absolute memory address (just as with non-indexed subroutines).

An indexed subroutine is only stored by **load_sub** under the following circumstances:

- If prior to loading, configuration of "List 1" and "List 2" resulted in a protected RTC6 List Memory area "List 3" of sufficient size. For example, if all memory is assigned to one or both lists, then no indexed subroutines can be stored. **get_list_space**, if called after a **load_sub** (but before the terminating **list_return**), can be used for querying the amount of still-available memory in the protected RTC6 List MemoryRTC6 List Memory area "List 3"
- If the indexed subroutine is terminated by a **list_return**
- If **list_return** is preceded by no other command for positioning the input pointer (for example, another **load_sub**, **set_input_pointer**, or **set_start_list_pos**)
- If the index is within the valid range (0…1023)

After a **list_return**, the input pointer becomes invalid. Any subsequent list commands are no longer stored.

Indexed subroutines are written to "List 3" by **load_sub** commands in order of entry. The starting address is automatically set after the end of the last subprogram.

If an indexed subroutine is stored using an already-existing index, then the prior subroutine with that same index is not overwritten. It remains in the protected RTC6 List Memory area "List 3", though it is no longer indexed. Therefore, it can no longer be called through its index by **sub_call** (whereas it can be still called through its absolute memory address by **list_call**).

Use **get_sub_pointer** to query whether a subroutine is referenced by a particular index. If no subroutine is referenced, **get_sub_pointer** returns the value "–1" (that is, 2^{32}–1).

To load an indexed subroutine into the protected RTC6 List Memory area "List 3" that is already fully loaded with indexed subroutines, you must first appropriately expand the size of the protected RTC6 List Memory area "List 3" by **config_list** and then defragmenting it by **save_disk**/**load_disk**. Note that expanding the size alone is not sufficient, see Section "Index Management and Defragmentation", page 118.

Notes on Not-Indexed Calls

- Index management or defragmentation, see Section "Index Management and Defragmentation", page 118, can result in a change of the indexed subroutine absolute memory address. SCANLAB therefore advises against calling an indexed subroutine by **list_call**.

Rules for Programming

Observe the following guidelines when programming indexed subroutines:

- In an indexed subroutine, **set_end_of_list** is replaced by a **list_nop**.
- Absolute jumps within or out from the protected RTC6 List Memory area "List 3" are ignored during processing, see Chapter 6.5.3 "Jumps", page 123. Therefore, absolute jumps cannot be used in indexed subroutines.
- When the subroutine is processed, also ignored are:
  - Relative jumps that exceed the boundaries of an indexed subroutine
  - Jump Commands which initiate a jump to themselves

General Information on Calling Subroutines

Nested calls up to a maximum depth of 63 are possible.

When calling with **sub_call** or **list_call**, only relative Jump Commands and Mark Commands may be used, if the subroutine execution is to be repeated at different Image Field places.

To be able to use the absolute Jump Commands and Mark Commands, which are often easier to handle, the so-called "AbsCalls" are provided.

"AbsCalls"

If the subroutines contain only relative Vector Commands and "Arc" Commands, see Chapter 7.1.1 "Marking with Vector Commands and "Arc" Commands", page 139, the corresponding processes can be repeated at different places in the Image Field.

With "AbsCalls" the current position is taken over as offset. This offset is then taken into account for all subsequent Vector Commands and "Arc" Commands in the subroutine.

Nested calls are taken into account when the offset is determined. This can be used, for instance, to define character sets by absolute vectors.

"AbsCalls" from subroutines are made with **list_call_abs** and **sub_call_abs**.

Conditional Calls

To enable calling of subroutines dependent on external control signals, additional commands are available for conditional branching during program execution, see Chapter 9.3.2 "Execution of Conditional Commands", page 317.

Repeatedly Executed Calls

sub_call_repeat, **sub_call_abs_repeat**, **list_call_repeat** and **list_call_abs_repeat** can be used to automatically execute the body of a subroutine several times.

Index Management and Defragmentation

Index Management

To duplicate, renumber or convert indexed subroutines, **copy_dst_src** is provided.

By **copy_dst_src**, an indexed subroutine is indexed one more time. **copy_dst_src** only alters the corresponding entries in the internal management table and does not modify the RTC6 List Memory content.

No longer needed indices (unneeded entries in the internal management table) can be deleted by **load_sub** directly followed by a **list_return**. Here, too, deletion occurs only in the internal management table, while the list commands of the previously indexed subroutine continue to reside in the RTC6 List Memory.

get_sub_pointer can be used to query whether a subroutine for a particular index exists. If no subroutine exists, **get_sub_pointer** returns a "–1" value (that is, 2^{32}–1).

A true duplicate of an indexed subroutine in the protected RTC6 List Memory area "List 3" can be created (after **copy_dst_src**) with **save_disk**/**load_disk**. Subroutines with multiple indices are thereby written several times to the RTC6 List Memory. Keep this in mind in order to prevent unintended memory overflow of the protected RTC6 List Memory area "List 3".

load_sub always enters a new indexed subroutine after the indexed subroutine with the highest memory address. Therefore, subroutines that are no longer required and are located in the protected RTC6 List Memory area "List 3" may block memory positions for further indexed subroutines.

For this reason, simply increasing the size of the protected RTC6 List Memory "List 3" by **config_list** fails to produce further usable memory for storing additional indexed subroutines (the protected RTC6 List Memory area "List 3" can only be expanded downward, not upward).

Defragmentation

This situation can be resolved by defragmenting with **save_disk**/**load_disk**.

Thereby, all indexed subroutines and (by **set_sub_pointer**) subsequently indexed subroutines are rewritten to the protected RTC6 List Memory area "List 3" (in index sequence, starting at the lowest memory position of the protected RTC6 List Memory area "List 3").

The now-available upper memory positions can then be used for storing additional indexed subroutines.

Notes

- Before calling **load_disk**, be sure the protected RTC6 List Memory area "List 3" is of sufficient size after configuration of "List 1" and "List 2". Indexed subroutines without sufficient space there are *not* stored by **load_disk**. **save_disk** returns the number of stored list commands. No-longer-needed subroutines should previously deleted from the index management by a **load_sub** which is directly followed by a **list_return**.
- In some circumstances, index management or defragmentation can alter the absolute memory address of an indexed subroutine. It is therefore not advisable to call an indexed subroutine by **list_call**.
- **save_disk** stores subroutines starting from the start address to the first-encountered **list_return**. Relative jumps are not evaluated. So do not use branches to several **list_return**. Instead, reclose eventual branches prior to one single **list_return**.

Subsequent Protection and Conversion of Non-Indexed Subroutines

Non-indexed subroutines can be (directly) written only to a RTC6 List Memory area ("List 1" or "List 2"), but not to the protected RTC6 List Memory area "List 3".

There are basically two methods to protect non-indexed subroutines subsequently:

(1) Changing the configuration

The part of the RTC6 List Memory area in which the non-indexed subroutine has been written is assigned to the protected RTC6 List Memory area "List 3" by **config_list**. The subroutine subsequently protected in this manner remains non-indexed (with unaltered memory address) and can, as before, be called by **list_call**.

(2) Converting to indexed subroutines

set_sub_pointer is used to index a non-indexed subroutine and thus include it in the memory management of the indexed subroutines. By **save_disk**/**load_disk**, it can subsequently be copied as an indexed subroutine to the protected RTC6 List Memory area "List 3", see Section "Defragmentation", page 118.

With a subsequent call by **sub_call** via the index, the subroutine in the protected RTC6 List Memory area "List 3" is then started.

If you want to subsequently protect a non-indexed subroutine – either by method 1 or method 2 – then be aware that absolute jumps within and out from the protected RTC6 List Memory area "List 3" are not allowed, see Chapter 6.5.3 "Jumps", page 123.

With converting to indexed subroutines (method 2), also all other programming rules for indexed subroutines must be observed, see Section "Rules for Programming", page 117.

Always try to use only one of the two methods. This avoids unintended data loss in the protected RTC6 List Memory area "List 3" by overwriting.

If you begin working with method 1 but later want to also use indexed subroutines: then you should convert all non-indexed subroutines residing in the protected RTC6 List Memory area "List 3" to indexed subroutines using method 2, before you define the first indexed subroutine by **load_sub**.

When doing so, observe the following Notes.

Notes

- If method 1 is used and you remove overwrite-protection for a part of the protected RTC6 List Memory area "List 3", then you risk overwriting indexed subroutines or previously protected subroutines.
- Non-indexed subroutines subsequently protected with method 1 can under some circumstances be overwritten by a later **load_sub** or **load_disk**.
- **set_sub_pointer** links the supplied index with the specified start address, even if an indexed subroutine had already been previously defined for this index. The original indexed subroutine with this index is then no longer indexed and no longer callable by the index.

- If using method 2, you should use it fully. If the **set_sub_pointer** alone is executed, then the subroutine is already callable by **sub_call** and its index, but the subroutine remains unprotected against overwriting. Protection is obtained only after the subroutine is subsequently copied as an indexed subroutine by **save_disk**/**load_disk** into the protected RTC6 List Memory area “List 3”.
- **save_disk** ignores all non-indexed subroutines, even those subsequently protected in the protected RTC6 List Memory area “List 3” by method 1. Be aware that they can be overwritten there by **load_disk**.
- **save_disk**/**load_disk** automatically replaces unallowed commands (for example,, **set_end_of_list**) with **list_nop** commands.
- Indexed subroutines repeatedly indexed with **copy_dst_src** are duplicated in the RTC6 List Memory at a subsequent **save_disk**/**load_disk**. This can result in a memory overflow in the protected RTC6 List Memory area “List 3”.
- Before executing **load_disk**, be sure the protected RTC6 List Memory area “List 3” is of sufficient size after configuration of “List 1” and “List 2” (**save_disk** returns the number of stored list commands). An indexed subroutine is *not* stored by **load_disk** if there is not sufficient memory.
- Conversion of a subroutine by method 2 changes the absolute memory address of the subroutine.

Unprotecting Subroutines

The protection of a subroutine stored in the protected RTC6 List Memory area “List 3” is removed, if it is assigned by **config_list** to one of the RTC6 List Memory area “List 1” or “List 2”.

The subroutine can then still be called using the same parameters (index or absolute memory address). But it no longer has protection against unintentional overwriting.

6.5.2 Character Sets and Text Strings

For marking tasks, it is convenient to use the RTC6 List Memory for storing command lists as separate subroutines that define how the scan system should mark the needed characters and/or text strings.

To simplify management of characters and text strings, the RTC6 PCIe Board provides the possibility of storing indexed character definitions and text string definitions in its protected RTC6 List Memory area "List 3" and calling them by simple commands.

Indexed character definitions and text string definitions are essentially indexed subroutines, but definable and callable by their own commands, and managed by a dedicated internal RTC6 PCIe Board management table – separately from indexed subroutines.

The individual character and text string definitions must specify the shape and orientation (for example, parallel to the x or y axis) of the characters or text strings. Both relative and absolute Vector Commands can be used for this. The end position of a character or a text string should be chosen to serve as the start position of a subsequent character. Each character definition or text string definition must be terminated with **list_return**.

Defining Indexed Character Sets

A sequence of character-defining list commands can be directly stored in the protected RTC6 List Memory area "List 3" by **load_char** (the resultant automatically-assigned memory address can be queried by **get_char_pointer**). Alternatively, a non-indexed subroutine can be subsequently indexed with **set_char_pointer** and then copied by **save_disk**/**load_disk** as an indexed character in the protected RTC6 List Memory area "List 3".

The RTC6 PCIe Board manages up to 4 character sets, each with 256 indexed characters.

Other than that, the same rules as for indexed subroutines are applicable, see Section "Indexed Subroutines", page 116 and Section "Subsequent Protection and Conversion of Non-Indexed Subroutines", page 119.

Notes

- \0 (NUL) is a markable character, too.
  \0 also serves as a text-output delimiter (for text strings), in which case it is not marked.
- Indexed character set definitions cannot use **mark_text**, **mark_time**, **mark_date** and **mark_serial**. Otherwise, improper marking might occur during execution of the indexed character.

Calling Indexed Characters

Marking of an individual character is started by calling **mark_char** (or the "AbsCall" command **mark_char_abs**) along with the index of the corresponding indexed character definition.

To label serial numbers, indexed characters (digits) can also be called up with **mark_serial**, see Chapter 7.5 "Marking Dates, Times and Serial Numbers", page 219.

The marking of entire text passages can be started by **mark_text** (or the "AbsCall" command **mark_text_abs**). The desired character set can be selected in advance by **select_char_set**.

When a **mark_text** is loaded, the to-be-marked text (if more than 12 characters in length) is split into blocks of 12 characters, with each block receiving its own **mark_text** in the RTC6 List Memory. Make sure that no unwanted memory overflow of the respective RTC6 List Memory area occurs.

Defining Indexed Text Strings for Time, Date and Serial Number

For the marking of times, dates and serial numbers, it can be useful to define text strings such as months ("January"…"December", "Jan."…"Dec.", "/01/"…"/12/" etc.) and days of the week ("Sunday"…"Saturday" or "Sun."…"Sat." etc.).

Here, you can likewise use previously-defined character sets with the **mark_char** and **mark_text**.

With **load_text_table**, a sequence of list commands defining a text string can be loaded directly into the protected RTC6 List Memory area "List 3" as an indexed text string (the resultant automatically-assigned memory address can be queried by **get_text_table_pointer**).

Alternatively, a non-indexed subroutine can be subsequently indexed with **set_text_table_pointer** and then copied by **save_disk**/**load_disk** as an indexed text string in the protected RTC6 List Memory area "List 3".

The RTC6 PCIe Board manages up to 42 indexed text strings.

Other than that, the same rules as for indexed subroutines are applicable, see Section "Indexed Subroutines", page 116 and Section "Subsequent Protection and Conversion of Non-Indexed Subroutines", page 119.

Notes

- **set_char_table** is synonymous with **set_text_table_pointer**.

Calling Indexed Text Strings

Indexed text strings can be called for marking times, dates and serial numbers by **mark_time**, **mark_date** and **mark_serial** (or the "AbsCall" commands **mark_time_abs**, **mark_date_abs** and **mark_serial_abs**), see Chapter 7.5 "Marking Dates, Times and Serial Numbers", page 219.

Managing Indexed Characters and Text Strings

The index management of indexed characters and indexed text strings occurs separately from the index management of indexed subroutines.

Index management by users (renumbering, duplicating, …) resembles index management of indexed subroutines, see Section "Index Management and Defragmentation", page 118, using **copy_dst_src**, **load_char**, **load_text_table**, **get_char_pointer**, **get_text_table_pointer** and **save_disk**/**load_disk**. Defragmentation of the protected RTC6 List Memory area "List 3" also includes indexed characters and text strings.

6.5.3 Jumps

list_jump_pos (synonymous with **set_list_jump**) and **list_jump_rel** allow the definition of jumps to a specified address which are carried out by the RTC6 PCIe Board at runtime.

With **list_jump_pos**, an absolute memory address within the configured RTC6 List Memory area ("List 1" and "List 2") can be specified. Jumps into and out of the protected RTC6 List Memory area "List 3" are not allowed with **list_jump_pos**. A **list_jump_pos** having such an unallowed jump address is ignored during execution.

With **list_jump_rel**, jump distances (that is, relative memory addresses) can be specified. **list_jump_rel** can be used in all RTC6 List Memory areas, even the protected RTC6 List Memory area "List 3". Nevertheless, when specifying jump addresses, you should be sure the jump does not exceed the boundary of the corresponding RTC6 List Memory area. Otherwise, **list_jump_rel** is ignored by the RTC6 PCIe Board during processing.

If **list_jump_rel** is used in an indexed subroutine, you must further ensure the jump does not exceed the boundaries of the subroutine. During processing of indexed subroutines, relative jumps that exceed the boundaries of a subroutine are ignored by the RTC6 PCIe Board.

Notes

- Reconfiguration of the RTC6 List Memory or conversion of a subroutine can result in an originally-valid jump address becoming invalid due to new list boundaries or an altered subroutine position in the memory. In this case, the RTC6 PCIe Board ignores the corresponding Jump Command – hence, the user program does probably no longer function as intended. Therefore, exercise care when programming Jump Commands.
- When conditional Jump Commands are used, execution of a jump is dependent on an external control signal, see Chapter 9.3.2 "Execution of Conditional Commands", page 317.
- Jump Commands initiating a jump to themselves as `list_jump_rel( 0 )` are ignored at runtime to prevent an infinite loop that excludes further activities. On the other hand, conditional Jump Commands as `list_jump_rel_cond( Mask1, Mask0, 0 )` are allowed, for example, to wait for confirmation of a signal.

6.5.4 RTC4-Circular Queue Mode

With the RTC6 PCIe Board, the RTC4 circular queue mode does not exist.

Nevertheless, users can actually replace this operational mode with the RTC6 PCIe Board by using an alternating list change and **load_list**.

`load_list ( 3, 0 )` ensures that new commands are loaded only into an already processed list (that is not BUSY list execution status), without needing to explicitly specifying the number of the list, see also Section "Alternating List Changes", page 114 and Section "Loading with Protection", page 109.

6.5.5 Loops

Although list jumps, see Chapter 6.5.3 "Jumps", page 123, and conditional jumps, see Chapter 9.3.2 "Execution of Conditional Commands", page 317, let you repeat any number of list commands limitlessly or under external control, precisely specifying the number of executions is not always reliably possible.

But this can be achieved by the command pair **list_repeat** and **list_until**. The command sequence between these two short list commands execute exactly as often as specified with the **list_until** command's parameter, but at least once. Here, nesting up to 8 loops deep is allowed.

list_repeat and **list_until** must always be used in pairs. Unpaired or supernumerous commands (**list_until** without an associated **list_repeat**, as well as **list_repeat** commands leading to a nesting depth greater than 8) are ignored. Empty loops (for example, **list_repeat** directly followed by **list_until**) terminate immediately and are not repeated.

The command pairs can be located both within lists and within subroutines.

Within subroutines, **list_until** performs a **list_jump_rel** to the address directly after the associated **list_repeat**. Loops do not function beyond the boundaries of a subroutine, because list jumps into or out of subroutines are not allowed, see Chapter 6.5.3 "Jumps", page 123.

Within lists, however, **list_until** executes a **list_jump_pos** (to the address directly after the associated **list_repeat**). Thus, **list_repeat** and **list_until** can even reside in two different lists, provided that list changing is ensured (by either an explicit list jump or an automatic list change).

If, on the other hand, a list actually has been terminated (as may be the case when using **auto_change_pos**), then the **list_repeat** stack gets automatically deleted and the started loop can no longer be ended because the next **list_until** no longer finds an associated **list_repeat**.

set_end_of_list deletes the entire loop management, if no automatic list change is pending, but **list_return** does not.

Explicit list jumps into or out of the body of a **list_repeat**/**list_until** loop are allowed because they cannot be monitored. Careless use could therefore compromise loop management integrity so severely that started loops do not execute as expected (but subroutine calls from inside a loop are always reliably possible as long as the subroutine itself contains no unpaired **list_repeat**/**list_until** commands).

If a **list_repeat**/**list_until** loop is to be executed with an initially unknown number of repetitions, a high value (for example, greater than the highest expected number) can be specified for the `Number` parameter of **list_until**. Within the loop, a conditional branch (for example, which is dependent on an external signal) can jump to a position outside the loop and leave the loop this way. At this point there should be a **list_until**(`Number` = 0) to end the just left loop properly.

6.6 Using Several RTC6 PCIe Boards in One PC

As many RTC6 PCIe Boards as the PCIe bus allows can be operated simultaneously in one PC. However, the RTC6 DLL internal card management can manage a maximum of 255 RTC6 PCIe Boards and RTC6 Ethernet Boards at the same time, see Chapter 16.5.3 "About the RTC6 Board Management", page 985.

All RTC6 PCIe Boards work independently of each other. The command lists of all boards can be loaded and executed at any time.

The RTC6 command set allows two methods to write user program when using several RTC6 PCIe Boards:

- "Multi-Board Programming", page 126
- "Single-Board Programming", page 127

6.6.1 Multi-Board Programming

With Multi-Board Programming, the multi-board commands (= single-board command names with prefix "**n_**", see Multi-board Com'd Name, page 340) are used.

Multi-board commands serve for the simultaneous use of several RTC6 boards, see also Chapter 6.6 "Using Several RTC6 PCIe Boards in One PC", page 126.

With Multi-board commands, in contrast to single-board commands (= command names without prefix "**n_**"), `CardNo` must be specified as first parameter. `CardNo` is the board number[1] to which the command is to be transmitted. All other parameters are identical.

The installed RTC6 PCIe Boards are numbered in the order found during initialization (starting with 1), see Chapter 16.5.3 "About the RTC6 Board Management", page 985.

The multi-board command **n_get_serial_number** can be used to determine which RTC6 PCIe Boards have been assigned numbers. See also example (3) below.

rtc6_count_cards returns the number of RTC6 PCIe Boards in the RTC6 board management.

Notes

- Multi-board commands are sent to the active board (default board), if the specified number is > 255 or 0 (real boards begin at 1).
- The Multi-board command is rejected, if no real board is entered in the RTC6 board management under the specified number.
- All multi-board commands are listed in Chapter 10 "RTC6 Commands". For almost every single-board command a corresponding multi-board command is available, see line Multi-board Com'd Name, page 340 in the corresponding command description.

Examples (Pascal)

(1) Write a Jump Command to the point (500, 500) into the current list of RTC6 PCIe Board #1:

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

(2) Process list with number `list_no` (1 or 2) on the RTC6 PCIe Board with the number specified by the variable `RTC6_no`:

```
n_execute_list(RTC6_no, list_no)
```

(3) Return the serial number of RTC6 board #1:

```
sn_1 := n_get_serial_number(1)
```

(1) As an unsigned 32-bit value.

6.6.2 Single-Board Programming

During single board programming, one of the inserted RTC6 PCIe Boards is defined with **select_rtc** as default card. All single board commands following **select_rtc** are sent to the defined board until **select_rtc** is called once again.

Multi-Board commands are not influenced by **select_rtc** (if the card number is valid, see above).

Care must be taken if a process uses multiple boards by multiple threads, because **select_rtc** is not thread-specific but board-specific. It immediately redirects the output of *all* currently running threads of a process to the specified RTC6 PCIe Board.

Notice!

- **select_rtc** defines the active RTC6 PCIe Board for all threads of one process (user program) that are currently running.
  In multi-threaded user programs, this can result in programming errors.

6.6.3 Master/Slave Operation

If several RTC6 PCIe Boards are to be operated clock-synchronized, then they must be connected pairwise with each other by the Master and Slave connectors and installed in preferably (recommended) adjacent PCIe slots. Connect the Master connector of one board to the Slave connector of another board. Suitable connection cables, see Figure 8, are available from SCANLAB.

An RTC6 PCIe Board automatically gets the master board of a master/slave chain, if a further RTC6 PCIe Board is connected to its Master connector but no further RTC6 PCIe Board is connected to its Slave connector. All other RTC6 PCIe Boards are slave boards. See also Chapter 4.4 "Master Socket Connector, Slave Socket Connector", page 68.

get_master_slave can be used to query separately for each RTC6 PCIe Board the master/slave status, that is, whether it is operated as a master, slave or single board.

For a source code example on how to check which RTC6 PCIe Board is the master and which one is slave, see Section "Example Code (Delphi)", page 130.

After synchronization, the clock phase of each board is delayed by 0…3 1/64 μs clock cycles (= approx. 0 ns…50 ns) in relation to the clock phase of the preceding (upstream) board.

Without synchronization, delays of up to 10 μs can occur. You can use **get_sync_status** to check if a slave board is synchronized to the master board (or to the preceding board in the master/slave chain).

Initialization with RTC6 Software Package ≥ 1.5.0 (≥ RBF 619)

On all RTC6 PCIe Boards of a master/slave chain must have been executed:

- **load_program_file**

Clock Phase Synchronization

The synchronization takes place automatically as soon as 2 RTC6 PCIe Boards are connected. **sync_slaves** does not have to be called anymore.

The synchronization status can be queried at any time with **get_sync_status** even without a prior call of **simulate_ext_start**.

If an RTC6 PCIe Board has been synchronized to external clock cycles in the meantime (see Chapter 7.4.10 "Synchronization of the RTC6 Clock Cycle and an External Clock Signal", page 216), it is automatically resynchronized with the master board upon leaving this state.

When **load_program_file** is executed, the synchronization of all subsequent slave boards is lost for a short time. However, it is automatically reestablished.

Notes

- The master board does not pass encoder signals to the slave board(s). They must always be individually supplied to the slave board(s). Here, you need to take into account the 0 ns…50 ns clock phase shift.
- The correction files and lists must be loaded separately onto all RTC6 PCIe Boards.
- By `get_sync_status( Bit #21…Bit #30 )`, the exact propagation time in 1/64 µs clock cycles between two RTC6 boards (outbound and return) can be read out.

Initialization with RTC6 Software Package < V1.5.0 (≤ RBF 618)

On all RTC6 PCIe Boards of a master/slave chain must have been executed:

- **load_program_file**
- **load_correction_file**

The synchronous timing with stable phase position of a master/slave chain is severed by the first not-initialized board. If an RTC6 PCIe Board is initialized by **load_program_file** but connected as slave to a board which has not been initialized by **load_program_file**, then it is subject to its own clock with a random phase position.

Clock Phase Synchronization

If the RTC6 PCIe Boards of a master/slave chain are to be synchronously clocked with a defined relative clock phase, then the boards must be correspondingly synchronized by **sync_slaves**.

For this, it is necessary to send **sync_slaves** one-time to the master board only. SCANLAB recommends performing the synchronization immediately after all boards have been initialized (by **load_program_file** and **load_correction_file**). It is sufficient to call **load_correction_file**(0,1,2) or to temporarily detach all *scan heads*.

Notes

- The master board does not pass encoder signals to the slave board(s). They must always be individually supplied to the slave board(s). Here, you need to take into account the 0 ns…50 ns clock phase shift.
- If a board in a synchronized master/slave chain is externally clocked separately by a cycle synchronization, then this clocking refers exclusively to this board. All other boards in the master/slave chain continue to be synchronized with the original clock of the master board. If cycle synchronization is then deactivated again, the affected card remains asynchronous. It can only be synchronized again by calling **sync_slaves** once more.
- The correction file and lists must be loaded separately onto all RTC6 PCIe Boards.

Synchronous Starts and Synchronous Stops

Within a master/slave chain, External Starts (if enabled with **set_control_mode**) and External Stops are passed on:

- from one board to all boards

Therefore, it can be triggered:

- A synchronous start of all synchronized boards within a master/slave chain (with presettable track delays) can be triggered by an external start signal, a **simulate_ext_start** or a **simulate_ext_start_ctrl** at any board
- A synchronous stop of all synchronized boards within a master/slave chain by an external stop signal or a **simulate_ext_stop** at any board

In contrast, *not* passed on are:

- Starts by **execute_list**
- Starts by **execute_at_pointer**
- Stops by **stop_execution**

Therefore, these must be separately executed even at master/slave-synchronized boards.

Notes

- See also Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 311.
- With **master_slave_config** the master/slave interface properties can be configured individually for each RTC6 board.

Example Code (Delphi)

The following example Delphi source code shows how to check which RTC6 PCIe Board is the master and which one is slave.

The code must be included in a user program, see Chapter 6.2.5 "Example Code (C)", page 103.

```
if init_rtc6_dll() <> 0 then halt;      // Initialize the RTC6 DLL
if rtc6_count_cards() <> 2 then halt;   // Are 2 RTC6 in the PC?

for CardNo := 1 to 2 do                 // Load program and 3D correction files onto both boards
begin
   // Stop RTC6 if a task is currently still running. Optional:
   // n_stop_execution(CardNo);
   if n_load_program_file(CardNo, nil) <> 0 then halt;
   if n_load_correction_file(CardNo, nil, 1, 2) <> 0 then halt;
end;

// Check for Errors (Bit #12...Bit #15)
if (n_get_sync_status(1) and $F000 = 0) and (n_get_sync_status(2) and $F000 = 0) then
begin
   // Detect master board
   // Is board 1 the master and board 2 a single slave?
   if (n_get_master_slave(1) = 2) and (n_get_master_slave(2) = 1) then
   begin
      Master := 1;
      Slave := 2;
   end else
   // Is board 2 the master and board 1 a single slave?
   if (n_get_master_slave(1) = 1) and (n_get_master_slave(2) = 2) then
   begin
      Master := 2;
      Slave := 1;
   end
   else
   halt;                                      // Something wrong with master-slave configuration
end else
halt;

n_select_cor_table(Master, 1, 0);
n_select_cor_table(Slave, 1, 0);
n_set_control_mode(Master, 1 + 8);           // Master slave, activate Start Stop control
n_set_control_mode(Slave, 1 + 8);
n_sync_slaves(Master);                       // Synchronize master and slave boards
// check synchronization status at any time
// provide an External Start or a simulated External Start before checking
n_simulate_ext_start_ctrl(Master); // for ≤ RBF 618
Result = n_get_sync_status(Master) and $3FF; // must be 640
Result = n_get_sync_status(Slave) and $3FF; // must be <4

// MasterCfg, SlaveCfg must be defined elsewhere, see manual
n_master_slave_config(Master, MasterCfg);
n_master_slave_config(Slave, SlaveCfg);
```

6.7 Usage of RTC6 PCI Express Boards by Several User Programs

Usage of RTC6 PCIe Boards by several user programs is coordinated by the (RTC6 DLL-internal) RTC6 board management, see also Chapter 16.5.3 "About the RTC6 Board Management", page 985. By **init_rtc6_dll**, it is initialized.

init_rtc6_dll automatically grants a user program access rights (by **acquire_rtc**) to the found boards, as long as the access right has not been already assigned to another user program (several RTC6 PCIe Boards or user programs can be used simultaneously, but no board can be simultaneously used by several user program).

Access rights (even if temporary) to boards are granted on an exclusive basis by the RTC6 DLL.

Multiple threads of *one* user program can use the same board, but can not send commands to it at the same time (the RTC6 DLL automatically serializes the command calls).

Without access rights, a board can only be accessed by a user program through purely RTC6 DLL-internal commands that do not require access rights (for example, **get_error**, **get_last_error** and **select_rtc**).

If a user program has granted access rights for a board, then this board can be acquired by another user program only after the original user program explicitly has been released it explicitly by **release_rtc** or **free_rtc6_dll**.

When a board is acquired by **acquire_rtc** (or **init_rtc6_dll** or **select_rtc**), a version check of the RTC6 DLL, `RTC6OUT.out`, and `RTC6RBF.rbf` is performed.

If these files are not loaded yet, then a version check cannot be explicitly performed but the check is still regarded as successful and does thus not hinder the board acquisition. If these files are loaded and the version check detects an error, then access is denied (**get_last_error** return code `RTC6_ACCESS_DENIED` | `RTC6_VERSION_MISMATCH`).

6.7.1 Notes on Board Acquisition by a User Program

init_rtc6_dll, **acquire_rtc**, **free_rtc6_dll**, **release_rtc** and **select_rtc** affect the access rights of RTC6 PCIe Boards. They do not initialize the RTC6 PCIe Boards.

A user program acquiring a board by **acquire_rtc** (or **init_rtc6_dll** or **select_rtc**) inherits its unadjusted memory contents and operational state. The user program therefore can use the stored data and settings of the board and could intervene in the flow of any list program started (by the previous user program).

If a user program releases a board by **release_rtc** and subsequently reacquires it – without it having been acquired in the meantime by another user program – then the RTC6 PCIe Board can be further used without changes, because in this situation all RTC6 DLL configuration data remain unaltered. However, the above does not apply, if the acquired board has been released by **free_rtc6_dll** and reacquired after **init_rtc6_dll**.

When an RTC6 PCIe Board is acquired by another user program, some important information managed by the previous user program only in the RTC6 DLL is *not* (automatically) taken over. The acquiring user program thereby lacks information related to memory configuration, protected-area management, or the operational status.

If board acquisition is followed by a board initialization by **load_program_file** and all settings are newly defined anyway, such missing information would not be relevant.

On the other hand, if the acquiring user program is supposed to further use the inherited state of the RTC6 PCIe Board, then it must explicitly query the missing RTC6 DLL information, receive it from the previous user program and explicitly re-establish it so that RTC6 DLL and board remain consistent. In this regard, observe the following notes.

Notes

- For a correct behavior of the input pointer at the list borders, the memory configuration currently set in the RTC6 DLL for the acquiring user program must be consistent to the current memory configuration of the acquired board. **get_config_list** obtains the current memory configuration of the board and sets it in the RTC6 DLL correspondingly.
- The management tables of protected functions (indexed subroutines, character sets and text strings) are located on the board.
  All protected functions stored on the board therefore remain callable. On the other hand, information on where the next protected function should be loaded is lost. This information can only be restored by **save_disk**/**load_disk**, see Section "Index Management and Defragmentation", page 118.
  An alternative restoration method is not possible. The intermixed loading of protected functions by differing applications should therefore be avoided.
- The input pointer is generally not inherited (the input pointer location currently saved in the RTC6 DLL for the acquiring user program is used, maybe corrected after **get_config_list**). On the other hand, output pointers of lists can be queried after an acquisition by **get_status**.
- After acquisition and until the next **load_…** call, the list status (in regards to LOAD list status and READY list status) might be incorrect. But for further execution this is not important, and the status is newly set after the next **load_…** call.
- Other settings such as **start_loop** or laser settings are not relevant to the RTC6 DLL. Though settings used by the previous user program can not generally be queried, new settings can of course be set as desired.
- Error handling is performed separately for each board and each user program. When access rights are exchanged, this data is not included.

6.8 Error Handling

In order to catch errors in the program flow, the RTC6 PCIe Board carries out a general error handling. With some commands there is also an individual error handling.

General errors occur, for example, if:

- the user program has no access rights for the board (RTC6_ACCESS_DENIED)
- the board fails to respond to a control command (RTC6_TIMEOUT)
- PCI communication problems occur during sending (RTC6_SEND_ERROR)

Individual errors occur, for example, if:

- calling a command with an unallowed (uncorrectable) parameter (RTC6_PARAM_ERROR, see, for example, **get_value** or **write_da_x**)
- rejected sending of a list command (RTC6_REJECTED, for example, due to an invalid input pointer)
- sending a control command at an improper time (RTC6_BUSY, for example, **goto_xy**, when a list is still being processed)

In such cases, the control commands are not executed and list commands are typically each replaced by **list_nop** (for example, for RTC6_PARAM_ERROR or RTC6_IGNORED, see **set_end_of_list** as an example).

The bits assigned to these errors are set or accumulated in the RTC6 DLL with each command in the board-specific error variables:

- LastError
  - Error code
  - Is automatically reset at the beginning of every command
  - Therefore, is a listing of occurred errors from the most recently executed command
  - Can be queried by **get_last_error**(1)
- AccError
  - Cumulative error code
  - Is reset when initializing the RTC6 DLL
  - Can be reset by the user program itself by **reset_error**
  - Are all accumulated error bits since the last error bit reset by **reset_error**
  - Can be queried by **get_error**(1)

Error handling takes place separately:

- for each board
- each user program

A **reset_error** does not delete the error code of another user program with current access rights to the specified board.

If access rights are exchanged, this data is not also exchanged.

Error handling only takes place during processing of commands within the RTC6 DLL and when sending to the RTC6 PCIe Board. Error handling takes place during execution of a list program.

An example code of how to incorporate board-specific error variables is provided in the command description of **get_error**.

(1) The described mechanism only applies for commands that establish communication with the RTC6 PCIe Board. Commands that *do not* establish communication with the RTC6 PCIe Board (for example, **rtc6_count_cards**, **set_rtc4_mode** or **get_serial_number**) neither generate nor alter LastError or AccError (see also comments in the corresponding Command Descriptions, page 342).

Some control commands (for example, **init_rtc6_dll**, **load_correction_file** or **load_program_file**) additionally return a special error code as the result value that is not buffered and must therefore be immediately evaluated or discarded by the user program.

6.8.1 Download Verification

Verification of RTC6 communication is vital particularly in medical applications. For this purpose, you can activate download verification separately for each board by **set_verify**.

However, this automatically results in extended download times.

If download verification is activated and an error is found, then the error code `RTC6_VERIFY_ERROR` is set, which can be queried by **get_last_error** or **get_error**. Certain operations might immediately be aborted and the board would then no longer be functional (for example, if **load_program_file** has been aborted).

With download verification activated, the following checks are performed (also note the comments in the command description of **set_verify**):

(1) Loading of list commands

For list-command downloads, each download is read back and compared (for equality) against the sent command. Here, only transfer to the RTC6 PCIe Board itself is checked; automatic parameter adjustments (for example, clipping) are not taken into account.

(2) Loading of control commands

For control commands, the corresponding parameters are read back and compared for equality against the sent parameters. Automatic parameter adjustments are not taken into account.

(3) load_program_file

For sending of **load_program_file**, the following is checked:

- `RTC6DAT.dat` is tested by a checksum for file correctness and PCI-transfer correctness.
- `RTC6RBF.rbf` is only checked by a bitwise transfer handshake. No other checking is possible.
- Each loaded section of `RTC6OUT.out` is immediately read back for checking. If an error is detected, then the loading process aborts.

(4) Loading of correction files

For loading by **load_correction_file**, the integrity of the to-be-loaded correction file is checked (by the checksum) and the transfer itself checked for correctness by an immediate read back of the correction table. For this function, the correction file must contain a checksum, see command description of **set_verify** and **verify_checksum**.

(5) Loading of tables

For loading other tables (for example, by **load_varpolydelay**), the transfer is checked for correctness by an immediate read back of the table. In addition to the **get_last_error** return code `RTC6_VERIFY_ERROR`, the corresponding error return value of the loading command is also get set.

6.8.2 Checking for Overruns

By **get_overrun**, it can be checked whether overruns of the 10 μs clock cycle have occurred. See also Section "Clock Overruns", page 187.

6.8.3 Example Code (C)

The following example C source code shows how to catch an error during initialization. It ensures the user program terminates with an error message, if:

- An error occurs during initialization with **init_rtc6_dll** (for example, if no RTC6 PCIe Board has been detected)
- The desired RTC6 PCIe Board (here: the board with serial number 12345) is not found
- Access is denied to the desired RTC6 PCIe Board
- An error occurs during **load_program_file** (for example, a version mismatch, file or system error)

The code must be included in a user program, see Chapter 6.2.5 "Example Code (C)", page 103.

```
UINT ErrorCode;
ErrorCode = init_rtc6_dll();
if ( ErrorCode )
{
   // Reading the number of RTC6 boards detected during initialization with init_rtc6_dll
   const UINT RTC6CountCards = rtc6_count_cards();
   if ( RTC6CountCards )
   {
      // Detailed error analysis for all detected boards
      UINT AccError( 0 );
      for ( UINT i = 1; i <= RTC6CountCards; i++ )
      {
         // Errors which occurred during execution of init_rtc6_dll
         const UINT Error = n_get_last_error( i );
         if ( Error != 0 )
         {
            AccError |= Error;
            const UINT SerialNumber = n_get_serial_number ( i );
            printf( "RTC6 board number %d (serial number %d): Error %d detected\n",
                        i, SerialNumber, Error );
            n_reset_error( i, Error );
         }
      }
      if ( AccError )
      {
         free_rtc6_dll();
         return;
      }
   }
```

```
   else
   {
      printf( "Initializing the DLL: Error %d detected\n", ErrorCode );
      free_rtc6_dll();
      return;
      }
   }
   else
   {
   // Reading the internal board number for the desired RTC6 board
   const UINT SerialNumberOfDesiredBoard ( 12345 );
   const UINT RTC6CountCards = rtc6_count_cards();
   UINT InternalNumberOfDesiredBoard ( 0 );
   for ( UINT i = 1; i <= RTC6CountCards; i++ )
   {
      if ( n_get_serial_number( i ) == SerialNumberOfDesiredBoard )
      {
         InternalNumberOfDesiredBoard = i;
      }
   }
   if ( InternalNumberOfDesiredBoard == 0 )
   {
      printf( "RTC6 board with serial number %d not detected.\n", SerialNumberOfDesiredBoard);
      free_rtc6_dll();
      return;
   }
   // Selecting the desired RTC6 board as the active RTC6 board for this user program
   if ( InternalNumberOfDesiredBoard != select_rtc( InternalNumberOfDesiredBoard ) )
   {
      // Errors which occurred during execution of select_rtc
      ErrorCode = n_get_last_error( InternalNumberOfDesiredBoard );
      if ( ErrorCode & 256 ) // RTC6_VERSION_MISMATCH
      {
         if ( ErrorCode = n_load_program_file( InternalNumberOfDesiredBoard, 0 ) )
         {
            printf( "n_load_program_file returned error code %d\n", ErrorCode );
         }
      }
      else
      {
         printf( "No access to RTC6 board with serial number %d\n", SerialNumberOfDesiredBoard );
         free_rtc6_dll();
         return;
      }
      if ( ErrorCode )
      {
         printf( "No access to RTC6 board with serial number %d\n", SerialNumberOfDesiredBoard );
         free_rtc6_dll();
         return;
      }
      else
      {
         // if n_load_program_file has been successful, select the desired board
         (void) select_rtc( InternalNumberOfDesiredBoard );
      }
   }
}
```

6.9 Miscellaneous

6.9.1 Free Variables

8 so-called "free" variables are available.

Users can freely assign data to them by the control command **set_free_variable** and the short list command **set_free_variable_list**.

These variable values can be:

- outputted at the McBSP interface, see also Chapter 9.1.7 "McBSP Interface", page 309
- read back by **get_free_variable** and **get_value** (see command descriptions)
- recorded by **set_trigger[*]** (see command descriptions)

The free variables let you, for example, transmit control commands over the McBSP interface to user hardware or document the operational states of the board (for example, with branches).

Notes

- You can use **set_free_variable** and **set_free_variable_list** to define any unsigned 32-bit values as variable values. However, the McBSP interface only outputs 24-bit values by **set_mcbsp_out_ptr** and 16-bit values by **set_mcbsp_out** (but **get_free_variable**, **get_value** and **set_trigger[*]** return full 32-bit values).

7 Basic Functions for Scan Head Control and Laser Control

7.1 Marking Dots, Lines and Arcs

7.1.1 Marking with Vector Commands and "Arc" Commands

As explained in Chapter 6.1 "RTC6 Software Concept Basics", page 96, positioning of the scan system axes (and thus of the laser beam) under RTC6 PCIe Board control is achieved by calling:

- Jump Commands
- Mark Commands
- Arc Commands
- Ellipse Commands

Each of these commands describes one vector or arc.(1) By using micro_vector[*] Commands, arbitrarily shaped trajectories(2) can be implemented.

Even numeric and alphabetic characters ultimately consist of the constituent lines, dots and arcs that define them, see Chapter 7.5 "Marking Dates, Times and Serial Numbers", page 219.

Vector Commands (Jump Commands, Mark Commands) require as parameters the coordinates of the *end* point of the corresponding vector(3). Each vector starts at the *current output position*, which is the end point of the preceding vector or arc.

"Arc" Commands (Arc Commands, Ellipse Commands) require parameters for the coordinates of the arc center and the arc angle(s). Circular arcs start at the current output position. The elliptical arc start at the position specified by the command parameters. A direct connection to the last output position must be explicitly ensured by the user program itself with suitable parameters. Otherwise, there is a "Hard Jump" there.

The output position after a RTC6 PCIe Board Hardware Reset is the center of the Image Field, that is, the point (0|0). Refer to Chapter 7.3 "Scan Head Control", page 172 for a description of the Image Field coordinate system.

At run-time, each vector or arc to be traced by the scan system gets divided by the RTC6 PCIe Board into Microsteps, see Chapter 7.1.2 "Microstepping", page 143(4).

Jump Commands serve to move the scan system axes to a new position while the laser is switched *off*.

In contrast, Mark Commands initiate a marking motion while the laser is switched *on* (see also the following description).(5)

To mark a point, Signals for "Laser Active" Operation must be switched on for the desired time period after a Jump Command or [*]mark[*] Command, see Chapter 7.1.3 "Marking Single Dots", page 144.

For line and arc marking, the RTC6 PCIe Board automatically switches Signals for "Laser Active" Operation on at the beginning of a Mark Command and later switches it back off (for example, at the beginning of a subsequent Jump Command).

The synchronization of scan head control and laser control can be adjusted by the user to the respective application by setting delays, see Chapter 7.2 "Delay Settings – Coordinating Scan Head Control and Laser Control", page 148.

Adjustment of laser parameters is described in Chapter 7.4 "Laser Control", page 189.

A thoroughly-commented example code for a basic marking task is shown in Chapter 7.1.4 "Example Code (C)", page 145.

(1) Here, wider line widths can be specified by **set_wobbel_mode**.

(2) See Glossary entry on page 31.

(3) 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]".

(4) Only iDRIVE Scan Systems which are equipped with an appropriate tuning can execute jumps also in Jump Mode, see Chapter 8.1.5 "Jump Mode", page 226.

(5) Outside a list, repositioning can be achieved by **goto_xy** or **goto_xyz** (even while the Laser Control Signals are on).

Jump Commands

A Jump Command (**jump_abs** or **jump_rel**(1)) causes the mirrors to move from the start point to the end point of a vector.

The Signals for "Laser Active" Operation are automatically switched off at the beginning of the vector and remain switched off during the jump, see also Chapter 7.2.1 "Laser Delays", page 148 and Chapter 7.2.2 "Scanner Delays", page 150. The jump speed is defined by **set_jump_speed** and **set_jump_speed_ctrl**.

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. See also the commands **home_position** and **home_position_xyz**.

Mark Commands

Upon execution of a [*]mark[*] Command (**mark_abs** or **mark_rel**(1)), the laser focus is moved linearly from the start point to the end point of the vector. The RTC6 PCIe Board automatically turns on the Signals for "Laser Active" Operation at the beginning of a [*]mark[*] Command, see also Section "Polylines", page 140.

The mark speed is defined by **set_mark_speed** and **set_mark_speed_ctrl**. It can be changed anywhere in a list by **set_mark_speed** or by **set_mark_speed_ctrl**, if no list is currently being processed.

Arc Commands

The Arc Commands **arc_abs** and **arc_rel** can be used for marking circular arcs(2). The parameters to be specified are coordinates of the arc center and the arc angle. The circular arc starts at the current output position, with angles counted positively and clockwise (contrary to the mathematical definition).

When an arc command is executed, the laser focus is guided along the specified arc at the specified speed. The Signals for "Laser Active" Operation are automatically switched on at the beginning of the execution of an arc command, see also Section "Polylines", page 140.

Polylines

If another [*]mark[*] Command (or "Arc" Command) follows immediately afterward ("Polyline"), the Signals for "Laser Active" Operation remain on.

Therefore, a continuous marking is possible by a direct line-up of [*]mark[*] Commands (and "Arc" Commands).

The Signals for "Laser Active" Operation are switched off at the beginning of the (normal) command that follows the last [*]mark[*] Command of a Polyline.

See also Section "`EdgeLevel`", page 155.

(1) For using abs and rel commands, see Section ""AbsCalls"", page 117.Additionally are available: timed vector commands, see Chapter 8.9 "Timed Commands", page 283, para vector commands, see Section "Vector-Defined Laser Control", page 214 and 3D vector commands, see Chapter "3D Commands", page 247.

(2) For using abs and rel commands, see Section ""AbsCalls"", page 117. Additionally, timed arc commands are available, see Chapter 8.9 "Timed Commands", page 283.

Ellipse Commands

The RTC6 command set also provides commands for marking elliptical arcs.

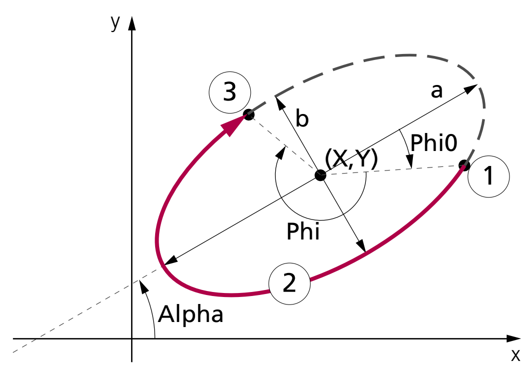
Here (unlike marking of vectors or circular arcs), you generally need to call two commands: **set_ellipse** as well as **mark_ellipse_abs** or **mark_ellipse_rel**[(1)].

By **set_ellipse** the arc shape is specified, see Figure 33:

- Lengths `a` and `b` of the ellipse half-axes
- The beginning phase angle `Phi0` (and thereby the arc starting point position relative to the end point of half-axis `a`)
- The arc angle `Phi` (and thereby the length of the to-be-marked ellipse section)

By **mark_ellipse_abs** or **mark_ellipse_rel**, position and orientation of the to-be-executed arc is specified, see Figure 33:

- The coordinates `(X,Y)` of the ellipse midpoint
- The angle `Alpha` between the ellipse half-axis `a` and the x axis



Legend
1 Starting point
2 Marked arc
3 End point

33

Marking ellipse-shaped arcs.

Notes

- By `a`, you can specify either the short or long half-axis (then use `b` for the other axis). `Phi0`, `Phi` and `Alpha` are always relative to axis `a`.
- `Phi0` and `Phi` are counted positively clockwise (in contrast to mathematical convention). In contrast, `Alpha` is counterclockwise (in accordance with mathematical convention).
- As with Mark Commands and Arc Commands, the laser focus moves with the specified mark speed along the specified arc when the Ellipse command is executed. The Signals for "Laser Active" Operation are automatically switched on at the beginning of an "Arc" Command, see also Section "Polylines", page 140.
- **set_ellipse** is a Short List Command. Therefore, it can be called between a Mark Command and an Ellipse command without thereby interrupting the Polyline (the laser remains on).
- Ellipse Commands always begin marking at the starting point determined by the above-mentioned parameters (in contrast to Mark Commands and Arc Commands which automatically begin marking at the current output position). If the starting point and current position do not match, then a Hard Jump to the starting point is executed at the beginning of marking (without a Jump Delay becoming effective).

(1) For using abs and rel commands, see Section ""AbsCalls"", page 117.

- Elliptical arcs can also be marked by circular Arc Commands (for example, **arc_abs**) if an appropriate coordinate transformation (for example, scaling that differs in the x direction/y direction) has been specified by **set_matrix**. Here, though, the effective mark speed varies along the arc, see also the note on page 236.

  This contrasts with **mark_ellipse_abs** and **mark_ellipse_rel**, where in 10 μs intervals the step length gets adjusted for the ellipse's shape at the current position such that the arc is marked with a (largely) constant mark speed.

  For very large eccentricities and also at high mark speeds, however, such stepwise ellipse approximation by a 10 μs clock can produce numerical inaccuracies in the end point regions of the large half-axis. Consequently, the effective mark speed there might not be precisely constant (for example, an eccentricity of $a/b = 2$ and 100 Microsteps per circumference would produce a speed deviation of approx. 3.7%). However, the outputted point always lies exactly on the ellipse.

  Moreover, as closed equations do not exist for calculating an ellipse arc length, the step length of the finally-marked Microstep is generally shorter and the mark speed correspondingly lower than specified. Nevertheless, the end position is always exact.

  Likewise, Sky Writing might produce run-in/run-out irregularities at the large half-axis. Users themselves must ensure that the parameter values used are consistent with the required precision.

[*]Para[*] Commands

- **para_jump_abs**
- **para_jump_abs_3d**
- **para_jump_rel**
- **para_jump_rel_3d**
- **para_laser_on_pulses_list** (special case)
- **para_mark_abs**
- **para_mark_abs_3d**
- **para_mark_rel**
- **para_mark_rel_3d**
- **timed_para_jump_abs**
- **timed_para_jump_abs_3d**
- **timed_para_jump_rel**
- **timed_para_jump_rel_3d**
- **timed_para_mark_abs**
- **timed_para_mark_abs_3d**
- **timed_para_mark_rel**
- **timed_para_mark_rel_3d**

If the Vector-Defined Laser Control is activated, these commands simultaneously vary a signal parameter linearly along the mark vector or jump vector, see Section "Vector-Defined Laser Control", page 214.

[*]para_mark[*] commands generally do not take Sky Writing into account.

7.1.2 Microstepping

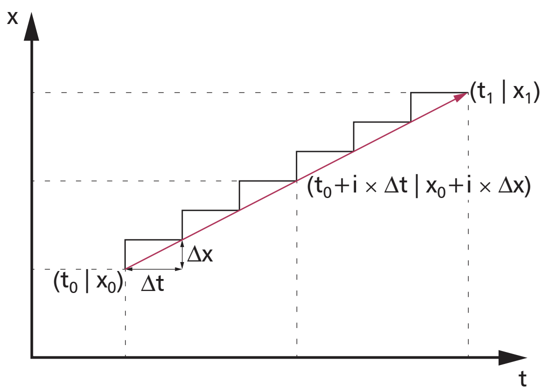
The RTC6 PCIe Board splits up each

- Jump Command,
- [*]mark[*] Command and
- "Arc" Command

into so-called

- Microsteps
  (not: Microvectors, see Chapter 8.8 "micro_vector[*] Commands", page 282).

The split-up of the x component of a vector into Microsteps is shown in Figure 34.



x_0 x coordinate of the current output position before scanning the vector.
x_1 x coordinate of the vector end position.
Δt Output interval 10 μs.

34

The x component of a vector is split-up into Microsteps. The y component is split-up in the same way.

All Microsteps are transferred to the scan head with a constant output interval (Δt) of 10 μs and *cannot* be changed.

The following applies:

Length Δs of a Microstep $= v \times \Delta t$ (1)
(v = current jump speed or mark speed)

Notes

- Custom curves can be implemented by using micro_vector[*] Commands, see Chapter 8.8 "micro_vector[*] Commands", page 282.
- Only iDRIVE Scan Systems which are equipped with an appropriate tuning can execute jumps also in Jump Mode, see Chapter 8.1.5 "Jump Mode", page 226.

(1) Alternatively see Chapter 8.9 "Timed Commands", page 283.

7.1.3 Marking Single Dots

To mark a single point, the Signals for "Laser Active" Operation must be switched on for the desired time period, see **laser_on_list**, **laser_on_pulses_list**, **para_laser_on_pulses_list** and Chapter 7.4 "Laser Control", page 189.

Alternatively, a single dot can also be marked by a timed Mark Command of length zero, see Chapter 8.9 "Timed Commands", page 283.

7.1.4 Example Code (C)

The following example C source code shows the commands of a simple laser scan application.

A point, a square and a circle are marked in CO_2 Mode. Here it is assumed that the RTC4 Compatibility Mode is activated.

The code must be included in a user program, see Chapter 6.2.5 "Example Code (C)", page 103.

```
// Scan system initialization
   // Loading and assigning a correction file
   ErrorCode = load_correction_file( 0,   // initialize like "D2_1to1.ct5",
                                     1,   // table (#1 is used by default)
                                     2 ); // use 2D only
   if ( ErrorCode )
   {
      printf( "Correction file loading error: %d\n", ErrorCode );
      free_rtc6_dll();
      return;
   }
   select_cor_table( 1, 0 ); // assigning correction table #1 to scan head connector #1 (default)

// Laser control initialization, see Chapter 7.4 "Laser Control", page 189.
   // Setting the CO2 Mode
   set_laser_mode( 0 );

   // Setting and enabling the Signals for "Laser Active" Operation
   set_laser_control( 0x18 ); // All Laser Control Signals active-LOW (Bit #3 and #4)
      // set_laser_control must be called at least once to activate Laser Control Signals.
      // Later on enable_laser/disable_laser would be sufficient.

   // Opens List 1
   set_start_list( 1 );

   // Setting the standby pulses
   set_standby_list( 800, 8 );
      // In RTC4 Compatibility Mode the standby parameters are specified in units of 1/8 µs.
      // The RTC6 board multiplies the specified values by 8 to convert them to
      // integer-multiple of 1/64 µs. Half of the standby output period = 100 µs.
      // Pulse length of the standby pulses = 1 µs.

// Timing, delay and speed preset
   // Setting the Scanner Delays (see Chapter 7.2.2 "Scanner Delays", page 150).
   set_scanner_delays( 25, 10, 5 );
      // Jump Delay = 250 µs (specified in [10 µs])
      // Mark Delay = 100 µs (specified in [10 µs])
      // Polygon Delay = 50 µs (specified in [10 µs])
```

```
    // Setting the jump speed and mark speed:
    set_jump_speed( 1000.0 );
    set_mark_speed( 250.0 );
       // In RTC4 Compatibility Mode the RTC6 board multiplies the speed values by 16.
       // Jump speed = 1000.0 bits/ms.
       // Marking speed = 250.0 bits/ms.

    // Setting the laser timing, see Chapter 7.4 "Laser Control", page 189.
    set_laser_pulses( 800, 400 );
       // In RTC4 Compatibility Mode the timing parameters are specified in units of 1/8 µs.
       // The RTC6 board multiplies the specified values by 8 to convert them to
       // integer-multiple of 1/64 µs. Laser HalfPeriod = 100 µs.
       // Laser pulse length = 50 µs.

    // Setting the Laser Delays, see Chapter 7.2.1 "Laser Delays", page 148.
    set_laser_delays( 100, 100 );
       // In RTC4 Compatibility Mode the Laser Delays are specified in units of 1 µs.
       // The RTC6 board multiplies the specified values by 32 to convert them to
       // integer-multiple of 1/64 µs.
       // LaserOn Delay = 100 µs.
       // LaserOff Delay = 100 µs.

    // Defining the end of the list and the end of command transfer to the RTC6 board
    set_end_of_list();

    // Execute the list commands for initialization
    execute_list( 1 );

 // Marking procedure
    // Waiting for list 1 to be not busy. (load_list( 1, 0 ) returns 1 if successful, otherwise 0);
    // if list 1 is not (no longer) busy:
    // opening the RTC6 List Memory for writing of list commands and setting the input pointer
    // to the start of list 1
    while ( !load_list( 1, 0 ) );

    // In the following the list commands for marking point, square and circle are defined and transferred
    // to the RTC6 board.

    // Marking the center point of the Image Field:
    jump_abs( 0, 0 ); // Jump to center point
       // A Jump Delay is automatically inserted after the jump.
    // Turning on the Laser Control Signals for 50 µs (+ LaserOff Delay - LaserOn Delay):
    laser_on_list( 5 );
```

```
// Marking a square around the center point:
jump_abs( -20000, -20000 ); // Jump to the bottom left corner of the square
   // A Jump Delay is automatically inserted after the jump.
mark_abs( -20000,  20000 ); // Marking the left edge of the square
mark_abs(  20000,  20000 ); // Marking the top edge of the square
mark_abs(  20000, -20000 ); // Marking the right edge of the square
mark_abs( -20000, -20000 ); // Marking the bottom edge of the square
   // The Laser Control Signals are automatically switched on
   // with the first [*]mark[*] Command after a LaserOn Delay and remain on for
   // all 4 [*]mark[*] Commands.
   // A Polygon Delay is automatically inserted after the first three [*]mark[*] Commands, each.
   // Initiated by the following non-marking command (Jump Command, see below), a Mark Delay
   // is automatically inserted after the last [*]mark[*] Command and the Laser Control Signals
   // are automatically switched off after a LaserOff Delay, because a Jump Command follows.

// Marking a circle around the center point:
jump_abs( 0, -10000 ); // Jump to the bottom edge of the circle
   // A Jump Delay is automatically inserted after the jump.
arc_abs( 0, 0, 360.0 ); // Marking the circle
   // The Laser Control Signals are automatically switched on with the arc command
   // after a LaserOn Delay.
   // Initiated by the following non-marking command (set_end_of_list, see below),
   // a Mark Delay is automatically inserted after the arc command and the Laser Control Signals
   // are automatically switched off after a LaserOff Delay, because a set_end_of_list follows.

// Defining the end of the list and the end of command transfer to the RTC6 board
set_end_of_list();

// Starting the transferred list
execute_list( 1 );
```

7.2 Delay Settings – Coordinating Scan Head Control and Laser Control

Scan head control and laser control should suit the dynamic behavior of the system components, that is, the

- response behavior of the laser
- response behavior of the galvanometer scanners (Tracking Error)
- the type of interaction between laser radiation and material

The following delays are available for this purpose:

- Laser Delays
  - LaserOn Delay
  - LaserOff Delay
- Scanner delays
  - Jump Delay (optionally: variable)
  - Mark Delay
  - Polygon Delay (optionally: variable)

7.2.1 Laser Delays

There are two different Laser Delays:

- LaserOn Delay
- LaserOff Delay

Laser Delays determine when the Signals for "Laser Active" Operation are switched on and off.

As a rule, Laser Delays have *no* influence on the total marking time. Exceptions:

- A negative `LaserOnDelay` value, see Section "LaserOn Delay", page 149
- Artificially inserted delays, see Section "Automatic Delay Adjustments", page 158

The LaserOn Delay and the LaserOff Delay are set by the undelayed short list command **set_laser_delays**. Their unit is *1/64 µs* each.

In order to avoid burn-in effects at start and end points of a marking, the laser focus should be moved at a speed which is as constant as possible. Therefore, the laser delay durations must be adjusted to the Tracking Error of the scan head and the set mark speed[1], see also Chapter 7.2.3 "Notes on Optimizing the Delays", page 158.

(1) In addition, automatic readjustment of the laser power during marking can be applied for optimization, see Chapter 7.4.9 ""Automatic Laser Control"", page 205.

LaserOn Delay

The LaserOn Delay is automatically inserted at the start of a single Mark Command and at the start of a series of Mark Command ("Polyline") and delays the switching on of the laser.

It can be used for several purposes:

- At the beginning of a marking, the mirrors must be accelerated to the specified mark speed (if necessary, from a halt), see Figure 37. This phase can be suppressed with a sufficiently high positive LaserOn Delay value until the mirrors have already reached a certain angular speed before the laser is switched on. On the other hand, the LaserOn Delay must not be too long, otherwise the first part of the marking is cut off.
- Some materials/applications (for example, welding) take some time until they react as desired to the exposure to laser radiation. In this case, it can be useful to "preheat" the starting point of the marking. This can be achieved by setting a *negative* LaserOn Delay value. However, this extends the total marking time because the LaserOn Delay is inserted prior to the current Mark Command as Scanner Delay.[1]

LaserOff Delay

The end of a marking is not defined by the marking itself, but by the first "normal" list command that is not a Mark Command, for example, a Jump Command.

The acceleration phase at the beginning of a motion leads to a time difference between the respective set position and the actual position of the mirrors, see Figure 37.

The laser is to be switched off until the *actual value* of the end position is reached (but not already at the *set value*). Therefore, a LaserOff Delay is inserted[2] automatically after the end of each marking by the first Normal List Command (no Short List Command), see also Section "Notes", page 152. This can be used to compensate for the Tracking Error of the scan head.

(1) This Scanner Delay is automatically extended, if a preceding LaserOff Delay has not yet been expired, see Section "Automatic Delay Adjustments", page 158.

(2) In DSP mode < 3 (see **set_dsp_mode**), the following applies for short marking vectors: if a preceding LaserOn Delay has not yet been expired, the LaserOff Delay is temporarily automatically extended accordingly, see Section "Automatic Delay Adjustments", page 158.

7.2.2 Scanner Delays

There are three different types of Scanner Delays:

- Jump Delays (optionally: variable)
- Mark Delays
- Polygon Delays (optionally: variable)

After each Jump Command and Mark Command, the RTC6 PCIe Board inserts one of these Scanner Delays before the next command is executed (unless otherwise specified).

These Scanner Delays are defined by **set_scanner_delays**. Their unit is *10 µs* each.

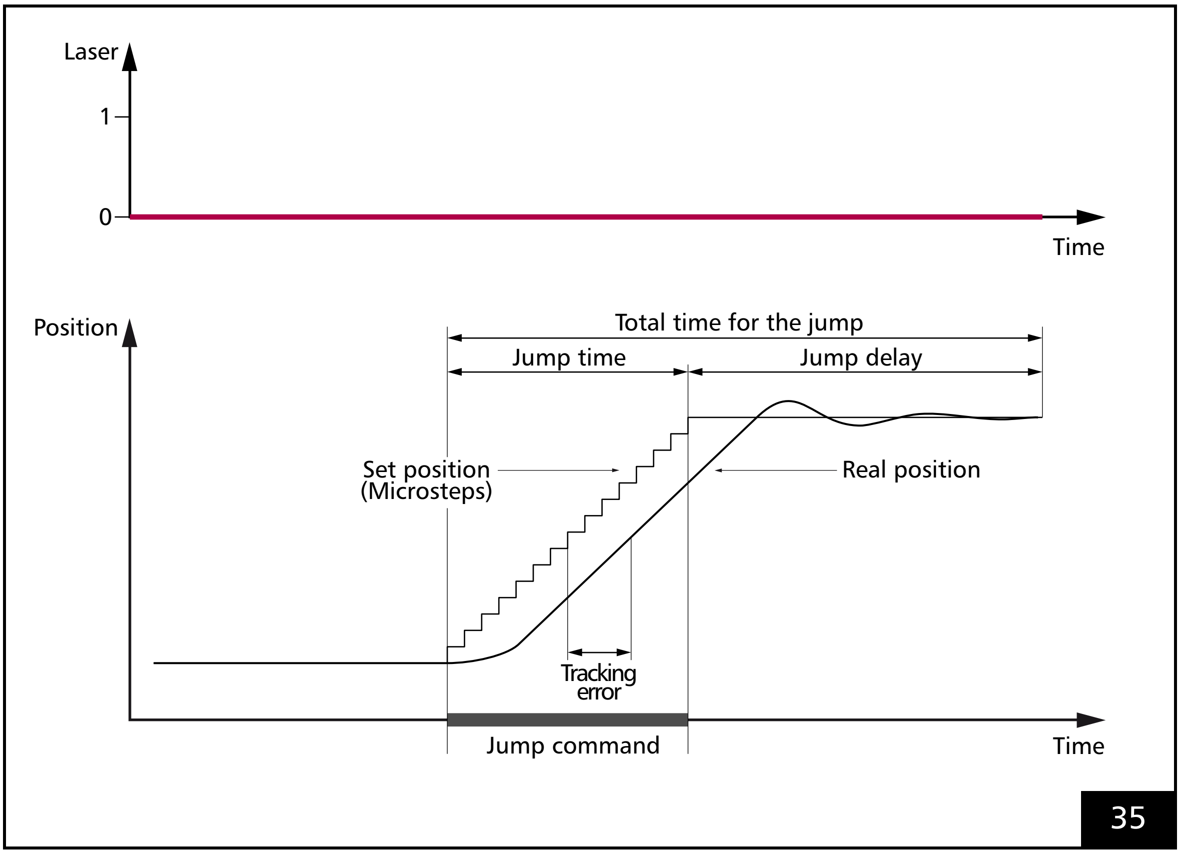
Jump Delay

The Jump Delay is specified by **set_scanner_delays** with the `Jump` parameter.

A typical course for a single Jump Command and the belonging Jump Delay is shown in Figure 35.

The total time for a jump is made up of the jump time and the duration of the Jump Delay.

The duration of the Jump Delay should suit the dynamic properties of the scan head (Tracking Error, tuning) and the jump speed set.



Scan head control during a Jump Command with a constant Jump Delay.
The laser remains off.

Variable Jump Delays

With short jumps, the scan head often does not reach the full jump speed. Then *shorter* Jump Delays are sufficient for settling of the mirrors:

- "Variable Jump Delays"

For this, there is:

- "Variable Jump Delays" mode

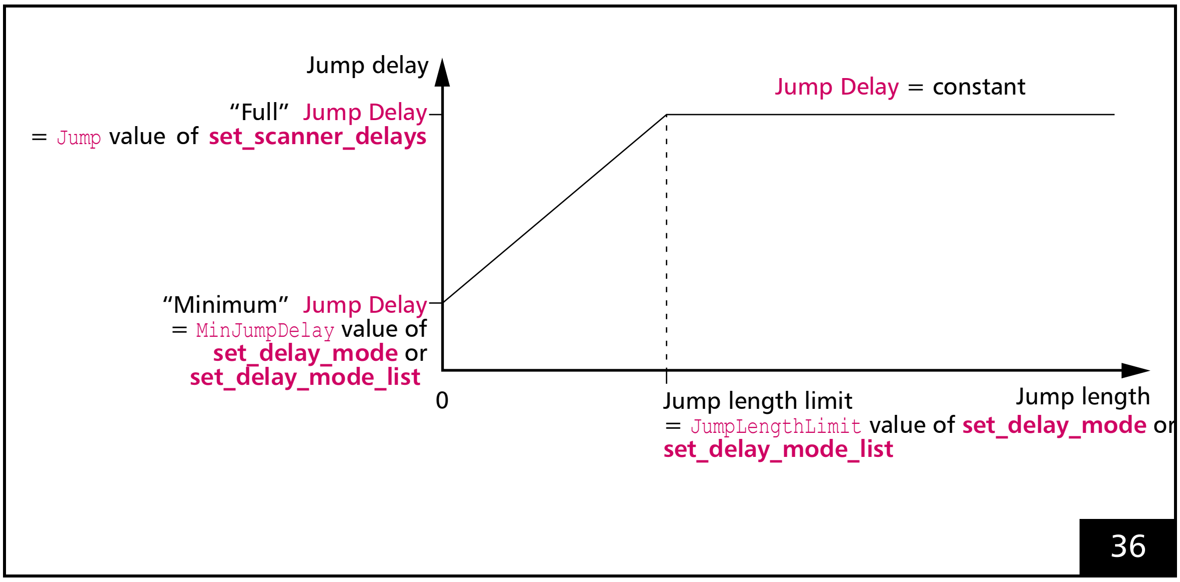
"Variable Jump Delays" mode is switched on by `JumpLengthLimit` > 0 (at **set_delay_mode** and **set_delay_mode_list**).

Then the RTC6 PCIe Board inserts a linearly interpolated Jump Delay between `MinJumpDelay` (from **set_delay_mode** or **set_delay_mode_list**) and `Jump` (Jump Delay from **set_scanner_delays**) for jump lengths between 0 and Jump length limit, see Figure 36.

With jump lengths larger than `JumpLengthLimit` a "full" Jump Delay is always inserted.

Notes

- With "Variable Jump Delays" mode, total marking time is reduced, especially when there are many short jumps.
- "Variable Jump Delays" mode is switched off by `JumpLengthLimit` = 0 (at **set_delay_mode** and **set_delay_mode_list**).
- After jump vectors of length 0, the duration of "Variable Jump Delays" is 0.
- The "minimum" Jump Delay should not be larger than the "normal" Jump Delay. Otherwise, "Variable Jump Delays" can become very large.



"Variable Jump Delays" mode: Jump Delay value depending on the jump length.

Mark Delay

The Mark Delay is specified by **set_scanner_delays** with the `Mark` parameter.

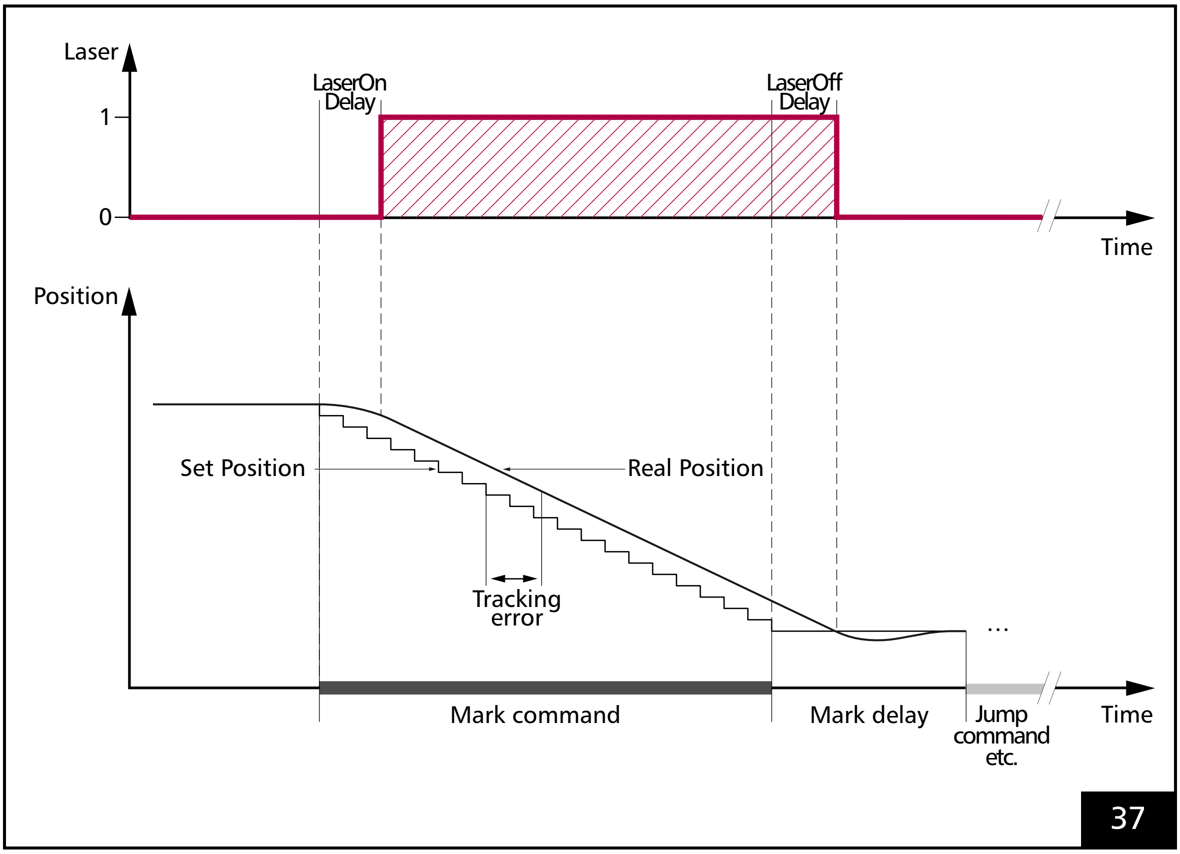
A typical course for a single Mark Command with the and the belonging Laser Delays is shown in Figure 37.

The duration of the Mark Delay should suit the dynamic properties of the scan head (Tracking Error, tuning) and the mark speed set.(1)

(1) About the interaction of Laser Delays and Scanner Delays in DSP mode < 3 (see **set_dsp_mode**), see Section "Automatic Delay Adjustments", page 158.

Notes

- If no further Mark Command follows a Mark Command, a Mark Delay is inserted automatically and the laser is switched off after a LaserOff Delay.
- If a further Mark Command follows a Mark Command (= "Polyline"), a (variable) Polygon Delay is inserted and the laser remains switched on, see Section "Variable Polygon Delays", page 154.
- Short list commands do not lead to insertion of a Mark Delay or Polygon Delay and not to a laser switch off.
- Note that a Mark Command of length 0 corresponds to a **list_continue**, if it is not timed, see Chapter 8.9 "Timed Commands", page 283.



Scan head control and laser control timing during a Mark Command or "Arc" Command with a Mark Delay.
The laser is on in the hachured period.

Polygon Delay

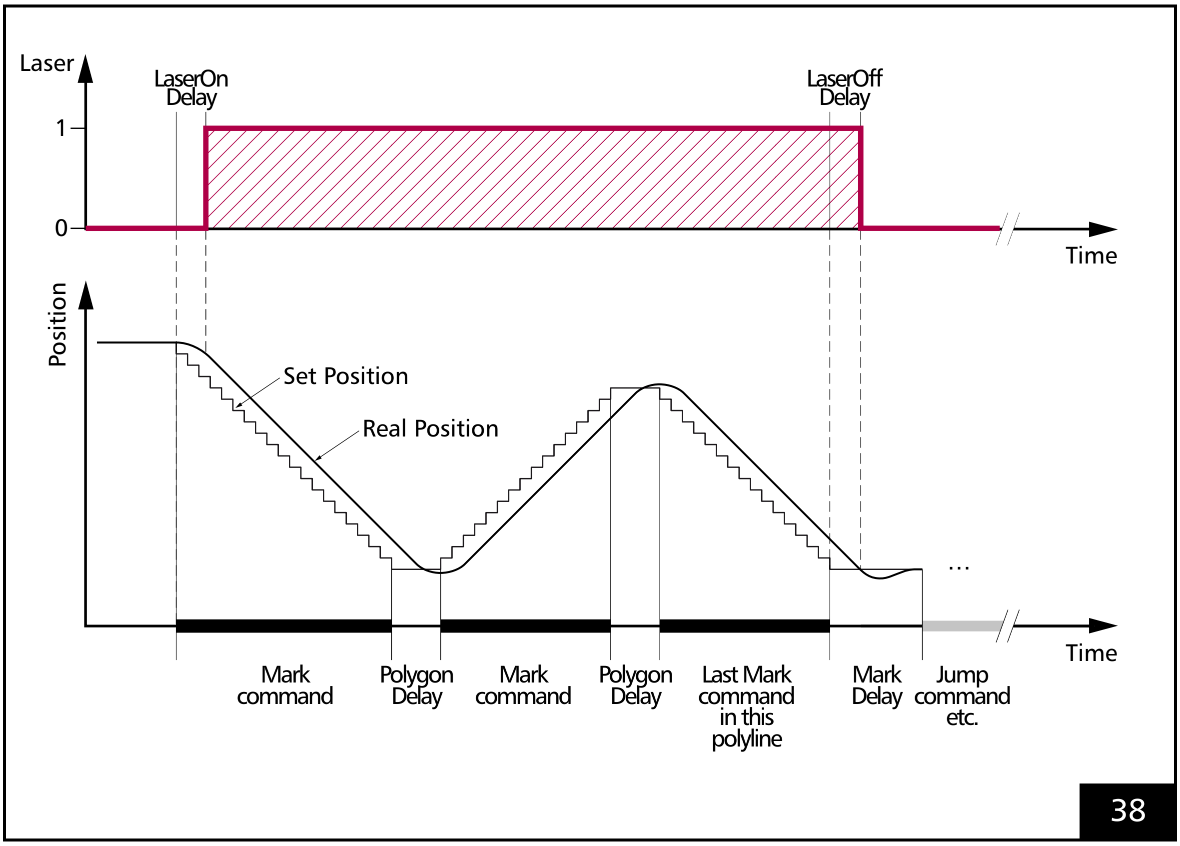
The Polygon Delay is specified by **set_scanner_delays** with the `Polygon` parameter.

A typical course for a series of Mark Commands ("Polyline") where Polygon Delays (instead of Mark Delays) are inserted between the individual Mark Command is shown in Figure 38.

Short list commands do not interrupt a Polyline.

If the (usually smaller) angles between the markings vary only slightly usually a Polygon Delay value can be specified that is smaller than the Mark Delay value.

If, on the other hand, the angles vary significantly, then a "Variable Polygon Delay" is recommended, see Section "Variable Polygon Delays", page 154.



Scan head control and laser control timing during a Polyline with a constant Polygon Delay.

Variable Polygon Delays

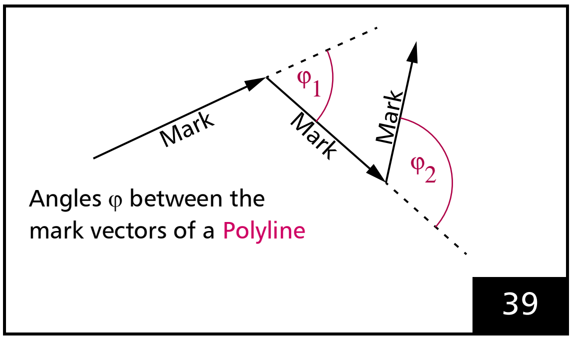
By setting the `VarPoly` value > 0 (at **set_delay_mode** or **set_delay_mode_list**) it is enabled:

- "Variable Polygon Delays" mode

Then, the RTC6 PCIe Board calculates the "Variable Polygon Delay" *v_delay*(φ) for every Polyline corner according to:

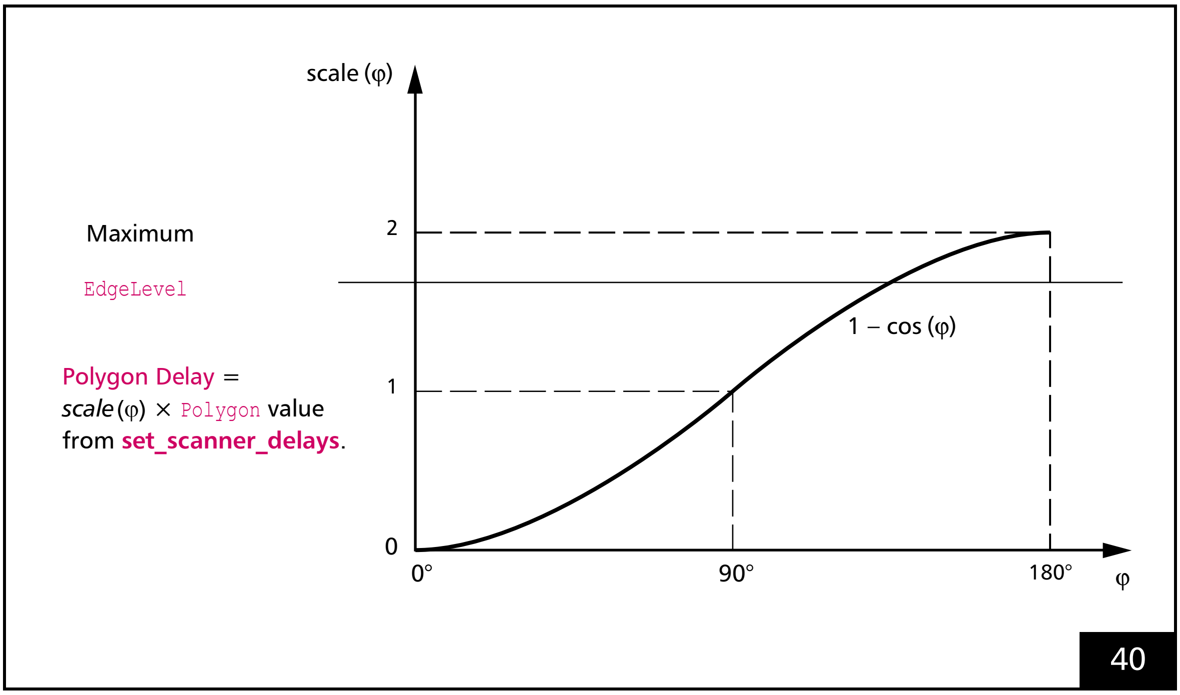
$$v\_delay(\varphi) = scale(\varphi) \times polygon\_delay$$

For the definition of the angle φ using the example of mark vectors, see Figure 39.



Variable Polygon Delays. Definition of the angle φ.

For circular arcs and ellipses, analogously the tangents at the connection point are relevant. *scale*(φ) is a scaling function for the `Polygon` value from **set_scanner_delays** (constraint $0 \le scale(\varphi) \le 2$), see Figure 40. The default scaling function after **load_program_file** is $scale(\varphi) = 1 - \cos(\varphi)$. It can be replaced by a user-defined scaling function, see Section "User-defined "Variable Polygon Delays"", page 156.



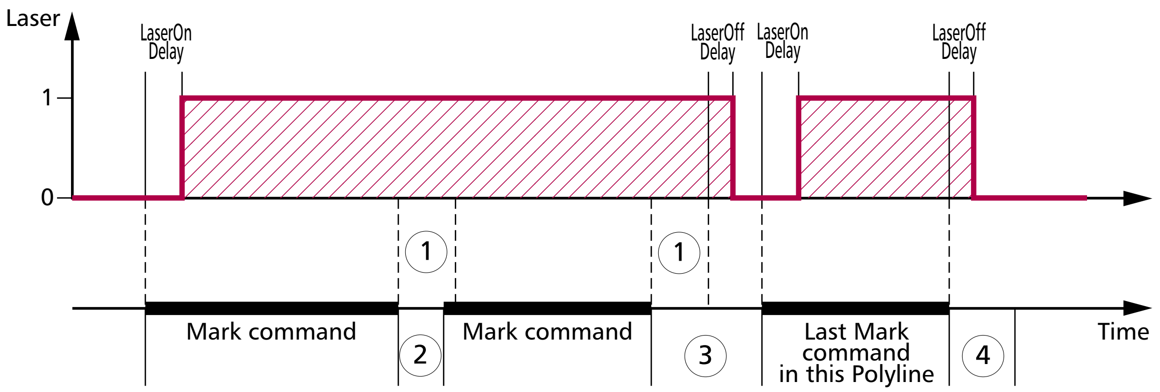
Variable Polygon Delays. Variation of the Polygon Delay (default scaling function after **load_program_file**).

`EdgeLevel`

The “Variable Polygon Delay” becomes quite long with angles φ close to 180°, see Figure 40. This might lead to burn-in effects in sharp corners of a Polyline.

If an `EdgeLevel` value > 0 is specified (at **set_delay_mode**), the RTC6 PCIe Board switches off the laser with a LaserOff Delay after `EdgeLevel` delay clock cycles at the latest and thus terminates the current Polyline, see Figure 41.

The next Mark Command starts as usual with the current “Variable Polygon Delay”.



Legend

1 `EdgeLevel`.
2 “Variable Mark Delay”, shorter than the specified `EdgeLevel` value.
3 “Variable Mark Delay”, longer than the specified `EdgeLevel` value. The laser is switched off between both Mark Commands.
4 Mark Delay.

41

Laser control timing during a Polyline with “Variable Polygon Delay”. An `EdgeLevel` value has been defined with **set_delay_mode**.

User-defined "Variable Polygon Delays"

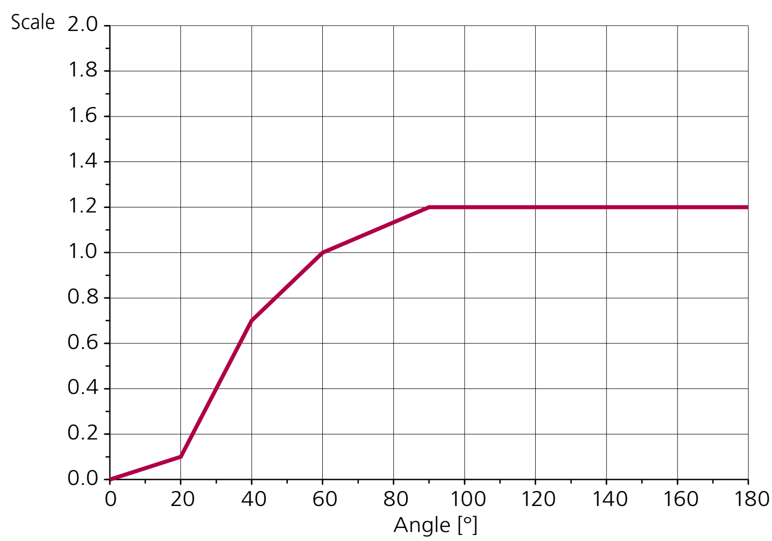
- For the *scale*(φ) scaling function, **load_varpolydelay** loads a table from an ASCII text file. See also Section "Variable Polygon Delays", page 154.
- The ASCII text file can contain one or more tables(1).
- Each table can contain up to 50 data points (φ | *scale*(φ)).
- The *scale*(φ) function is linearly interpolated from the data points.
- As an example, Figure 42 shows a table with 4 data points and the corresponding scaling function.

(1) Even of another type, see table 1, page 156.

Table 1: Possible table types in the ASCII text file.
Each type can occur more than once.

[VarPolyTable<No>]	User-defined "Variable Polygon Delays", see page 156
[PositionCtrlTable<No>]	Scaling function, see page 207
[AutoLaserCtrlTable<No>]	Nonlinearity curve, see page 212
[JumpTable<No>]	Jump Delay values, see page 228
[StretchTable<No>]	2D stretch correction table, see page 249
[Fly2DTable<No>]	2D compensation table, see page 259

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



42

Example: Table for User-defined "Variable Polygon Delays" with 4 data points (left) and corresponding scaling function *scale*(φ) (right).

For the tables, the following rules apply:

- Each table must begin with the line:
  `[VarPolyTable<No>]`
  `<No>` represents the table number.
- If the table contains multiple `[VarPolyTable<No>]` entries with the same `<No>`, then only the lines after the first entry are used. Only lines up to the next '[' character (that is not preceded by a semicolon) are used.
- Each data point (φ | *scale* (φ)) is defined as follows:
  `Angle<n> = <Value>`
  `Scale<n> = <Value>`
  where `<n>` must be replaced by a number (1 ≤ `<n>` ≤ 50) which denotes the number of the data point. The values `<Value>` for the angle φ (in degrees) and the scaling factor can be specified as (unsigned) floating point numbers. Decimal separator: decimal point (.).
- If the table contains multiple data points with the same Index `<n>`, then the most recently read one is used and the previous ones are ignored.
- If the table contains multiple data points with the same angle φ, then the data point with the largest Index `<n>` is used and the others ignored. Equality is checked to within ±0.01°.
- For `<Value>`, the following ranges apply: 0.0° ≤ φ ≤ 180.0° and 0.0 ≤ *scale* (φ) ≤ 2.0.
- Each instruction must be in a separate line.
- Spaces and tabs in a line (for example, between '=' and `<Value>`) are ignored.
- Empty lines are ignored.
- Data points with invalid values are ignored.
- The data point of a particular index `<n>` is ignored if the corresponding `Angle<n>` and/or `Scale<n>` definition is missing.
- The semicolon ';' can be used for comments. All characters in a line following a semicolon are ignored.
- The instructions for data points in the table can be ordered as desired.
- Indices for data point pairs in the table can be selected as desired within the range [1…50] (the table is then automatically sorted by ascending angles).
- If the table contains *no* valid data point, then **load_varpolydelay** has no effect (return value 1 or 13).
- The angle φ = 0° means that two successive vectors are parallel and are marked in the same direction.
  If the table contains no explicit data for φ = 0° (equality is checked to within ±0.01°), then a data point for φ = 0° with the scaling factor *scale* (0°) = 0 is added.
- The angle φ = 180° means that two successive vectors are marked in opposite directions.
  If the table contains no explicit data for φ = 180° (equality is checked to within ±0.01°), then a data point for φ = 180° with the largest scaling factor found in the table for *scale* (180°) is added.

After initialization by **load_program_file**, the RTC6 PCIe Board uses the internal (default) table for the “Variable Polygon Delay” (1–cos(φ), see Figure 40). Alternatively, this can also be achieved with `Name` = NULL in **load_varpolydelay**.

The table can be saved by **create_dat_file**.

7.2.3 Notes on Optimizing the Delays

The delays are set by **set_scanner_delays** and **set_laser_delays**.

They should be appropriate for:

- the set jump speed
- the set mark speed
- the Tracking Error of the used scan head
- the `ControlPreview` time of the used Preprocessing System, if **set_controlpreview_compensation_ctrl** is not called

Non-optimized delays may lead to marking results of insufficient quality and excessive scan times, see also Section "Potential Errors when Optimizing the Delays", page 160.

Notes

- Laser Delays are specified in units of 1/64 µs.
- Scanner Delays (Jump Delay, Mark Delay, Polygon Delay) are specified in units of 10 µs.

Recommended Optimization Sequence

(1) Laser Delays.

It is recommended to set Jump Delays and Mark Delays to a high value. The laser delay lengths have no influence on the total scan time as long as positive values are specified (see also following section).

(2) Scanner Delays.

Automatic Delay Adjustments

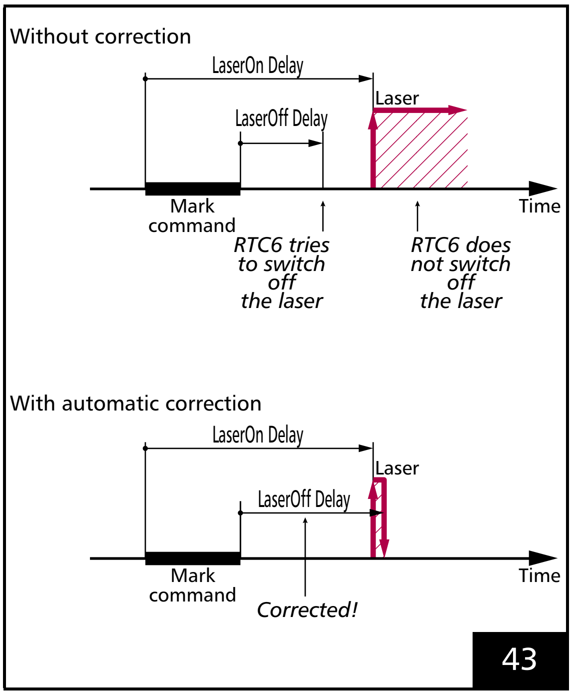
In principle, all delays can be set for any value within the corresponding allowed range. In some situations the laser control could be disturbed, for example, when Laser-On and Laser-Off overlap. Therefore, for each command that switches the laser, the RTC6 PCIe Board checks the set Laser Delays, corrects them and, if necessary, inserts a Scanner Delay "artificially". These situations, which are listed below, should actually already be avoided in the user program, because the "corrected" marking does not necessarily meet the requirements.

Laser-On and Laser-Off Overlap

The RTC6 PCIe Board corrects the faulty laser delay so that:

- the Laser-On occurs 1/64 µs after the currently effective LaserOff Delay
- or the Laser-Off occurs 1/64 µs after the still running LaserOn Delay

In both cases, the chronological next mark is cut off somewhat, see also Figure 43.



Automatic adjustment of LaserOff Delay.

Notes

- For further information on the general avoidance of such automatic adjustments in RTC5 Compatibility Mode, see RTC5 Manual, Chapter 7.2.3 "Notes on Optimizing the Delays".

Laser-On and Laser-On Overlap

If the LaserOn Delay is switched from a long delay to a short delay between two markings, the short LaserOn Delay could switch on the laser before the long LaserOn Delay of the previous marking has expired, if the laser has been switched off between the markers (otherwise the laser would stay on and the new LaserOn Delay would not be effective at this point).

To avoid this overlap, the RTC6 PCIe Board in this case (like the RTC5) inserts a Scanner Delay “artificially”. This increases the processing time and changes the dynamics of the galvanometer scanner motion.

Laser-Off and Laser-Off Overlap

If the LaserOff Delay is switched from a long delay to a short delay, the short LaserOff Delay could switch off the laser before the long LaserOff Delay of the previous marking has expired, if a marking had switched on the laser in the meantime (otherwise the laser would stay off and the new LaserOff Delay would not be effective at this point)

To avoid this overlap, the RTC6 PCIe Board in this case (like the RTC5) extends the short LaserOff Delay so long that it only expires after the previous one (and of course only after the intermediate LaserOn Delay).

Several Simultaneously Expiring Laser Delays

This can happen if a laser delay is only effective beyond the length of the next command. The RTC6 PCIe Board can process up to 256 LaserOn Delays and LaserOff Delays simultaneously in a pipeline, as long as Laser-On and Laser-Off alternate and Laser-On—Laser-On and Laser-Off—Laser-Off run one after the other (see automatic adjustments above). This allows markings (interrupted lines) to be “pre-programmed” up to at least 2.56 ms.

Notes

- In contrast to the RTC6 PCIe Board, the RTC5 can only process one delay of the same type at a time. The delays are adjusted automatically. If necessary, Scanner Delay are inserted “artificially”. In DSP mode < 3 (see **set_dsp_mode**), the RTC6 PCIe Board simulates the behavior of the RTC5 in order to display the same time sequences. For further details, see RTC5 Manual, Chapter 7.2.3.

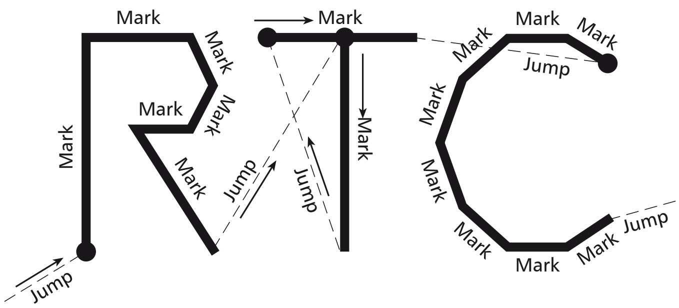
Potential Errors when Optimizing the Delays

The following figures show the various effects of unsuitable delays on the marking result, in this example on the lettering "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.

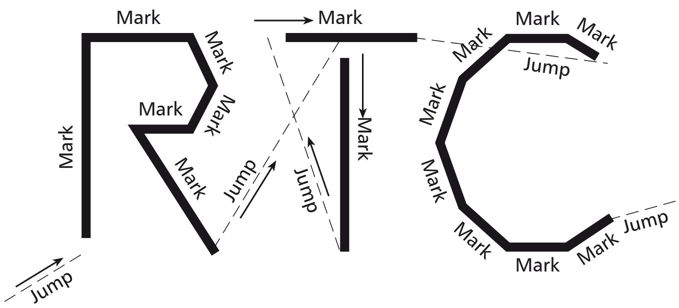
Burn-in effects at the start points of the respective vectors result.



LaserOn Delay too long

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

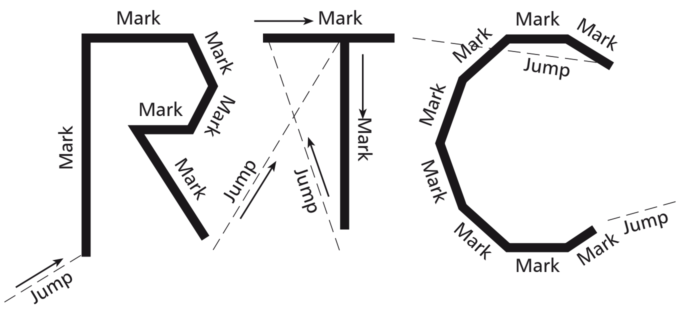
The first part of the vector is not marked.



LaserOff Delay too short

The laser is switched off after a mark command before the mirrors have reached the vector end position.

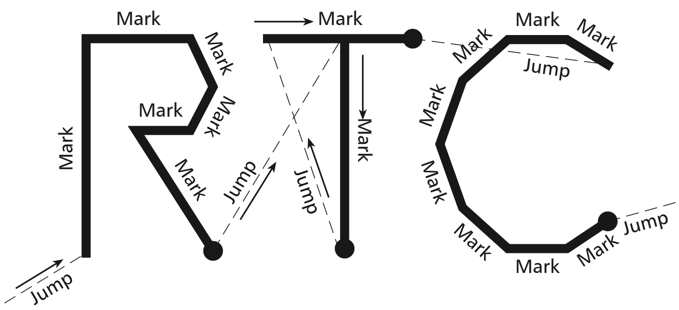
The respective vectors are not marked completely.



LaserOff Delay too long

The laser is switched off too late after a Mark command. The laser is still on, although the mirrors are already slowed down considerably or have already stopped.

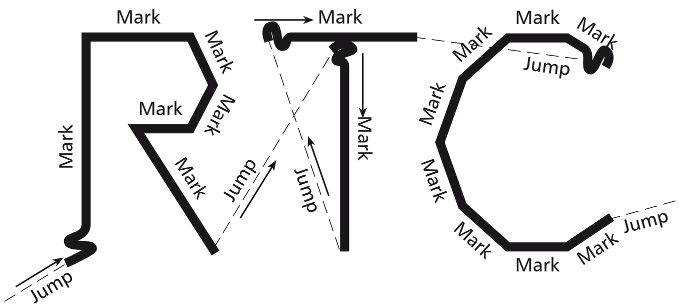
The results are burn-in effects at the end points of the respective vectors.



Jump Delay too short

The mark vector that follows a Jump Command has already started although the scanners have not yet settled.

A running-in oscillation or overshoot is visible.



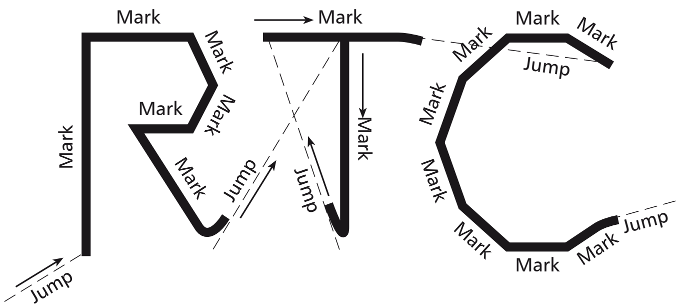
Jump Delay too long

There are no visible effects if the Jump Delay is too long. However, the scanning time is extended.

Mark Delay too short

The traversing of a vector is started before the mirrors have reached the end position of the preceding mark vector.

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



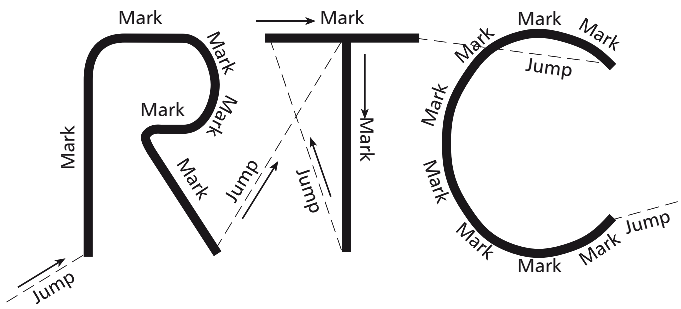
Mark Delay too long

There are no visible effects if the Mark Delay is too long, but the scanning time is 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.

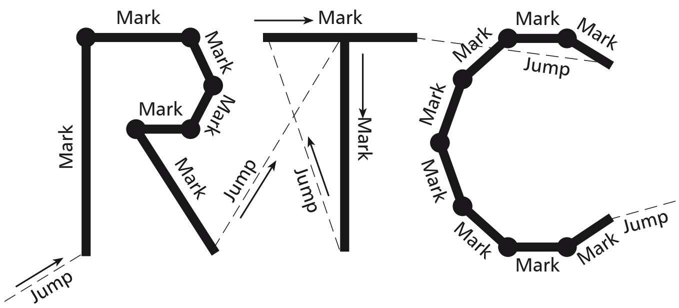
The corners of the Polyline are rounded off.



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 "Variable Polygon Delays" mode, a maximum length ("`EdgeLevel`") can be defined, see Section "`EdgeLevel`", page 155 for details.

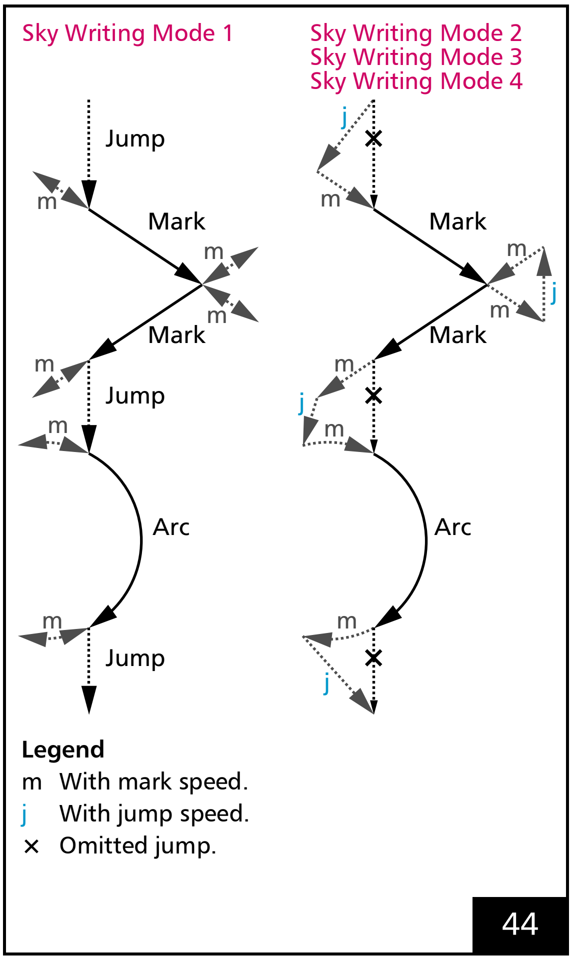
7.2.4 Sky Writing

For applications with elevated accuracy requirements the so-called Sky Writing can be switched on. Then, every mark vector is precisely executed at a constant mark speed over the entire vector length.

For Mark Commands with Sky Writing enabled, the RTC6 PCIe Board automatically performs non-marking Sky Writing motions before and after the to-be-executed vectors and arcs, see Figure 44.

The following are always executed without Sky Writing:

- micro_vector[*] Commands
- [*]para[*] Commands



Sky Writing motions (grey). For Sky Writing Mode 3 and Sky Writing Mode 4 see also Figure 45.

By the Sky Writing command parameters you can specify:

- the duration of these run-in motions
- the duration of these run-out motions
- the synchronization between scan motion and laser control

Sky Writing is available as:

- Sky Writing Mode 1, page 163
- Sky Writing Mode 2, page 164
- Sky Writing Mode 3, page 165
- Sky Writing Mode 4, page 166

Sky Writing Mode 1

In Sky Writing Mode 1, the RTC6 PCIe Board performs the following Sky Writing motions for each Mark Command – regardless of previous or subsequent commands:

- In the run-in phase, the vector/arc is preceded by a forerun motion performed by the galvanometer scanners at mark speed, see Figure 44: the galvanometer scanners are driven a short distance parallel to the vector (or along the arc extension), initially from the startpoint in the opposing direction, then back to the startpoint.
- After the vector/arc has been processed at mark speed, it is gets a short deceleration and retrace motion of the galvanometer scanners (at mark speed) appended in the run-out phase.

Sky Writing Mode 1 can be switched on (off) by:

- **set_sky_writing**
- **set_sky_writing_list**
- **set_sky_writing_para**
- **set_sky_writing_para_list**

Sky Writing Mode 2

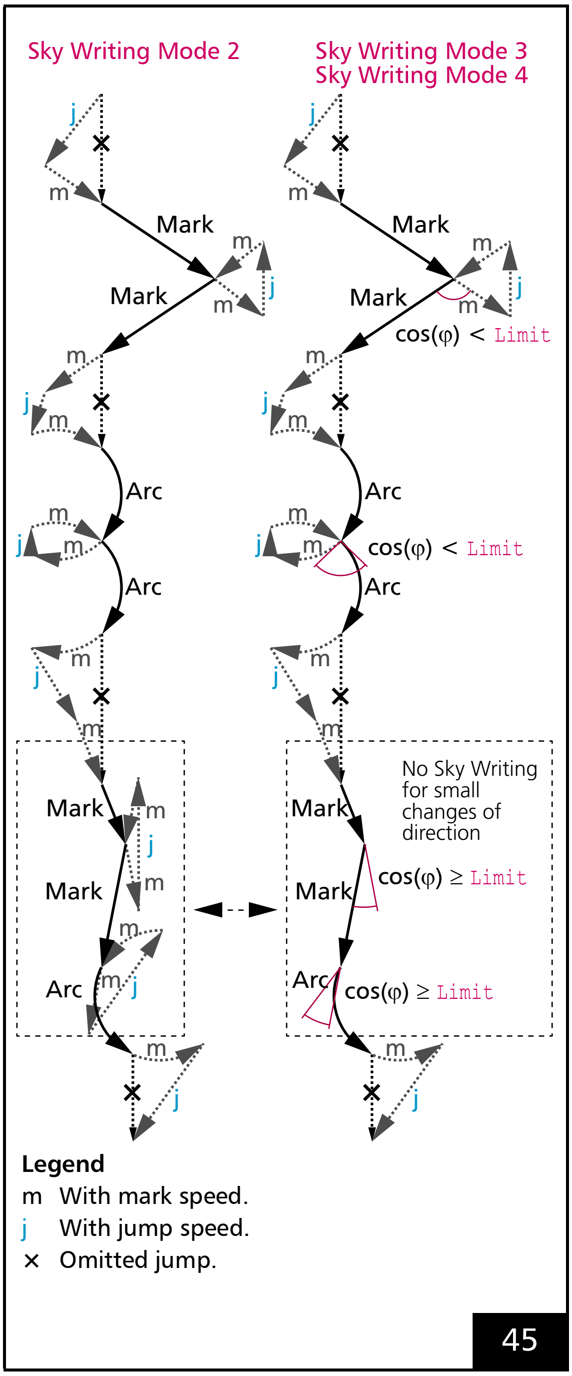
In Sky Writing Mode 2, the RTC6 PCIe Board calculates *time-shortened* Sky Writing motions.

Here, too, each to-be-executed vector/arc gets preceded and appended with a run-in motion and a run-out motion in extension of the vector/arc at mark speed. Within a Sky Writing Mode 2 marking sequence, however, neither forerun motion (in the run-in phase) nor retrace motion (in the run-out phase) of the galvanometer scanners occur. Instead, the RTC6 PCIe Board executes Sky Writing jumps (at the currently specified jump speed) from jump vector startpoints to run-in startpoints, from run-out endpoints to run-in startpoints, and from run-out endpoints to jump vector endpoints, see Figure 44.

set_sky_writing and **set_sky_writing_para** always switch on Sky Writing Mode 1. Therefore, it is only possible to *change the mode afterwards* to Sky Writing Mode 2 by **set_sky_writing_mode** or **set_sky_writing_mode_list**.

Sky Writing Mode 2 activation affects the same commands as Sky Writing Mode 1 (except ellipse commands, see below). *Time-shortened* Sky Writing, however, can only occur within a sequence of non-parameterized [*]mark[*] Commands, arc commands and Jump Commands (may also contain timed or 3D commands). If such a sequence gets interrupted by some other "non-Sky Writing Mode 2-capable" list command (for example, a [*]para[*] Command, ellipse command or any short list command), then the RTC6 PCIe Board suspends Sky Writing Mode 2 and complete the preceding command in Sky Writing Mode 1 (with a retrace motion of the galvanometer scanners). This suspension of Sky Writing Mode 2 does not deactivate it: subsequent "Sky Writing Mode 2-capable" commands are started at in Sky Writing Mode 1 (with a forerun motion of the galvanometer scanners) and then executed again in Sky Writing Mode 2.

Even with Sky Writing Mode 2 switched on, ellipse commands are always executed in Sky Writing Mode 1.



Sky Writing motions (grey) in Sky Writing Mode 2 and Sky Writing Mode 3 and Sky Writing Mode 4.

Sky Writing Mode 3

The time cost of Sky Writing motions for vectors and arcs having only small directional changes within a Polyline is probably disproportionately high for the gained accuracy.

Therefore, **set_sky_writing_mode** or **set_sky_writing_mode_list** can be used to switch to Sky Writing Mode 3 or Sky Writing Mode 4. A switching limit angle can be specified for them by **set_sky_writing_limit** or **set_sky_writing_limit_list**, see also Figure 45:

- For larger angle deviations `cos(φ) < Limit` between successive Mark Commands of a Polyline a Sky Writing motion is executed as in Sky Writing Mode 2
- For smaller angular changes `cos(φ) ≥ Limit` a variable polygonal delay is executed, see Section "Variable Polygon Delays", page 154

Notes

- Notes, page 166 on Sky Writing Mode 4 apply.

Sky Writing Mode 4

- Prerequisite: DLL 646, OUT 649
- Like Sky Writing Mode 3
  – Plus allows short list commands(1) between Jump Commands/Mark Commands

Notes

- In Sky Writing Mode 4, a Delayed Short List Command is executed in the same way as an Undelayed Short List Command, that is, after the run-out of the preceding vector.
- If there are several short list commands(1) between 2 vectors: Make sure that no overruns occur by **get_overrun**.
- To calculate the Sky Writing motions, the RTC6 board searches for the next non-Short List Command in the RTC6 List Memory (if none is available: endless).
- Do not call **set_mark_speed** between jump vectors and mark vectors.
  Otherwise, the calculated run-in position of the jump to the subsequent mark vector would be incorrect because it is still based on the original mark speed.
- If a Polyline ends with a short mark vector (shorter than mark speed × `Timelag`) - and a Polygon Delay has been executed but not a Sky Writing motion - then the length of the short vector (caused by the Tracking Error) may possibly not achieve the precision expected with Sky Writing.
- In Sky Writing Mode 3 and Sky Writing Mode 4 a Sky Writing Mode 1 motion (see Section "Sky Writing Mode 2", page 164) occurs:
  – At the beginning of a Polyline
  – At the end of a Polyline
  – With a Polyline interruption due to "non-Sky Writing Mode 2-capable" list commands

(1) See Short List Command, page 335.

Synchronization

The timing diagram for scan-head control and laser control in Sky Writing Mode 1 is shown in Figure 46.

The timing diagram for Sky Writing Mode 2, Sky Writing Mode 3 and Sky Writing Mode 4 is similar but there are no galvanometer scanner motions in opposite directions in run-in and run-out.

The relevant parameters for Synchronization are:

- `Nprev`
- `Npost`
- `Timelag`
- `LaserOnShift`

These are specified by **set_sky_writing_para**, **set_sky_writing** and the corresponding list commands:

- The parameter `Nprev` is a whole number in units of 10 µs and defines the duration of the run-in:

Mode	Duration
Sky Writing Mode 1	20 × `Nprev` [µs]
Sky Writing Mode 2	10 × `Nprev` [µs]
Sky Writing Mode 3	10 × `Nprev` [µs]
Sky Writing Mode 4	10 × `Nprev` [µs]

- The `Npost` parameter `Npost` is a whole number in units of 10 µs and defines the duration of the run-out:

Mode	Duration
Sky Writing Mode 1	20 × `Npost` [µs]
Sky Writing Mode 2	10 × `Npost` [µs]
Sky Writing Mode 3	10 × `Npost` [µs]
Sky Writing Mode 4	10 × `Npost` [µs]

- The parameters `Timelag` (64-bit IEEE floating point value rounded to integer multiple of 1/64 µs) and `LaserOnShift` (whole number in units of 1/64 µs) define the delay of the Signals for "Laser Active" Operation switch-on time points and switch-off time points relative to the starting set position and ending set position:
  - Delay of switch-on time point relative to the starting set position / µs = `Timelag` + 1/64 µs × `LaserOnShift`
  - Delay of switch-off time point relative to the ending set position / µs = `Timelag`

For `LaserOnShift` = 0, the Signals for "Laser Active" Operation are switched on (off) with a delay of `Timelag` relative to the starting set position (ending set position).

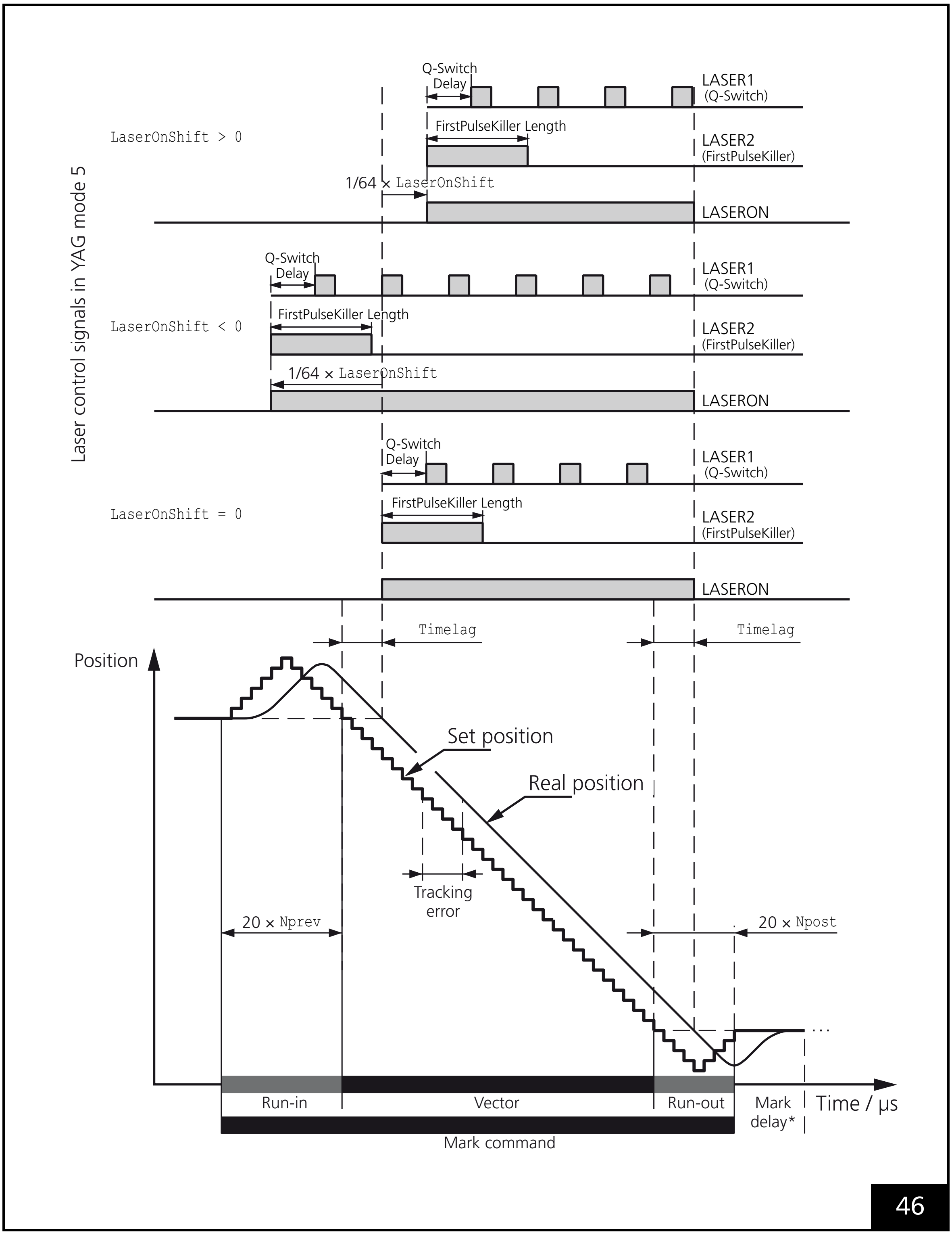
By setting the parameter `Timelag` to the actual Tracking Error (and the parameters `Nprev` and `Npost` to sufficiently large values), it is ensured that:

- Signals for "Laser Active" Operation switch on/off precisely at the endpoints of the desired vector
- Tracking Error becomes constant even before the vector's startpoint is reached
- The vector is scanned right up to its endpoint with constant mark speed

With Sky Writing, the Laser Delays from **set_laser_delays** are not effective.

The switch-on time point of the Signals for "Laser Active" Operation can be adjusted with the parameter `LaserOnShift` (for example, to the `HalfPeriod` of **set_laser_pulses** or to the reaction times of the laser itself), so that the first laser pulse occurs exactly with the starting set position:

- Positive `LaserOnShift` values delay the switching on of the Laser Control Signals.
- Negative `LaserOnShift` values advance the switching on of the Laser Control Signals (at the earliest, however, until the beginning of the run-in).
  Negative `LaserOnShift` values are also necessary to "make room" for a Q-Switch delay or a FirstPulseKiller signal in a YAG Mode.



Scan-head and laser control in Sky Writing Mode 1 (with parameters `Timelag`, `Nprev`, `Npost` and `LaserOnShift`)
(the Laser Control Signals LASER1 and LASER2 in YAG Mode 5 are exemplarily shown)
(* This Mark Delay is only effective with Sky Writing Mode 3 and Sky Writing Mode 4).

Notes

- By **set_sky_writing_mode** and **set_sky_writing_mode_list**, you can switch back and forth between Sky Writing Mode 1, Sky Writing Mode 2, Sky Writing Mode 3 or Sky Writing Mode 4.
  Even temporarily switching off is possible, as long as `Timelag` > 0 is set.
- When searching for appropriate Sky Writing parameters, initially **set_sky_writing** and Sky Writing Mode 1 should be used: as start value for the `Timelag` parameter, the Tracking Error of the galvanometer scanners. should be set. Then `Timelag` (with `LaserOnShift` = 0) can be fine-tuned such that marking vector *ends* are exactly marked and only afterwards the parameter `LaserOnShift` should be adjusted (keeping constant the parameter `Timelag`), so that the marking vectors' *beginnings* are exactly marked, too.
  In a second step **set_sky_writing_para** can be used to optimize the parameters `Nprev` and `Npost`: `Nprev` and `Npost` may be decreased (relating to the default settings of **set_sky_writing**, see below) to minimize marking times. Alternatively, `Nprev` may be increased to enable a correspondingly large negative `LaserOnShift`.
- With **set_sky_writing** and **set_sky_writing_list**, `Nprev` and `Npost` are predefined:
  – `Nprev` = approx. 0.15 × `Timelag`
  – `Npost` = approx. 0.10 × `Timelag`
  This results in the following run-in and run-out durations:

	Mode	Duration
run-in	Sky Writing Mode 1	3 × `Timelag`
	Sky Writing Mode 2	1.5 × `Timelag`
	Sky Writing Mode 3	1.5 × `Timelag`
	Sky Writing Mode 4	1.5 × `Timelag`
run-out	Sky Writing Mode 1	2 × `Timelag`
	Sky Writing Mode 2	1 × `Timelag`
	Sky Writing Mode 3	1 × `Timelag`
	Sky Writing Mode 4	1 × `Timelag`

- With **set_sky_writing_para** or **set_sky_writing_para_list**, you can also specify other run-in and run-out phases, for example, shorter ones to reduce marking times (but this may be at the expense of accuracy).
  Shorter run-in and run-out phases are particularly beneficial when lines are marked in only one direction without interruption and jump speed and mark speed are set equally.
  Note that in Sky Writing mode no automatic adjustment of Laser Delays and Scanner Delays is performed:
  – If the next Mark Command's run-in phase begins when the preceding Mark Command's LaserOn Delay has not yet expired, then the laser does not switch on for the preceding command.
  – If the next Mark Command's run-out phase begins when the preceding Mark Command's LaserOff Delay has not yet expired, then the laser does not switch off for the preceding command.
  – In Sky Writing Mode 1, a positive `LaserOnShift` time (in µs) should exceed the smallest occurring marking time by no more than (20 × `Npost` – `Timelag`) – and in Sky Writing Mode 2, Sky Writing Mode 3 and Sky Writing Mode 4 by no more than (10 × `Npost` – `Timelag`). Otherwise, the laser is not switched on for this marking.

- Pulse lengths and frequencies (in a YAG Mode additionally the Q-Switch delay and the FirstPulseKiller signal parameters) previously defined for the Signals for "Laser Active" Operation are kept unchanged in the Sky Writing mode. On the other hand, previously defined LaserOn, LaserOff, mark, polygon or "Variable Polygon Delays" *are not* taken into account in the Sky Writing mode. They are fully functional again after deactivation of Sky Writing mode. Deactivation of Sky Writing mode results in addition of a Mark Delay defined prior to activation (see Figure 46).
- In Sky Writing Mode 1, the run-in and run-out phases take place fully within the respective Mark Command. As a result, execution of the Mark Command is extended by a time period (in 10 µs) of (2 × `Nprev` + 2 × `Npost`). In Sky Writing Mode 2, Sky Writing Mode 3 and Sky Writing Mode 4 execution is extended by a time period (in 10 µs) of at least (`Nprev` + `Npost`).
- Jump Delays are performed normally in Sky Writing Mode 1. The Jump Delay value (in 10 µs, see **set_scanner_delays**) may be reduced by approx. (2 × `Nprev`) or even to 0.
- Jump Delays are automatically (internally, dynamically) reduced by up to `Nprev` in Sky Writing Mode 2, Sky Writing Mode 3 and Sky Writing Mode 4
- Time-based [*]mark[*] Commands and "Arc" Commands can also be performed in Sky Writing mode, see Chapter 8.9 "Timed Commands", page 283. Here, however, a shorter marking period is coupled with a higher mark speed and therefore with longer distances and acceleration phase in run-in and run-out phase. Herefore, `Nprev` and `Npost` can be separately adjusted with respect to the specified mark speed.
- the Sky Writing mode is not taken into account (but also not deactivated)
  - During execution of [*]para[*] Commands, for example, Vector-Defined Laser Control is active
  - During execution of micro_vector[*] Commands, see Chapter 8.8 "micro_vector[*] Commands", page 282
- With Sky Writing Mode 2, Sky Writing Mode 3 or Sky Writing Mode 4 activated, the following functionalities of the 2D Jump Commands **jump_abs** and **jump_rel** are *not* executed:
  - Automatic tuning switching in Jump Mode, see Chapter 8.1.5 "Jump Mode", page 226
  - Coordinate transformations with `at_once` = 2, see list item "With `at_once` = 2 …", page 235

Sky Writing with Minimum Mark Speed

To reduce the process time with Sky Writing motions, **set_sky_writing_min_speed_ctrl** can be used to shift part of the acceleration phase and deceleration phase into the mark vector.

To achieve this, the RTC6 PCIe Board automatically shortens `Nprev` and `Npost` and thus the Sky Writing motions. Therefore, at the mark vector start and mark vector end, only the specified `Speed` value (= minimum mark speed for Sky Writing) is reached rather than the full mark speed.

Notes

- See **set_sky_writing_min_speed_ctrl**, Comments, page 805.
- On *short* mark vectors the full mark speed may not never reached. Marking results may then not be as expected, even though the RTC6 PCIe Board automatically adjusts the laser switching times.

  With optimum marking results, the shortest mark vector length and `Speed` value are in a suitable ratio.

  – If the `Speed` value is a given –
  select the minimum mark vector length (l_{min}) so that the 100% mark speed ($V_{100\%}$) is reached on the mark vector, according to:

  $$l_{min} > ({V_{100\%}}^2 - Speed^2)/(A_{max})$$

  – If the minimum mark vector length (l_{min}) is a given –
  select the `Speed` value so that the 100% mark speed ($V_{100\%}$) is reached on the mark vector, according to:

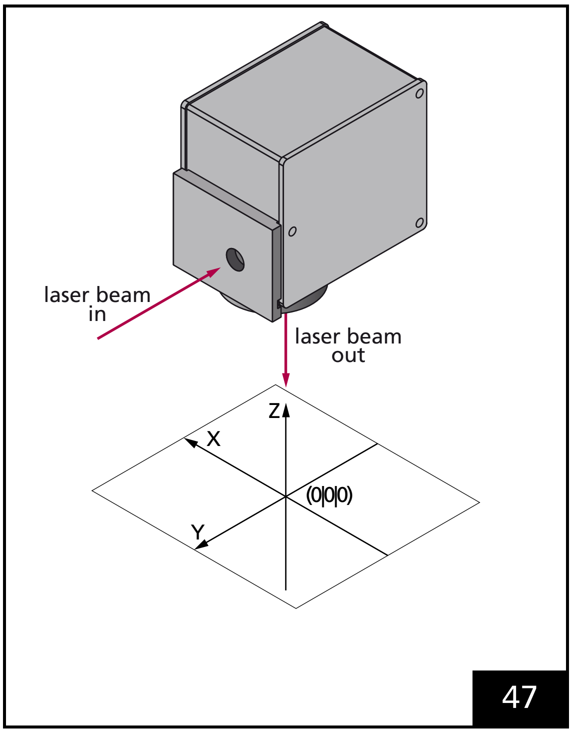
  $$Speed > \sqrt{{V100\%}^2 - (Amax \times lmin)}$$

7.3 Scan Head Control

7.3.1 Reference System

The reference system for the Image Field which is used by the RTC6 PCIe Board is shown in Figure 47.

The y axis points in the *reverse* direction of the input laser beam, the z axis points in the *reverse* direction of the output laser beam. x axis, y axis and z axis form a right-handed reference system. The origin of the reference system, that is, the point (0|0|0), is in the center of the Image Field.



Reference system for the RTC6 coordinates.

7.3.2 Image Field Size and Image Field Calibration

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 (that is, the distance between the input laser beam axis and the Image Field).

The x, y and z coordinates of a vector must be specified as signed 20-bit values (that is, as numbers between –524,288 and +524,287).

The calibration factor K defines the ratio of the digital point coordinates in *bits*(1) and the actual position of the point in *millimeters*.

Let a_0 denote the side length of the Image Field given by the maximum scan angle. The theoretical calibration factor is then $K_0 = 2^{20}/a_0$ [bits per mm(1)].

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 read out from a used correction table by **get_table_para** or **get_head_para**.

Given the calibration factor K, the side length a of the usable Image Field in *millimeters* can be calculated:

$$a = \frac{2^{20}}{K}$$

(1) The expression "bits" is here synonymous with "digital control value" (see footnote (3) on page 139).

Typical Image Field

In general, the size of the usable (or "typical") Image Field – dependent on the objective and the optical configuration of the scan system – is smaller than the maximum adjustable Image Field.

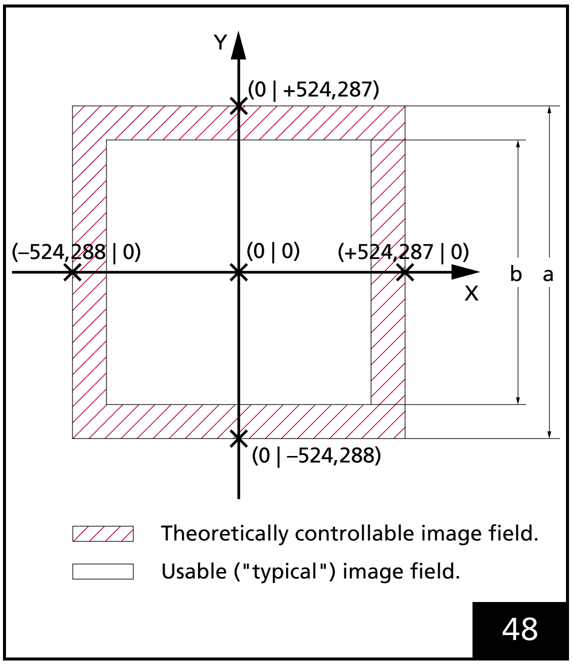


Image field size.

The scan head has a usable Image Field. If the laser focus moves outside this 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. Refer to your scan head operating manual's section on objectives.

- Compare the calculated side length *a* of the maximum adjustable Image Field with the side length *b* of the usable Image Field given in the technical specifications of your scan head manual (see Figure 48).
- If the laser focus is to be restricted to points within the usable 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 usable Image Field:

$$M = K \times b/2$$

Compatibility Modes

RTC6 Standard Mode

(Default after **load_program_file** and **set_rtc6_mode**)
The Image Field coordinates for the x axis, y axis and z axis and all related parameters (for example, jump speed or wobbel amplitude) are to be specified as signed 20-bit values.

RTC5 Compatibility Mode

(**set_rtc5_mode**)
The Image Field coordinates for the x axis and y axis and all associated parameters are to be specified as signed 20-bit values – as in RTC6 Standard Mode.
See also Chapter 2.12.2 "Adapting RTC5 Source Code for the RTC6 PCIe Board", page 62.

The Image Field coordinates of the z axis are to be specified as signed 16-bit values. The RTC6 PCIe Board automatically multiplies all z coordinates by 16(1).
As the z calibration factor, a value 16 times smaller must be used.

RTC4 Compatibility Mode

(**set_rtc4_mode**)
The Image Field coordinates for the x axis, y axis and z axis and all related parameters are to be specified as signed 16-bit values. The RTC6 PCIe Board automatically multiplies them by 16(1).
As the xy calibration factor and z calibration factor, a value 16 times smaller must be used. See also Chapter 2.10.2 "Porting RTC4 Source Code to the RTC6 PCIe Board", page 50.

(1) The allowed value range decreases accordingly.

7.3.3 Virtual Image Field

In particular for Processing-on-the-fly applications, a virtual Image Field of size 29-bit(1) is available.

Therefore, Vector Commands and "Arc" Commands can be loaded for objects up to 512 times larger than the real Image Field (in the Processing-on-the-fly direction). For details on Processing-on-the-fly in the virtual Image Field, see Chapter 8.6.6 "Virtual Image Field with Processing-on-the-fly", page 261.

At runtime, the current coordinates are clipped to the real Image Field [–524,288…+524,287], see #2…#6 in Chapter 7.3.6 "Output Values to the Scan System", page 186.

In addition, the extended value range of the virtual Image Field can also be used for utilizing the complete real 20-bit Image Field during coordinate transformations (such as rotations, shrinkages or shifts, see Chapter 8.2 "Coordinate Transformations", page 233). Otherwise, certain edge areas of the real Image Field may remain inaccessible during such coordinate transformations.

Coordinate Transformations in the Virtual Image Field

"Virtual coordinate transformations" (particularly translations and rotations) are sometimes needed to compensate certain mechanical tolerances of object positioning when performing continuous marking larger than the real Image Field.

See #3 in Chapter 7.3.6 "Output Values to the Scan System", page 186.

"Virtual coordinate transformations" are defined by:

- `set_matrix`(`HeadNo` = 4)
- `set_offset`(`HeadNo` = 4)
- `set_offset_xyz`(`HeadNo` = 4)
- `set_angle`(`HeadNo` = 4)

They let you define matrix coefficients and 2D offsets (with **set_offset_xyz**, `ZOffset` is ignored). With `at_once` = 1, they become effective immediately, if no list is currently being processed. Otherwise, they are only saved as with `at_once` = 0.

Stored transformations are automatically activated when a Processing-on-the-fly Session is restarted (the changed output position is reached at jump speed). They remain active(2) even after the end of this Processing-on-the-fly Session.

As long as a Processing-on-the-fly Session is active, the parameters for the "virtual coordinate transformation" cannot be changed, they can only be saved.

By **set_matrix**(4, 0.0, 0.0, 0.0, 0.0), the "virtual coordinate transformation" can be deactivated again, see Section "Clock Overruns", page 187.

The adjustment range for offsets is ±28 bits. Matrix coefficients must not exceed an absolute value of 2.0. The offset is applied after the matrix operation.

For "virtual coordinate transformations" *cannot* be used:

- **set_scale**
- all list commands for coordinate transformations

With RTC6 Software Package ≥ V1.6.1, "virtual coordinate transformations" can be also defined by a "Global Online Positioning", see Chapter 8.3.2 ""Global Online Positioning"", page 240.

(1) Up to DLL 608: 24-bit.

(2) With RTC6 Software Package < V1.5.0, "virtual coordinate transformations" are only available during a **set_fly_2d** Processing-on-the-fly Session.

7.3.4 3D Image Field

Carrying Out Adjustment

SCANLAB 3D correction tables are calculated such that the plane of the middle focus position is controlled with z = 0.

Therefore, the mechanical distance between scan system and working plane must be adjusted accordingly. With the vario*SCAN* series, a specific distance to the scan system has to be maintained in addition. The values are to be taken from the manual of the respective 3-axis scan system or vario*SCAN*.

Deviations from these should preferably be set via the user program by **set_defocus**, **set_defocus_offset**, **set_offset_xyz** or the corresponding list commands.

Once the working distance and distance to the scan system are correctly adjusted, then subsequently the laser focus position should be fine-tuned. For this, a test pattern is to be marked in the middle of the z = 0 working plane. The optimum laser focus position for processing results can be achieved:

- With the vario*SCAN* series, by manually turning the focusing ring, see corresponding Manual
- With a vario*SCAN* FLEX, by adjusting the focusing optic position, see Manual for "RTC5/6 varioSCAN 40 FLEX Extension" Extension Board (#0128683)
- With a vario*SCAN* FC and intelli*WELD* FC by adjusting the coefficient *A* of the used 3D correction table[(1)]

Checking the z axis Calibration

The optimum output values for the z axis also depend on various parameters such as beam divergence of the used laser and tolerances of the optical components.

Such information is generally not available to SCANLAB and therefore, are not included in the calculation of 3D correction tables.

Therefore, in some cases the calculated 3D correction table might not fit optimally the individual scan system. To test whether this is the case, run a laser marking test that covers the entire 3D Image Field.

Check if the laser focus meets the requirements of your application. If you find that the spot diameter varies considerably, then a recalibration of the z axis correction may help under certain circumstances.

The aim of the z axis calibration procedure is to determine suitable coefficients *A*, *B* and *C*. These can be transferred to the RTC6 PCIe Board by **load_z_table**.

A, *B* and *C* are coefficients of the parabolic function

$$z_{out} = A + B\mathrm{l} + C\mathrm{l}^2$$

(focus length value *l* in the RTC4 compatibility range [–32,768…+32,767], see **load_z_table_no**; See also Notes below).

They determine the relationship between the z output value z_{out} and the focus length value l. For each point (x|y|z) in the 3D Image Field, the focus length value l corresponds to the focus length difference between the specified point (x|y|z) and the point(0|0|0).

The ABC coefficient values of a correction file on the PC can be read out directly with **read_abc_from_file** and written into with **write_abc_to_file**, see following Notes.

(1) Coefficients *A*, *B* and *C* can be queried by **get_head_para** and newly set by **load_z_table**. Only *A* should be modified. *B* and *C* should be passed over unchanged – as queried – by **load_z_table**.

Notes

- As of RTC6 Software Package V1.12.0, you can leave the focus length value *l* in RTC6 20-bit range [–524,288…+524,287], if you use **load_z_table_20b** / **load_z_table_no_20b** and **read_abc_from_file_20b** / **write_abc_to_file_20b**.
- ABC values from the `*_ReadMe.txt` file of correction file are to be interpreted as 16-bit values.

Procedure

- Refer also to the "Calibrating a 3-Axis Laser Scan System" Manual.
- See also Notes, page 176.

(1) Adjust the correct mechanical distance between the scan system and the z = 0 working plane and the correct distance between the vario*SCAN* device and the scan system(1).

(2) Load the 3D correction file by **load_correction_file**.

(3) Assign the 3D correction table by **select_cor_table**.

(4) Read out the assigned coefficients *A*, *B* and *C* by **get_head_para**.

(5) vario*SCAN* FC and intelli*WELD* FC only: proceed with step 10. Steps 6…9 do not need to be performed.

(6) Move the laser spot to the reference point(2) by **goto_xyz**.

(7) Set the z axis to the neutral (middle) position by **load_z_table_20b**(0, 0, 0).

(8) Place a test object at the reference point.

(9) Adjust the laser focus, see page 175.

(10) Move the focus to an arbitrary point (x|y|z) within the (required) 3D Image Field by **goto_xyz**(3).

(11) Query the focus length value l (in RTC6-20-bit range [–524,288…+524,287]) set by the RTC6 PCIe Board for this point(4) by **get_z_distance**.

(12) Optimize the laser focus at this point: vary the Z output value z_{out} until the quality of the laser focus meets your requirements(5) by **load_z_table_20b**(A=z_{out}, 0, 0).
A starting value can be calculated in accordance with $A + Bl + Cl^2$ by using the previously read focus length value l and the previously read or used values A, B and C.

(13) Repeat steps 10…12 for as many locations (x|y|z) as possible(6) and write down the values (l|z_{out}) for each new point. If possible, seek to thereby cover the entire 3D Image Field required by your application.

(14) Fit the function $z_{out} = A + Bl + Cl^2$ to your value pairs (l|z_{out}).

(15) Use the resulting coefficients *A*, *B* and *C* to adjust the 3D correction table by **load_z_table_20b**(*A*, *B*, *C*).

Values loaded by **load_z_table_20b** are overwritten by a subsequent **load_correction_file** (**load_correction_file** sets the three coefficients *A*, *B* and *C* to the values of the loaded correction table). After each **load_correction_file** or **select_cor_table**, you should therefore also call **load_z_table_20b**(*A*, *B*, *C*) again.

By **load_z_table_no_20b**, the three coefficients `A`, `B`, `C` can be assigned to correction table `No`. They are switched with **select_cor_table**.

Alternatively, the ABC values can be written permanently to the header of the correction file by **write_abc_to_file_20b**, see also parameter 5…7 in Section "ct5 Correction File Header", page 183.

(1) See the corresponding manual.

(2) The reference point is a point in the z = 0 working plane for which a middle focus length value is required for a sharp laser focus. The laser beam should be focused to the reference point when the z axis is set to the neutral position (Z output value z_{out} = 0). The coordinate values of the reference point (in mm) are provided in the `*_ReadMe.txt` file that accompanies the 3D correction file or in the 3-axis scan system's or the vario*SCAN* user manual. If you are using an F-Theta objective, the reference point is generally the origin (0|0|0).

(3) If you are using a scan system with an F-Theta objective, it is sufficient to select various points (0|0|z) on the z coordinate axis. The maximum possible 3D Image Field is specified in the corresponding user manual.

(4) The focus length value l can be positive or negative, see notes on **get_z_distance**.

(5) The optimal z_{out} output value can be positive or negative.

(6) Note that larger z control values lead to shorter working distances and that the focus thereby shifts toward the scan system. The test object might have to be tracked accordingly.

Testing 3-Axis Scan Systems with F-Theta Objective

- With the adjusted 3D correction table and 3D vector commands, see Chapter "3D Commands", page 247, mark two identical large squares. Mark one of the squares with the work piece in z position $z = z_{min}$, the other one with $z = z_{max}$.
- These two squares should match exactly. If this is not the case, your correction file is not perfectly suited to your objective. To solve this problem, measure the size (x and y) of both squares and report these values to SCANLAB. You will then receive a new 3D correction file.
- See also Section "Enhanced 3D Correction", page 249.

7.3.5 Image Field Correction and Correction Tables

Field Distortion

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[(1)], see Figure 49.

(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 mark 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 plane, a varying spot size results.

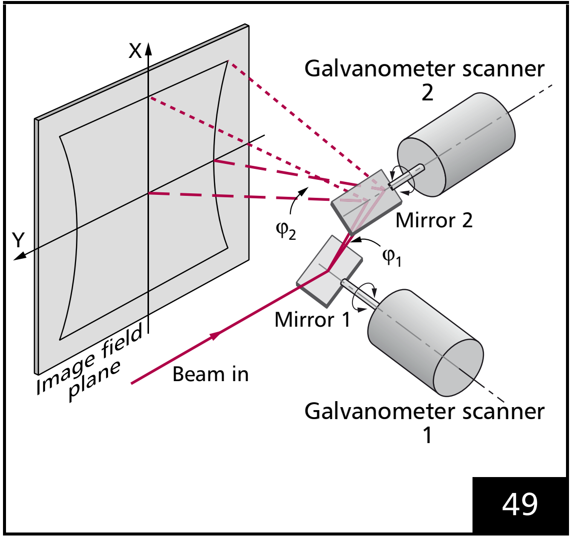


Image field distortion when deflecting a beam in a two-mirror deflection system.

By focusing the deflected laser beam with an F-Theta objective, effect 2 and effect 3 can be avoided. However, this causes a barrel-shaped distortion of the Image Field, see Figure 50.

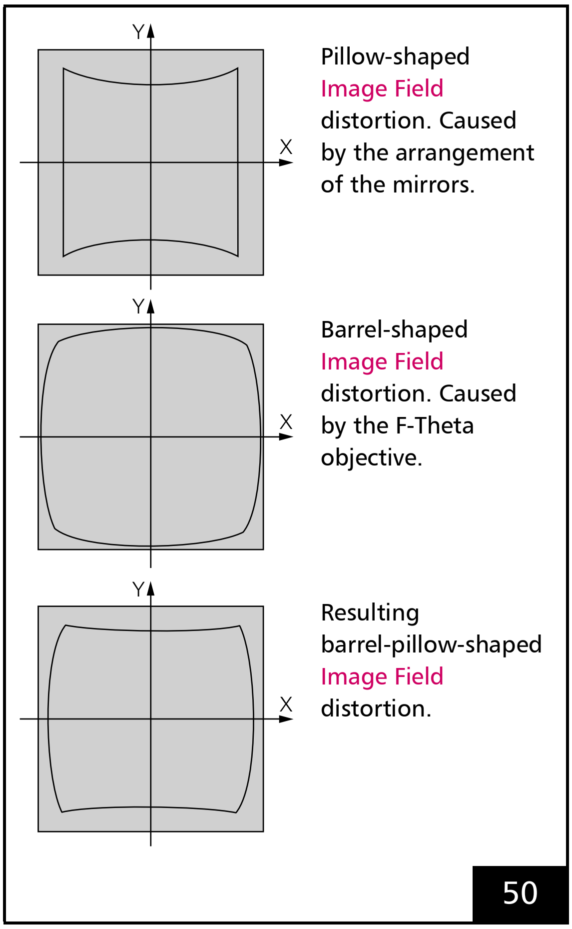


Image field distortion caused by the arrangement of the mirrors and by the F-Theta objective.

(1) Cause: 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.

Image Field Correction Algorithm

For these field distortions, the RTC6 PCIe Board board internally uses a algorithm to compensate for it. The algorithm is based on a correction table.

An orthogonal grid of 257 × 257 points is superimposed on the ideal square Image Field. The adjusted x and y coordinates for a corrected output of these grid points are stored in a correction table.

To move the focus to any point within the Image Field, the RTC6 PCIe Board calculates the corrected coordinates by interpolating from the grid points in the correction table.

The correction is executed for every single Microstep, see also Chapter 7.1.2 "Microstepping", page 143.

SCANLAB creates a correction file for every system type. For this purpose, the correction table is calculated based solely on general system data (such as mirror geometry, calibration and the objective 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 the correXion program line. These generate RTC correction files based on test measurement data. For further information, refer to the corresponding manual or contact SCANLAB).

Activating Image Field Correction

To activate Image Field correction, at the beginning of a user program the required correction tables must:

(1) be loaded from the corresponding correction files into RTC6 PCIe Board memory (by **load_correction_file**)

(2) assigned to the used scan head connectors (by **select_cor_table** or **select_cor_table_list**)

2D correction tables activate an Image Field correction for x and y coordinates,
3D correction tables additionally for z coordinates.

2D and 3D Correction Files

SCANLAB supplies 2D and 3D correction files. They have the extension `*.ct5` and the following naming scheme applies:

- `D2_xxx.ct5` for 2D correction files
- `D3_yyy.ct5` for 3D correction files

Here, `xxx` and `yyy` are numbers. Each correction file is calculated for a specific optical configuration. The configuration is specified in the accompanying `*_ReadMe.txt` file and in parameters of the correction file header, see also Section "ct5 Correction File Header", page 183.

Loading Correction Tables

By **load_correction_file**, up to 8 correction tables can be loaded from their corresponding correction file into the RTC6 PCIe Board memory, see also Chapter 8.5 "Controlling 2D Scan Systems and 3D Scan Systems", page 245.

If the Option "3D" is *not* enabled, then only 2D correction tables can be loaded.

2D correction tables can also be loaded from 3D correction files (by **load_correction_file** with `Dim` = 2). The 3D part is ignored.

If the Option "3D" is enabled, a 3D correction table can even be loaded from a 2D correction file (by **load_correction_file** with `Dim` = 3). Thereby, the 2D correction table is automatically supplemented with a linear z correction.

The actually suitable Z correction can be loaded by **load_z_table_no** or **load_z_table** after the assignment by **select_cor_table** (see below), see Chapter 7.3.4 "3D Image Field", page 175.

Notes

- RTC5/RTC6 correction files (file extension `*.ct5`) differ from those of prior RTC products (file extension `*.ctb`) in terms of size, structure and content.
- If *.`ct5` and *.`ctb` correction files have the same file name (except for the different extensions), then they were calculated for the same optical configuration.
- For converting older correction files, see Section "Converting Correction Files", page 185.

Assigning Loaded Correction Tables

Loaded correction tables are assigned to the two scan head connectors by **select_cor_table** or **select_cor_table_list**.

If neither the Option "Second Scan Head Control" nor the Option "3D" is enabled, then a 2D correction table can be assigned only to the Connector for First Scan Head.

If the Option "Second Scan Head Control" is enabled, then the two scan head connectors can each be assigned their own 2D correction table.

If the Option "3D" is enabled but not the Option "Second Scan Head Control", then a 2D correction table or a 3D correction table can be assigned exclusively to the Connector for First Scan Head. The Connector for First Scan Head outputs position signals for an xy scan system.

With a 3D correction table, the Connector for Second Scan Head outputs additionally position signals for the z axis (for example, vario*SCAN*) via both channels.

If the Option "Second Scan Head Control" and the Option "3D" are enabled, up to two (even different) 2D correction tables or a single 3D correction table can be assigned.

In order to control a 3D system, the (only) 3D correction table must be assigned exclusively to the connector for the xy scan system. There is no assignment for the z axis. The output for the z axis occurs automatically on both channels of the other port (`select_cor_table( n, 0 )` or `select_cor_table( 0, n )`).

Notes

- If no correction table has been loaded into the RTC6 PCIe Board memory, then the board outputs unforeseen values to the scan system. Therefore, at least a 1to1 correction table must be loaded (as 2D or 3D correction table), see Section "1to1 Correction Tables", page 182, if no 2D correction file or 3D correction file is available that has been calculated for the specific optical configuration.
- Correction files should have been assigned[1]:
  - before a list is started for the first time or
  - before the galvanometer scanners of a scan system are moved by a control command (like **goto_xy**)
- If only one scan head connector has an 2D correction table assigned, then the other scan head connector outputs no *position* signals.
- During initialization, **load_program_file** assigns a correction table according to `select_cor_table( 1, 0 )`. However, it leaves the galvanometer scanners at position (0,0). **load_program_file** cannot determine whether a correction table #1 is loaded at all. Therefore, there is no internal jump to the output position that is specified in the correction file. Its contents could be random (see above).
- **load_program_file** does not initialize the memory contents of the correction tables. The correction table values are random immediately after switching on the RTC6 PCIe Board. It is recommended to explicitly call **select_cor_table**. Before returning, **select_cor_table** itself internally waits for the end of the jump.
  - If **load_correction_file** is called after **load_program_file**, then:
    - **load_correction_file** automatically executes a `select_cor_table( HeadA, HeadB )` with the last used values for `HeadA` and `HeadB`
    - **load_correction_file** positions the galvanometer scanners with an internal jump at the currently set jump speed to the output position specified there
  - If **load_correction_file** is called prior to **load_program_file**, then the correction table is only saved. There can be no internal jump to the intended output position.

(1) Correction files can also be loaded before the RTC6 files have been loaded by **load_program_file**.

1to1 Correction Tables

1to1 correction files

- Are not assigned to a particular scan system
- In general, serve test purposes only
- Alternatively, can be loaded by **load_correction_file** with parameter `Name` = NULL.
  - According to the further `Dim` parameter a 2D or 3D 1to1 correction table is being generated internally.

Because some user programs require a file name and do not accept NULL as the name, the 1to1 correction file `Cor_1to1.ct5` is supplied within the RTC6 Software Package. This contains a calibration factor of value 0. If a user program strictly needs an "authentic" calibration factor, a corresponding value can be specified with `CorrectionFileConverter.exe`, see Section "Folder CorrectionFileConverter", page 33 (button **Show File Header** > field 'Field Calibration [Bit/mm]').

Inverse Tables

The ct5-correction file format supports "inverse tables". These are required for back transforming actual scan head positions to the associated Image Field coordinates.

For further information, see:

- Parameter 11, see page 184
- Chapter 8.1.3 "Monitoring the Positioning", page 223
- Section "Converting Correction Files", page 185

ct5 Correction File Header

The ct5-correction file header contains 16 parameters, see following table.

These can be read out and thus directly incorporated into a user program:

- from the currently loaded correction tables by **get_table_para**
- from assigned correction tables by **get_head_para**

Notes

- With ctb-correction files, some parameter values can exclusively be taken from its associated `*_ReadMe.txt` file. These must be transferred manually to the user program.
- A `ct5` file created by converting another `ctb` file, see Section "Converting Correction Files", page 185, does not contain all parameter data.
- A 1to1 correction file does not contain all parameter data.
- Parameters 3…7 are only relevant for 3D marking. In 2D correction tables, they are 0.
- For the same 3-axis scan system, the ABC coefficients (parameter 5…7) in ctb-correction files and ct5-correction files are identical.

Parameter[a]	Description
0	Type of the correction table. • = 0.0: 2D correction table • = 1.0: 3D correction table
1	Calibration factor K_{xy} [bit/mm]. See Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 172.
2	Focal length or working distance [mm]. • For a configuration with a scan objective: the effective focal length of the objective [mm]. • For a configuration without a scan objective: the working distance A [mm]. A = distance from the optical axis of the incident laser beam at the first deflection mirror to the image plane.
3	Stretch factor for the x direction. Compensates the pyramid-shaped Image Field change which exists in the z direction of 3D markings.
4	Stretch factor for the y direction. See parameter 3.
5	Coefficient *A* of the polynomial for z axis control, see page 175: offset part, ±26 Bit.
6	Coefficient *B* of the polynomial for z axis control, see page 175: linear part, ±11 Bit.
7	Coefficient *C* of the polynomial for z axis control, see page 175: square part, ±4 Bit.
8	Number of the correction file. With correction files supplied by SCANLAB, the parameter corresponds to the number in the file name (for example, 145 for `D2_145.ct5` or `D3_145.ct5`).

Parameter[a] (cont'd.)	Description (cont'd.)
9	Differentiation between correction with or without an F-Theta objective. The following applies: Parameter = $10 \times P_{Obj} + P_{Typ}$ with • P_{Obj}= 0: Correction without F-Theta objective P_{Obj}= 1: Correction with F-Theta objective • If correction with F-Theta objective: P_{Typ} = 0.0: without distortion data P_{Typ} = 1.0: with F-Theta's F-stop progression condition P_{Typ} = 2.0: with image height table
10	Indicator for the source of the correction file. The following applies: Parameter = $1000 \times P_{Orig} + P_{Ver}$ with • P_{Orig} = 10000: originally calculated file P_{Orig} = 20000: converted from `ctb` file P_{Orig} = 30000: reconstructed from txt file By manipulating a correction file using correction programs available from SCANLAB, P_{Orig} is increased by 1 in each case. – P_{Ver} = Version number of the program used to create the correction file
11	Information about the inverse table. The following applies: Parameter = $P_{Exist} + 2 \times P_{Calc}$ with • P_{Exist} = 1.0: valid inverse table is present P_{Exist} = 0.0: no valid inverse table present • If valid inverse table is present: P_{Calc} = 0: inverse table calculated ab initio P_{Calc} = 1: inverse table numerically calculated
12	Angle calibration of the scan system. Mechanical angle deflection in [± °] at 96% of the maximum control.
13	Code for the scan head geometry used for the calculation (for internal use only), for example, • = –1.0: unknown geometry (for example, for a table converted from a `ctb` file) • = 0.0: standard geometry
14	Indicator for whether an additional protective window has been taken into account. The following applies: Parameter = $1{,}000{,}000 \times P_T + 1{,}000 \times P_I$ with • P_T = Protective window thickness in mm (max. 2 decimal places) • P_I = Refraction index (max. 3 decimal places) Example: The value 3,521,450.0 corresponds to a protective window thickness of 3.52 mm and a refraction index of 1.450.
15	Indicator for whether the Image Field size has been limited in the correction file. • = 0.0: without field size limit • = 2.0: with field size limit

(a) Numbering according **get_table_para** and **get_head_para**.

Converting Correction Files

The RTC6 Software Package includes the program `CorrectionFileConverter.exe` for converting `ctb` correction files to `ct5` correction files and vice versa. The corresponding manual is supplied in English and German.

When converting a `ctb`-correction file into a `ct5`-correction file, the `ct5`-correction file header receives a corresponding origin indicator (parameter 10, page 184, P_{Orig} = 20000).

Such conversions do *not* produce fully complete `ct5`-correction files because the file header lacks several parameters. These can be subsequently entered manually by `CorrectionFileConverter.exe` (via button **Show File Header**).

Moreover, it may happen that the inverse correction table to be calculated numerically during the conversion does not cover the entire Image Field or cannot be calculated. In this case, a 1to1 correction table is inserted instead, see also parameter 11, page 184.

In contrast, converting a `ct5`-correction file into a `ctb`-correction file results in a fully complete `ctb`-correction file.

Parameter information from the `ct5`-correction file header are:

- no longer contained in a `ctb`-correction file for 2D correction table
- contained partially in a `ctb`-correction file for 3D correction tables

7.3.6 Output Values to the Scan System

Calculation

Calculation steps for generating the RTC output values to the scan system from the current x, y, x coordinate values
(1) Decomposing vectors and arcs to Microsteps[(a)]
(2) If applicable: Applying a wobbel motion[(b)] – for example, from **set_wobbel_mode** call
(3) If applicable: Applying a global coordinate transformation in virtual Image Field[(c)] – for example, from calls of **set_matrix**, **set_angle**, **set_offset** with `HeadNo` = 4
(4) If applicable: Applying the set Processing-on-the-fly correction[(d)] – for example, from calls of **set_fly_x** or **set_fly_x_pos**
(5) If applicable: Applying coordinate transformations to align the 1 (2) scan system(s) relative to the Image Field[(e)] (a) Transforming x coordinate values and y coordinate values as per matrix × angle × scale ("2D transformation") • for example, from calls of **set_matrix**, **set_angle**, **set_scale** and corresponding list commands (b) Applying an offset to x coordinate values and y coordinate values as well as z coordinate values[(f)] • for example, from calls of **set_offset** or **set_offset_xyz** and corresponding list commands
(6) Clipping x coordinate values and y coordinate values to the boundary values of the real 20-bit Image Field as soon as they exceed the real Image Field range (virtual Image Field)[(g)]
(7) If applicable: Correcting x coordinate values and y coordinate values according to the assigned correction table ("2D Image Field correction")[(h)] – From **select_cor_table** call or **select_cor_table_list** call
(8) If applicable: Applying the focus shift to z coordinate values – for example, from **set_defocus** call or **set_defocus_list** call
(9) If applicable: Correcting z coordinate values according to the assigned correction table ("3D Image Field correction")[(h)] – From **select_cor_table** call or **select_cor_table_list** call
(10) Clipping z coordinate values to the boundary values of the real 20-bit Image Field as soon as they exceed the real Image Field range (virtual Image Field)[(i)]
(11) If applicable (only some legacy scan systems): Applying an explicitly[(j)] or automatically[(k)] set gain correction and offset correction for the x galvanometer scanner and y galvanometer scanner – From **set_hi** call (explicit) – From **auto_cal** call (automatic)

(a) See Chapter 7.1.2 "Microstepping", page 143.
(b) See Chapter 8.4 "Wobbel Mode", page 241.
(c) See Section "Coordinate Transformations in the Virtual Image Field", page 174.
(d) See Chapter 8.6 "Processing-on-the-fly", page 251.
(e) See Chapter 8.2 "Coordinate Transformations", page 233 and Chapter 8.5.2 "3D Scan Systems", page 246.
(f) A complete 3D calculation is always performed. If the Option "3D" is not enabled, z values are just not outputted.
(g) See Chapter 7.3.3 "Virtual Image Field", page 174.
(h) See Chapter 7.3.5 "Image Field Correction and Correction Tables", page 178.
(i) See Chapter 7.3.3 "Virtual Image Field", page 174.
(j) See Section "Customer-Specific Calibration", page 287.
(k) See Chapter 8.10 "Automatic Self-Calibration", page 284.

Notes

- Overflowing values are always clipped to the boundary values of the maximum possible value range.
- A complete 3D calculation is always performed. If the Option "3D" is not enabled, z values are just not outputted.

Value Ranges

For all scan system axes, signed 20-bit values (–524,288…+524,287) are outputted. This also applies to values which the scan system returns to the RTC6 PCIe Board, see also:

- Section "RTC4 Compatibility Mode", page 173
- Section "RTC5 Compatibility Mode", page 173
- Chapter 4.5.2 "XY2-100 Converter (Accessory)", page 71

Precalculation and Diagnosis

With **get_galvo_controls** the output values resulting for the given coordinates and setting parameters in the current configuration (correction table, coordinate transformations) can be determined without the galvanometer scanners moving.

Clock Overruns

The 10 µs clock cycle might not always suffice for calculating all data required by the computation-intensive Jump Commands, Mark Commands and Arc Commands if several of the available command options are utilized simultaneously – for example, simultaneous control of two scan systems, wobbel motion, coordinate transformation in the virtual Image Field, Processing-on-the-fly for two axes (correction by McBSP interface), "Automatic Laser Control", para vectors, data recording, "Variable Polygon Delay", short list commands (for example, in a Polyline), control commands during list execution.

This overrun situation can be internally detected and counted. You can appropriately test your user program by using **get_overrun** to count the number of overruns. Such overruns result in one or several peripheral ports not being accessible during the current 10 µs clock cycle, possibly including output to the scan head. The galvanometer scanner motion might also could pause for 10 µs.

7.3.7 Status Monitoring and Diagnostics

For status monitoring and diagnostic purposes, **get_value** or **get_values** can be used to read out a variety of signals:

- A 20-bit status word which is returned by the scan system, see Section "Status Information Returned from the Scan System", page 188
- The current LASERON signal
- The current Cartesian control values (that is, the so-called "sample values" common to both scan head connectors, see #2, #3 and #4.
- The control values specific to each scan head connector which take into account
  - any coordinate transformations defined by **set_matrix**, **set_scale**, **set_angle** or by the corresponding list commands, see #5a.
  - any z coordinate offset defined by **set_offset_xyz** or **set_offset_xyz_list**, see #5b.
  - any focal length offset defined by **set_defocus** or **set_defocus_list** and any loaded correction table, see #8 and #9.

Each of these signals can be recorded on the RTC6 PCIe Board for a longer time period by **set_trigger[*]** – with a selectable sample period. After that, **get_waveform** can be used to transfer them to the PC for analysis.

By **set_trigger**(`Period` Bit #31 = 1) (ring buffer functionality), practically arbitrary amounts of data can be recorded. The recorded values can be transferred to the PC in blocks during list execution using **get_waveform_offset**. After the measurement data memory has reached its capacity, measurement value recording automatically starts from the beginning.
The current status and progress of a measurement session started with **set_trigger[*]** can be queried by **measurement_status**.
The values returned by the scan system are always 20-bit values.

Status Information Returned from the Scan System

The scan system transmits the following signals to the RTC6 PCIe Board every 10 μs via the SL2-100 protocol:

- A 20-bit status word.
  It can mean the following data types:
  - The actual 20-bit status word (the upper 16 bits are identical to the 16-bit status word of the XY2-100 protocol (XY2-100 status word), for a description see **get_head_status**).
  - Only with iDRIVE Scan Systems, see Chapter 8.1 "iDRIVE Functions", page 221:
    a different data type selectable with **control_command**.

  The 20-bit status word can:
  - be read out by **get_value**/**get_values**
  - be recorded by **set_trigger[*]**
- 6 additional status bits.
  With iDRIVE Scan Systems with SL2-100 protocol, the status bits (PowerOK, TempOK and PosAck; per axis) are transferred in parallel to and independently from the 20-bit status word. Therefore, these status bits can even be used to monitor the scan system (see Section "Automatic Suppression of Laser Control Signals", page 192) if, for example, an "Automatic Laser Control" (see Chapter 7.4.9 ""Automatic Laser Control"", page 205) is active that requires a different return data type.
  The 6 additional status bits can be read out by **get_head_status**.

Notes

- Scan systems with XY2-100 protocol require the use of the XY2-100 Converter (Accessory).
- For iDRIVE Scan Systems with XY2-100 protocol, the actual XY2-100 status word must be set as the to-be-returned data type (default type after switching on).

7.4 Laser Control

At certain output ports, the RTC6 PCIe Board outputs signals which can be used for controlling various laser types ("Laser Control Signals" LASERON, LASER1, LASER2). These output ports are at the:

- LASER Connector, page 75
- EXTENSION 2 Socket Connector, page 83

The laser control mode is set by **set_laser_mode**:

- The CO_2 Mode (laser mode 0; default after **load_program_file**) is designed for controlling a CO_2 laser.
- YAG Mode 1 (laser mode 1), YAG Mode 2 (laser mode 2), YAG Mode 3 (laser mode 3), YAG Mode 5 (laser mode 5) are designed for controlling lasers from the Nd:YAG family (and related).
- Laser Mode 4 and Laser Mode 6 are designed for controlling free-running lasers.(1)

All Laser Control Signals are TTL signals. By **set_laser_control**, it is set whether they are active-HIGH or active-LOW, see also Chapter 4.6.1 "LASER Connector", page 75. Their current setting can be queried by **get_startstop_info** (Bit #13).

For the maximum current load values, see Chapter 15 "Technical Specifications – RTC6 PCIe Board", page 950.

(1) For synchronization to externally controlled free-running lasers, see Chapter 7.4.10 "Synchronization of the RTC6 Clock Cycle and an External Clock Signal", page 216.

7.4.1 Enabling, Activating and Switching Laser Control Signals

All Laser Control Signals are suppressed:

- After a hardware reset and
- After initialization with **load_program_file**.

Then, all laser control signal output ports (LASERON, LASER1 and LASER2) are in high impedance mode.

Before Laser Control Signals can be outputted at all, their polarity must first be set by **set_laser_control**. This cancels the tristate state at the same time (= "global unblock"). By default, LASERON and LASER1/LASER2 are set to their respective "Off" levels.

The tristate state is only set again:

- after a hardware reset (restart)
- a call of **load_program_file**

Furthermore, real Laser Control Signals must be enabled:

- either simultaneously by **set_laser_control**(Bit #2 = 0) or
- afterwards with the separate command **enable_laser**

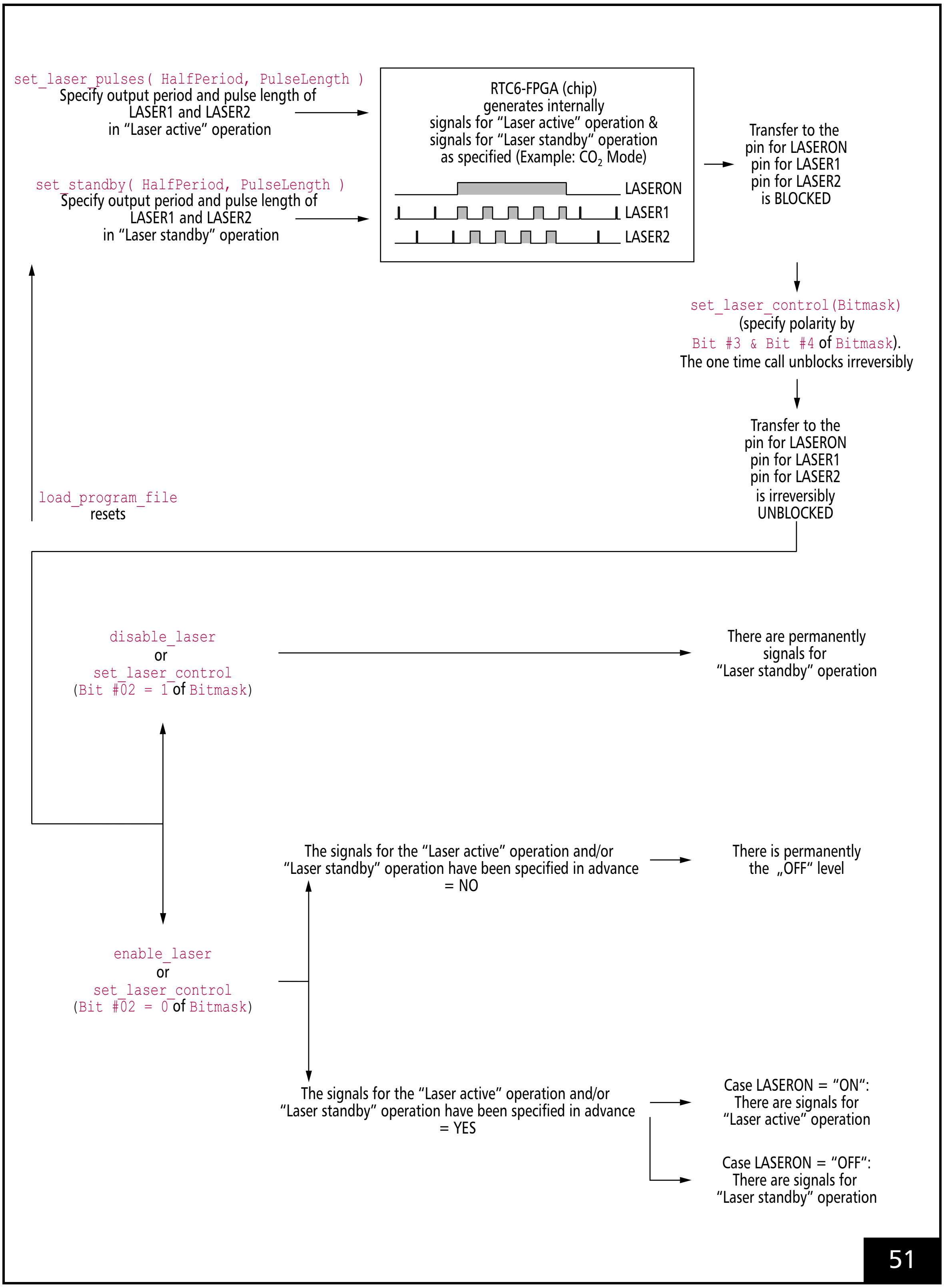
These can be suppressed again by **disable_laser** or **pause_list**. In both cases, all Laser Control Signals are set to their respective "Off" levels.

By **get_startstop_info** (Bit #9, Bit #10, Bit #13, Bit #14) it can be read out:

- The "global unblock" by **set_laser_control**
- The polarities of LASERON and LASER1/LASER2
- The enabling of the Signals for "Laser Active" Operation by **enable_laser**

General notes on the Laser Control Signals can be found in Section "Signals for "Laser Active" Operation", page 191 and Section "Signals for "Laser Standby" Operation", page 191. Mode-specific details depend on the set laser mode, see Chapter 7.4.3 "CO_2 Mode", page 193 to Chapter 7.4.8 "Pulse Picking Laser Mode", page 203.

The assignment of the Laser Control Signals to the respective output pins at the LASER Connector can be configured, see Chapter 7.4.2 "Configuring the LASER Connector", page 192.



RTC6 board hardware and laser control-related RTC6 commands.

Signals for "Laser Active" Operation

By **set_laser_pulses**, **set_laser_pulses_ctrl** or **set_laser_timing**, the Signals for "Laser Active" Operation can be

- activated: `HalfPeriod` ≠ 0 and `PulseLength` ≠ 0
- deactivated: `HalfPeriod` = 0 and/or `PulseLength` = 0

Even if the Signals for "Laser Active" Operation have been enabled and activated, they *are only outputted at the output ports, if they are switched on by further commands*.

They are automatically switched on, when:

- Mark Commands are called, see Chapter 7.1.1 "Marking with Vector Commands and "Arc" Commands", page 139

They are automatically switched off, when:

- a Mark Command is followed by a normal non-mark command
- a list is terminated by **set_end_of_list** or **stop_execution**
- a list is temporarily suspended by **set_wait**, **pause_list** or **stop_list**

In the latter case, the Signals for "Laser Active" Operation are switched on again if the list is continued by **release_wait** or **restart_list**.

They can also be switched on and off within a list with **laser_signal_on_list** and **laser_signal_off_list** or outside a list with **laser_signal_on** and **laser_signal_off** for an unlimited time.

Pulse Completion

Whether a modulation pulse (Q-Switch pulse) started with the LASERON signal switched on is still completely executed or cut off at LASER1 if it has not yet been fully processed when the LASERON signal is switched off, can be set by **set_laser_control**(Bit #0).

Signals for "Laser Standby" Operation

By **set_standby** or **set_standby_list**, the Signals for "Laser Standby" Operation can be

- activated: `HalfPeriod` ≠ 0 and `PulseLength` ≠ 0
- deactivated: `HalfPeriod` = 0 and/or `PulseLength` = 0

The Signals for "Laser Standby" Operation are continuously outputted at the output ports after their activation (without further commands), if no signals for the "Laser active" operation are switched on.

If the Signals for "Laser Active" Operation are deactivated (for example, by `PulseLength` = 0), there is no changeover. Then the Signals for "Laser Standby" Operation are continuously outputted even during the execution of Mark Commands.

If the Signals for "Laser Standby" Operation are deactivated, all output ports are set to the "Off" level in the "Laser standby" mode.

If activated signals for the "Laser standby" operation are to be deactivated when stopping a list with **pause_list** (here, only the Signals for "Laser Active" Operation are automatically deactivated), users must explicitly initiate this by calling **set_standby**(`PulseLength` = 0).

The current "Laser standby" parameters that may have been changed within a list can be read out by the control command **get_standby**.

Pulse Completion

With the Signals for "Laser Standby" Operation, pulse completion, see Pulse Completion, page 191, is not supported.

Automatic Suppression of Laser Control Signals

Case: Scan System Status Errors

By **set_laser_control** (Bit #16…Bit #27), it can set that the Laser Control Signals are to be automatically suppressed when the corresponding scan-system status signal (PowerOK, TempOK, PosAck of axis X/Y of head A/B) indicates an error (that is, is 0; "NOK").

As soon as at least one of the specified status signals is 0, then:

- Output of the Laser Control Signals are automatically interrupted. They are only continued, if all selected status signals are simultaneously 1 (Laser Control Signals disabled by **set_laser_control**, **disable_laser** or **pause_list** remain disabled regardless of the status signals' current value)
- Internal error bits are (cumulatively) set (which can be read out by **get_marking_info**)
- If accordingly set by **set_laser_control**(Bit #28 = 1), a **stop_execution** is automatically executed (the list stops, Laser Control Signals get permanently switched off)
- If accordingly set by **set_laser_control**(Bit #29 = 1) in addition, the **stop_execution** is forwarded as /Master-STOP (see Figure 70) to all Master/Slave connected RTC6 boards. Whether the signal is effective there can be set individually for each RTC6 board with **master_slave_config**. This ensures that the laser is automatically switched off even if the error occurs on an RTC6 board that is not intended for controlling the laser.

Case: Galvanometer Scanner Position Exceedances

range_checking can be used to define that the Laser Control Signals are to be automatically suppressed as soon as a galvanometer scanner exceeds a predefined range limit.

They are automatically switched on again as soon as the next Mark Command starts within the permitted range. This means that an interrupted Polyline remains suppressed for the rest of the Polyline.

7.4.2 Configuring the LASER Connector

LASER Connector pin (01), (02) and (09) can be configured by **config_laser_signals** and **config_laser_signals_list**, see Figure 17.

If you employ a variety of lasers or laser operational modes, then these commands might eliminate the need to configure at the hardware level (that is, using various cables and switches).

Whereas the default setting (for normal markings) outputs the LASERON signal as a laser start signal on the LASERON channel, you could, for example, configure the LASER2 signal as a gate signal outputted on the LASERON channel in Pixel Output Mode with pulse picking.

For other operational modes, you could also configure the FirstPulseKiller signal, page 369 as the laser start signal on the LASERON channel. This way, LASER1 signals can be outputted even before a delayed switch-on of the laser (which is not possible in the default setting).

The following description of the various laser modes applies to the default setting for laser control signal output.

7.4.3 CO_2 Mode

`set_laser_mode( Mode = 0 )` sets CO_2 Mode ("Laser Mode 0").

The timing diagram, see Figure 52, shows the corresponding signals using the example of an isolated mark command.

For "laser active" operation:

- The LASERON signal is switched on
- 2 alternating modulation signals are outputted at the LASER1 output port and LASER2 output port. Their pulse length and period duration can be defined by **set_laser_pulses**, **set_laser_pulses_ctrl** or **set_laser_timing**.

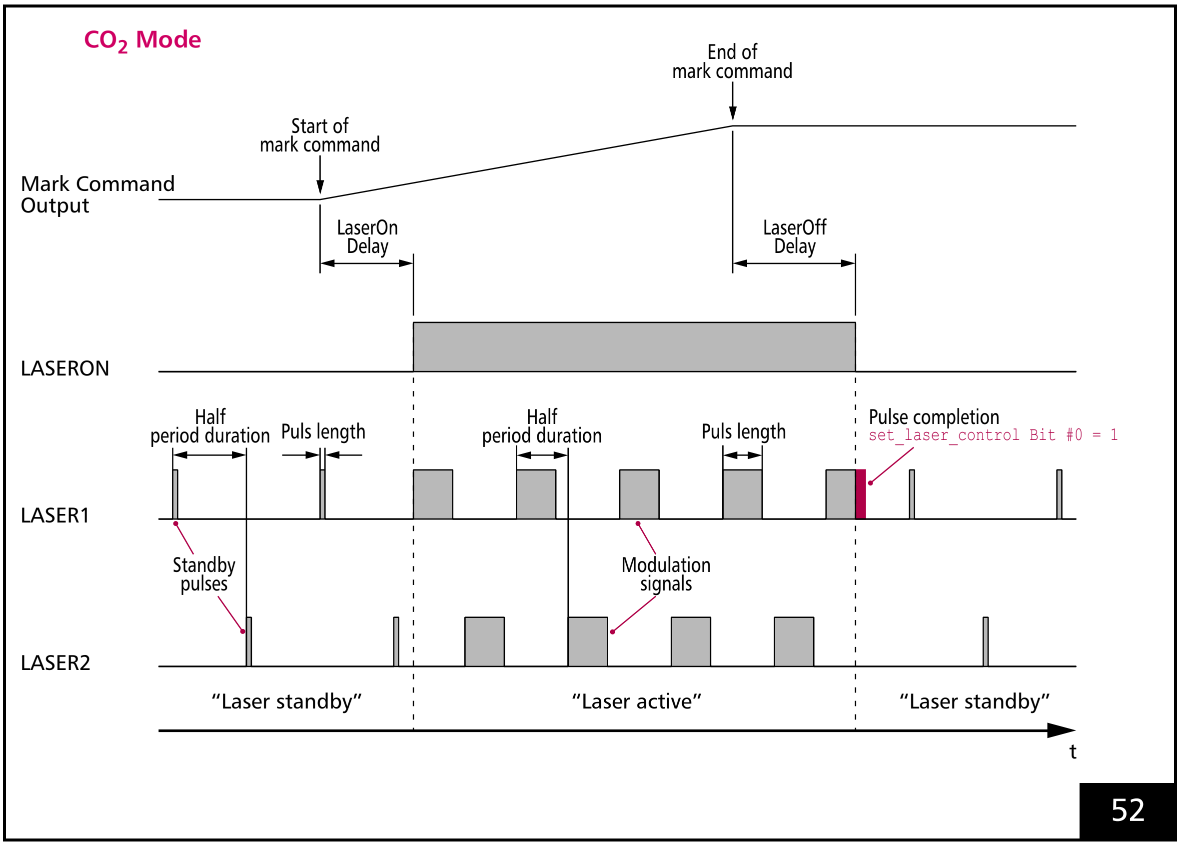
For "laser standby" operation:

- The LASERON signal is switched off
- Alternating standby pulses are outputted at the LASER1 output port and LASER2 output port. Their pulse length and period duration can be defined by **set_standby** or **set_standby_list**.

Notes

- LASER1 signals and LASER2 signals are activated by:
  `HalfPeriod` ≠ 0 and `PulseLength` ≠ 0
- LASER1 signals and LASER2 signals are deactivated by:
  `HalfPeriod` = 0 and/or `PulseLength` = 0
  (default setting after **load_program_file**)
- The LASER2 signal is phase-shifted by half a signal period in relation to the LASER1 signal. It can be used for the control of a second laser tube. By **set_laser_control** (Bit #1 = 1), both signals can be exchanged with each other. To control laser power, the pulse length of the LASER1 and LASER2 signals can be varied. Both signals share the same pulse lengths and periods.
- For the LASER1 signals and LASER2 signals, *half* of the output period must be specified.
- **set_laser_pulses** and **set_laser_timing** are Delayed Short List Commands. They can also be used to change the laser parameters between two Mark Commands.

  The time base for the signals is always:
  - 1/64 µs in RTC6 Standard Mode, see also Section "RTC6 Standard Mode", page 173
  - 1/8 µs in RTC4 Compatibility Mode, see also Section "RTC4 Compatibility Mode", page 173
- Pulse Completion, page 191 affects also LASER2 signals.



Timing diagram of the signals in CO_2 Mode (with active-HIGH Laser Control Signals).
Example: isolated mark command.

7.4.4 YAG Mode 1, YAG Mode 2, YAG Mode 3, YAG Mode 5

By **set_laser_mode**, a YAG Mode can be set:

YAG Mode	Call
YAG Mode 1	`set_laser_mode( Mode = 1 )`
YAG Mode 2	`set_laser_mode( Mode = 2 )`
YAG Mode 3	`set_laser_mode( Mode = 3 )`
YAG Mode 5	`set_laser_mode( Mode = 5 )`

The timing diagram, see Figure 53, shows the corresponding signals using the example of an isolated mark command.

In each YAG Mode, the following applies for “laser active” operation:

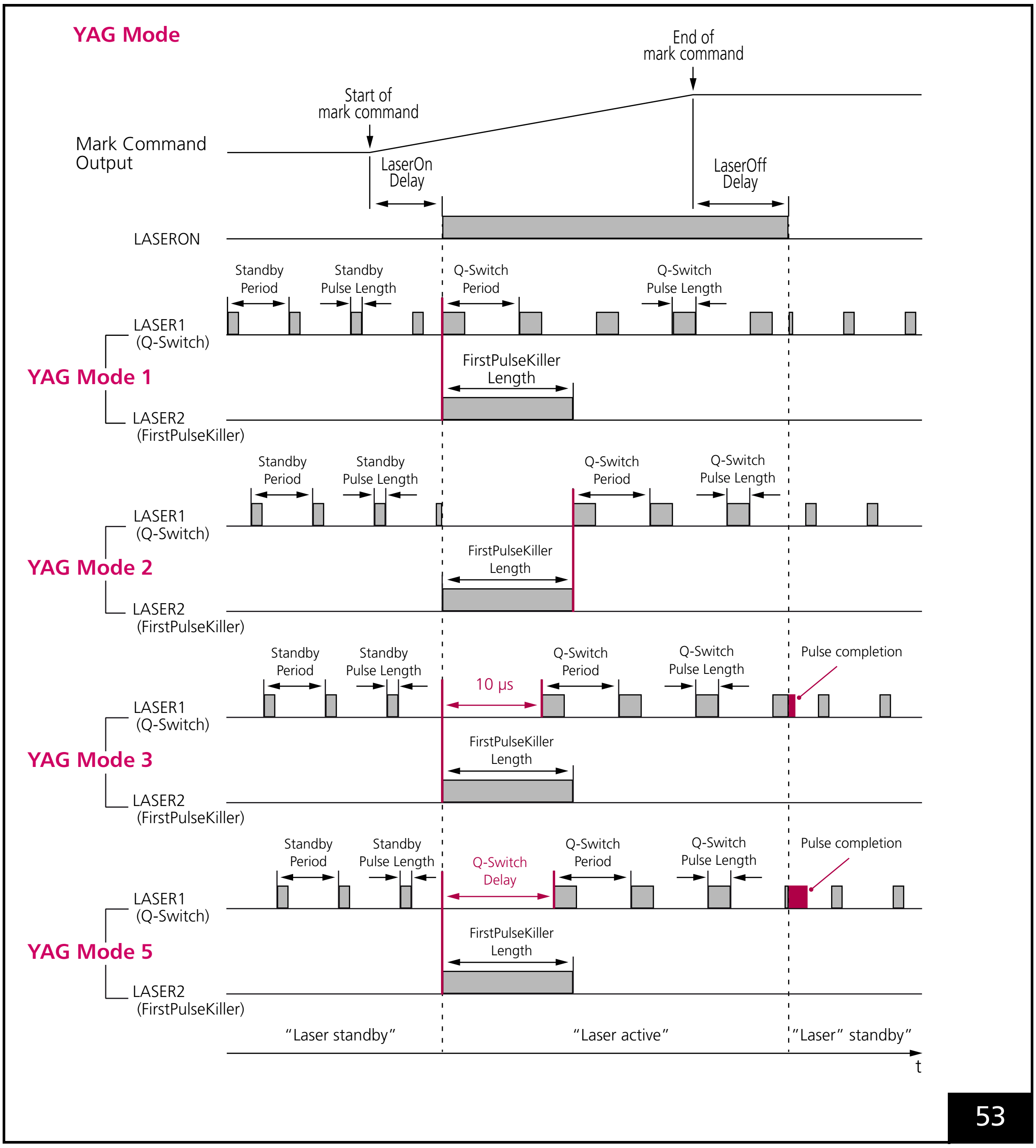
- The LASERON signal is switched on
- A Q-Switch Signal is outputted at the LASER1 output port, see Q-Switch Signal, page 197.
- A adjustable FirstPulseKiller signal, page 369 is outputted at the LASER2 output port, see also Section “FirstPulseKiller Signal”, page 197

In each YAG Mode, the following applies for “laser standby” operation:

- The LASERON signal is switched off
- Standby pulses are outputted at the LASER1 output port
  - Their pulse length and period duration can be set by **set_standby** or **set_standby_list**.

Notes

- LASER1 signals are activated by:
  `HalfPeriod` ≠ 0 and `PulseLength` ≠ 0
- LASER1 signals are deactivated by:
  `HalfPeriod` = 0 and/or `PulseLength` = 0
  (default setting after **load_program_file**)



Timing diagram of the signals in YAG Mode 1, YAG Mode 2, YAG Mode 3, YAG Mode 5 (with active-HIGH Laser Control Signals). Standby signals are not synchronized with the “laser-active” signals and can have arbitrary phase alignments. Example: isolated mark command.

Q-Switch Signal

The Q-Switch Signal serves to control the quality switch of the laser.

By **set_laser_pulses**, **set_laser_pulses_ctrl** or **set_laser_timing** are set:

- Q-Switch period duration
- Q-Switch pulse length

Notes

- See also Section ”Pulse Completion”, page 191 (Signals for “Laser Active” Operation).
- See also Section ”Pulse Completion”, page 191 of (Signals for “Laser Standby” Operation).

FirstPulseKiller Signal

The FirstPulseKiller signal, page 369 serves to signal the first laser pulses of a pulse sequence, if they do not correspond to the usual continuous power.

The FirstPulseKiller signal, page 369 is only provided. The laser itself must respond appropriately.

The FirstPulseKiller signal, page 369 is outputted automatically together with the LASERON signal.

The length of the FirstPulseKiller signal, page 369 is set with **set_firstpulse_killer** or **set_firstpulse_killer_list**.

See also **config_laser_signals**, FirstPulseKiller signal, page 369.

YAG Mode Differences

YAG Mode 1, YAG Mode 2, YAG Mode 3 and YAG Mode 5 only differ in the relative start time of the first Q-Switch pulse with reference to the FirstPulseKiller signal, see also the timing diagrams in Figure 53.

After the Q-Switch delay has expired, the first Q-Switch pulse starts. The Q-Switch delay value can be specified by **set_qswitch_delay** and **set_qswitch_delay_list** or alternatively by selecting the YAG Mode:

- YAG Mode 1: 0
- YAG Mode 2: FirstPulseKiller signal length
- YAG Mode 3: 10 µs
- YAG Mode 5: value from **set_qswitch_delay** or **set_qswitch_delay_list**

By **set_laser_mode**, the Q-Switch delay is merely *preset*. Therefore, in YAG Mode 1, YAG Mode 2, YAG Mode 3, the Q-Switch delay can be subsequently changed by **set_qswitch_delay** or **set_qswitch_delay_list**.
In YAG Mode 2, the Q-Switch delay is also adjusted accordingly with each **set_firstpulse_killer** or **set_firstpulse_killer_list**.

Lamp Current (Laser Power)

To control the lamp current of a YAG laser, 12-Bit Analog Output Port 1 ANALOG OUT1 or 12-Bit Analog Output Port 2 ANALOG OUT2 can be used. They are available at the LASER Connector, see Figure 17. 12-Bit Analog Output Port 2 ANALOG OUT2 is also available at the MARKING ON THE FLY Socket Connector, see Figure 25.

To define the analog output signal, the control command **write_da_x** and the Delayed Short List Command **write_da_x_list** are provided.

Alternatively, the lamp current can be controlled digitally by the 8-Bit Digital Output Port (EXTENSION 2 Socket Connector), see also Section ”8-Bit Digital Output Port”, page 84.
For setting the 8-Bit Digital Output Port, **write_8bit_port**, **write_8bit_port_list** or **write_port_list** are available.

When the lamp current is changed, list execution can be halted by **long_delay** until a constant laser power has been achieved.

7.4.5 Laser Mode 4

`set_laser_mode( Mode = 4 )` sets Laser Mode 4.

The timing diagram, see Figure 54, shows the corresponding signals using the example of an isolated mark command.

At LASER1 output port, for “laser active” operation as well as for “laser standby” operation standby pulses are outputted continuously.
Pulse length and period duration of the standby pulses can be set by **set_standby** or **set_standby_list**.

For “laser active” operation, the following applies:

- The LASERON signal is switched on
- A programmable FirstPulseKiller signal, page 369 is outputted at the LASER2 output port, see also Section ”FirstPulseKiller Signal”, page 198

For “laser standby” operation, the following applies:

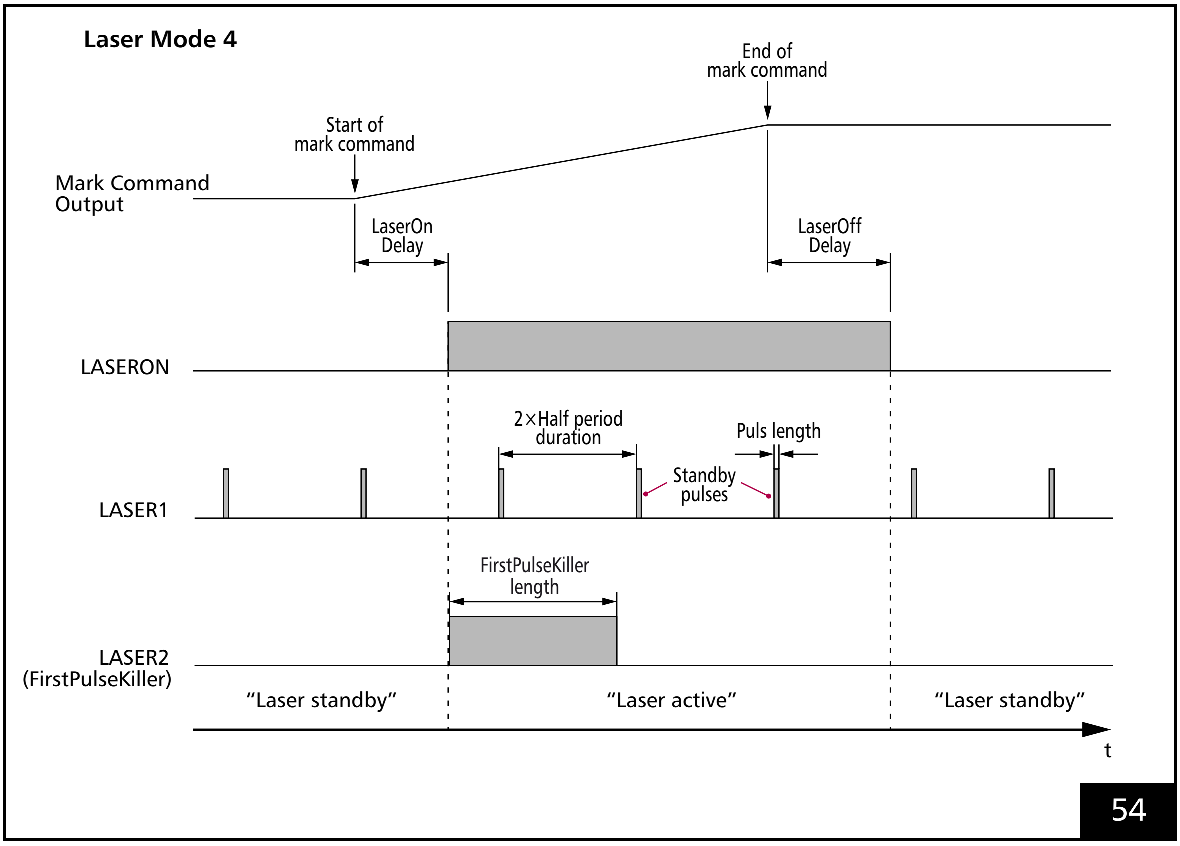
- The LASERON signal is switched off
- The LASER2 signal is switched off

FirstPulseKiller Signal

- Like Section ”FirstPulseKiller Signal”, page 197.

Notes

- LASER1 signals are activated by:
  `HalfPeriod` ≠ 0 and `PulseLength` ≠ 0
- LASER1 signals are deactivated by:
  `HalfPeriod` = 0 and/or `PulseLength` = 0
  (default setting after **load_program_file**)
- Laser Mode 4 is used for some fiber lasers.



Timing diagram of the signals in Laser Mode 4 (with active-HIGH Laser Control Signals).
Example: isolated mark command.

7.4.6 Laser Mode 6

`set_laser_mode( Mode = 6 )` sets Laser Mode 6.

Laser Mode 6 is like Laser Mode 4 and is provided for certain free-running lasers whose gate signals (LASERON) must *not* be changed during the duration of a pulse.

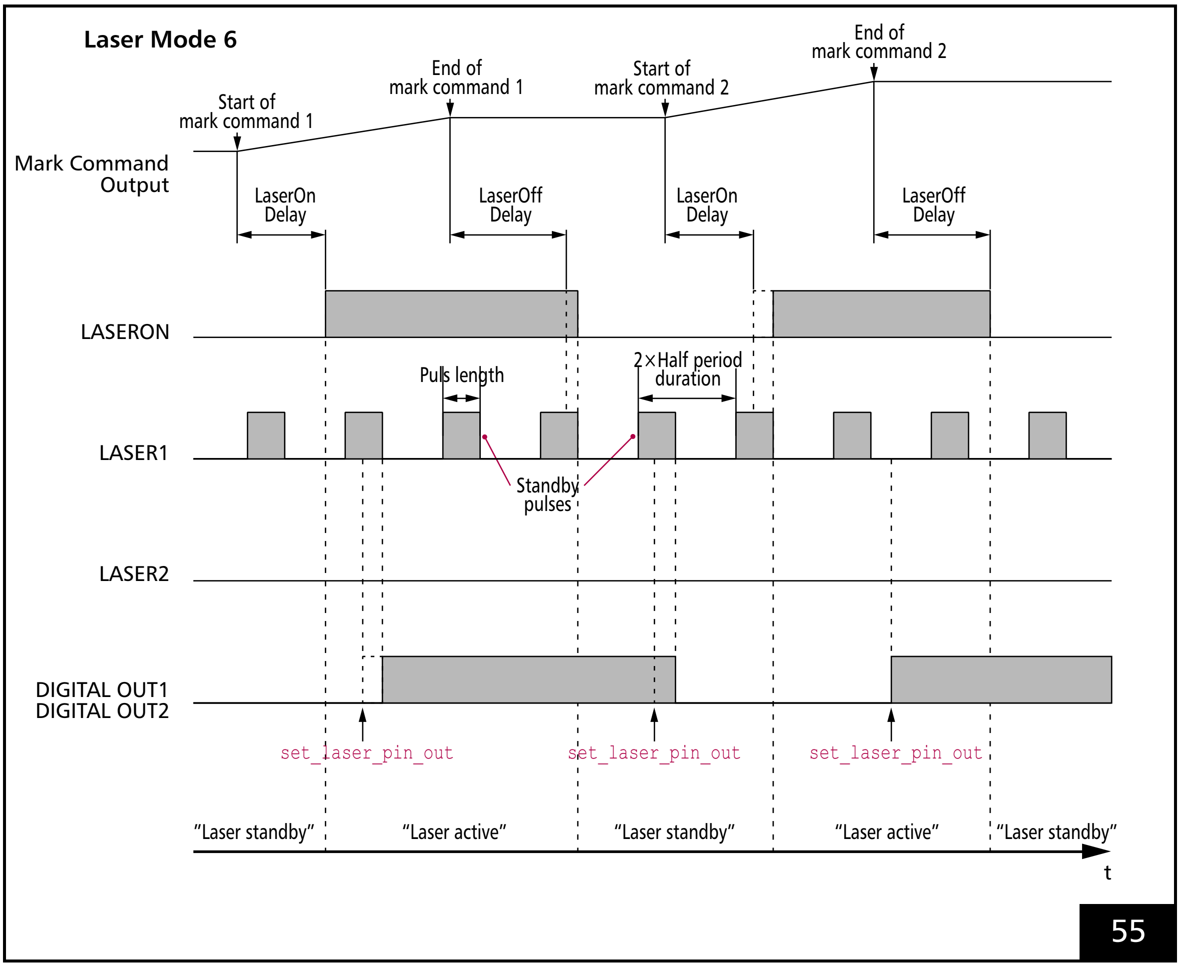
The timing diagram, see Figure 55, shows the corresponding signals using the example of an isolated mark command.

Laser Mode 6 signals and Laser Mode 4 signals are the same (see also Notes there, page 198) with the following exception:

- As long as a standby pulse is active, switching of the LASERON signal is delayed accordingly. LASERON switches 5/64 μs after the end of the standby pulse.

Notes

- No LASER2 signal is outputted.
- The delay of the switching time when a standby pulse is still active is also valid for switching the output values at the 2-Bit Digital Output Port by **set_laser_pin_out**, **set_laser_pin_out_list** or **write_port_list**, see Figure 55.



Timing diagram of the signals in Laser Mode 6 (with active-HIGH Laser Control Signals).
Example: isolated mark commands.

7.4.7 Softstart Mode (not yet implemented)

Not implemented yet.

7.4.8 Pulse Picking Laser Mode

Pulse Picking Laser Mode can be set by **set_pulse_picking** or **set_pulse_picking_list**.

The timing diagram, see Figure 56, shows the corresponding signals using the example of an isolated mark command.

For "laser active" operation, the following applies:

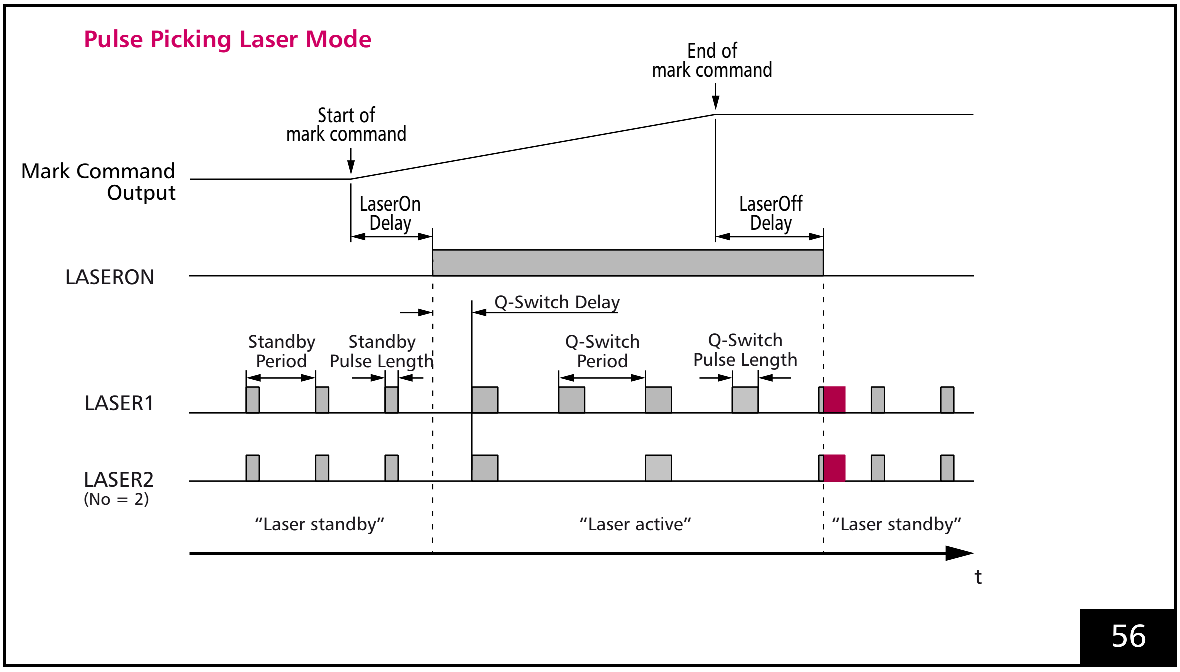
- The LASERON signal is switched on
- A modulation signal is provided at the LASER1 output port
  - Its pulse length and period duration can be set by **set_laser_pulses**, **set_laser_pulses_ctrl** or **set_laser_timing**
- At the LASER2 output port every `No`th pulse of the signals is outputted at the LASER1 output port (in phase)
  - `No` is set with **set_pulse_picking** or **set_pulse_picking_list**

For "laser standby" operation, the following applies:

- The LASERON signal is switched off
- Standby pulses are provided in phase at LASER1 output port and LASER2 output port
  - Their pulse length and period duration can be set by **set_standby** or **set_standby_list**
  - Standby pulses cannot be "pulse-picked"

Notes

- With **set_laser_control**(Bit #7 = 1) the Pulse Picking signal at LASER2 can be set to a constant length independent of the LASER1 signal, which is set with **set_pulse_picking_length**.
- **set_pulse_picking** and **set_pulse_picking_list** overwrite a laser mode previously set with **set_laser_mode**. Vice versa, **set_laser_mode** switches off the Pulse Picking Laser Mode.
- For the LASER1 signals, half the period duration must be specified.
- A Q-Switch delay is effective, see also Section "YAG Mode Differences", page 197.
- A FirstPulseKiller signal, page 369 is not outputted.
- The setting sequence of the LASER1 signals and the Pulse Picking Laser Mode is irrelevant.



Timing diagram of the signals in Pulse Picking Laser Mode (with active-HIGH Laser Control Signals and `No` = 2). Example: isolated mark command.

7.4.9 “Automatic Laser Control“

By **set_auto_laser_control**, automatic readjustment of the Signals for “Laser Active” Operation – even dynamically during execution of Mark Commands can be achieved:

- `Ctrl` parameter determines the output port
- `Value` parameter determines the output value for 100% power at the target (set) mark speed at the Image Field center
- Remaining parameters see command description

This can be used to compensate for variations in laser energy input caused by:

- Changing power density
- Changing Spot size
- Changing processing speed

For “Automatic Laser Control“, a position-, speed- or vector-controlled correction of the Laser Control Signals can be activated and combined:

- See Table “Automatic Laser Control“ Type, page 206

Selectively, “Automatic Laser Control“ adjusts one of the following signal parameters:

- 12-bit output value at the ANALOG OUT1 output port or ANALOG OUT2 output port of the LASER Connector, see also Section ”12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2”, page 77
- Output value at the 16-Bit Digital Output Port of the EXTENSION 1 Socket Connector, see also Section ”16-Bit Digital Input Port and 16-Bit Digital Output Port”, page 81
- Output value at the 8-Bit Digital Output Port of the EXTENSION 2 Socket Connector, see also Section ”8-Bit Digital Output Port”, page 84
- Pulse length (`PulseLength`) or output period (`HalfPeriod`) of the Laser Control Signals LASER1 and LASER2

The automatically readjusted value can be recorded by **set_trigger[*]** (`Signal n` = 24).

Notes

- “Automatic Laser Control“ does *not* compensate for an explicit change in mark speed between two Mark Commands caused by a **set_mark_speed** call.
- “Automatic Laser Control“ and Pixel Mode should not affect the same output port at the same time.
- “Automatic Laser Control“ cannot be combined with variable laser control via the McBSP interface (see **set_multi_mcbsp_in**).

“Automatic Laser Control” Type	Purpose
Position-Dependent Laser Control, page 207	Allows compensation of radial laser energy input variations during execution of Mark Commands. Such variations may be caused, for example, by a objective edge diminution or different incidence angles of the laser beam.
Speed-Dependent Laser Control, page 209	Allows compensation of laser energy input variations during execution of Mark Commands that resulting from an uneven galvanometer scanner motion (acceleration phase and deceleration phase, time-dependent Mark Commands). In addition, the galvanometer scanner speed can also be transformed back into a mark speed depending on the position. Speed-Dependent Laser Control and Encoder-Speed-Dependent Laser Control can be combined, if both galvanometer scanner and workpiece move simultaneously.
Encoder-Speed-Dependent Laser Control, page 212	Allows the laser energy input to be controlled based on the currently present encoder speed. Speed-Dependent Laser Control and Encoder-Speed-Dependent Laser Control can be combined, if both galvanometer scanner and workpiece move simultaneously.
Vector-Defined Laser Control, page 214	Allows linear readjustment of a signal parameter along parameterized mark vectors or jump vectors (see Section “[*]Para[*] Commands”, page 142). This signal parameter can be combined with the laser power control if the control parameter `Ctrl` matches `Ctrl` from **set_auto_laser_control** (the current parameter Para dynamically replaces the 100% value from **set_auto_laser_control**) or as an independent control (for example, as defocus).
2D Position-Dependent Laser Control, page 215	With 2D Position-Dependent Laser Control, laser energy input is varied in 2D based on a table of scaling factor values.

General Notes

- Each individual contribution as well as the total correction cannot exceed a factor of 4.0 (clipping[1]).
- If laser power and/or energy input into the to-be-processed material is not strictly proportional to the output values of the selected signal parameter, then **load_auto_laser_control** can be used to load a nonlinearity curve that defines this (application-specific) relationship, see Section "Loading and Determining the Nonlinearity Curve", page 212.
- In addition, the selected signal parameter can neither exceed the value range allowed with **set_auto_laser_control** (parameter `MinValue` and `MaxValue`) nor the value range allowed for the respective output port. It is clipped correspondingly.
- For the laser control signal a corresponding default value is outputted (see **set_port_default** or **set_laser_off_default**), if:
  – Signals for "Laser Active" Operation are switched off when "Automatic Laser Control" is active
  – "Automatic Laser Control" itself is deactivated

  If no default value has been explicitly defined, the permitted maximum value is outputted. As substitute for the control parameters `HalfPeriod` and `PulseLength`, the parameter `Value` (the 100% value) from **set_auto_laser_control** is used.
- Once an "Automatic Laser Control" has been switched on with **set_auto_laser_control**, then the following can be changed subsequently by **set_auto_laser_params** or **set_auto_laser_params_list**:
  – the signal parameter
  – the 100% value
  – limit values assigned to it

  The `Mode` parameter cannot be changed subsequently.

Position-Dependent Laser Control

To activate the Position-Dependent Laser Control for the output port specified by **set_auto_laser_control**, a user-defined scaling function must be loaded by **load_position_control**, see Section "Notes on Loading a Scaling Function", page 207. This scaling function represents a scaling factor as a function of the distance to the center of the Image Field.

After **load_program_file**, it is initialized for all distances with "factor 1.0". The "position-dependent" laser control is de facto deactivated by that. The table can be saved by **create_dat_file**.

When calculating the correction for Position-Dependent Laser Control, the current Cartesian control values are used as a basis:

- Including wobbel correction (#2 in Chapter 7.3.6 "Output Values to the Scan System", page 186)
- Including coordinate transformation in the virtual Image Field (#3)
- Including Processing-on-the-fly correction (#4)
- Excluding head-specific coordinate transformation (#5a and #5b)
- Excluding Image Field correction (#7)

Notes on Loading a Scaling Function

- For the *Scale(Position)* scaling function, **load_position_control** loads a table from an ASCII text file.
- The ASCII text file can contain one or several tables.[2]
- Each table can contain up to 50 data points (*Position* | *Scale(Position)*).
- The *Scale(Position)* function is linearly interpolated from the data points.
- 2D Position-Dependent Laser Control and Position-Dependent Laser Control cannot be combined. Loading a user-defined ASCII text file for a 2D Position-Dependent Laser Control deactivates Position-Dependent Laser Control and vice versa.

(1) < DLL 612: overflow.

(2) Even of another type, see table 1, page 156.

For the scaling function tables, the following rules apply:

- Each table must begin with the line:
  [PositionCtrlTable<No>]
  <No> represents the table number.
- If the table contains multiple [PositionCtrlTable<No>] entries with the same <No>, then only the lines after the first entry are used. Only lines up to the next '[' character (that is not preceded by a semicolon) are used.
- Each data point (*Position* | *Scale* (*Position*)) is defined as follows:
  ```
  Position<n> = <Value>
  Scale<n> = <Value>
  ```
  where <n> corresponds to the index (1 ≤ <n> ≤ 50) of the data point. The values <Value> can be specified as (unsigned) floating point numbers. Decimal separator: decimal point (.).
- If the table contains multiple data points with the same Index <n>, then the most recently read one is used.
- If the table contains multiple data points with the same position value *Position*, then the data point with the largest Index <n> is used. Equality is checked to within ±0.01.
- The position value is specified radially as the distance between the to-be-marked point and the coordinate midpoint ($= (x^2 + y^2)^{1/2}$) as percent of half the image-field side length.
  Example: (X_{max} | 0) corresponds to 100%, (X_{max} | Y_{max}) corresponds to $2^{1/2} \times 100\%$.
- For <Value>, the following ranges apply:
  0.0 ≤ *Position* ≤ 150.0 and
  0.0 ≤ *Scale* (*Position*) ≤ 4.0.
- Each instruction must be in a separate line.
- Spaces and tabs in a line (for example, between '=' and <Value>) are ignored.
- Empty lines are ignored.
- Data points with invalid values are ignored.
- The data point of a particular index <n> is ignored if the corresponding Position<n> and/or Scale<n> definition is missing.
- The semicolon ';' can be used for comments. All characters in a line following a semicolon are ignored.
- The instructions for data points in the table can be ordered as desired.
- Indices for data point pairs in the table can be selected as desired within the range [1…50] (the table is then automatically sorted by ascending position values).
- If the table contains *no* valid data point, **load_position_control** has no effect (return value 1 or 13).
- If there is no entry for *Position* = 0.0, then an entry with Scale = Min(Scale<i>) is inserted (the smallest allowed value defined in the table is used for lower positions values). Likewise for *Position* = 150.0 with Max(Scale<i>).
- If the selected text file only contains a single valid data point with Scale<n> = S, then (for the entire position range) the scaling function *Scale* (*Position*) = S is loaded. The correction has a multiplicative effect on the laser control signal. For S = 1.0, position-dependent correction is therefore switched off. Alternatively, this can also be achieved with Name = NULL in **load_position_control**.
- The table can be saved by **create_dat_file**.

Speed-Dependent Laser Control

When activating the Speed-Dependent Laser Control for the output port specified by **set_auto_laser_control**, the `Mode` parameter is used to select which input parameters are to be used for calculating the correction.

As reference value for the 100% speed, always the set (target) mark speed (set by **set_mark_speed** or **set_mark_speed_ctrl**) applies (its change is never compensated by the "Automatic Laser Control").

- `Mode` = 1 is intended (for consistency reasons) especially for analog scan systems. The current Microstep length per 10 μs is used as input. It may deviate from the target speed due to the 10 μs clock pulse rounding or with **[*]timed[*]** commands. Variations in the acceleration and deceleration phases cannot be compensated.
- `Mode` = 2 is intended as basic mode for iDRIVE Scan Systems. It can be combined with other special corrections (see below).
- Extensions to `Mode` = 2 (any combination possible):
  - +0
    Basic mode for iDRIVE Scan Systems
  - +4
    Combines the speeds from `Mode` = 2 and `Mode` = 5 to a total speed. The 100% reference speed from `Mode` = 5 remains unconsidered. Instead, the encoder speeds are converted into galvanometer scanner bit speeds with the fly scaling factors and then vectorially added to the galvanometer scanner speed. A corresponding Processing-on-the-fly Session must be active.
  - +16
    Basic mode for SCANahead Systems
  - +32
    Position-dependent correction of the galvanometer scanner speed (in angular units) in a one for the Image Field ("inverse speed correction table")

Notes

- Usually **set_auto_laser_control**(`Mode` = 2) requires a special intelliSCAN firmware, which returns an actual speed corrected by the signal runtimes between the RTC6 PCIe Board and the scan head. If you have any questions as to whether your intelliSCAN is equipped with it or is upgradeable, contact SCANLAB.
- Usually a combination of Speed-Dependent Laser Control and a negative LaserOn Delay does not make sense.

"Spot Distance Control"

The "Spot Distance Control" functionality is only available for SCANahead Systems.
Also refer to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.

With **set_auto_laser_control**(`Ctrl` = 7) ("Spot Distance Control"), temporal output pulses are controlled to a geometrically constant pulse distance. This pulse distance is defined by **spot_distance** or **spot_distance_ctrl**. The parameters `Value`, `MinValue` and `MaxValue` have no meaning with the "Spot Distance Control" functionality.

The speed correction occurs with `Mode` = 2 + 16 + (optionally) 32, (see above).(1)
"Spot Distance Control" cannot be combined with `Mode` = 1, 2 or 5.

The correction is more precise and dynamic than the comparable control with `Ctrl` = 5 (HalfPeriod). For successful use, a laser with "pulse-on-demand" functionality should be used that outputs laser pulses immediately when triggered by the LASER1 signal. Frequency modulations are set with an accuracy of 1/64 μs for the pulse distance.

A comparison of a typical marking result with "Automatic Laser Control" (`Ctrl` = 7) and without "Automatic Laser Control" is shown in Figure 57.

As an alternative to **set_auto_laser_control** with `Ctrl` = 7, the marking shown in Figure 58 could also be carried out with Sky Writing switched on. However, when using **set_auto_laser_control** with `Ctrl` = 7 the process time is often shorter.

Notes

- **set_auto_laser_control**(`Ctrl` = 7) ("Spot Distance Control") *cannot* be combined with:
  - Vector-Defined Laser Control by **set_vector_control** (`Ctrl` = 5) (`HalfPeriod`)
  - Laser Mode 4(2)
  - Laser Mode 6(3)
- **set_trigger[*]** (signal 24) records an internal control signal but *not* `HalfPeriod`.

(1) See page 662: "As a result, **control_command** is unavailable for the Connector for First Scan Head as long as this `Mode` selection is in effect (because the current speed is utilized – which is transmitted by the scan system)."

(2) In Laser Mode 4, only a standby signal is outputted that cannot be varied.

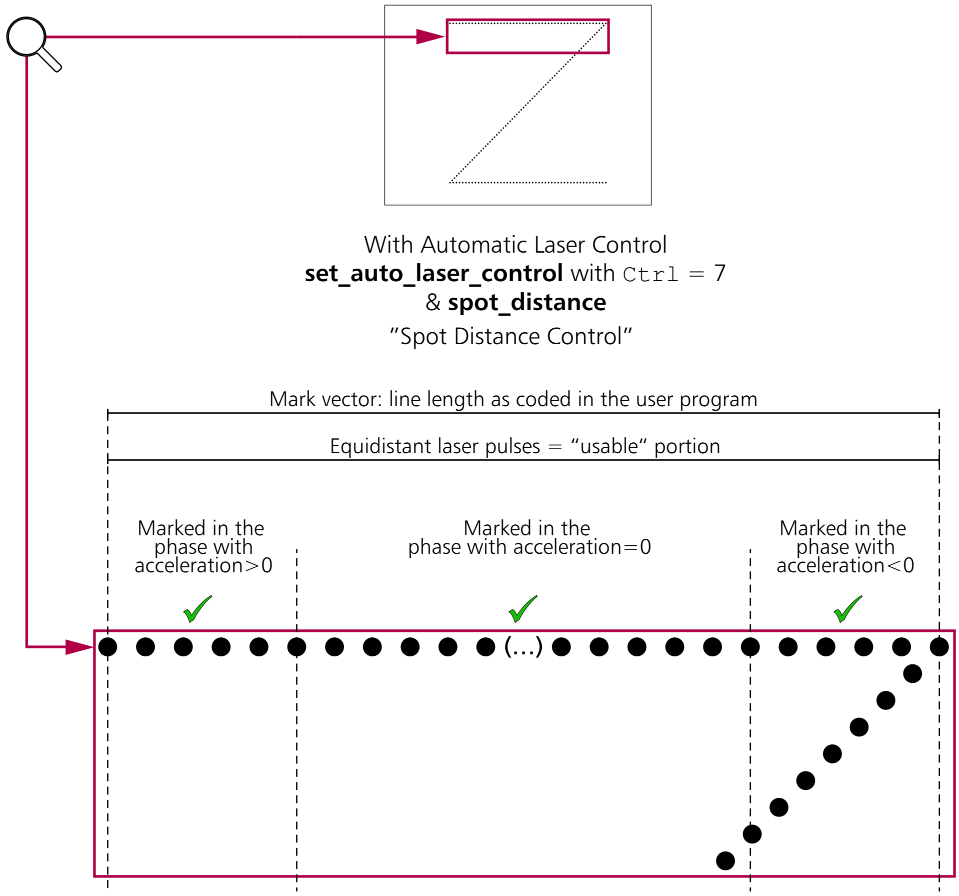
(3) In Laser Mode 6, only a standby signal is outputted that cannot be varied.

Legend

1. With **set_auto_laser_control** with `Ctrl` = 7 and with equidistant geometric pulse spacing specified at **spot_distance**/**spot_distance_ctrl**.
2. No "Automatic Laser Control" is active (spot spacing may vary).

57

Example marking result: "sharp" corner.



58

For **set_auto_laser_control** with `Ctrl` = 7 the marking result shows *largely* equidistant spots as long as the acceleration does not change too much within a 10 μs cycle.

Encoder-Speed-Dependent Laser Control

set_auto_laser_control(`Mode` = 5) is intended for a pure encoder speed-dependent correction, if only the workpiece moves and the galvanometer scanners (ideally) are idle.

The 100% reference speed is defined by **set_encoder_speed_ctrl** or **set_encoder_speed**. Processing-on-the-fly should not be active at the same time.

For a combination of galvanometer scanner speeds and encoder speeds in a Processing-on-the-fly Session, see `Mode` = 6 = 2 + 4.

Loading and Determining the Nonlinearity Curve

- For the *Scale(Percent)* nonlinearity curve, **load_auto_laser_control** loads a table from an ASCII text file.
- The ASCII text file can contain one or several tables.(1)
- Each table can contain up to 50 data points *(Percent | Scale(Percent))*.
- The *Scale(Percent)* function is linearly interpolated from the data points.

For the tables, the following rules apply:

- Each table must begin with the line:
  `[AutoLaserCtrlTable<No>]`
  `<No>` represents the table number.
- If the table contains several `[AutoLaserCtrlTable<No>]` entries with the same `<No>`, then only the lines after the first entry are used. Only lines up to the next '[' character (that is not preceded by a semicolon) are used.
- Each data point (*Percent* | *Scale*(*Percent*)) is defined as follows:
  `Percent<n> = <Value>`
  `Scale<n> = <Value>`
  where `<n>` corresponds to the index (1 ≤ `<n>` ≤ 50) of the data point. The values `<Value>` can be specified as (unsigned) floating point numbers. Decimal separator: decimal point (.).
- If the table contains multiple data points with the same Index `<n>`, then the most recently read one is used.
- If the table contains multiple data points with the same percent value *Percent*, then the data point with the largest Index `<n>` is used. Equality is checked to within ±0.01°.

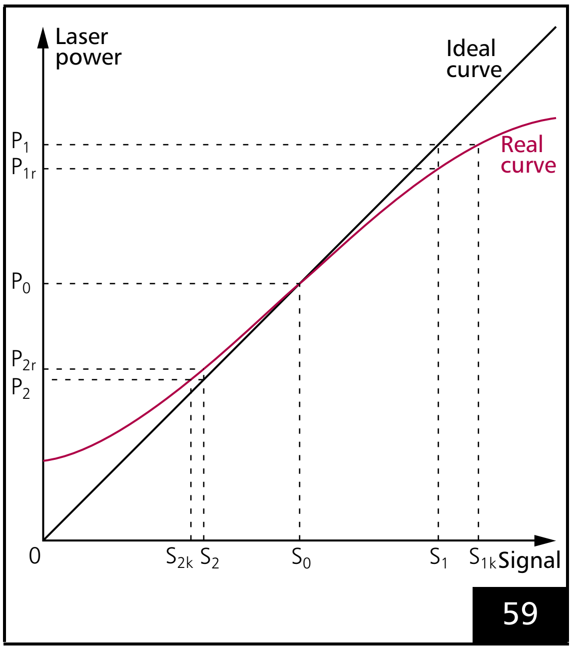
(1) Even of another type, see table 1, page 156.

- The percent value is relative to the 100% value from **set_auto_laser_control** (Parameter `Value`) or dynamically from a Vector-Defined Laser Control.
  In the following example, a nonlinearity factor of 1.2 is set for a 1.5x multiple of the set value:
  `Percent<n> = 150`
  `Scale<n> = 1.2`
- For `<Value>`, the following ranges apply
  $0.0 \le Percent \le 400.0$ and
  $0.0 \le Scale(Percent) \le 4.0$.
- Each instruction must be in a separate line.
- Spaces and tabs in a line (for example, between '=' and `<Value>`) are ignored.
- Empty lines are ignored.
- Data points with invalid values are ignored.
- The data point of a particular index `<n>` is ignored if the corresponding `Percent<n>` and/or `Scale<n>` definition is missing.
- The semicolon ';' can be used for comments. All characters in a line following a semicolon are ignored.
- The instructions for data points in the table can be ordered as desired.
- Indices for data point pairs in the table can be selected as desired within the range [1…50] (the table is then automatically sorted by ascending percent values).
- If the table contains *no* valid data point, then **load_auto_laser_control** has no effect (return value 1 or 13).
- If there is no entry for *Percent* = 0.0, then an entry with `Scale` = Min(`Scale<i>`) is inserted (the smallest allowed value defined in the table is used for lower percent values). Likewise for *Percent* = 400.0 with Max(`Scale<i>`).

After **load_program_file** this function is initialized for all percentage values with "Factor 1.0", the nonlinear laser control is "deactivated". Alternatively, this can also be achieved with `Name` = NULL in **load_auto_laser_control**.

The table can be saved by **create_dat_file**.

The example diagram in Figure 59 illustrates how the nonlinearity curve can be determined.



Laser power progression – example of determining a nonlinearity curve.

The straight line in the diagram describes an ideal relationship between laser power and the laser control signal parameter (here, the term laser power also represents the pulse frequency = 0.5/`LaserHalfPeriod`), the curved line simulates a realistic relationship.

S_0 is the signal parameter value defined as the set value and P_0 is the associated laser power. At point ($S_0 \mid P_0$) (this corresponds to the data point `Percent0` = 100, `Scale0` = 1.0) the two curves are normalized to each other. The combination of a nonlinearity curve with a Vector-Defined Laser Control is therefore, generally not recommended.

An increase (decrease) of the signal parameter to S_1 (S_2) results in an ideal laser power P_1 (P_2) and a real laser power P_{1r} (P_{2r}). For the actually desired laser power P_1 (P_2), a corrective signal parameter value S_{1k} (S_{2k}) is needed. The following two value pairs are then to be entered as data points for the nonlinearity curve:

`Percent1` = $S_1/S_0 \times 100 = P_1/P_0 \times 100$
`Scale1` = S_{1k}/S_1

`Percent2` = $S_2/S_0 \times 100 = P_2/P_0 \times 100$
`Scale2` = S_{2k}/S_2

Vector-Defined Laser Control

Vector-Defined Laser Control allows a a signal parameter to be varied linearly along a mark vector or jump vector.

To initialize Vector-Defined Laser Control, **set_vector_control** must be used to specify which signal parameter is to be varied with which initial value (parameters `Ctrl` and `Value`).

Then the signal parameter is varied linearly along a parameterized mark or jump vector. The end value is automatically the start value for a subsequent [*]para[*] Command.

Notes

- List commands for explicitly changing the signal parameter output value are temporarily effective or not at all.
- [*]para[*] Commands always use the end value of the previous [*]para[*] Command as their start value.
  Control commands that write to the same output port should be avoided while processing a list of [*]para[*] Commands.
- If the same output port is selected via **set_auto_laser_control** for `Ctrl`, the control parameter from the [*]para[*] Commands acts as 100% value for the laser control.
  Special care should be taken with **set_vector_control** (`Ctrl` = 7) (Defocus): The setting of the signal value is always done immediately. This can lead to Hard Jumps on the vario*SCAN*.
- During execution of [*]para[*] Commands, the Sky Writing mode, see Chapter 7.2.4 "Sky Writing", page 163, is *not* taken into account (but also not deactivated).

2D Position-Dependent Laser Control

With 2D Position-Dependent Laser Control, laser energy input is varied in 2D based on a table of scaling factor values.

To activate the 2D Position-Dependent Laser Control for the output port specified by **set_auto_laser_control**, a user-defined ASCII text file must be loaded by **load_position_control_2d_ctrl**, see:

- Requirements for the User-Defined ASCII Text File
- Notes on Loading the User-Defined ASCII Text File

When calculating the correction for 2D Position-Dependent Laser Control, the current Cartesian control values are used as a basis:

- Including wobbel correction (#2 in Chapter 7.3.6 "Output Values to the Scan System", page 186)
- Including coordinate transformation in the virtual Image Field (#3)
- Including Processing-on-the-fly correction (#4)
- Excluding head-specific coordinate transformation (#5a and #5b)
- Excluding Image Field correction (#6, #7)

At runtime, the "Automatic Laser Control" output value is multiplied by the calculated(1) scaling factor.

Notes on Loading the User-Defined ASCII Text File

- **load_position_control_2d_ctrl** loads 257 × 257 scaling factor values from a user-defined ASCII text file.
- Comment lines are ignored when reading in.
- Scaling factor values outside the allowed value range are clipped.
- The RTC6 board checks whether "K = value" (see below) and the value from the correction table assigned to the scan head 1 match, see `get_head_para( HeadNo = 1, ParaNo = 1 )`.
- 2D Position-Dependent Laser Control and Position-Dependent Laser Control cannot be combined. Loading a user-defined ASCII text file for a 2D Position-Dependent Laser Control deactivates Position-Dependent Laser Control and vice versa.

Requirements for the User-Defined ASCII Text File

- End of line format: \n or \r\n
- 1 own row with the calibration factor (anywhere in the file).
  Form: "K = value".
  Example: K = 10000.
- 257 rows with 257 columns each (= 66,049 grid points) represent the actual table. Only 1 table may be present.(2)
  – Each grid point is a scaling factor value. Allowed value range: 0.0000… 4.0000
  – Decimal separator: decimal point (.)
  – Column separator: space ( )
  – Maximum line length: 2000 characters.
- 0…n comment lines. Start: "//" or ";"

(1) Scaling factor calculation occurs – analogous to the image field correction algorithm – by bilinear interpolation of grid points contained in the user-defined ASCII text file, see Requirements for the User-Defined ASCII Text File.

(2) In contrast to other table types, see table 1, page 156.

7.4.10 Synchronization of the RTC6 Clock Cycle and an External Clock Signal

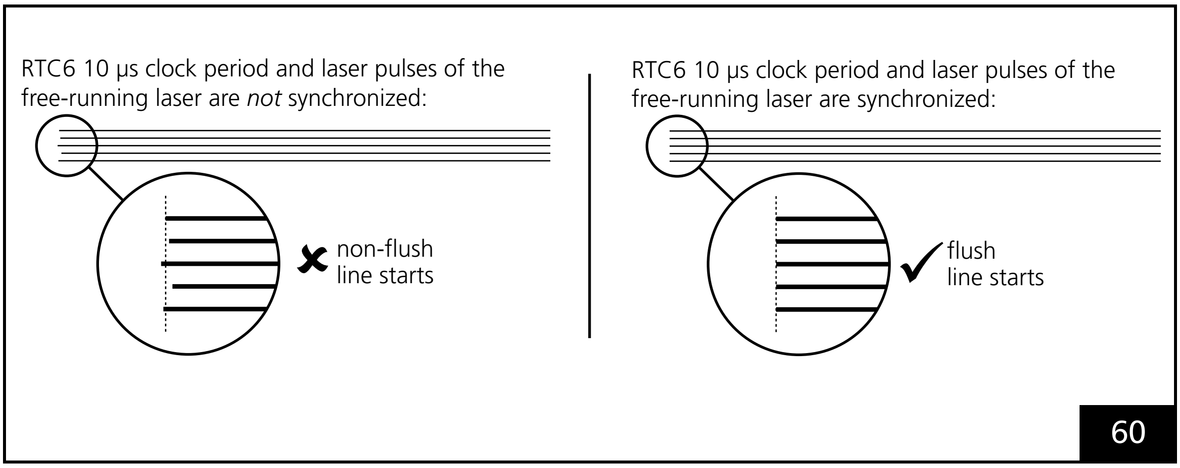
The laser pulse signals of a free-running laser and the laser control of the RTC6 PCIe Board can be synchronized. Non-flush line starts are thereby avoided, see Figure 60.

Here, the RTC6 PCIe Board accepts the external clock signal as the master clock. It must be supplied at the digital input port DIGITAL IN1 of the LASER Connector.

The synchronization is switched on and off by Bit #6 of **set_laser_control**. By Bit #5, it can be set whether to use the rising or falling edge of the external clock for synchronization. This automatically synchronizes the RTC6-internal Laser Control Signals to the external clock signals.

Notes

- The frequency of the master clock must meet the following requirements:
  $f = k \times 100$ kHz ($k \leq 64$) and
  $f = 64$ MHz / n (k and n integer).
  Therefore, only the following frequencies can be used: 100 kHz, 200 kHz, 400 kHz, 500 kHz, 800 kHz, 1 MHz, 1.6 MHz, 2 MHz, 3.2 MHz, 4 MHz, 6.4 MHz. The allowed external frequency deviation from an integer multiple of 100 kHz is ±15.625 ns per 10 μs.
- The supplied clock signal must be a TTL signal. The minimum pulse length or pulse pause of the clock signal should be 80 ns. See also Chapter 4.6.1 "LASER Connector", page 75, Section "2-Bit Digital Input Port", page 77.
- No synchronization is carried-out, if the synchronization is activated but there is no valid clock signal at DIGITAL IN1 of the LASER-connector. In this case the RTC6 PCIe Board uses its internal 10 μs clock cycle.
- If the synchronization is activated by **set_laser_control** by setting Bit #6, then the synchronization of RTC6 PCIe Board and external clock is subtly (depending on the respective phase position within 1.2 ms at most).



Free-running laser and RTC6 PCIe Board – example of marked lines.

7.4.11 Pulse Synchronization Mode

The Pulse Synchronization Mode:

- Delays the output of a LASER1 pulse until a new clock signal pulse of the laser at DIGITAL IN1 pin[1] is detected, see also Figure 61.
- Is enabled by **set_laser_pulse_sync**(Mode > 0)
- Is disabled by **set_laser_pulse_sync**(Mode = 0)

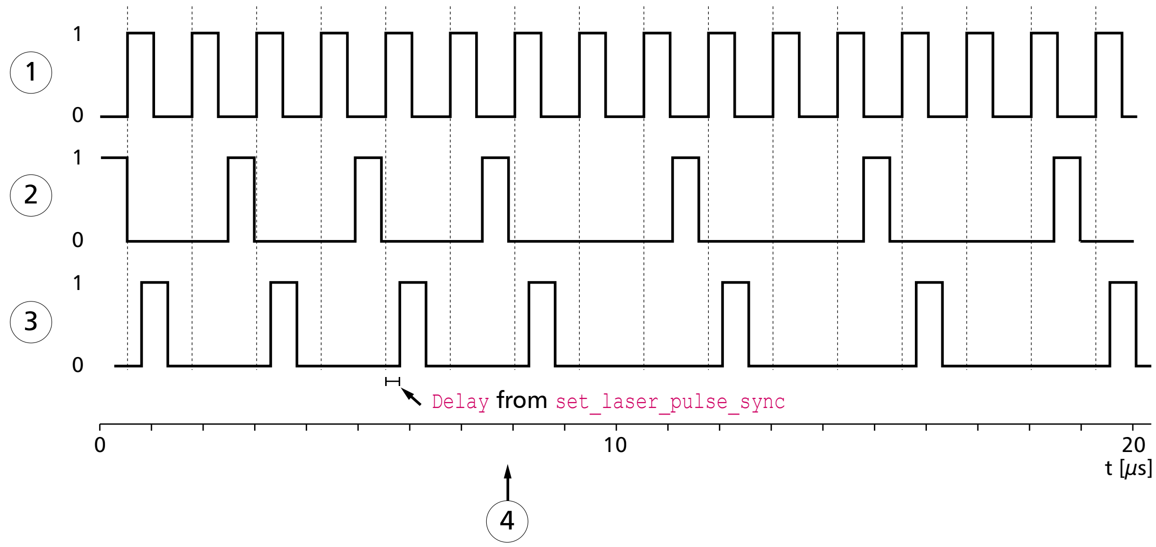
Observe the following when selecting the parameter values:

- By **set_laser_pulse_sync**(Delay), the LASER1 signal can be delayed by a fixed value. The Delay value must be ≤ the period set by
  - **set_laser_pulses**
  - **set_laser_timing**
  - **set_laser_pulses_ctrl**
  - **set_auto_laser_control**
- For a Delay value ≥ (2 × HalfPeriod from **set_laser_pulses**), the following applies:
  - The first LASER1 pulse is delayed by the Delay value
  - The subsequent LASER1 pulse is ignored
- The LASER1 signal pulse length remains unchanged and depends on the preceding configuration (for example, by **set_laser_pulses**).

Notes

- The supplied clock signal must be a TTL signal. The minimum pulse length or pulse pause of the clock signal should be 80 ns. See also Chapter 4.6.1 "LASER Connector", page 75, Section "2-Bit Digital Input Port", page 77.
- By Bit #5 of **set_laser_control**, it can be set whether to use the rising or falling edge of the external clock for synchronization.
- Pulse Synchronization Mode can be combined with "Spot Distance Control" as well as with the other variants of "Automatic Laser Control" such as PulseLength or HalfPeriod, see Chapter 7.4.9 ""Automatic Laser Control"", page 205.
- Pulse Synchronization Mode can be combined with Laser Mode 4 or Laser Mode 6.
- Pulse Synchronization Mode *cannot* be combined with laser modes that use LASER2 signal (CO_2 Mode, Pulse Picking Laser Mode). Pulse Synchronization Mode because:
  - It only affects the LASER1 signal
  - It does not change the LASERON signal
  - It does not change the LASER2 signal
- Pulse Synchronization Mode *should not* be combined with cycle synchronization, see Chapter 7.4.10 "Synchronization of the RTC6 Clock Cycle and an External Clock Signal", page 216.

(1) At the LASER Connector, see Figure 17.



Legend

1. Clock signal of the laser@DIGITAL IN1 pin.
2. Pulse Synchronization Mode disabled: LASER1 pulses@LASER1 pin, not synchronized with 1.
3. Pulse Synchronization Mode enabled: LASER1 pulses@LASER1 pin, synchronized with 1.
4. Change of HalfPeriod, for example, due to a **set_laser_pulses** call.

61

Pulse Synchronization Mode. For more information, see text.

7.5 Marking Dates, Times and Serial Numbers

The RTC6 DLL contains a series of commands to mark the current time, current date or the serial number of products.

Before times, dates and serial numbers can be marked, the required characters and text strings must be defined as indexed characters and indexed text strings.
Separate text strings can be defined for marking times/dates and serial numbers. See Section "Defining Indexed Text Strings for Time, Date and Serial Number", page 122.

7.5.1 Marking the Time and Date

By **time_update** the PC time/date is transferred to the RTC6 PCIe Board. This is necessary after every PC restart. After that, the board (as long as it remains energized) internally counts the date/time with the quartz-controlled 10 μs clock to the second.
By **time_fix**, **time_fix_f** or **time_fix_f_off** the current time/date of the board is hold in a cache.

The time (hours, minutes, seconds) can be marked by **mark_time** or **mark_time_abs** and the date (year, month, day, day-of-the-week) by **mark_date** or **mark_date_abs**.

To mark the date and time, the Gregorian or Julian date can be set as well as the 12- or 24-hour format.

For marking dates of expiry, **time_fix_f_off** is available to fix a forward date based on the current date and current time.

7.5.2 Marking Serial Numbers

By **mark_serial** and **mark_serial_abs**, up to 12-digit serial numbers can be marked. It can be specified how leading zeros are handled.

The RTC6 PCIe Board manages up to 4 serial-number-sets (each with its own serial number and increment size). Serial-number-set 0 is selected at initialization with **load_program_file**.

Other serial-number-sets must be selected in advance by **select_serial_set** or **select_serial_set_list** (see notes below).

By **set_serial**, **set_serial_step** or **set_serial_step_list**, a starting serial number (max. 10 digits) and an increment size for each serial-number-set can be specified. At initialization with **load_program_file** all starting serial numbers are set to 0 and all increment sizes are set to 1.

Each call of **mark_serial** or **mark_serial_abs** causes the current serial number to be incremented (yet before execution of the serial number marking) by the specified increment size.

If a serial number is to be omitted a blank marking can be executed (`Digits` = 0), which increments the serial number by 1 (*not* by the specified increment size).

Notes

- If a serial-number-set is to be marked by **mark_serial** or **mark_serial_abs**, then you can only select that set by the list command **select_serial_set_list**. **mark_serial**, **mark_serial_abs** and **set_serial_step_list** are always applied to the serial-number-set most recently selected by **select_serial_set_list**.
- You can use the control command **get_list_serial** to query the number of the serial-number-set most recently selected by **select_serial_set_list** as well as the current serial number of that set. This also lets you determine (among other things) whether the current number has been or has not been incremented after an uncontinued aborted list.
- The control command **select_serial_set** lets you select (independently of selection by **select_serial_set_list**) a serial-number-set for the control commands **set_serial_step** and **set_serial** (Note that the RTC6 PCIe Board does not prohibit modifying parameters of the serial-

number-set currently being marked). The control commands **set_serial_step** and **set_serial** always apply to the serial-number-set most recently selected by **select_serial_set**.

- **get_serial** returns the current serial number of the serial-number-set selected by **select_serial_set** (if multiple serial-number-sets exist, then the returned serial number is not necessarily the most recently marked serial number).
- **set_max_counts** allows specification of the maximum number of External Starts and thus the maximum number of externally started markings. The number of already occurred External Starts can be obtained with **get_counts**. When a single serial-number-set is used, these commands let you indirectly set the maximum serial number and query the current serial number. But when multiple serial-number-sets are used, these commands do not differentiate between the various serial-number-sets.

8 Advanced Functions for Scan Head Control and Laser Control

8.1 iDRIVE Functions

SCANLAB iDRIVE Scan Systems utilize the i*DRIVE* technology. This servo and control approach exploits the advantages of fully digital servo electronics to deliver significantly expanded functionality. An enhanced transfer protocol between the servo electronics and RTC board facilitates support of all the new features (see also the following section).

This allows users to adjust a number of settings of the respective scan system, for example,

- To select which data it has to transmit to the RTC board
- To choose from different dynamic settings (tunings)
- To set the PosAck limit value
- To set the effective calibration of the scan system
- To set the start behavior of the scan system
- To perform a fault diagnosis
- To perform a functional test of the data transfer

For more information, refer to the manual of the scan system.

i*DRIVE* functions are executed by **control_command**(1).

8.1.1 Transfer Protocol

Data transfer between RTC6 PCIe Board and scan system is carried out according SL2-100 protocol.

The SL2-100 protocol supports the full functionality of i*DRIVE* technology.

With iDRIVE Scan Systems this protocol allows, for example, status signals of the x axis and y axis to be separately and simultaneously evaluated. For a 3D scan system, z axis status signals can be simultaneously evaluated by the channel of the scan head connector to which the z axis is connected.

The XY2-100 Converter (Accessory) can be used with iDRIVE Scan Systems equipped with an interface for XY2-100 protocol or XY2-100 Enhanced protocol.

(1) See also Section "Folder iSCANCfg", page 33.

8.1.2 Configuring the Data Signal Transmission Behavior of the Scan System

Setting Data Types

The digital servo architecture of iDRIVE Scan Systems allows a wide variety of data signals to be returned from the scan system to the RTC board.

Each axis has its own status channel on which data is transmitted to the RTC board every 10 μs:

- STATUS channel
  - Designed for the x axis (Galvanometer scanner 2)
- STATUS1 channel
  - Designed for the y axis (Galvanometer scanner 1)

This opens up possibilities such as monitoring the actual values of the galvanometer scanners during an application or carrying out comprehensive troubleshooting in case of operational malfunction.

control_command(`Data` = 05nn$_H$) allows to set which data the scan system has to transmit to the RTC board. The available data types are described in detail in the manual of the respective scan system as well as (in parts) in Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards", page 1057.
The set data type is only transferred until another one is set.

After every power-up or reset (after the initialization has been completed), the scan system transmits (on all receive channels) the XY2-100 status word.

Reading Out Data

At any time, data received by the RTC6 PCIe Board can be:

- Read out asynchronously by **get_value**, **get_values** or **get_head_status**
- Synchronously recorded by **set_trigger[*]**

See also Chapter 7.3.7 "Status Monitoring and Diagnostics", page 188.

Note that switching of the data source is followed by a short (serial transmission-related) delay of typically 50 μs before the first data is transmitted, see also comment at **control_command**, page 1098.

The value ranges of these data and the possible status states are described in Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards", page 1057.

Notes

- **get_head_status** queries the XY2-100 status word. With SL2-100 protocol-compliant data transfer, the scan system always transfers the XY2-100 status word in parallel with other status information.
  Important: If the XY2-100 Converter (Accessory) is used for control, then it must explicitly be set that the scan system has to transfer the XY2-100 status word to the RTC6 PCIe Board. Otherwise, **get_head_status** returns unusable values.

8.1.3 Monitoring the Positioning

For some applications, it is important to monitor and, if necessary, to document the scan system positioning even during operation.

For this purpose, the actual position of the scan axes must be set to be returned from the scan system by **control_command**. The returned actual position values can be queried then by **get_values** or recorded by **set_trigger[*]** (1).

If the returned actual positions of the scan axes are to be compared with the Cartesian target coordinate values (X, Y, Z), then these must be transformed back by the corrections made on the RTC6 PCIe Board, Chapter 7.3.6 "Output Values to the Scan System", page 186.

For runtime reasons, the backward transformation needs to subsequently be performed by the PC rather than on the RTC6 PCIe Board itself. For this, all correction and transformation settings currently assigned to the scan system can be transferred from the RTC6 PCIe Board to the PC by **upload_transform**. Afterwards, an individual xy value pair or an individual z value can be backward transformed by **transform**.

By **get_transform**/**get_transform_offset** (see also **get_waveform**), an entire series of xy value pairs or z values previously recorded by **set_trigger[*]** can be backward transformed.

Notes

- For some forward transformations, a backward transformation is not possible:

Forward transformation	**Backward trans-formation possible?**
Wobbel motion (#2 on page 186)	no
Global coordinate transformation (#3 on page 186)	no
Processing-on-the-fly correction (#4 on page 186)	no
Coordinate transformations (total matrix and offset) to the x and y coordinates (#5a and #5b on page 186)	yes
Offset to the z coordinate (#5b on page 186)	yes
Clipping to the limits of the controllable Image Field (#6 on page 186)	no
2D Image Field correction (#7 on page 186)	yes
3D Image Field correction (#9 on page 186)	yes
Gain and offset correction of automatic self-calibration (#11 on page 186)	yes
Clipping to the maximum possible control values	no

- Furthermore, backward transformation is not possible if a noninvertable transformation matrix has been defined during forward transformation (notice: clipping to ±50 for each individual matrix coefficient; see Chapter 8.2 "Coordinate Transformations", page 233). The non-invertability is already reported by **upload_transform**.

(1) Due to communication runtimes, the currently returned actual positions are several clock pulses later than the currently outputted control signals.

- If forward transformation included a clipping to the edges of the positionable Image Field or to the edges of the maximum possible range of control values, then backward transformation (ideally) calculates the Cartesian coordinates of these edge values instead of the original values.
- By **get_values**, 4 arbitrary signals can be queried at the same time. Example:
  - the actual position of galvanometer scanner 2 by StatusAX
  - actual position of galvanometer scanner 1 by StatusAY
  - the actual z axis position by StatusBX
  - an additional desired signal, for example, LaserOn

  In contrast, **get_value** (not: **get_values**) is not useful for monitoring xy positioning because it is only meant for querying a single signal and multiple calls unavoidably lead to xyz values across different points of time.
- By **set_trigger**, you can simultaneously record 2 arbitrary signals by 2 measurement channels (**set_trigger4** 4 signals by 4 measurement channels; **set_trigger8** 8 signals by 8 measurement channels).
- The assignment of the values read out by **get_values** or **set_trigger** to x, y or z for the backward transformation can be specified with parameter Code at:
  - **transform**
  - **get_transform**/**get_transform_offset**
- In addition, it is possible to specify which partial transformations are to be performed with:
  - **transform**
  - **get_transform**/**get_transform_offset**
- Values queried by **get_values**, or arbitrary synthetic values can be backward transformed by **transform**. During backward transformation of synthetic values beyond the forward transformation's achievable Image Field, values sometimes are only calculated by extrapolation, due to possible range exceedances or other errors.
- If the user program binarily stores both the recorded values and the transferred transformation data (see **get_waveform**), then subsequent backward transformation by **transform** (not **get_transform**/**get_transform_offset**) can also be executed offline, hence without needing to further access a RTC6 PCIe Board.
- If **control_command** is used to specify positioning error rather than actual position as the to-be-returned data type by the scan system, then it is not possible to directly compare the originally defined pattern with the marked pattern. But you can check if the scan system correctly processed the RTC6 PCIe Board output values. This is particularly useful if backward transformation of actual values is not (fully) possible or when it cannot be determined if deviations between backward-transformed actual positions and originally defined coordinate values are due to scan system error or clipping during forward transformation.

8.1.4 Selecting the Tuning (Dynamics Setting)

SCANLAB can optimize the dynamics setting of scan systems (tuning) to accommodate differing requirements of diverse applications regarding the laser positioning dynamics, for example:

- Vector tuning
  to execute vectors or circular arcs at a constant processing speed
- Jump tuning
  to execute jumps of minimized duration

iDRIVE Scan Systems can be optionally equipped with several tunings. For different applications, the suitable tuning can be switched to – separately for each axis – by **control_command**(`Data` = 11nn_H).

For scan systems equipped with one or several Jump tunings, you can also activate Jump Mode (and hereby tuning autoswitching) for 2D jumps, see Chapter 8.1.5 "Jump Mode", page 226.

The default set start behavior is that the scan system starts with tuning number 0 upon power-up or after a reset.

8.1.5 Jump Mode

For applications such as drilling holes with defined spacing (whereby laser processing is actually point-by-point rather than along lines and curves), you can optimize process times by activating the so-called "Jump Mode".

This requires the scan system to be equipped with a Jump tuning, see also Section "Requirements and Activation", page 227.

Functional Principle

In the default setting (after **load_program_file**), both Jump Commands and Mark Commands are executed in vector mode:

- The jump length gets subdivided into individually executable Microsteps in accordance with the current jump speed. If the scan system is only equipped with a Jump tuning, then the Microsteps execute using this tuning.
- A Jump Delay defined by **set_scanner_delays** is executed before a subsequent list command.

In contrast, when Jump Mode is enabled and activated by **set_jump_mode** or **set_jump_mode_list**, every 2D jump (see below) is executed as follows:

- The entire jump length of the 2D jump is controlled as a "Hard Jump" over a time dimensioned jump of 10 μ duration. The target position is executed without Microstepping.
- The jump executes with a Jump tuning. **set_jump_mode** can be used to designate which Jump tuning to use. If a different tuning has been set before the jump, then the RTC6 PCIe Board automatically switches at the beginning of the jump to the tuning specified by **set_jump_mode**.
- At the end of the 2D jump, the RTC6 PCIe Board automatically switches to a Vector tuning (if the scan system is equipped with one and if a corresponding setting has been made by **set_jump_mode**).
- At the end of the 2D jump, a jump-length-dependent Jump Delay occurs. This Jump Delay can be specified for the corresponding jump length by **load_jump_table_offset** or **set_jump_table**, see also Section "Jump-Length-Dependent Jump Delays", page 227. Here, an external Jump Delay specified by **set_scanner_delays** is *not* taken into account.

Notes

- Jump Mode works exclusively on
  - **jump_abs**, **jump_rel**, **goto_xy** (*not* on the corresponding 3D, para or timed commands)
  - home jumps and home returns (see **home_position**)
- If a 2D jump occurs where the jump length limit (`Length` parameter) specified by **set_jump_mode** is *not* reached or exceeded on at least one of the two axes, then the jump executes in vector mode even if Jump Mode has been enabled and activated. This allows exploitation of the fact that *short* jumps can in some circumstances execute faster by Vector tuning than with Jump tuning. But if no Vector tuning is installed or none specified, then you should set the `Length` parameter to 0.
- Each switch between different tunings (servos) requires an additional 10 μs clock cycle. For applications such as pure drilling, this can be avoided by not specifying a Vector tuning to switch back to when you call **set_jump_mode**.
- When you deactivate or disable Jump Mode (by **set_jump_mode** or **set_jump_mode_list**), then subsequent jumps again execute in vector mode (split-up into Microsteps and without further servo autoswitching). Here, the Vector tuning is used that has been most recently set at the end of Jump Mode, unless deactivation has been followed by selection of a different tuning by **control_command**. Moreover, the most recently set jump speed is again used and jumps are followed by the Jump Delay specified by **set_scanner_delays**.

Requirements and Activation

The following are required for enabling and activating Jump Mode:

- At least one of the two scan head connectors must have been assigned a correction table.
- At least one of the two scan head connectors must be connected to an intelli*SCAN*, intelli*cube*, intelli*WELD* or intelli*DRILL* scan system.
- As of scan system firmware ≥ 2078.
- The attached scan system must be equipped with at least one Jump tuning. In contrast, a Vector tuning is not absolutely required.
- The tuning numbers specified by **set_jump_mode** must match those stored on the board.
- The tunings specified by **set_jump_mode** must be of the proper type – Vector tuning or Jump tuning – (the Tuning type is stored in the scan system firmware) and must be suitable for rapid switching.

Before Jump Mode can be activated by **set_jump_mode_list**, it must have been successfully enabled at least once by **set_jump_mode** (see command description).

The **set_jump_mode** control command (but not the **set_jump_mode_list** list command) performs an appropriate check if Jump Mode has been not already enabled.

Jump-Length-Dependent Jump Delays

When executing a "Hard Jump", it takes the scan head some time to reach the specified position.

The RTC6 PCIe Board takes this delay (also called step response) into account by appending a Jump Delay at the end of the jump.

Point-by-point laser processing does not need to take other Scanner Delays into account and you can generally set Laser Delays to 0.

The specific step response behavior of the respective scan system (step response time vs. jump length) can be stored on the RTC6 PCIe Board in a user-specific Jump Delay table. With Jump Mode enabled, the RTC6 PCIe Board uses the specified Jump Delay table to determine the appropriate Jump Delay value for each 2D jump in accordance with the jump's longer edge (that is, either the x or y component of the jump).

You can determine the step response behavior experimentally and then load it onto the board as a table of values using **load_jump_table_offset**. Alternatively, the Jump Delay table can also be automatically determined by **load_jump_table_offset** (parameter `Name` = NULL). Additionally, the currently loaded Jump Delay table can be retrieved as a binary table by **get_jump_table** and reloaded onto the board by **set_jump_table**.

The step response time (at least for longer jumps) typically scales with the squareroot of the jump length, and **load_program_file** accordingly initializes the internal jump table – with an end value of 10.24 ms for a jump length of 2^{20} bits.

When the Jump Delay table has been loaded and a new `RTC6DAT.dat` file is created by **create_dat_file**, then this table is automatically loaded upon the next **load_program_file**.

Determining Jump Delay Values Experimentally

The user manual of the scan system typically specifies the step response times for each Jump tuning at selected jump lengths.

To experimentally determine the step response behavior, you need to have the scan system perform jumps of various lengths and query the resulting position values by the status channel for analysis.

After you activate Jump Mode, perform the jumps by using **jump_abs** or **jump_rel**. The scan system should have been previously set to return the actual-position data type by **control_command**. You can then record the latter by **set_trigger[*]** and retrieve it by **get_waveform**.

The determined Jump Delay values must be supplied in an ASCII text file. If the step response behaviors of both axes differ, then the higher of the two axes' Jump Delay values should be supplied in the ASCII text file.

Notes on Loading Determined Jump Delay Values

- For jump length values and Jump Delay values, **load_jump_table_offset** loads a table from an ASCII text file.
- The ASCII text file can contain one or several tables.(1)
- Each table can contain up to 50 data points *(Length | Delay(Length))*.
- The complete (internal) Jump Delay table *Delay*(*Length*) is linearly interpolated from the data points.

For the tables, the following rules apply:

- Each table must begin with the line:
  [JumpTable<No>]
  <No> represents the table number.
- If the table contains multiple [JumpTable<No>] entries with the same <No>, then only the lines after the first entry are used. Only lines up to the next '[' character (that is not preceded by a semicolon) are used.
- Each data point (*Length* | *Delay*(*Length*)) is defined as follows:
  Length<n> = <LengthValue>
  Delay<n> = <DelayValue>
  where <n> is the data point index (1 ≤ <n> ≤ 50).
  The <Value> numbers can be supplied as (unsigned) floating point numbers. Decimal separator: decimal point (.).
- If the table contains multiple data points with the same index <n>, then the most recently read one is used and the previous ones ignored.
- If the table contains multiple data points with the same jump length value *Length*, then the data point with the largest index <n> is used and the others ignored. Equality is checked to within ±0.01.

(1) Even of another type, see table 1, page 156.

* For <Value> the following ranges apply:
$0.0 \leq Length \leq 1048576.0$
$0.0 \leq Delay(Length) \leq 65535.0$
Delay values are supplied in units of 10 µs, jump lengths in bits.
* Each instruction must be in a separate line.
* Space characters and tabs within a line (for example, between '=' and <Value>) are ignored.
* Empty lines are ignored.
* Data points with invalid values are ignored.
* The data point of a particular index <n> is ignored if the corresponding `Length<n>` and/or `Delay<n>` definition is missing.
* The semicolon ';' can be used for comments. All characters in a line following a semicolon are ignored.
* The instructions for data points in the table can be ordered as desired.
* Indices for data point pairs in the table can be selected as desired within the range [1…50] (the table is then automatically sorted by ascending position values).
* If the table contains *no* valid data points, then **load_jump_table_offset** has no effect (return value 1 or 13).
* If there is no entry of *Length* = 0.0, then one with `Delay` = Min(`Delay<i>`) is inserted (the smallest valid value encountered is filled downward). The same applies to *Length* =524288.0 with Max(`Delay<i>`).
* If the specified text file contains only one valid data point with `Delay<n> = D`, then the Jump Delay table *Delay*(*Length*) `= D` (for the whole jump length range) is loaded.

Automatic Determination of the Jump Delay Table

If previously Jump Mode has been successfully enabled and activated by **set_jump_mode**(`Flag` = 1), an automatic determination of the Jump Delay table can be started by **load_jump_table_offset**(`Name` = NULL).

For automatic determination, "Automatic Laser Control" is deactivated, the data type to be returned by the scan system is set to target position and the tuning set to the Jump tuning that had been defined by **set_jump_mode** (the original settings are restored when **load_jump_table_offset** completes).

For automatic determination, several diagonal jumps of varying lengths are performed. For each jump, the target position returned by the scan system is recorded by **set_trigger** (not: **set_trigger4**, **set_trigger8**) and retrieved by **get_waveform**. The data is analyzed for the timepoint at which the specified position tolerance (`PosAck` parameter) has been last exceeded, that is, when the target position persistently remained in the jump target positional range ±`PosAck`. This value is then reserved as the Jump Delay value associated with the corresponding jump length.

Notes

- Before automatic determination, you must absolutely switch off the laser.
- Data recording requires execution of a list with a total of six commands. The commands are automatically written to RTC6 List Memory and processed. The parameter `ListPos` indicates the position in RTC6 List Memory ("list 1" or "list 2") in which storage is to occur. This position should be such that any previously entered list commands can be harmlessly overwritten.
- For automatic determination, the longest jump is performed first, followed by increasingly shorter jumps (with a maximum of up to 16 different jump lengths). For the first (longest) jump length (jump across the entire image diagonal), the measurement period is specified by the parameter `MaxDelay` [10 μs]. `MaxDelay` should be chosen to be adequate but not significantly larger than the Jump Delay for the longest jump. A larger `MaxDelay` increases the total required execution time. With `MaxDelay` = 500, the total execution time for automatic determination is typically a few seconds.
- For statistical noise reduction, 4 identical jumps are performed for each jump length and the results averaged. Additionally, the values for each individual measurement are low-pass filtered (2-point smoothing). This permits selection of a position tolerance `PosAck` that can also be somewhat (but not substantially) under the expected noise level but note that only whole multiples of 16 are returned with an XY2-100 Converter (Accessory). Here, a noise level of $\pm 3 \times 16$ bits can be expected).
- If the `PosAck` range is not persistently reached within the measurement period, then `MaxDelay` becomes the determined Jump Delay.
- If the determined Jump Delay is smaller than `MinDelay`, then `MinDelay` becomes the determined Jump Delay and the measurement terminates. Then, shorter jumps are no longer performed and `MinDelay` is also the determined Jump Delay for these shorter jump lengths. A longer `MinDelay` reduces the total execution time for automatic determination.

- If an offset is specified for automatic determination (by the according **load_jump_table_offset** parameter), this offset is added to all automatically determined delay values before the overall Jump Delay table gets calculated by linear interpolation and loaded onto the board (in addition, the delay values are clipped to the value range 0…65,535).
  The Offset can be used to compensate for measurement runtime latencies (for example, caused by an XY2-100 Converter (Accessory), by tuning switching or by a runtime latency of the signal returned by the scan system) when calculating the Jump Delay table. It can also be used to add a safety margin to the delay values to compensate for noise-induced random deviations.
  load_jump_table and **load_jump_table_offset**(`Offset` = 0) are identical.
- For simultaneous control of two scan systems, you should determine the Jump Delay values for both systems and, after comparing, use the values of the slower system.
- The resulting table can be retrieved in binary form by **get_jump_table** and reloaded onto the board by **set_jump_table**.

8.1.6 Configuring the PosAck Limit Value

control_command(`Data` = 15nn$_H$) can be used to set the PosAck limit value nn. The default start behavior is for the scan system to set the limit value to 0.28% of the full position range after every power-up.

If other limit values are desired, they must be separately set for each axis.

8.1.7 Configuring the Effective Calibration

By **control_command**(`Data` = 12nn$_H$), the scan system control electronics can be set to scale down position values received from the RTC6 PCIe Board by a certain factor. The scaling factor can be set to the values 1, 1/2, 1/4 or 1/8.

Position values sent back from the scan system (optional) to the RTC6 PCIe Board are backward transformed by the reciprocal of the scaling factor.

However, the effective calibration can be thereby reduced to confine the scan area to a smaller angular range – with a higher angular resolution.

The default start behavior is for the scan system to start with a scaling factor of 1 upon power-up.

By **control_command**(`Data` = 053F$_H$), the currently set scaling factor can be read out.

8.1.8 Configuring the Start Behavior

The default configuration of iDRIVE Scan Systems is set as follows:

- Tuning number 0, see also Chapter 8.1.4 "Selecting the Tuning (Dynamics Setting)", page 225
- PosAck limit value is $B8_H$ (corresponds to 0.28% of the full position range of 2^{16} bit), see also Chapter 8.1.6 "Configuring the PosAck Limit Value", page 231
- Scaling factor = 1, see also Chapter 8.1.7 "Configuring the Effective Calibration", page 231

These settings can be changed by **control_command**. The changed settings are only temporary, however they can be additionally saved as starting settings for subsequent power-ups by **control_command**(`Data` = $0A00_H$).

The return behavior of the scan system can only be temporarily changed. After a power-up, the scan system transmits the XY2-100 status word, see also Section "Configuring the Data Signal Transmission Behavior of the Scan System", page 222.

8.1.9 Fault Diagnosis and Functional Test

The various feedbacks of iDRIVE Scan Systems can be used for diagnostics in the event of a fault.

If a problem occurs, the versatile status return functions of the can be used for scan system diagnosis, too.

For example, an event code can be queried that indicates which event has been responsible for the change to an error state.

To verify that data transfer capability between the RTC6 PCIe Board and a scan system is intact, by **control_command**(`Data` = $21nn_H$) an 8-bit value nn – separately for each axis– can be transmitted to the scan system. Subsequently, a 20-bit value is returned on the corresponding status channel: If data transfer is error-free, then the upper 8 bits of the returned 20-bit value is identical with the originally sent 8-bit value, and the next lower 8 bits are identical with the complement of the sent 8-bit value.

This 20-bit value is returned until **control_command**(`Data` = $05nn_H$) is used to select another data type to be returned by the scan system, see Section "Configuring the Data Signal Transmission Behavior of the Scan System", page 222.

Prior to a transfer test, the data type currently selected for transmission can be cached by **control_command**(`Data` = $17FF_H$). After the test, **control_command**(`Data` = 1700_H) restores the same transfer behavior as before the test.

8.2 Coordinate Transformations

For precise set-up of the scan system relative to the Image Field (or, if the Option "Second Scan Head Control" is enabled, *two scan heads* can be adjusted relative to a *common* Image Field), a linear coordinate transformation can be defined (separately for the Connector for First Scan Head and Connector for Second Scan Head) for all x and y output coordinates (x|y) defined by Vector Commands or "Arc" Commands:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = M \times \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

The (2 x 2) total matrix M is thereby automatically calculated by the RTC6 PCIe Board as a product of a scaling matrix M_S, a rotation matrix M_R and a general transformation matrix M_T:

$$M = M_T \times M_R \times M_S$$

The coefficients of the three matrices (M_T, M_R, and M_S) and the offset values ($x_0|y_0$) can be individually defined for the Connector for First Scan Head and Connector for Second Scan Head.

The offset ($x_0|y_0$) is set by **set_offset** or **set_offset_list**.

For 3-axis scan systems, **set_offset_xyz** or **set_offset_xyz_list** enables setting of an offset z_0 for the z coordinate, too (z_0 has the opposite effect of **set_defocus** or **set_defocus_list**).
The following applies:

$$z' = z + z_0$$

The coefficients of the scaling matrix M_S are set by **set_scale** or **set_scale_list** using a scaling factor *k* that is common to both axes:

$$M_S = \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix}$$

The coefficients of the rotation matrix M_R are set by **set_angle** or **set_angle_list** by specifying a rotation angle α (in accordance with mathematical convention: positive angles produce counterclockwise rotation):

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

The coefficients $m_{11}...m_{22}$ of the general transformation matrix M_T are set by **set_matrix** or **set_matrix_list**:

$$M_T = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$$

With the general transformation matrix M_T, the two above matrices (M_S and M_R, as special case) as well as further transformations for scaling, rotating, mirroring or skewing objects can be defined:

- Scaling by the factors k_x and k_y:

$$M_T = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}$$

- Rotation by the angle α:

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

Example:
`set_matrix( 1, 0.5, -0.866, 0.866, 0.5, 1 )`
defines a rotation by 60° (counterclockwise) around the center of the Image Field for the Connector for First Scan Head.
This can also be achieved by `set_angle(1,60)`.

- Mirroring around the y axis (flipping in the x direction):

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

- Mirroring around the x axis (flipping in the y direction):

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

- Mirroring around the first dimension diagonal (exchanging the x and y coordinates):

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

- Skewing in the x direction by the angle α (slanting):

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

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

A general transformation defined by **set_matrix** or **set_matrix_list** can also represent a combination of various transformations (users can calculate the corresponding matrix M_T by multiplying the corresponding individual matrices in the correct order).

Notes

- The described coordinate transformations are primarily intended for small corrections when setting up the scan system relative to the Image Field. Separate settings for scaling and rotations thereby provide more handling flexibility in comparison to a single matrix setting.
- Initialization by **load_program_file** results in an offset of (0|0|0) and in matrices M_S, M_R and M_T, each predefined as identity matrices.
- Each matrix or offset definition overwrites prior definitions.
- The RTC6 PCIe Board calculates the total matrix M independently of the order in which the individual transformation matrices were defined.
- The value range of scaling factor *k* for the scaling matrix M_S is [–16…+16]. The value range for the coefficients of the general transformation matrix M_T is [–50…+50]. Also be sure that the value range [–50…+50] for individual coefficients of the total matrix M are not exceeded; otherwise calculation of the corrected coordinates might result, under some circumstances, in overflows.
- Rotations take place exclusively around the centerpoint of the Image Field; mirroring is relative to the axes.
- For each definition, the parameter `at_once` can be used to specify whether the new setting should have immediate effect on the current position (`at_once` = 1 or 3) or whether it should only be provisionally accumulated and cached (`at_once` = 0 or 2). The most recently called `at_once` parameter value determines when a (accumulated) transformation takes effect.

- Before the total transformation is applied to the current position, the Signals for "Laser Active" Operation are switched off with `at_once` = 0…2. They remain unchanged with `at_once` = 3.
- With `at_once` = 1 or 3, all settings (only) accumulated until then are processed immediately and simultaneously. In the process, the scan system axes are moved from the current position to the corrected position at the defined jump speed. Consequently, this can require some clock cycles.
  This needs to be observed especially when RTC6 PCIe Boards are master/slave synchronized and are supposed to execute different jump lengths. With this use case, the coordinate transformations should be executed prior the synchronized start with `at_once` = 1 and their end should be waited for.
  Any Scanner Delays are not initialized. The INTERNAL-BUSY list execution status is set while the jump to the corrected position is executed.
- With `at_once` = 2, the accumulated settings only become effective upon execution of the next **jump_abs**, **jump_rel**, **goto_xy** or **goto_xyz** (but not other Jump Commands such as **jump_abs_3d** or **jump_rel_3d**) unless afterwards another call triggers immediate execution. Correction of the current output position then occurs together with the specified coordinate jump. This eliminates unnecessary galvanometer scanner motions (incl. delays).
  Example: The following command sequence produces a first jump to (0, 0), followed by – if `at_once` = 1 or `at_once` = 3 – a second jump from (0, 0) to (1000, 500) and a third from (1000, 500) to (0, 500). But if `at_once` = 2, then only a second jump occurs from (0, 0) to (0, 500).

```
jump_abs( 0, 0 );
set_offset_list( 1000, 500, at_once );
jump_abs( -1000, 0 );
```

Notes on data recording:

- The galvanometer scanner motion of **jump_abs** or **jump_rel** at the beginning of the list can be recorded with **set_trigger** (signal type 7, 8, 9), if *no* coordinate transformation with `at_once` = 2 has been applied before.
- In case of motions *after* a coordinate transformation (for example, by **set_offset_list** with `at_once` = 2) the "sample values" (**set_trigger** signal type 7, 8, 9) are immediately set to the target coordinates and remain unchanged for the duration of the motion. The motion recorded here does not appear like Microstepping[(1)] has been performed, but rather like a Hard Jump. The real galvanometer scanner motion can be recorded with **set_trigger**, signal type 25 and 26 (transformed control values).
- In case of motions caused by coordinate transformation in a list (for example, by **set_offset_list** with `at_once` = 1), the "sample values" (**set_trigger** signal type 7, 8, 9) remain unchanged for the duration of the motion. The real galvanometer scanner motion can be recorded with **set_trigger**, signal type 25 and 26 (transformed control values).

- Settings via control commands by `at_once` = 0 are only saved as long as no list is running. When the execution is started or the list is already running, the settings take effect immediately before the next list command.
  Settings via list commands with at_once = 0 are only saved until they are retrieved elsewhere (`at_once` > 0 or by a control command).
- If no correction table is assigned to the corresponding scan head connector, then the new settings for the coordinate transformations are only stored on the RTC6 PCIe Board. They take effect when a correction table is assigned.

(1) See Chapter 7.1.2 "Microstepping", page 143.

- Coordinate transformations are applied to all to-be-outputted coordinates from all Vector Commands ([*]**jump**[*] or [*]**mark**[*] list commands, but also **goto_xy** or **goto_xyz**) and "Arc" Commands. See also Chapter 7.3.6 "Output Values to the Scan System", page 186.
- With a scaling matrix, the effective jump speed and mark speed changes.
- With 3D vector commands:
  - The transformation matrix only affects the x and y components
  - The offset affects all three components
- In order to utilize the complete real Image Field with coordinate transformations (such as rotations, shrinkages or shifts), the extended value range of the virtual Image Field can be used even without Processing-on-the-fly, see Chapter 7.3.3 "Virtual Image Field", page 174.
- Coordinate transformations for the virtual Image Field can be defined, see Section "Coordinate Transformations in the Virtual Image Field", page 174.
  See also 4 in Chapter 7.3.6 "Output Values to the Scan System", page 186.

Notes for RTC4 Users

- With RTC6 PCIe Boards, coordinate transformations can no longer be set upon loading a correction file by **load_correction_file**. Instead, coordinate transformations are separately defined for the Connector for First Scan Head and Connector for Second Scan Head and serve the same purpose as those of **load_correction_file** of the RTC4.
- RTC6 PCIe Boards do not support the coordinate transformations (collectively for both *scan heads*) before Microstepping which is possible with the RTC4.
  The global coordinate transformations have a slightly different effect than the coordinate transformations above, see #3, #5a and #5b in Chapter 7.3.6 "Output Values to the Scan System", page 186.

8.3 Online Positioning

The preceding Chapter 8.2 "Coordinate Transformations", page 233 details how to precisely align a scan system relative to the Image Field, see also Section "Coordinate Transformations in the Virtual Image Field", page 174.

The user program can, for example, determine the required transformation values by automatic position analysis for a workpiece on a conveyor belt and then execute the associated transformations.

However, it is not easy to achieve well-controlled timing (referenced to the 10 µs clock of the RTC6 PCIe Board) while positioning a workpiece and aligning the scan system by the control commands described in Chapter 8.2 "Coordinate Transformations", page 233. For applications in which such timing is important, commands for so-called Online Positioning are available. Here, data for an offset and/or rotation coordinate transformation or a general matrix operation can be inputted by the McBSP interface.

The following variants are provided for different use cases:

- The previous "Local Online Positioning" concerns the scan system-specific coordinate transformations in the real Image Field analogous to **set_offset**, **set_angle** and **set_matrix** with parameter `HeadNo` = 1 or 2 according to #5a and #5b in Chapter 7.3.6 "Output Values to the Scan System", page 186. It is described in Chapter 8.3.1 ""Local Online Positioning"", page 237.
- The "Global Online Positioning" concerns global coordinate transformations in the virtual Image Field analogous to **set_offset**, **set_angle** and **set_matrix** each with `HeadNo` = 4 according to #3 in Chapter 7.3.6 "Output Values to the Scan System", page 186. It is described in Chapter 8.3.2 ""Global Online Positioning"", page 240.

Online Positioning cannot be combined with Processing-on-the-fly applications with position information, but it can be combined with encoder-based Processing-on-the-fly applications.

Notes

- Since coordinate transformations #5a and #5b are calculated after the Processing-on-the-fly correction #4, larger Image Field rotations are not well compatible with linear Processing-on-the-fly corrections. In such cases, "Global Online Positioning" is preferable.

8.3.1 "Local Online Positioning"

Reading in data for "Local Online Positioning" via the McBSP interface needs to be configured by **set_mcbsp_x**, **set_mcbsp_y** and/or **set_mcbsp_rot** or **set_mcbsp_matrix** (or by the equivalent list commands).

With **apply_mcbsp** or **apply_mcbsp_list**, you can acquire the most recent fully transferred values and define the required coordinate transformations (as with **set_offset** and/or **set_angle** or **set_matrix** or the equivalent list commands). Here, as with the commands described in Chapter 8.2 "Coordinate Transformations", page 233 an `at_once` parameter can be used to specify when the newly defined (total) transformation should take effect.

For precise timing, execution of the list command that triggers the transformation (depending on the `at_once` parameter, this would be **apply_mcbsp_list** or **jump_abs** or **jump_rel** or any other list command) can be made dependent on the input of an external control signal (for conditional command execution, see Chapter 9.3.2 "Execution of Conditional Commands", page 317).

The McBSP interface is described in Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87.

Configuring "Local Online Positioning"

set_mcbsp_x, **set_mcbsp_y** and/or **set_mcbsp_rot** (or the equivalent list commands) determine both how the values inputted at the McBSP interface are interpreted by the RTC6 PCIe Board and which internal memory location is used to read the values:

- Depending on which of the above commands is called, the RTC6 PCIe Board interprets the inputted values as offsets in the x direction and/or y direction and/or as rotation values or as matrix coefficients. The desired scaling factor always needs to be supplied as a command parameter (except with **set_mcbsp_matrix**, see command description).
  The three options **x,y** and **rot** can be used either separately or in any desired combination. By the appropriate command, each option can be enabled or disabled independently of the other two. In contrast, the **matrix** option cannot be used in conjunction with other options.
- As soon as one of the 4 options becomes activated, all values subsequently inputted at the McBSP interface are internally stored in memory location 1 or 2 (see below). Transferred values can subsequently be queried by **read_mcbsp** or applied in coordinate transformations by **apply_mcbsp** or **apply_mcbsp_list**.
- **x or y *or* rot**
  If you activate only *one* of the three options, then x or y offset correction values can be supplied as signed 32-bit values or rotation correction values as unsigned 32-bit values.
  The McBSP input values are transferred to internal memory location 1.
- **x and y (without rot)**
  If you activate x and y offset corrections, but no rotation correction, then the two offset correction values must be supplied as a signed 16-bit value, each, combined to a 32-bit value (the x value in the lower 16 bits and the y value in the upper 16 bits).
  The McBSP input values are transferred to internal memory location 1.
- **x or y and rot**
  If you activate an X or a Y offset correction together with a rotation correction, then the offset and rotation correction values should be alternatingly supplied as 32-bit values. The McBSP input values are then alternatingly transferred to internal memory locations 1 and 2. The RTC6 PCIe Board identifies the data type by examining the coding Bit #31 (Bit #31 = 0 for offset values, Bit #31 = 1 for rotation correction values).
  Signed 31 bits are effectively available for transferring offset values. 31 bits *without* sign are available for rotation correction values.

- **x and y and rot**
  If you activate all three options together, then the two offset correction values must be combined and supplied as one 32-bit value alternatingly supplied with the rotation correction value as a second 32-bit value. The McBSP input values are likewise alternatingly transferred to internal memory locations 1 and 2.
  The RTC6 identifies the data type by examining the coding Bit #31 (Bit #31 = 0 for offset values, Bit #31 = 1 for rotation correction values). Signed 15 bits are effectively available for transferring offset values (whereby x values reside in the lower 16 bits and y values in the upper 16 bits). 31 bits *without* sign are available for rotation correction values.
  The last two cases designate the data type by coding Bit #31. Though this makes the order of transmission irrelevant, always *both* data types nevertheless must be transmitted (preferably always alternatingly). If a request is made by **apply_mcbsp** (or **apply_mcbsp_list**) when two values of the same data type exist in both memory locations, then the most recently transferred value is always used. If two identical Bit #31 codings are present, then the last transfer should have already ended at the time of the request.
- **matrix**
  With this option activated, matrix coefficients are then transferable as signed 32-bit values. The indices are encoded in the data word (see command description). McBSP input values get transferred to internal memory location 1.

Notes

- You can use "Local Online Positioning" in conjunction with an encoder-controlled Processing-on-the-fly application, but *not* in conjunction with a Processing-on-the-fly application controlled by McBSP signals:
  – When you use the commands for configuring "Local Online Positioning", then Processing-on-the-fly correction activated by **set_fly_x_pos**, **set_fly_y_pos**, **set_fly_rot_pos**, **set_mcbsp_in**, **set_mcbsp_in_list**, **set_multi_mcbsp_in** or **set_multi_mcbsp_in_list** gets automatically deactivated. Subsequently, McBSP input values are copied to internal memory locations 1 and 2 (see above) and are then available for "Local Online Positioning".
  – In reverse, **set_fly_x_pos**, **set_fly_y_pos**, **set_fly_rot_pos**, **set_mcbsp_in**, **set_mcbsp_in_list**, **set_multi_mcbsp_in** or **set_multi_mcbsp_in_list** deactivate a previously activated "Local Online Positioning". Subsequently, McBSP input values are then transferred to the internal memory locations 0 to 3 depending on the command and are then available for the Processing-on-the-fly application.
  – If you switch off (intentionally or with an invalid scaling factor) "Local Online Positioning" by **set_mcbsp_x**, **set_mcbsp_y** or **set_mcbsp_rot**, then the data is continued to be copied to internal memory locations 1 or 1 and 2 (as long as data is transmitted), but it is no longer applied (**apply_mcbsp** has no effect).

8.3.2 "Global Online Positioning"

"Global Online Positioning" (available as of RTC6 Software Package ≥ V1.6.1) needs to be activated by one of the following commands:

- **set_mcbsp_global_matrix**
- **set_mcbsp_global_rot**
- **set_mcbsp_global_x**
- **set_mcbsp_global_y**
- **set_mcbsp_global_matrix_list**
- **set_mcbsp_global_rot_list**
- **set_mcbsp_global_x_list**
- **set_mcbsp_global_y_list**

Once activated, all other McBSP processings are deactivated ("Processing-on-the-fly" applications controlled by McBSP signals, "Local Online Positioning" as described in Chapter 8.3.1 ""Local Online Positioning"", page 237, processings activated by **set_mcbsp_in** or **set_multi_mcbsp_in**) and vice versa.

All subsequent data transferred via McBSP are internally handled in the same way as with "Local Online Positioning" (copied to internal memory locations 1 and possibly 2; readable by **read_mcbsp**),
but instead of the scan system-specific coordinate transformations in the real Image Field (see Chapter 8.3.1 ""Local Online Positioning"", page 237) automatically used for coordinate transformations in the virtual Image Field instead.

Calling **apply_mcbsp** or **apply_mcbsp_list** is no longer necessary and even has no effect.

Coordinate transformations in the virtual Image Field (see also Chapter 8.6.4 "Compensating 2D Motions", page 258) are automatically applied, as soon as a Processing-on-the-fly application is activated *afterwards*.

During a Processing-on-the-fly application, new data can only be sent and stored, but not applied, see also:

- `set_matrix( HeadNo` = 4)
- `set_offset_xyz( HeadNo` = 4)
- `set_angle( HeadNo` = 4)

"Global Online Positioning" is compatible with Processing-on-the-fly applications controlled by encoder signals.

> **Notice!**
>
> - The latest transferred data value is used immediately according to the current "Global Online Positioning" mode. Therefore, make sure to transmit correct values after changing that mode and before applying the data by starting a Processing-on-the-fly session.
> - Example: **set_mcbsp_global_x** and **set_mcbsp_global_y** combine the xy offsets in the lower and upper half word of the transmitted data.
>   **set_mcbsp_global_y**`( Scale = 0.0 )` disables the y offset and the latest sent data value is used *in total* as x offset. Make sure to transmit the correct x offset again.

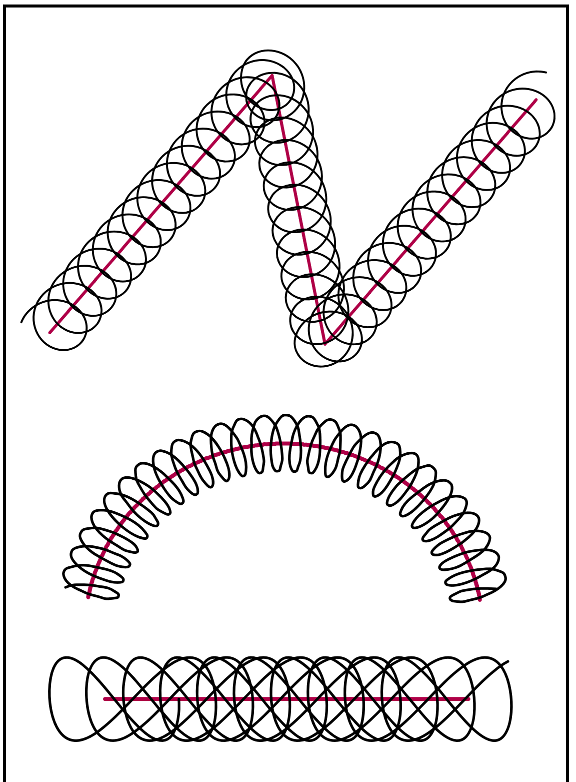
Configuring "Global Online Positioning"

The "Global Online Positioning" is configured the same way as the "Local Online Positioning", see Chapter 8.3.1 ""Local Online Positioning"", page 237.

8.4 Wobbel Mode

The Wobbel Mode allows varying the line width for laser marking.

For this purpose, for example, an ellipse-shaped motion is added to the regular, linear motion of the output position. This results in a spiral motion of the laser focus in the Image Field, see Figure 62. Alternatively, the motion can be combined with a horizontal or vertical figure-of-8.



62

Principle of the Wobbel Mode. Top: circular wobbel. Middle: ellipse-shaped wobbel. Bottom: figure-of-8 wobbel (horizontal 8).

A broadening of the original line is obtained by choosing suitable values for the transverse and longitudinal amplitudes and the frequency of the wobbel motion. With figure-of-8s, broader mid-line processing can be achieved by appropriate parameter values. If the specified transverse and longitudinal amplitudes are identical, then the wobbel shape remains stationary in space; otherwise the orientation of the wobbel shape follows the current direction of motion. If there is no direction of motion, no wobbel motion is performed.

During Processing-on-the-fly correction by the McBSP interface or encoder interface where the scan head only compensates differences between actual external motions and intended total motion, see Chapter 8.6.2 "Compensating Linear Motions", page 252, a slight jitter in the direction of galvanometer scanner motion might occur, particularly during exact external path motions. Here, the wobbel motion superimposed onto the direction of motion does correspondingly jitter, too. You can avoid such jitter by specifying a fixed (instead of the momentary) direction of motion for the wobbel shape by the **set_wobbel_direction** list command. This is also particularly important if the complete translation motion takes place outside and the galvanometer scanners only have to carry out the actual sweep motion.

After **set_wobbel** or **set_wobbel_mode**, the wobbel start point is always set for the same value relative to the vector/arc startpoint and direction. The Wobbel phase is then continued both within an uninterrupted Polyline and after interruptions (for example, by a Jump Command) until **set_wobbel** or **set_wobbel_mode** are called again.

The Wobbel Mode cannot be combined with:

- Sky Writing
- Pixel Output Mode
- Jumps[(1)]
- **laser_on_list**

For further details, see **set_wobbel** and **set_wobbel_mode**.

(1) See also **set_wobbel_mode_phase**.

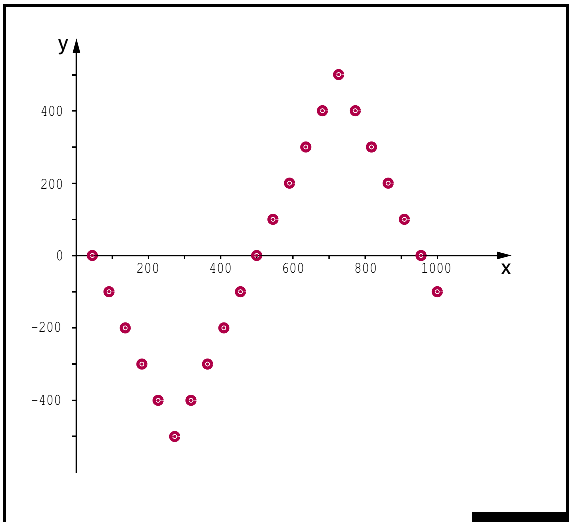
For optimum marking results, the wobbel frequency and mark speed must complement each other, see Chapter 8.4.1 "Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values", page 243.

For many welding applications, the default set of "classic" wobbel shapes (circle, ellipse, sine, figure-8) does not produce optimal results in the area of the weld seam. For example, high-speed motion occurs parallel to translation motions and low-speed motion occurs in the opposing direction.

set_wobbel_vector lets you define a wobbel shape customized for your user program, consisting of up to 1023 piecewise linear sections, while also specifying variation of laser power along that shape (see also **set_wobbel_control**). By **create_dat_file**, this "freely definable wobbel shape" is saved, see comment on page 377.

However, **set_wobbel_vector** cannot be combined with automatic laser control or Vector-Defined Laser Control.

The present wobbel excursion from **set_wobbel** or **set_wobbel_mode** can be recorded by **set_trigger[*]** (signal 53). The format of the data is ((transversal << 16) + longitudinal).



63

See Section "Example Code (C++)", page 242.

Example Code (C++)

The following example C++ source code shows a zigzag pattern, see Figure 63.

The code must be included in a user program, see Chapter 6.2.5 "Example Code (C)", page 103.

```
// Zigzag pattern

// Transversal micro step
   const double dTrans( 100.0 );
// Longitudinal micro step
   const double dLong( 0.0 );

// Number of steps
   const uint16 Period( 5 );

// Start a new wobbel shape
   set_wobbel_vector( 0, 0, 0, 0);

// 1st wobbel vector
   set_wobbel_vector( dTrans, dLong,
                      Period, 0.0 );
// 2nd wobbel vector
   set_wobbel_vector( -dTrans, dLong,
                      Period * 2, 0.0 );
// 3rd wobbel vector
   set_wobbel_vector( dTrans, dLong,
                      Period, 0.0 );
// if laser power variation is needed,
include here
// set_wobbel_control( Ctrl, Value,
                      MinValue, MaxValue );

// Activate freely definable wobbel shapes
   set_wobbel_mode( 100, 0, 100, 2 );

// Mark a vector
   timed_mark_rel( 1000, 0, 22*10 );
```

8.4.1 Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values

"Classic" Wobbel Shapes

"Classic" wobbel shapes are defined by **set_wobbel_mode** or **set_wobbel**.
Only assign hardware appropriate values to the `Transversal`, `Longitudinal` and `Freq` parameters.

> **Notice!**
>
> - Too big values define control situations where very high waste heat is produced. Galvanometer scanner and digital control board/amplifier board overheating and permanent damage may occur even in short-term operation (overload!). Take the highest possible dynamics of scan head and laser into account.
> - If the frequency values are too high the galvanometer scanners may not be able to follow the nominal curve. This may lead to unexpected marking results.

Rule of thumb to estimate appropriate maximum values:

(1) Maximum frequency: $F = 1/(10 \times T)$
where T = Tracking Error[1]

(2) Wobbel amplitude[2]: $A = T \times V$
where V = typical positioning speed[1]

Notes

- Also take the path velocity on the wobbel shape itself into account. For circular wobbel shapes it is calculated as follows:

(3) Path velocity $V_{Path} = 2 \times \pi \times A \times F$.

- Make sure that the combination of the used values and the trajectory[3] velocity are suitable for a long-term operation without causing damages (process safety!).
- Check the temperature status already during an evaluation period (see **get_head_status**) in order to recognize potential overload situations at an early stage. With iDRIVE Scan Systems you can also read out the present temperatures of galvanometer scanners and/or digital control boards (see **control_command**).

Example of Use

System Specification

- Tracking Error T = 0.33 ms.
- Calibration $\pm 0.349\ rad_{optical}$ $(= 10°_{mechanical})$ at ±503,316 bit.
  - This is an angle control AC of $\approx 1.44 \times 10^6\ bit/rad_{optical}$ $(\approx 50{,}300\ bit/°_{mechanical})$.
- Calibration factor[4] K = 5,000 bit/mm.
- Typical positioning speed $V_{rad} = 100\ rad_{optical}/s$.
  - Converted to control values this corresponds to a typical positioning speed of $V = V_{rad} \times AC$ $\approx 1.44 \times 10^8$ bit/s = 1,440 bit/10 µs.
  - In the Image Field this corresponds to a typical positioning speed of V/K =28.8 m/s.

(1) See technical specifications in the scan head manual.

(2) At the maximum frequency estimated with (1).

(3) See Glossary entry on page 31.

(4) See `*_ReadMe.txt` file of correction file or **get_table_para**.

Wobbel Parameters

- The maximum frequency estimated with rule of thumb 1:
  $F = 1/(10 \times 0.33\times10^{-3}\ s) \approx 300$ Hz.
- Wobbel amplitude estimated with rule of thumb 2:
  $A = 0.33\times10^{-3}\ s \times 1.44\times10^{8}$ bit/s
  $\approx 47{,}500$ bit.
  – In the Image Field this corresponds to a wobbel amplitude of $A/K \approx 9{,}5$ mm.
- Path velocity of circular wobbel shapes as given by rule of thumb 3 is:
  $V_{Path} \approx 895$ bit/10 μs.
- The *maximum* mark speed results from V_{Path} /K (in this example 89,500,000 bit/s / 5,000,000 bit/m ≈ 17.9 m/s. Note that this value is a rough estimate. Furthermore, it is dependent on a position due to the correction file (which is non-linear).
  The calculated path velocity for wobbel only (!)
  – should be lower than the specified maximum positioning speed,
  – but there may be certain circumstances where it is much higher than the *typical* mark speed.
  Make sure that your values are suitable for operation (temperature status and so on, same as above).

"Freely Definable Wobbel Shapes"

When defining "freely definable wobbel shapes" (see **set_wobbel_vector**) take the dynamics of the scan head into account.

- The maximum repetition rate for "Freely Definable Wobbel Shapes" corresponds to the maximum wobbel frequency estimated by the rule of thumb 1, page 243.
- The sum of the path (Trajectory) velocity and the velocity on the freely defined wobbel figure should not exceed the maximum positioning velocity.
- The acceleration should not significantly exceed the value of 2.5 V/S (where V = typical positioning speed and S = Tracking Error).
- Also with "freely definable wobbel shapes", observe the heating of the scan system for freely definable wobbel figures, see safety notice on page 243 and rule of thumb 3.

8.5 Controlling 2D Scan Systems and 3D Scan Systems

8.5.1 2D Scan Systems

1 RTC6 PCIe Board can control 1…2 2D scan systems.

The second 2D scan system requires the Option "Second Scan Head Control", page 39.

When controlling 2 2D scan systems, a separate Image Field correction can be carried out for each of them.

By **select_cor_table** or **select_cor_table_list**, the two correction tables are assigned to the respective scan head connector, see also Section "2. SCANHEAD Socket Connector", page 70).

To use two correction tables in a double scan system configuration

(1) Load each of the desired 2D correction files by `load_correction_file(Name, n, 2)` (`n` = 1…8).
See also Chapter 8.5.3 "Using Several Correction Tables", page 250.

(2) Assign a loaded 2D-correction table to the first and the second scan head each by calling `select_cor_table(HeadA, HeadB)` with (`HeadA` and `HeadB` = 1…8).

(3) By **set_offset**, **set_scale**, **set_angle** or the corresponding list commands, specify offset, scaling factor and rotation to align the two Image Fields precisely with respect to each other.

Notes

- If you are not using one of the scan head connectors: assign the correction table 0 to it.
- The default setting for **select_cor_table** and **select_cor_table_list** is `(1,0)`:
  - For the Connector for First Scan Head, correction table #1 is used
  - At the Connector for Second Scan Head, there are *no position outputs*
- The RTC6 PCIe Board returns to this setting after every **load_program_file**. If a different setting is to be used, **select_cor_table** or **select_cor_table_list** must be called again.
- The scan head connectors *cannot* be simultaneously assigned:
  - two 3D correction tables
  - a 2D correction table *and* a 3D correction table

8.5.2 3D Scan Systems

An RTC6 PCIe Board can also be used to control a 3-axis scan system[(1)]. This requires the Option "3D", page 39.

Intended Use

3-axis scan systems can be used for positioning the laser focus within a flat processing field without the need for a flat field objective. Therefore, they are frequently used in applications for which flat field objectives are not available.

3-axis scan systems can also be used as 3D beam deflection systems. Here, the laser focus is guided along the contour of the workpiece being processed, thus enabling workpiece processing in 3 dimensions.

SCANLAB offers dynamic focusing units[(2)] that extend xy *scan systems* to 3-axis scan systems.

Notice!

- The standard RTC6 DLL (`RTC6DLL.dll`) supports all RTC6 commands for controlling 3-axis scan systems.
  However, the full functionality of the RTC6 commands – the actual *output* of z coordinates – is only available with enabled Option "3D", page 39.

Connection and Initialization

In order to control a 3-axis scan system by an RTC6 PCIe Board:

- The Option "3D", page 39 of the RTC6 PCIe Board must be enabled (this can be checked by **get_rtc_version**)
- The xy scan system must be connected to the Connector for First Scan Head
- The z axis must be connected to the Connector for Second Scan Head
- A 3D correction table (`D3_*.CT5`) must exclusively be assigned to the Connector for First Scan Head (by **select_cor_table** or **select_cor_table_list**).
- If, in addition, the Option "Second Scan Head Control", page 39, is enabled, it is alternatively possible to connect:
  – the xy scan head to the Connector for Second Scan Head
  – the z axis to the Connector for First Scan Head
  – In this constellation the 3D correction table must be assigned to the Connector for Second Scan Head

See also Chapter 4.5 "Interfaces to Scan System", page 69 and Section "2D and 3D Correction Files", page 179.

Other than that, no additional drivers or software files are needed for controlling a 3-axis scan system. Even initialization and program launching remain unchanged, see Chapter 6.2 "Initialization and Program Start-Up", page 98.

Several 3-axis scan systems can be simultaneously controlled (by multi-board commands) from a single PC. This requires a corresponding number of RTC6 PCIe Boards with enabled Option "3D"s to be installed in that PC.

(1) Consisting of an xy scan system and a z axis dynamic focusing unit (see footnote 2) as z axis.

(2) See Glossary entry on page 27.

3D Commands

These are, for example, the following RTC6 commands:

- **[*]3d[*]**
- **goto_xyz**
- **set_offset_xyz** and **set_offset_xyz_list**

Except for the additional motion in the third dimension, the 3D commands function identically to their corresponding 2D commands:

- Specified vectors and arcs are split-up into Microsteps, see Chapter 7.1.2 "Microstepping", page 143
- The jump speed and mark speed are specified by **set_jump_speed** and **set_mark_speed**, see Chapter 7.1.1 "Marking with Vector Commands and "Arc" Commands", page 139. As for timed Vector Commands, the speeds are automatically determined from the specified jump or marking duration
- 3D Image Field correction is applied in accordance with the 3D correction table assigned by **select_cor_table** or **select_cor_table_list**, see Chapter 7.3.5 "Image Field Correction and Correction Tables", page 178
- For Jump Commands and [*]mark[*] Commands, the Laser Control Signals are switched on and off while taking delay settings into account, see Chapter 7.2 "Delay Settings – Coordinating Scan Head Control and Laser Control", page 148

For the value range of the data, see Section "Compatibility Modes", page 173.

The calibration factor K_{xy} can be read out from the correction table used by **get_table_para**.

In RTC4 Compatibility Mode and RTC5 Compatibility Mode, the 3 coordinate values are always scaled up to 20-bit values internally, so that the three spatial directions are treated equally with regard to the jump speed or mark speed.

The 3 coordinate values are internally always scaled up to 20-bit values, so that the three spatial directions are treated equally with regard to the jump speed or mark speed.

With big z jumps, observe the different dynamics of vario*SCAN* and 2D scan systems.

Notes

- After Vector-Defined Laser Control has been activated by **set_vector_control** (`Ctrl` = 7), the para-Mark Command and para-Jump Commands can be used to set a dynamic focus shift linearly along the mark or jump vector, see Section "Vector-Defined Laser Control", page 214. See also **set_defocus** or **set_defocus_list**.
- The size of the usable Image Field(1) and the focus shift in the z direction(2) (height of the usable 3D Image Field) can be obtained from the `*_ReadMe.txt` file supplied with the 3D correction file as well as from the user manual of the 3-axis scan system/vario*SCAN* ("Technical Specifications" chapter).
- If a z axis is to be used to hold the laser focus only in a certain plane (especially in 3D systems without a lens), then even 2D vector commands can be used. 2D vector commands leave the z position previously set with a 3D vector command unchanged and only adjust the focus length.
- If the Option "3D", page 39 is not enabled or no 3D correction table has been assigned:
  - The split-up into Microsteps is calculated including the z axis (which influences the effective jump speed and mark speed in the xy plane.
  - There is merely no output for the z axis

(1) Line "Max. Field Size (z=0): nnn.nnn mm".

(2) Line "Max. Z-Range: +/- nn.n mm".

- By **set_offset_xyz** or **set_offset_xyz_list**, an offset can be defined for the z coordinate. In addition, **set_defocus** or **set_defocus_list** can be used for defining an offset to the calculated focal length, which then appropriately affect the z output value (in the direction opposite to the offset).
- During a marking process, the scan system focal length is continuously readjusted. During execution of a Jump Command (or **goto_xyz**) this readjustment can be switched off. With `DirectMove3D` = 1 in **set_delay_mode** or **set_delay_mode_list**, the z output value is directly (linearly) taken to its final value during the jump.

Adjusting Tracking Errors

If the xy scan head and the Dynamic Focusing Unit unit of a 3D scan system have different Tracking Errors, **set_timelag_compensation** can be used to *delay* the output of the faster component to that of the slower component.

The laser control signal output is synchronized with the slower component. This allows, for example, Sky Writing with 3D commands without permanent defocusing.

In case of a SCANahead System, the `PreviewTime` parameter from **set_scanahead_params** is automatically used as “Tracking Error“.

Enhanced 3D Correction

- For more information on the actual procedure, refer to the "Calibrating a 3-Axis Laser Scan System" Manual, Chapter 5.4 "Step 4: Correcting Stretch".

The image size of 3D scan systems without objectives or with non-telecentric F-Theta objectives depends on its distance to the scan head objective (z coordinate).

The Image Field size typically gets stretched or squeezed linearly with the z value. Therefore, SCANLAB 3D correction files include stretch correction factors to compensate for this effect, see Section "ct5 Correction File Header", page 183.

With some objectives, the typical stretching is combined with a change in the image geometry. Then additional stretch corrections are necessary which compensate the image geometry changes xy position-dependent but linear in z (2D stretch correction table). The stretch correction values are meaning the Z gradient for the location deviation at this point.

Assumption: for any chosen non-zero Z plane, (X, Y) is the desired position and (X', Y') the measured position. The corrections `<StretchX>` and `<StretchY>` are then calculated as follows:

```
<StretchX> = (X-X')/Z
<StretchY> = (Y-Y')/Z
```

You can then use **load_stretch_table** to load the enhanced 3D correction onto the RTC6 PCIe Board from an ASCII text file and assign it to an already loaded 3D-correction table. This ASCII text file must contain corrections for a complete rectangular grid. The number of gridlines and their spacings can be freely defined and even differ in X and Y. If the absolute values of the corrections exceed 0.03125, then they are clipped to this limit value. The corrections are bilinearly interpolated for data points within the specified grid and linearly extrapolated for data points outside the grid.

Enhanced 3D correction is not active by default after **load_program_file**. It becomes active upon loading a valid table onto the RTC6 PCIe Board. It can be deactivated by calling **load_stretch_table** using a NULL pointer instead of the filename.

For the 2D stretch correction tables, the following rules apply:

- The ASCII text file can contain one or several tables.(1)
- The 2D stretch correction table must begin with the line:
  `[StretchTable<No>]`
  `<No>` represents the table number to be specified by **load_stretch_table**.
- This is directly followed by a block of data points.
- If several tables with the same number exist, then only data from the first encountered table is read. The others are ignored.
- Reading of the text data terminates upon end of file or upon a line containing a caption.
- All characters to the right of a semicolon are treated as comments and ignored.
- The order of data points is up to you.
- The maximum length of a data line is 255 characters.
- A data line contains two position coordinates (in bits, signed integers) and two correction values (dimensionless, signed floating point numbers, use the period `(.)` or comma as the decimal separator), each separated by spaces or tabs:
  `<Xpos> <Ypos> <StretchX> <StretchY>`
- If a data point reoccurs, then the most recently read value is used.
- Empty lines or incomplete data lines are invalid and are ignored.

(1) Even of another type, see table 1, page 156.

8.5.3 Using Several Correction Tables

The RTC6 PCIe Board memory can store 8 different correction tables at the same time.

It can be useful to work with several different correction tables even if only a single scan system (2D or 3D) is used. Example: a pilot laser and a processing laser with different wavelengths are used.

Special applications sometimes also require different correction tables in quick succession. This change can be done, for example, by `select_cor_table( n, 0 )` or `select_cor_table_list( n, 0 )` (`n` = 1…`number_of_correction_tables`).

number_of_correction_tables serves to protect other commands (for example, **load_correction_file** and **select_cor_table**) from unwanted table numbers.

There is no need to change existing user programs except when user input is to be rejected in the future (by means of explicit RTC6 error messages).

8.6 Processing-on-the-fly

- "Processing-on-the-fly" means the combination of workpiece motion and scan system motion.
- The use of the *Processing-on-the-fly* functionality requires the Option Processing-on-the-fly.
- Whether the Option Processing-on-the-fly is enabled can be checked by **get_rtc_version**.
- Chapter 8.6.12 ""Fly Extension" Commands", page 268
- Available are (not mixable):
  - "Classic" Processing-on-the-fly commands[1]
  - "Fly Extension" Commands, page 268[2]

8.6.1 Intended Use and Initialization

With its Option Processing-on-the-fly enabled, the RTC6 PCIe Board allows processing of parts in motion (for example, parts on a conveyor belt, rotating plate or xy positioning stage), as well as stationary parts with a moving scan system (for example, by a robot arm).

To adjust laser scan processes to the current workpiece position relative to the scan system, the position of the workpiece or scan system can be forwarded to the RTC6 PCIe Board:

- indirectly by encoder counters
- directly by the McBSP interface

If the Processing-on-the-fly correction is active, the coordinate values of all Vector Commands and "Arc" Commands are transformed based on the forwarded position values.

Upon forwarding the motion as encoder pulses, RTC6-internal encoder counters are triggered. The counter values correspond to the current position. They get a scaling factor (from certain RTC6 commands) assigned and are then used as *Processing-on-the-fly* correction.

See also Chapter 9.3.3 "Synchronization by Encoder Signals", page 320.

Upon forwarding the position values via the McBSP interface, the input values get a scaling factor assigned and are then used as *Processing-on-the-fly* correction.

See also Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 322.

Simultaneous usage of both forwarding methods for Processing-on-the-fly correction of two independent motions is not possible, see Section "Overview", page 251.

The McBSP interface cannot be simultaneously used for a Processing-on-the-fly application and an Online Positioning, see Notes, page 239.

Processing-on-the-fly correction is activated and deactivated by list commands. The parameters required for activation may have to be determined beforehand by a calibration process (see below).

If both scan head connectors each have a 2D correction table assigned, then a Processing-on-the-fly correction has the same effect at both scan head connectors.

For the z axis, a Processing-on-the-fly correction can be activated as well, see Chapter 8.6.11 "Processing-on-the-fly Correction for the z Axis", page 266.

Overview

The following encoder-based Processing-on-the-fly corrections of motions are available:

- **set_fly_x**, **set_fly_y**, even in combination
  To compensate linear motions of the workpiece (for example, by conveyor belt or xy positioning stage), see Chapter 8.6.2 "Compensating Linear Motions", page 252.
- **set_fly_rot**
  To compensate rotary motions of the workpiece (for example, by rotating plate), see Chapter 8.6.3 "Compensating Rotary Motions", page 256.
- **set_fly_2d**
  To compensate 2D motions of the workpiece (for example, xy positioning stage), see Chapter 8.6.4 "Compensating 2D Motions", page 258.

(1) See "Classic" Processing-on-the-fly Control Commands, page 328 and "Classic" Processing-on-the-fly List Commands, page 333.

(2) See "Fly Extension" List Commands, page 333.

The following McBSP-based Processing-on-the-fly corrections of motions are available:

- **set_fly_x_pos**, **set_fly_y_pos** even in combination
  To compensate 1D or 2D motions of the scan system (for example, robot arms), see Chapter 8.6.2 "Compensating Linear Motions", page 252.
- **set_fly_rot_pos**
  To compensate rotary motions of the scan system (for example, rotary table), see Chapter 8.6.3 "Compensating Rotary Motions", page 256.
- **set_mcbsp_in**, **set_mcbsp_in_list**
  To compensate 1D or 2D motions or rotary motions (for example, robot arms, rotary table), see Chapter 8.6.2 "Compensating Linear Motions", page 252 and Chapter 8.6.3 "Compensating Rotary Motions", page 256.

The following Processing-on-the-fly corrections can be combined:

- **set_fly_x** with **set_fly_y**.
- **set_fly_x_pos** with **set_fly_y_pos**
- For the special case **set_fly_z**, see Chapter 8.6.11 "Processing-on-the-fly Correction for the z Axis", page 266
- For linear 3D Processing-on-the-fly corrections with positional values, see Section "Correction via McBSP Interface with Enhanced McBSP Input", page 255.

The following Processing-on-the-fly corrections *cannot* be combined:

- Encoder-based with McBSP-based
- linear and rotating

The last command always determines the overall correction unless it can be combined with previous settings.

However, mutual synchronization of any Processing-on-the-fly corrections by **wait_for_encoder_mode**, **wait_for_encoder_in_range** and **wait_for_mcbsp** is possible, see Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 261.

8.6.2 Compensating Linear Motions

Processing-on-the-fly correction for linear motions (translations) can be activated(1) by:

- **set_fly_x** and/or **set_fly_y**
- **set_fly_x_pos** and/or **set_fly_y_pos**
- **set_mcbsp_in** (`Mode` = 1…3)
- **set_mcbsp_in_list** (`Mode` = 1…3)

A scaling factor must thereby be specified in each case.

With **set_multi_mcbsp_in** or **set_multi_mcbsp_in_list**, you can activate additional Processing-on-the-fly correction with positional values for linear motion in all three coordinate directions without bit-resolution restrictions. No scaling factor is required.

Processing-on-the-fly correction can be deactivated (simultaneously for both directions) by **fly_return**. See the corresponding notes in Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 260.

Correction via Encoder Counters

To be able to pass position values via the board-internal encoder counters(2), the Processing-on-the-fly correction must be activated by:

- **set_fly_x** and/or **set_fly_y**

Here, the scaling factor [in bits per count] defines the relation between the shift [in bits] of the current output position in the Image Field and one counter pulse (count) of the corresponding encoder counter. See also Section "Determining the Scaling Factors", page 253.

By **set_fly_x**, the encoder counter "Encoder0" is reset to zero. By **set_fly_y** the encoder counter "Encoder1" is reset to zero.(3)

(1) Different Processing-on-the-fly corrections cannot be combined arbitrarily, see the Section "Overview", page 251.

(2) See Notice!, page 59.

(3) By **set_control_mode** `Bit #9` it can be set beforehand whether the counter is reset at the respective Processing-on-the-fly command or at the start trigger.

Thereafter, the output value is calculated from the current output position by adding (for all spatial directions) the product of the scaling factor and the current counter value.

This correction takes place every 10 μs.[1]

Notes

- **set_fly_x** and **set_fly_y** can be used in combination for 1D as well as for 2D Processing-on-the-fly applications (for example, an xy positioning stage), particularly for separate or separable marking tasks in the real Image Field.
- For continuous marking in the virtual Image Field, SCANLAB recommends to preferably use **set_fly_2d**, see Chapter 8.6.4 "Compensating 2D Motions", page 258.

Determining the Scaling Factors

The scaling factors can be determined experimentally:

(1) Read out the counter start value by **get_encoder**.

(2) Start the motion.

(3) Stop the motion.

(4) Read out the counter end value by **get_encoder**[2].

(5) Measure the distance traveled in mm.

(6) Calculate the encoder increment i as follows:

$$i = \frac{(\text{counter end value} - \text{counter start value})}{\text{distance travelled}}$$

(7) Calculate the scaling factors `Scalex` and `Scaley` [in bits per count] as follows:

$$\texttt{Scalex} = \frac{K}{i_x}$$

$$\texttt{Scaley} = \frac{K}{i_y}$$

Whereby K is the calibration factor [in bits per mm], see Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 172 and i is the encoder increment from Step 6.

If the workpiece moves at a constant speed v_x or v_y [in mm per second] and an encoder simulation has been activated by **simulate_encoder**, then the scaling factors are calculated as follows:

$$\texttt{Scalex} = \frac{K \times v_x}{(1.000.000 \text{ counts / s})}$$

$$\texttt{Scaley} = \frac{K \times v_y}{(1.000.000 \text{ counts / s})}$$

(8) Adjust the signs of the scaling factors according to the motion direction.

(1) The encoder counters are signed 32-bit counters with overflow (= after reaching the maximum (minimum) counter value, counting continues at the minimum (maximum) counter value).

(2) Alternatively, the counter start and end values can be stored in a cache on the RTC6 PCIe Board by the list command **store_encoder** and then retrieved from there by the control command **read_encoder**.

Correction via McBSP Interface

To be able to pass position values via the McBSP interface, the Processing-on-the-fly correction must be activated by:

- **set_fly_x_pos** and/or **set_fly_y_pos**
- **set_mcbsp_in** for positional values in 2 spatial directions
- **set_multi_mcbsp_in** for positional values in all 3 spatial directions (requires no scaling factors)

Here, the scaling factor [in bits per bits] defines the relation between the shift [in bits] of the current output position in the Image Field and the input value [in bits] at the McBSP interface. See also Section "Determining the Scaling Factors", page 254.

Thereafter, the output value is calculated from the current output position by adding (for all spatial directions) the product of the scaling factor and the current input value at the McBSP interface.

This correction takes place every 10 μs.

At the McBSP interface, see Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87, there are available:

- For 1D corrections – signed 32-bit values
- For 2D corrections – signed 16-bit values per axis (the y value is in the lower 16 bits and the y value in the upper 16 bits of the value at the McBSP interface)

Notes

- After **set_fly_x_pos** and **set_fly_y_pos**, the input values received at the McBSP interface automatically get copied to internal memory location 0. This still applies even after Processing-on-the-fly correction gets switched off by **fly_return**, **set_fly_x_pos**, **set_fly_y_pos** or **set_fly_rot_pos**, as well as after **load_program_file**. The current data at memory location 0 can be queried by `read_mcbsp(0)`.
- In contrast, activation of Processing-on-the-fly correction by **set_mcbsp_in** or **set_mcbsp_in_list** also result in McBSP input values being copied to internal memory locations 1, 2 and/or 3, see Section "Correction via McBSP Interface with Additional McBSP Input", page 255.
- After Processing-on-the-fly correction is activated by **set_multi_mcbsp_in** or **set_multi_mcbsp_in_list**, the transmitted data gets consecutively written to memory locations 0…3. From there, they can be read out unsorted by **read_mcbsp** or sorted by **read_multi_mcbsp**.

Determining the Scaling Factors

The scaling factors can be determined experimentally:

(1) Read out the start input value by **read_mcbsp**.

(2) Start the motion.

(3) Stop the motion.

(4) Read out the end input value by **read_mcbsp**.

(5) Measure the distance traveled in mm.

(6) Calculate the position increment i as follows:

$$i = \frac{(\text{end input value} - \text{start input value})}{\text{distance travelled}}$$

(7) Calculate the scaling factors `Scalex` and `Scaley` [in bits per bits] as follows:

$$\texttt{Scalex} = \frac{K}{i_x}$$

$$\texttt{Scaley} = \frac{K}{i_y}$$

Whereby K is the calibration factor [in bits per mm], see Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 172 and i is the encoder increment from Step 6.

Correction via McBSP Interface with Additional McBSP Input

To activate Processing-on-the-fly correction for linear motions using McBSP input values (as an alternative to **set_fly_x_pos** or **set_fly_y_pos**), you can also call **set_mcbsp_in** or **set_mcbsp_in_list** (`Mode = 1…3`).

These commands offer the advantage of using the McBSP interface to input additional desired signals that should not be subjected to Processing-on-the-fly correction even when it is activated.

For this, all McBSP input values must be coded by Bit #31.

- Bit #31 = 0: The input value gets copied to internal memory location 0 and is applied for Processing-on-the-fly correction.
- Bit #31 = 1: The input value gets copied to internal memory location 3 but is not applied for Processing-on-the-fly correction.

Notes

- All input values always get copied alternatingly to internal memory locations 1 and 2 and subsequently, in accordance with their Bit #31 coding, to internal memory locations 0 and 3. But after deactivation of correction by `set_mcbsp_in(0)` or `set_mcbsp_in_list(0)`, they only get copied to memory locations 1 and 2.
- You can query the data currently stored at internal memory locations 0 - 3 by **read_mcbsp**.
- A scaling factor is specified at **set_mcbsp_in** or **set_mcbsp_in_list**. For 2D corrections (`Mode = 3`), the scaling factor applies to both axes simultaneously. Calibration for determining the scaling factor is the same as for **set_fly_x_pos** and **set_fly_y_pos** (see above).
- For transferring 1D correction values, 31 bits with sign are available (Bit #31 is reserved as the coding bit). In contrast, only 15 bits with sign are available per axis for transferring 2D correction values, whereby the x value lies in the lower 16 bits and the y value in the upper 16 bits of the value at the McBSP interface. For a description of the interface, see Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87.

Correction via McBSP Interface with Enhanced McBSP Input

As an alternative to the previous chapter's methods, you can also activate Processing-on-the-fly correction of linear motion with **set_multi_mcbsp_in** or **set_multi_mcbsp_in_list**.

These commands have an advantage over **set_mcbsp_in** or **set_mcbsp_in_list** in that they offer Processing-on-the-fly correction not only in the x direction and y direction, but also in the z direction and with laser power variation. Furthermore, up to 4 additional signals can be transmitted for usage as you wish.

Each 10 µs, data asynchronously transmitted to McBSP memory locations 0 through 3 gets sorted and copied to an additional internal memory location. From there it is available for final usage and can be read out sorted by signal types using **read_multi_mcbsp**.

8.6.3 Compensating Rotary Motions

Before activating Processing-on-the-fly correction for rotary xy motion, you must define the rotation center by **set_rot_center** or **set_rot_center_list**. The rotation center may also be situated outside the Image Field.

The Processing-on-the-fly correction itself can be activated by:

- **set_fly_rot**
- **set_fly_rot_pos**
- **set_mcbsp_in** (`Mode = 4`)
- **set_mcbsp_in_list** (`Mode = 4`)

Processing-on-the-fly correction can be stopped by **fly_return** (see the corresponding notes on Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 260).

Notes

- A Processing-on-the-fly rotation correction:
  - Cannot be combined with other Processing-on-the-fly corrections, see Section "Overview", page 251.
  - Simultaneously for both scan head connectors is only practical, if both attached scan systems are aligned to exactly the same rotation center.

Correction via Encoder Counter

To be able to pass rotation angles via the board-internal incremental encoder, encoder input port ENCODER X[1] must be connected, see also Section "Input Ports for External Encoder Signals", page 321.

Then, Processing-on-the-fly correction must be activated by **set_fly_rot**. The parameter `Resolution` [in counts per revolution] must thereby be specified. See also Section "Determining the Resolution Parameter", page 256.

set_fly_rot resets the encoder counter "Encoder0" to zero.[2]

The output value is calculated from the current output position via a rotation matrix around the specified rotation center with rotation angle / 360° = current "Encoder0" counter reading / `Resolution`).

This correction is performed every 10 µs.[3]

Determining the Resolution Parameter

The `Resolution` parameter an be determined experimentally:

(1) Read out the counter start value by **get_encoder**.

(2) Start the rotation.

(3) Count the number of revolutions.

(4) Stop the rotation.

(5) Read out the counter end value by **get_encoder**.[4]

(6) Calculate the parameter `Resolution` [in counts per revolution] as follows:

$$\texttt{Resolution} = \frac{(\text{counter end value} - \text{counter start value})}{\text{number of revolutions}}$$

`Resolution` = (counter end value – counter start value) / number of revolutions

If the workpiece rotates at a constant speed ω [in number of revolutions per second] and an encoder simulation has been activated by **simulate_encoder**, then the `Resolution` parameter can be calculated as follows:

$$\texttt{Resolution} = \frac{(1.000.000 \text{ counts / s})}{\omega}$$

(7) Adjust the sign of `Resolution` to the direction of rotary motion.

(1) ENCODER X = Encoder0.

(2) By **set_control_mode** `Bit #9` it can be set beforehand whether the counter is reset at the respective Processing-on-the-fly command or at the start trigger.

(3) The encoder counters are signed 32-bit counters with overflow (= after reaching the maximum (minimum) counter value, counting continues at the minimum (maximum) counter value).

(4) Alternatively, the counter start and end values can be stored in a cache on the RTC6 PCIe Board by the list command **store_encoder** and then retrieved from there by the control command **read_encoder**.

Correction via McBSP Interface

If angle-position values for Processing-on-the-fly correction of rotary motion are forwarded by the McBSP interface, then Processing-on-the-fly correction must be activated by **set_fly_rot_pos**.

The required `Resolution` parameter has the same meaning as with **set_fly_rot** and is similarly determined, see Section "Determining the Resolution Parameter", page 256.

McBSP input values are read out by **read_mcbsp**.

Notes

- McBSP input values get copied to board-internal memory location 0, see notes in Section "Correction via McBSP Interface", page 254.

Correction via McBSP Interface with Additional McBSP Input

A Processing-on-the-fly correction for rotary motions with McBSP input values can be activated by:

- **set_mcbsp_in** (`Mode = 4`)
- **set_mcbsp_in_list** (`Mode = 4`)

These commands offer the advantage of using the McBSP interface to input additional desired signals that should not be subjected to Processing-on-the-fly correction even when it is activated.

For this, all McBSP input values must be coded by Bit #31.

Notes

- All input values always get copied alternatingly to internal memory locations 1 and 2 and subsequently, in accordance with their Bit #31 coding, to internal memory locations 0 and 3. But after deactivation of correction by `set_mcbsp_in(0)` or `set_mcbsp_in_list(0)`, they only get copied to memory locations 1 and 2.
- You can query the data currently stored at internal memory locations 0 - 3 by **read_mcbsp**.
- By **set_mcbsp_in** or **set_mcbsp_in_list**, a rotation resolution can be specified. Calibration for determining the rotation resolution is the same as for **set_fly_rot_pos**, see Section "Determining the Resolution Parameter", page 256.
- For transferring rotary correction values, 31 bits with sign are effectively available.

8.6.4 Compensating 2D Motions

You can use the **set_fly_2d** command to activate encoder-based 2D Processing-on-the-fly correction (for example, for a xy positioning stage), particularly for continuous marking in the virtual Image Field.

Compared to an activation by **set_fly_x** and **set_fly_y**, this has the following advantages:

- **set_fly_2d** simultaneously resets both encoders (whereas the separate commands **set_fly_x** and **set_fly_y** do so with a slight temporal offset between both channels)
- **set_fly_2d** allows compensation of non-linear relations between encoder values and actual xy positioning stage motions, see Section "2D Encoder Compensation for xy Positioning Stages", page 258
- Even during an interruption of a **set_fly_2d** marking by **wait_for_encoder_mode** or **wait_for_encoder_in_range**, the galvanometer scanner positions receive continuous Processing-on-the-fly correction in accordance with the current xy positioning stage encoder values (whereby the laser focus remains stationary relative to the xy positioning stage). Thus, unnecessary jumps are avoided after xy positioning stage forwarding motions, see Chapter 8.6.6 "Virtual Image Field with Processing-on-the-fly", page 261

Notes

- A **set_fly_2d** correction *cannot* be combined with other Processing-on-the-fly corrections, see Section "Overview", page 251.
- Coordinate transformations within the virtual Image Field, see Section "Coordinate Transformations in the Virtual Image Field", page 174 can be activated when starting a Processing-on-the-fly Session with changed values, if they have been loaded before with the corresponding control commands.

2D Encoder Compensation for xy Positioning Stages

For particularly demanding marking requirements, it might be necessary to also compensate mechanical deviations of a xy positioning stage.

For this purpose, **load_fly_2d_table** can be used to load up to two 2D compensation tables, see page 259, onto the RTC6 PCIe Board.

Then – during a Processing-on-the-fly application initiated by **set_fly_2d** – encoder values are 2D interpolated and compensated just like with field correction tables.

To avoid unnecessary initialization motions, you can use **init_fly_2d** to define a desired positioning stage start position as the reference value for 2D encoder compensation (and to store it on the RTC6 PCIe Board) and to select simultaneously one of the both correction tables.

Upon every subsequent encoder reset by **set_fly_2d**, the current position is automatically applied as the new reference position, so that the relation between current encoder values and the absolute position of the positioning stage is retained for compensation. This relation remains even upon termination with **fly_return** or upon a new start with **set_fly_2d**.

This relation does not remain, if you meanwhile activate other Processing-on-the-fly corrections or reset the encoders by a /START (see **set_control_mode** (`Bit #9` = 1).

At any time, you can query the currently valid reference values by **get_fly_2d_offset**.

Encoder compensation is applied exclusively to Processing-on-the-fly correction. The original uncompensated encoder values use:

- **get_encoder**
- **store_encoder**
- **read_encoder**
- **wait_for_encoder**
- **wait_for_encoder_mode**
- **wait_for_encoder_in_range**
- Recording by **set_trigger** (`Signal1`/`Signal2` = 43 or 44)
- Recording by **get_value**

For load_fly_2d_table, you need to provide an ASCII text file that contains a 2D compensation table.

A 2D compensation table must have encoder compensations for reference points on a rectangular grid. The number of grid lines and their spacing is up to you (both may also differ in x and y). Missing reference point data is automatically assigned a compensation of 0. The largest occurring encoder reference points form a frame within which encoder compensation values are bilinearly interpolated. Encoder values outside this frame is clipped to this frame prior to interpolation. Therefore, the reference points should cover at least the range of the xy positioning stage required by your application. Reference points with values exceeding ±524,288 can result in some loss of precision.

After load_program_file, 2D encoder compensation is inactive by default.

It becomes active as soon as a valid 2D compensation table from an ASCII text file gets loaded onto the RTC6 PCIe Board. You can subsequently deactivate it by calling load_fly_2d_table using a NULL pointer instead of the filename.

For the 2D compensation tables, the following rules apply:

- The ASCII text file can contain one or several tables.(1)
- The 2D compensation table must begin with the line:
  `[Fly2DTable<No>]`
  `<No>` stands for the table number `No`, which must be specified with load_fly_2d_table. The first of the two tables is loaded with `No` = `<No>` and the second with `No` = `<No>` + 65,536.
- This is directly followed by a block of data points.
- If several tables with the same number exist, then only data from the first encountered table is read. The others are ignored.
- Reading of the ASCII text file terminates upon end of file or upon a line containing a caption.
- All characters to the right of a semicolon are treated as comments and ignored.
- The order of data points is up to you.
- The maximum length of a data line is 255 characters.
- A data line contains the two reference point coordinates for "Encoder0" and "Encoder1" (signed integers) and two compensation values (signed integers), each separated by spaces or tabs:
  `<Encoder0> <Encoder1> <Encoder0-delta> <Encoder1-delta>`
- If a data point reoccurs, then the most recently read value is used; the others are ignored.
- Empty lines or incomplete data lines are invalid and are ignored.

(1) Even of another type, see table 1, page 156.

8.6.5 Deactivating Processing-on-the-fly Correction

All Processing-on-the-fly corrections can be deactivated (simultaneously for both spatial directions) by **fly_return**.

fly_return requires a new output position to be specified, which then is reached by a normal jump.

Processing-on-the-fly correction that has been activated by **set_multi_mcbsp_in** should be terminated by **fly_return_z**.

In all other cases (see below) a "Hard Jump" to a new output position may occur.

Notes

- If Processing-on-the-fly correction is not explicitly deactivated[1], then:
  - it is also effective during execution of subsequent lists
  - it is not effective in the pause between two lists[2]
- Processing-on-the-fly correction enabled by **set_fly_x**, **set_fly_y**, **set_fly_2d**, **set_fly_rot**, **set_fly_x_pos**, **set_fly_y_pos** or **set_fly_rot_pos** also gets deactivated if the same command is called again but with invalid parameter values. This could lead to a jump to an uncorrected output position (also refer to the command descriptions).
- If Processing-on-the-fly has been activated by **set_fly_x**, then **set_fly_y** *does not* deactivate it and vice versa. The same applies to **set_fly_x_pos** and **set_fly_y_pos**. Other than that, every Processing-on-the-fly command automatically deactivates any Processing-on-the-fly correction activated by another Processing-on-the-fly command, see also Section "Overview", page 251. This could lead to a "Hard Jump" to a new output position.
- Processing-on-the-fly correction activated by **set_mcbsp_in** or **set_mcbsp_in_list** also gets deactivated if you call **set_mcbsp_in** with `Mode` = 0 or **set_mcbsp_in_list** with `Mode` = 0. This could lead to a "Hard Jump" to an uncorrected output position
- Processing-on-the-fly correction activated by **set_fly_x_pos**, **set_fly_y_pos**, **set_fly_rot_pos**, **set_mcbsp_in** or **set_mcbsp_in_list** also gets deactivated if you call the command for configuring Online Positioning, see Section "Notes", page 239. This could lead to a "Hard Jump" to a new output position.
- A deactivation of a Processing-on-the-fly correction occurs only after the Scanner Delay.
- If Processing-on-the-fly has been activated by **set_mcbsp_in** or **set_mcbsp_in_list**, then the **fly_return** command deactivates Processing-on-the-fly correction, but it does not deactivate copying to internal memory locations (just like with `set_mcbsp_in(5)`). To afterward also deactivate copying to internal memory locations, you can use `set_mcbsp_in(0)` or `set_mcbsp_in_list(0)`.
- Processing-on-the-fly correction also gets deactivated (for both spacial directions simultaneously) by **stop_execution** or an External Stop.

(1) **set_end_of_list** does not deactivate Processing-on-the-fly correction.

(2) Therefore, the correction does not affect the control commands **goto_xy** and **goto_xyz**.

8.6.6 Virtual Image Field with Processing-on-the-fly

With Processing-on-the-fly applications, the full value range (29-bit) of the virtual Image Field can be used, see Chapter 7.3.3 "Virtual Image Field", page 174 to load and process command lists with objects that are up to 512 times larger than the real 20-bit Image Field.

With 1D Processing-on-the-fly applications (for example, workpieces on a conveyor belt), objects can only exceed the real Image Field in the very dimension parallel to the Processing-on-the-fly direction.

With 2D Processing-on-the-fly applications (for example, with an xy positioning stage), the to-be-marked objects of a command list may be distributed across the entire virtual Image Field.

If an entire marking task consists of several partial markings ("tiles") whose extent does not exceed the real Image Field, the Processing-on-the-fly functionality can also only be used to move the partial marking to be marked into the real Image Field and then process it there.

Coordinate values which are still outside the real image *after* Processing-on-the-fly correction are clipped to the boundary values of the real Image Field Here, the **get_marking_info** error bits get set automatically, see Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 264.

If there is a risk of the real 20-bit Image Field being exceeded during list execution, specifically a forwarding motion of the conveyor belt or xy positioning stage should be executed. The forwarding motion can be waited for with **wait_for_encoder**, **wait_for_encoder_mode**, **wait_for_encoder_in_range** or **wait_for_mcbsp** by interrupting the list execution until certain encoder values or McBSP values are reached, see Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 261.

The appropriate break up of the entire marking task and synchronization with the 1D or 2D Processing-on-the-fly motion is the task of the user program.

8.6.7 Synchronizing Processing-on-the-fly Applications

Processing of command lists can be started by either a RTC6 command or an external start signal, see Chapter 6.4.4 "Starting and Stopping Lists", page 112.

If the command list is not to be started immediately with the start signal, then an appropriate delay may be implemented as follows:

- When transferring positions via the McBSP interface – by a **wait_for_mcbsp** at the beginning of the command list.
- When transferring positions via encoder counters – by a **set_fly_x**/**set_fly_y**, **set_fly_2d** or **set_fly_rot** (to reset the counter(s)) and a subsequent **wait_for_encoder_mode** or **wait_for_encoder_in_range** at the beginning of the command list.
  wait_for_encoder_in_range is useful for 2D encoder-based Processing-on-the-fly applications. **wait_for_encoder_in_range** waits until both encoders are within the specified range at the same time and thus does not depend on the actual Trajectory that used to reach this range(1)
  When transferring positions via encoder counters, a track delay can be configured (even independently of an active Processing-on-the-fly Session) to delay the execution of the actual start relative to the triggering start signal or RTC6 command, see Section "External Start", page 312.

(1) If you would use **wait_for_encoder** or **wait_for_encoder_mode** instead, you would need to call these commands individually and consecutively for each encoder. Here, simultaneous fulfillment of both encoder criteria would depend on the xy positioning stage motion's explicit Trajectory and you would need to define and reproduce an appropriate Trajectory even before loading the lists.

External Starts triggered by an external start signal (or by **simulate_ext_start** or **simulate_ext_start_ctrl**) that do not execute immediately because of the track delay setting are held in a queue that can accommodate up to 8 starts (each start trigger is automatically generated when the delay has expired, see also Section "External Start", page 312). This helps avoid dead time between the execution of multiple (Processing-on-the-fly) list programs.

Moreover, the list command **simulate_ext_start** can be used to execute several lists at defined intervals.

If **wait_for_encoder**, **wait_for_encoder_mode**, **wait_for_encoder_in_range** and **wait_for_mcbsp** are used to interrupt a list for intermediate forwarding motions, see Chapter 8.6.6 "Virtual Image Field with Processing-on-the-fly", page 261, then the Signals for "Laser Active" Operation are not changed before the forwarding motion.

With encoder-based Processing-on-the-fly applications, the laser focus with **park_position** can be moved to a safe parking position before an interruption and back to the starting position or any other position after an interruption with **park_return**. See command description for more details.

The behavior of the galvanometer scanners during such a forwarding motion depends on the Processing-on-the-fly correction used:

- With McBSP-based Processing-on-the-fly correction as well as the encoder-based Processing-on-the-fly corrections **set_fly_x**, **set_fly_y** and **set_fly_rot**, the galvanometer scanners are stationary relative to the real Image Field during list interruption. As a rule, a jump to the next marking must then be made.
- With the encoder-based **set_fly_2d** Processing-on-the-fly correction, the positions of the galvanometer scanner are continuously Processing-on-the-fly-corrected (the laser focus thus remains stationary relative to the to-be-marked object). The subsequent jump to the next marking or continuation of the same is usually very small.
- With **set_fly_2d**, clipping might occur if the Processing-on-the-fly-corrected coordinate values exceed the real Image Field during a long forwarding motion. To avoid this, it might make sense to switch off Processing-on-the-fly correction before the forwarding motion (by **fly_return**, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 260; the galvanometer scanners then remain stationary). To switch correction back on after the forwarding motion, you can use **activate_fly_2d** instead of **set_fly_2d** (or **activate_fly_xy** instead of **set_fly_x** and **set_fly_y**). These commands do not reset the encoders, but instead convert the last Processing-on-the-fly-uncorrected coordinate values such that the Processing-on-the-fly-corrected output matches the current output. If an error occurs when restarting with these commands (for example, because the recalculated coordinate values would then be outside the 29-bit virtual Image Field, or because another Processing-on-the-fly mode has been activated in the meantime), then an error bit gets set. This error bit can be read out by **get_marking_info**(`Bit #9`).

If, during list processing, it is necessary to immediately respond to this error, then the list command **if_not_activated** can be called to possibly jump to an error-handling routine.

- **activate_fly_2d_encoder** or **activate_fly_xy_encoder** can be used to continue at another positioning stage position when switching on again. They reset the encoders and simulate a positioning stage position using the encoder values passed as parameters. This avoids larger forwarding motions for initialization.

8.6.8 Encoder Resets

Many encoder-based Processing-on-the-fly applications (for example, a conveyor belt continuously traveling in the same direction) need to occasionally reset the encoders (for example, before a new marking sequence). For this, you can integrate Processing-on-the-fly reactivations into your lists by **set_fly_x**, **set_fly_y**, **set_fly_rot** or **set_fly_2d**.

In encoder-based Processing-on-the-fly applications with continuous marking (for example, using an xy positioning stage in the virtual Image Field), however, the relationship between the encoder values and the absolute position of the xy positioning stage should usually be preserved.

For this purpose, **set_fly_2d** should be used for Processing-on-the-fly activation. Before a long interruption, you can deactivate Processing-on-the-fly correction with **fly_return**. And after the interruption you can use **activate_fly_2d** or **activate_fly_2d_encoder** to resume correction. See also Chapter 8.6.4 "Compensating 2D Motions", page 258 and Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 261. Processing-on-the-fly correction can also be resumed with **activate_fly_xy** or **activate_fly_xy_encoder**, but some functions are no longer available, particularly those necessary for the relation between current encoder values and absolute xy positioning stage positions, see the Section "2D Encoder Compensation for xy Positioning Stages", page 258.

By **set_control_mode** `Bit #9` it can be set beforehand whether the counter is reset at the respective Processing-on-the-fly command or at the start trigger.

8.6.9 Monitoring Processing-on-the-fly Corrections

The Image Field boundaries might be reached, if:

- A Processing-on-the-fly user program is not optimized for the motion of the workpiece or scan system
- Considerable unintended change of workpiece or scan system speed occurs during a Processing-on-the-fly operation

Because the RTC6 PCIe Board clips coordinate values at the boundaries to prevent unallowed values, this could cause some parts of the to-be-marked pattern to not be scanned.

To allow user programs to monitor Processing-on-the-fly applications, the RTC6 PCIe Board sets internal error bits (Bit #0…Bit #3) if the Image Field boundaries are exceeded (and coordinate values get clipped). These can be read out by **get_marking_info**.

To avoid boundary exceedance, you should repeatedly call **get_marking_info** during test runs or normal operation of your Processing-on-the-fly application and modify your user program accordingly.

Notes

- The error bits Bit #0…Bit #3 can be used to determine *which* edge of the Image Field has been exceeded. Each boundary exceedance results in setting of the corresponding error bit.
- The error bits Bit #0…Bit #3 are reset during initialization by **load_program_file** and by **get_marking_info**. Therefore, **get_marking_info** returns information about errors that occurred since the last initialization or the last call of **get_marking_info**. Subsequent transformations of type #5a and #5b…#11, see Chapter 7.3.6 "Output Values to the Scan System", page 186, are not taken into account here.
- During the adjustment phase of a Processing-on-the-fly correction, coordinate points at the edge of the Image Field should therefore be approached and checked for possible limitations.
- The error bit Bit #9 indicates if the 29-bit virtual Image Field range has been exceeded during resumption of Processing-on-the-fly correction by **activate_fly_2d** or **activate_fly_xy**, see Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 261.

Customer-Defined Monitoring Area

For Processing-on-the-fly applications, the RTC6 PCIe Board also checks for exceedance of a second value range. Its boundaries can be specified by **set_fly_limits**.

Such exceedances likewise result in setting internal error bits (Bit #4…Bit #7). These can be read out by **get_marking_info**.

Moreover, the following conditional commands allow execution of any list command to be made dependent upon whether boundary exceedance in a customer-defined monitoring area occurred or not:

- **if_fly_x_overflow**
- **if_fly_y_overflow**
- **if_not_fly_x_overflow**
- **if_not_fly_y_overflow**

If the condition specified in the command parameter to the error bits Bit #4…Bit #7 is fulfilled, the immediately following list command is executed. Otherwise, it is skipped.

Notes

- Boundary exceedance of a customer-defined monitoring area does not necessarily result in clipping of the output coordinate values. Clipping (and setting error bits Bit #0…Bit #3) only occurs if the maximum Image Field (–524,288…+524,287 bit) is exceeded (the customer-defined monitoring area is typically smaller).
- The error bits Bit #4…Bit #7 are reset during initialization (by **load_program_file**), but *not* by **get_marking_info**. Individual error bits get implicitly reset by the conditional commands and can also be explicitly reset by **clear_fly_overflow** or **clear_fly_overflow_ctrl** (see command description).
- The error bits Bit #4…Bit #7 do *not* take into account transformations of type #5a and #5b…#11, see Chapter 7.3.6 "Output Values to the Scan System", page 186.
- Also for Rotation-fly applications it is possible to monitor a rectangular area, but not a rotation angle range.

8.6.10 Tracking Error Compensation of Encoder Values for Processing-on-the-fly Applications

- Prerequisite:
  RTC6 Software Package ≥ 1.18.0.

Particularly in Processing-on-the-fly applications with variable encoder speed, the Tracking Error of the scan system galvanometer scanners leads to certain position errors.

Therefore, for applications with higher accuracy requirements, a Tracking Error compensation can be switched on by **set_fly_tracking_error**.

8.6.11 Processing-on-the-fly Correction for the z Axis

The encoder-based Processing-on-the-fly correction for the z axis (referred to as “FlyZ correction” in the following) can be activated by **set_fly_z**.

You can add FlyZ to Processing-on-the-fly correction for the x axis and y axis that had been activated by **set_fly_x**/**set_fly_y**/**set_fly_rot** or can be effective on its own.

This makes dynamic marking possible if the workpiece plane (z = 0) and the scan system do not move parallel to each other.

With “FlyZ correction” activated, the z component workpiece motion gets compensated by re-adjustment of the z axis (see Dynamic Focusing Unit) in accordance with the current encoder signals.

Prerequisites

- Both, Option Processing-on-the-fly as well as the Option “3D” must be enabled.
- The desired marking must function properly when static (that is, without workpiece motion):
  - The user program must model workpiece skew by using suitable 3D vector commands.
  - The vario*SCAN* must be capable of tracking the xy galvanometer scanner motions of the scan system.
- For moving workpieces, the Dynamic Focusing Unit must also be capable of following workpiece motion. Take care to exceed neither the maximum focus range in z direction nor the depth of field of the objective. Ensure that the precision of the correction table is sufficient in terms of edge sharpness, stretching etc. When used with **set_fly_rot**, ensure that the rotation angle across the Image Field is small enough and the rotation center is sufficiently far away. Users are responsible for appropriate testing. SCANLAB provides no additional functionality for this. A virtual Image Field for the z coordinate does *not* exist.
- See also the operational prerequisites for 3-axis scan systems in Section ”Connection and Initialization”, page 246.

Notes on Usage

- For general information on Processing-on-the-fly correction and determining scaling factors, see Chapter 8.6 "Processing-on-the-fly", page 251.
- When activating "FlyZ correction" by **set_fly_z**, you can specify which of the two encoder counters should be used. Because **set_fly_x** already uses encoder counter "Encoder0" and **set_fly_y** uses encoder counter "Encoder1" (**set_fly_rot** uses "Encoder0"), you might need to use one of the two encoders twice.
- "FlyZ correction" can be applied together with **set_fly_x**/**set_fly_y**/**set_fly_rot**.
- "FlyZ correction" can be also be used alone (only encoder-based z variation).
- Activated "FlyZ correction" can be deactivated by **fly_return_z** or **fly_return**. As a result, all Processing-on-the-fly corrections for all three spatial directions are simultaneously deactivated. You can supply a command parameter for a new output position (only the x and y coordinates for **fly_return**, in addition the z coordinate for **fly_return_z**).
- Activated "FlyZ correction" can also be deactivated by **set_fly_z** in conjunction with an invalid parameter (for example, `set_fly_z( ScaleZ=0 )`).
  - If only z correction has been activated here, then a jump to an uncorrected output position might occur.
  - If Processing-on-the-fly corrections were also activated for other spatial directions (by **set_fly_x**/**set_fly_y**/**set_fly_rot**), then these do not get deactivated here (and no jump to an uncorrected output position occurs).
- If correction for all three spatial directions is activated by **set_fly_x**, **set_fly_y** and **set_fly_z**, then a subsequent `set_fly_x( ScaleX=0 )` only deactivates X correction without affecting Y and Z correction (likewise for `set_fly_y(ScaleY=0)`). But if only z correction (by **set_fly_z**) or 2D correction (by **set_fly_z** and **set_fly_x** *or* **set_fly_y**) is activated, then a subsequent **set_fly_x**, **set_fly_y** or **set_fly_rot** with invalid parameter value (for example, `set_fly_x( ScaleX=0 )`) also deactivates z correction. In the latter case, a jump to a (partially) uncorrected output position might occur.
- **set_fly_z** deactivates a Processing-on-the-fly correction with positional values via McBSP.
- Conversely, Processing-on-the-fly correction activated by **set_fly_z** gets deactivated by such a correction. Here, a "Hard Jump" to a new output position might occur.
- To avoid "Hard Jumps", use only **fly_return_z** to deactivate Processing-on-the-fly correction.
- **get_marking_info** also provides information on possible range exceedances during "FlyZ correction" (Bit #22…Bit #25).
- **set_fly_limits_z**, **if_fly_z_overflow** and **if_not_fly_z_overflow** are available for defining a customer-specific monitoring range for "FlyZ correction" and for corresponding conditional command execution, see Chapter 9.3.2 "Execution of Conditional Commands", page 317. You can use **clear_fly_overflow** and **clear_fly_overflow_ctrl** to reset the error bits (Bit #24…Bit #25 from **get_marking_info**) for customer-specific monitoring of FlyZ applications.
- A 3D Processing-on-the-fly correction with positional values via McBSP can be activated by **set_multi_mcbsp_in** and **set_multi_mcbsp_in_list**.

8.6.12 "Fly Extension" Commands

- In order to be able to use the Processing-on-the-fly functionality flexibly and generically, the "Fly Extension" Commands are available in RTC6 Software Package ≥ V1.6.1 (DLL 617, OUT 617, RBF 623), see Table 2.

Table 2: Number of supported axes in "Fly Extension" Commands.

"Fly Extension" Command[a]	Axes
set_fly_1_axis	1
fly_return_1_axis	1
activate_fly_1_axis	1
park_position_1_axis	1
park_return_1_axis	1
wait_for_1_axis	1
set_fly_2_axes	2
fly_return_2_axes	2
activate_fly_2_axes	2
park_position_2_axes	2
park_return_2_axes	2
wait_for_2_axes	2
set_fly_3_axes	3
fly_return_3_axes	3

(a) Only list commands, no control commands.

- The Processing-on-the-fly functionality[1] itself remains unchanged. Only its control has been made more flexible.
- "Fly Extension" Commands are a flexible system of Processing-on-the-fly applications.
  - Most of them affect:
    - only 1, only 2 or all 3 spatial axes
    - alternatively 1 Rotary axis
  - At the same time the following can be freely assigned (Table 4)
    - Encoder counter "Encoder0" or "Encoder1"
    - and/or an McBSP input on each of theses axes, even mixed

- Important: "Fly Extension" Commands are in no case compatible with the "Classic" Processing-on-the-fly commands[2].
- A Processing-on-the-fly application with "Classic" Processing-on-the-fly commands[2] must have been explicitly been terminated, before the functionality of one of the "Fly Extension" Commands can be activated, and vice versa.
- With "Fly Extension" Commands, axes are specified according to Table 3.

Table 3: Parameter `Axis` of the "Fly Extension" Commands.

`Axis` Parameter value	Axis
1	x axis
2	y axis
3	z axis
4	Rotary axis

- With the "Fly Extension" Commands, linear axes `1`, `2` and `3` cannot be combined simultaneously with a Rotary axis (`4`).
- "Fly Extension" Commands support 1, 2 or 3 Axes, see Table 2.

(1) As described in Chapter 8.6.1 "Intended Use and Initialization", page 251…Chapter 8.6.11 "Processing-on-the-fly Correction for the z Axis", page 266.

(2) See Footnote, page 251.

Notes on How to Use "Fly Extension" Commands

- `set_fly_1_axis( Axis, Mode, Scale )` replaces one of the "Classic" Processing-on-the-fly commands:
  - **set_fly_x**
  - **set_fly_y**
  - **set_fly_z**
  - **set_fly_rot**
  - **set_fly_x_pos**
  - **set_fly_y_pos**
  - **set_fly_rot_pos**
- **set_fly_2_axes** combines the activation of any 2 linear axes and replaces for example, even **set_fly_2d**.
- **fly_return_1_axis**, **fly_return_2_axes** and **fly_return_3_axes** extend the functionality of the "Classic" Processing-on-the-fly commands **fly_return** and **fly_return_z**.
- **activate_fly_2_axes** replaces the "Classic" Processing-on-the-fly commands
  - **activate_fly_2d**
  - **activate_fly_2d_encoder**
  - **activate_fly_xy**
  - **activate_fly_xy_encoder**
- **wait_for_1_axis** can "wait" for any encoder value or a value transmitted by McBSP, thus substituting **wait_for_encoder_mode** and **wait_for_mcbsp**.
- **wait_for_2_axes** stands for **wait_for_encoder_in_range** and **wait_for_encoder_in_range_mode** and are able to wait for McBSP values in addition.
- The laser control (switch off the laser or leave it unchanged) can be set individually at **wait_for_1_axis** (**wait_for_2_axes**) with parameter `LaserMode` (`LaserMode`).
- The galvanometer scanner motion (stand still ore move along) can be specified with **wait_for_1_axis** (**wait_for_2_axes**) by parameter `WaitMode` (`WaitMode`). Thus, it is no longer bound to **set_fly_2d** and **set_fly_x**/**set_fly_y**.
- "Fly Extension" Commands are also compatible with McBSP transmissions from **set_mcbsp_in** and **set_multi_mcbsp_in**. The Processing-on-the-fly corrections can be subsequently overwritten, for example, by Encoder-based corrections. However, the McBSP transmission to the memory locations themselves cannot be changed.
- "Fly Extension" Commands are – like "Classic" Processing-on-the-fly commands[1] – *not* compatible with an Online Positioning.
- "Fly Extension" Commands are able to "wait" for McBSP values, see Table 4.
- Modes and Encoders supported with "Fly Extension" Commands are shown in Table 4.

(1) See Footnote on page 268.

Table 4: Supported Modes and Encoders with "Fly Extension" Commands.

Mode (parameter value `Mode`)	Encoder ("source")	Effect
0	Reserved.	Generates **get_last_error** return code `RTC6_PARAM_ERROR`.
1	Encoder0.	Only for Processing-on-the-fly-correction. Encoder counter "Encoder0"-scaled. `PreviewTime`-corrected.
2	Encoder1.	Only for Processing-on-the-fly-correction. Encoder counter "Encoder1"-scaled. `PreviewTime`-corrected.
3	Encoder2.	Only for Processing-on-the-fly-correction. Tied to encoder counter "Encoder0"-scaled. `PreviewTime`-corrected.
4	Encoder2.	Only for Processing-on-the-fly-correction. Tied to encoder counter "Encoder1"-scaled. `PreviewTime`-corrected.
5	Reserved.	Generates **get_last_error** return code `RTC6_PARAM_ERROR`.
6	McBSP 32 Bit.	Processing-on-the-fly correction with position specifications, **read_mcbsp**(0).
7	McBSP 31 Bit.	**set_mcbsp_in** (`Mode` 1, 2 or 4) and **read_mcbsp**(0).
8	McBSP 29 Bit.	Memory location for the x coordinate of the Processing-on-the-fly application from **set_multi_mcbsp_in**, **read_multi_mcbsp**(0).
9	Reserved.	Generates **get_last_error** return code `RTC6_PARAM_ERROR`.
10	McBSP LOW 16 Bit.	Processing-on-the-fly correction with position specifications at 2 Axes.
11	McBSP LOW 15 Bit.	From **set_mcbsp_in** (`Mode`= 3).
12	McBSP 29 Bit.	Memory location for the y coordinate of the Processing-on-the-fly application from **set_multi_mcbsp_in**, **read_multi_mcbsp**(1).
13	Reserved.	Generates **get_last_error** return code `RTC6_PARAM_ERROR`.
14	McBSP HIGH 16 Bit.	Processing-on-the-fly correction with position specifications at 2 Axes.
15	McBSP HIGH 15 Bit.	From **set_mcbsp_in** (`Mode`= 3).
16	McBSP 29 Bit.	Memory location for the z coordinate of the Processing-on-the-fly application from **set_multi_mcbsp_in**, **read_multi_mcbsp**(2).
17	Encoder0.	Only for **wait_for_1_axis** and **wait_for_2_axes**. Not for Processing-on-the-fly corrections. Encoder counter "Encoder0"-unscaled and `PreviewTime`-corrected.
18	Encoder1.	Only for **wait_for_1_axis** and **wait_for_2_axes**. Not for Processing-on-the-fly corrections. Encoder counter "Encoder1"-unscaled and `PreviewTime`-corrected.
19	Encoder0.	Like 17, but not `PreviewTime`-corrected.
20	Encoder1.	Like 18, but not `PreviewTime`-corrected.

Notes on Table 4

- Each of these sources can be recorded by **set_trigger[*]**.
- Mode 1…4 as well as 17…18 are also suitable for SCANahead Systems.
- With **wait_for_1_axis** and **wait_for_2_axes**, Mode 17…20 must be specified but not 1…4.
- With the other "Fly Extension" Commands where an Encoder specification is possible, Mode 1…4 must be specified but not 17…20.
- With the McBSP modes:
  - Users must make sure that the data is actually transferred in the correct format
  - Users must additionally set (exception: `Mode` = 6) a corresponding Processing-on-the-fly correction in **wait_for[*]** commands (by **set_fly[*]**, **set_mcbsp_in** or **set_multi_mcbsp_in**)
  - the RTC6 board "waits" for the Encoder values even without explicit activation of the Processing-on-the-fly correction
- A **get_last_error** return code `RTC6_PARAM_ERROR` is generated, if a Mode is not allowed.
- Further details can be found in the respective command descriptions.

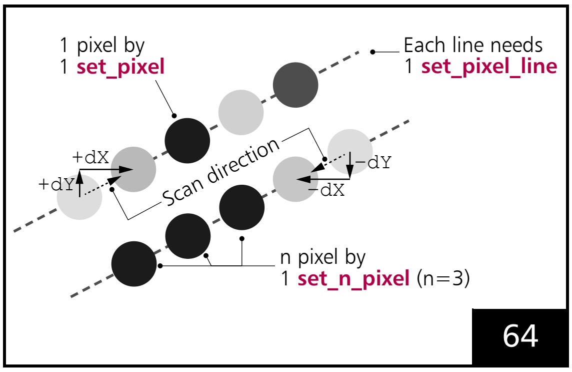
8.7 Pixel Output Mode

The Vector Commands described in Chapter 7.1 "Marking Dots, Lines and Arcs", page 139 are intended for scanning vector based images. Furthermore, the RTC6 PCIe Board allows marking pixel images (bitmaps). With suitably adjusted lasers, black-and-white images or greyscale images can be obtained. Raster and vector based images can be combined as desired.

8.7.1 Principle of Operation

Pixel images are created image line by image line, see Figure 64. Each line consists of a number of equidistant pixels. A line is generated in a single pass. During this pass, the laser focus moves – as with a normal [*]mark[*] Command – at an (approximately) constant velocity along the entire image line (the motion is split-up into Microsteps). The individual pixels are marked in passing: each pixel gets a laser pulse assigned at the appropriate location. By varying the laser energy from pixel to pixel, greyscale images (including black & white images) are produced.

Different output ports can be used to control the laser depending on mode and pixel output frequency, see command descriptions.



An individual **set_pixel_line** is required for each image line. The individual pixels of this line are then defined by successive **set_pixel**/**set_n_pixel**.

- By **set_pixel_line**, a single line is configured:
  - The *spatial* distance between the individual pixels with the parameters `dX`, `dY` (with **set_pixel_line_3d** additionally `dZ`)
  - The *temporal* distance between the individual pixels with the parameter `HalfPeriod`
  - The parameter `Channel` is extended compared to the RTC5 to Channel= Mode + Port, where now the (pixel) mode is defined via the value Mode and the Output Port is defined via the value Port, see Table 5 and 6

Notes

- The Pixel Output Mode can be combined with Processing-on-the-fly, see Chapter 8.6 "Processing-on-the-fly", page 251.
- The Pixel Output Mode *should not* be used in conjunction with "Automatic Laser Control" (see Chapter 7.4.9 ""Automatic Laser Control"", page 205), if there is a readjustment of the port output, pulse length (`PulseLength`) or output period (`HalfPeriod`) of the Laser Control Signals LASER1 and LASER2.
- The Pixel Output Mode *cannot* be combined with Sky Writing.(1)
- The Pixel Output Mode *cannot* be combined with Wobbel Mode.

8.7.2 RTC6 Commands

Before writing an image line, a jump to the beginning of the line should be executed by a Jump Command.

At the beginning of each image line, the Pixel Output Mode is activated by **set_pixel_line** or **set_pixel_line_3d**. The pixel distance and pixel output period (and, resultingly, the speed at which the image line is traversed) are simultaneously set as well.

(1) However, the Pixel Output Mode can also be executed so that automatically generated (Sky Writing Mode 1-like) pre and post motions are included. This makes it easier to output pixel images with accurate positioning, see Chapter 8.7.4 "Synchronization", page 277.

The pixel output period is defined by the parameter HalfPeriod (half pixel output period). This is also half the laser period. The pixel distance between two adjacent pixels in the line – and thus also the marking direction – is defined by a:

- 2D vector (dX,dY) with **set_pixel_line**
- 3D vector (dX,dY,dZ) with **set_pixel_line_3d**

Directly after **set_pixel_line**/**set_pixel_line_3d**, **set_pixel** has to be called separately for each pixel of the line. This defines the laser energies to be outputted at the corresponding pixel locations.

Pixel pulses are always outputted at the LASER1 output port, even for the purpose of synchronizing the laser (gating). If PulseLength is not specified as the port, the very PulseLength is outputted that has been specified at last by **set_laser_pulses**, however, at least 1/64 µs.

As an alternative to a sequence of n identical **set_pixel** calls, **set_n_pixel** can be used.

Then only one command is stored on the RTC6 PCIe Board, but during list processing the corresponding **set_pixel** is executed n times. Particularly for black & white images, this can drastically reduce the size of lists. Do not confuse **set_n_pixel** with **n_set_pixel** (multi-board command of **set_pixel**).

Prior to the end of an image line, no command other than **set_pixel** or **set_n_pixel** must be inserted into the list. After **set_pixel_line**/**set_pixel_line_3d**, the first list command that is *not* a **set_pixel** or **set_n_pixel** command stops the Pixel Output Mode and thus processing of the image line.

Each **set_pixel**/**set_n_pixel** call that does not follow another **set_pixel**/**set_n_pixel** or **set_pixel_line**/**set_pixel_line_3d** call is ignored during executing and is thus a Short List Command.

Notes

- The number of pixels in an image line is limited only by the capacity of the RTC6 List Memory, see Chapter 15 "Technical Specifications – RTC6 PCIe Board", page 950. 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 **set_pixel_line** or **set_pixel_line_3d**.
- The pixel distance (dX, dY) in the x direction and y direction (in bits) (and with **set_pixel_line_3d** also dZ) can be specified with floating point numbers. This allows scaling and rotating the image without rounding errors.
- Depending on the Pixel Output Mode, the half pixel output period can be any integer-multiple of 1/64 µs. It is independent from the 10 µs position output period of the galvanometer scanners' (split-up into Microsteps) motion. Very low pixel output frequencies result in multiple galvanometer scanner steps per pixel, higher frequencies result in multiple pixel pulses per galvanometer scanner step.
- You can implement a Pixel Output Mode in a 10 µs raster with variable speed and/or curvilinear paths with the help of micro_vector[*] Commands, see Chapter 8.8 "micro_vector[*] Commands", page 282.

8.7.3 Laser Control

Depending on the type of laser employed, the laser energy outputted at each pixel position can be varied by:

- laser pulse length
- laser power per pixel

"Classic Mode"

The "Classic Mode" (Mode = 0, see `Channel`) corresponds to the Pixel Output Mode of the RTC4 and RTC5. Maximum pixel output frequency: 400 kHz.

- Variation Of Laser Pulse Length: **set_pixel** defines – for each pixel – the length of the pixel pulse (in units of 1/64 µs), which is outputted at the LASER1 port. For synchronization, see Chapter 8.7.4 "Synchronization", page 277.
- Variation Of Laser Power: **set_pixel** allows specification of a 12-bit analog voltage level for each pixel. The value is transferred either to the ANALOG OUT1 or ANALOG OUT2 output port (see above). For synchronization, see Chapter 8.7.4 "Synchronization", page 277. **set_n_pixel** defines the analog voltage level for `n` directly successive pixels of an image line.

"Extended Mode"

In "Extended Mode" (Mode = 16, see page 777), two pixels (32-bit values) are outputted at the specified port with each **set_pixel**. Maximum pixel output frequency: 800 kHz.

"Fast Mode"

In "Fast Mode" (Mode = 32, see page 777), 4 pixels of 16 bits each are outputted at the specified port with each **set_pixel**. Maximum pixel output frequency: 1.6 MHz.

"Ultra Fast Mode"

In "Ultra Fast Mode" (Mode = 64, see page 777) 8 pixels of 8 bits each are outputted at the specified port with each **set_pixel**. Maximum pixel output frequency: 3.2 MHz.

Notes

- Pixel output frequencies > 800 kHz ("Fast Mode" and "Ultra Fast Mode") require the Option "UFPM". Otherwise, the pixel output frequency is clipped to 800 kHz.
- It is recommended that some experiments be performed to determine an appropriate gradation curve for producing smooth greyscales. The resulting pixel greyscale values ("colors") strongly depend on the employed material and the laser.
- The LASERON signal – as with a normal [*]mark[*] Command – switches to "On" after the LaserOn Delay has expired and remains "On" for the entire pixel line. After the last pixel has been outputted, it automatically goes to "Off". See Chapter 7.4 "Laser Control", page 189.
- The LASER1 signal depends on the respective laser mode and the associated settings, as well as on the Mode (see page 777) set by **set_pixel_line**/ **set_pixel_line_3d**: A periodic signal with `HalfPeriod` from **set_pixel_line** and a `PulseLength` from **set_laser_pulses** (however, at least 1/64 µs long) is outputted. The following exceptions apply:
  - In Laser Mode 4 and Laser Mode 6, only a standby signal is outputted that cannot be varied. Power changes are only possible via the output ports ANALOG OUT1, ANALOG OUT2, 8-Bit Digital Output Port (EXTENSION 2 Socket Connector) or 16-Bit Digital Output Port (EXTENSION 1 Socket Connector).
  - If an output to `PulseLength` is specified as the Mode, `PulseLength` is varied per pixel.
  - No Softstart is performed, see Chapter 7.4.7 "Softstart Mode (not yet implemented)", page 202.
- The LASER2 signal follows the specifications of the respective laser mode.
- Pixel Output Mode and Sky Writing cannot be combined. However, a separate Sky Writing Mode 1-like galvanometer scanner motion can be inserted, see Figure 65.
- The values for `HalfPeriod` and `PulseLength` set prior to **set_pixel_line** are not restored again. **set_laser_pulses** must explicitly be called again.

Table 5:

Mode	Max. freq	No. of pixels per set_pixel/ set_n_pixel	1st parameter of set_pixel/set_n_pixel (name: `PortOutValue1`[a])	2nd parameter of set_pixel/set_n_pixel (name: `PortOutValue2`[b])
0, 256[c]	400 kHz	1	Pixel 1 (defined by the pulse length). The output is done as signal LASER1. [Pixel 1 (1st parameter: 32-bit; 2nd parameter: up to 16 bit, depending on port)]	Port output (but not port 5) for Pixel 1
16	800 kHz	2	Port output for Pixel 1 (up to 32 bits per pixel, depending on port) [Pixel 1]	Port output for Pixel 2 (up to 32 bits per pixel, depending on port) [Pixel 2]
32[d]	1.6 MHz	4	Port output for Pixel 1, 2 (up to 16 bits per pixel, depending on port) [Pixel 2 | Pixel 1]	Port output for Pixel 3, 4 (up to 16 bits per pixel, depending on port) [Pixel 4 | Pixel 3]
64[d]	3.2 MHz	8	Port output for Pixel 1, 2, 3, 4 (up to 8 bits per pixel, for all ports) [Pixel 4 | Pixel 3 | Pixel 2 | Pixel 1]	Port output for Pixel 5, 6, 7, 8 (up to 8 bits per pixel, for all ports) [Pixel 8 | Pixel 7 | Pixel 6 | Pixel 5]
			Bit 31 .... Bit 0	Bit 31 .... Bit 0

(a) `PulseLength` in RTC6 Software Packages < v1.3.3.

(b) `AnalogOut` in RTC6 Software Packages < v1.3.3.

(c) The galvanometer scanner motion is *not* continuous in Mode = 0 and continuous in Mode = 256.

(d) > 800 kHz only possible with Option “UFPM”.

Table 6:

Output Port	Where at the RTC6 PCIe Board	
1	12-Bit Analog Output Port 1	LASER Connector, page 75, ANALOG OUT1 See Figure 17.
2	12-Bit Analog Output Port 2	LASER Connector, page 75, ANALOG OUT2 See Figure 17. MARKING ON THE FLY Socket Connector, page 85, ANALOG OUT2 See Figure 25.
3	8-Bit Digital Output Port	EXTENSION 2 Socket Connector, page 83 See Figure 24.
4	16-Bit Digital Output Port	EXTENSION 1 Socket Connector, page 81 See Figure 22.
5[a][b]	Pulse length	LASER Connector, page 75, LASER1 See Figure 17.

(a) Port = 5 cannot be combined with Mode = 0.

(b) Port = 5 cannot be combined with Mode = 256.

8.7.4 Synchronization

The pixel output timing diagram for one image line with 3 pixels is shown in Figure 66.

To prepare the laser control, **set_pixel_line**/**set_pixel_line_3d** switches off the Signals for "Laser Active" Operation from a previous Mark Command after a LaserOff Delay (as with a Jump Command) and waits until the laser is actually off.

During this waiting period, the galvanometer scanners do not move in "Classic Mode" (Mode = 0, see page 777) only. In all other modes, they continue to move at the speed defined by `HalfPeriod` and pixel distance in equidistant Microsteps. This allows to program jerk-free forerun motions and post motions. Initial acceleration phases can be hidden by a LaserOn Delay (as with a normal [*]mark[*] Command) or by a corresponding number of "idle pixels" (see below).

After that, depending on the laser mode, pixel output starts immediately or after a Q-Switch delay, see Figure 66. Analog signals at ANALOG OUT1 or ANALOG OUT2 change synchronously with the leading edge of each pixel pulse. The digital-to-analog converter requires about 1.5 µs…3 µs to produce a stable analog output signal. With pixel output frequencies above around 100 kHz (`HalfPeriod` < approx. 320) digital-to-analog conversion cannot always be fully completed. At such pixel output frequencies, it must be carefully checked whether the functionality is sufficient for the intended purpose.

Bit #1 in `set_laser_control( Ctrl )` can be used to shift the laser pulse and thus the start of the digital-to-analog conversion by half a pixel period.

The pixel line ends with the first list command that is not a **set_pixel** or **set_n_pixel**.

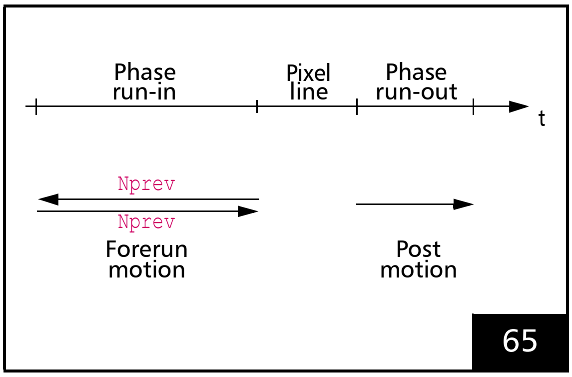
For the laser to be switched off even in the middle of a 10 µs cycle, a default pixel is automatically outputted after the last pixel. This is repeated as often as necessary until a started 10 µs cycle is finished. Then the laser is finally switched off.

The default pixel should be defined appropriately prior to **set_pixel_line**/**set_pixel_line_3d** in order to achieve a non-visible laser marking (= "idle pixels") in the run out, see **set_port_default** and **set_default_pixel**.

The RTC6 board waits – especially at pixel output frequencies < 100 kHz – until the default pixel is outputted.

The galvanometer scanners continue to run during this time, to ensure jerk-free connection motions (programmed by the user). No Scanner Delay is automatically inserted. In "Classic Mode" (Mode = 0, see page 777), the galvanometer scanners remain idle for a few clock cycles. With Mode + 256 this can be suppressed (even for the run-in phase).

The Tracking Error and the hidden acceleration phase mean a pixel line shift", see Figure 66. Normally, this needs to be compensated for by an adjusted run-in. To make this easier, Mode + 512 can be used to switch on Sky Writing Mode 1-similar motions in the run-in and run-out phase and thus place the pixel line with pinpoint accuracy, see Figure 65.



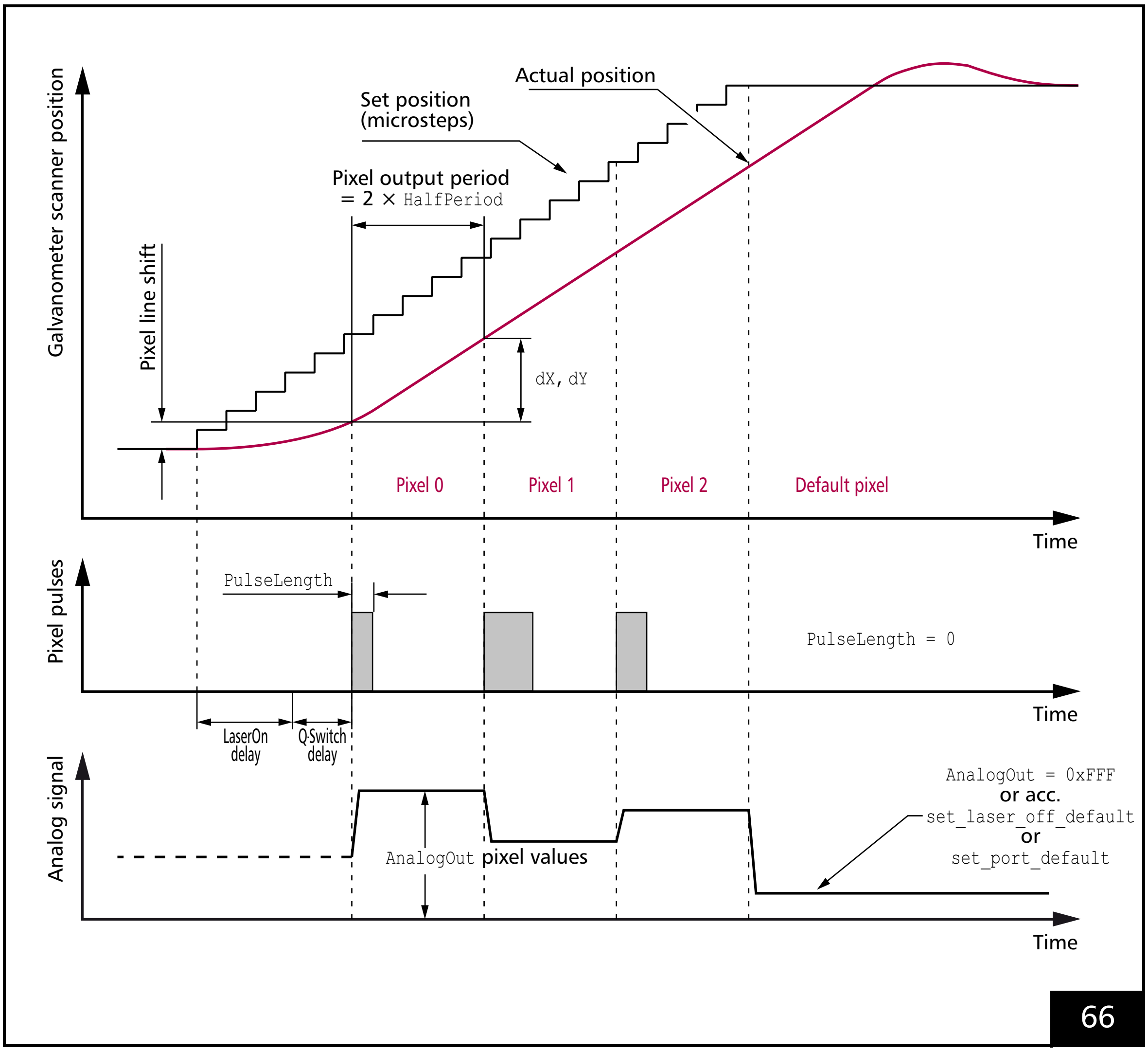
Mode + 512

These motions cannot be combined with Sky Writing Mode 2. The duration of the forerun motion must be explicitly defined via `Nprev` of **set_sky_writing_para** in advance.(1)

(1) With SCANahead Systems, Sky Writing and automatic delay calculation need to be switched on, see "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.

Notes

- With HalfPeriod < PulseLength / 2 the laser does not switch off between the pixels.
- When specifying HalfPeriod and pixel distance, observe the dynamics of the scan system (mark speed) and the properties of the laser system (power modulation).



Timing of the galvanometer scanner positions and the Laser Control Signals in the Pixel Mode Mode = 0and a YAG Mode, see Chapter 7.4.4 "YAG Mode 1, YAG Mode 2, YAG Mode 3, YAG Mode 5", page 195.
The pixel output period in this example is approx. 4.5 Microsteps.

Example Code

The example code must be included in a user program.

```
//  UINT = uint32_t
//  LONG =  int32_t

// laser parameters
UINT LaserMode = 0;
UINT LaserControl = 0x0;
LONG LaserOnDelay  = (LONG)round(100.0*64.0);  // LaserOnDelay = 100 µs
UINT LaserOffDelay = (UINT)round(100.0*64.0);  // LaserOffDelay = 100 µs
UINT HalfPeriod  = (UINT)round(5.0 * 32.0);    // Period = 5 µs, PixelFrequency = 200 kHz
UINT PulseLength = (UINT)round(0.5 * 64.0);    // PulseLength = 0.5 µs

// Pixel Output Mode variables
enum E_PixelModes
{
    STANDARD      = 0,    // like RTC5-Mode; up to 400 kHz; 1 pixel per set_pixel command; 2 values of 32 bit per pixel
    ENHANCED      = 16,   // up to 800 kHz; 2 pixel per set_pixel command; 1 value of 32 bit per pixel
    FAST          = 32,   // up to 1.6 MHz; 4 pixel per set_pixel command; 1 value of 16 bit per pixel
    ULTRAFAST     = 64,   // up to 3.2 MHz; 8 pixel per set_pixel command; 1 value of 8 bit per pixel
    STANDARD_MOVE = 256   // like STANDARD, but with continuous galvanometer scanner motion
} Mode;

enum E_PixelPorts
{
    NO_OUT_PORT   = 0,
    ANALOG_OUT1   = 1,    // LASER Connector ANALOG OUT1
    ANALOG_OUT2   = 2,    // LASER Connector ANALOG OUT2
    DIGITAL_8Bit  = 3,    // EXTENSION 2 socket connector (8-bit)
    DIGITAL_16Bit = 4,    // EXTENSION 1 Socket Connector (16-bit)
    PULSE_LENGTH  = 5     // not allowed for mode = 0 or mode = 256
} Port;

Port = DIGITAL_8Bit;          // pixel output port number
Mode = STANDARD;              // Pixel Output Mode
UINT Channel = Port + Mode;   // pixel channel

UINT Number = 1;              // set_pixel repetition number

UINT PixelArray[17] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 };   // port output values
UINT PortOutValue1 = PixelArray[0];
UINT PortOutValue2 = PixelArray[0];

LONG dx = (LONG)round(0.01 * CalibrationFactorXY);   // 10 um spot distance in x direction
LONG dy = (LONG)round(0.0  * CalibrationFactorXY);   //  0 um spot distance in y direction

LONG PixelLineStartPosX = (LONG)round(0.0 * CalibrationFactorXY);
LONG PixelLineStartPosY = (LONG)round(0.0 * CalibrationFactorXY);

// basic configuration
set_laser_mode(LaserMode);
set_laser_control(LaserControl);

// RTC-List
set_start_list_pos(1, 0);
set_laser_delays(LaserOnDelay, LaserOffDelay);
set_default_pixel_list(0);     // sets pixel default PulseLength
UINT PortDefault = Port - 1;   // CAUTION: port numbers differ between set_port_default/set_port_default_list and
                               //                                          set_pixel_line
set_port_default_list(PortDefault, PixelArray[0]); // sets default value for specified port
```

```
switch (Mode)
{
case STANDARD:
case STANDARD_MOVE:
{
    // standard Pixel Output Mode up to 400 kHz (Mode = 0, 256)
    // 2 values of 32 bit per pixel
    // 1 pixel per set_pixel command
    //--------------------------------
    // |           pixel1            |
    // |   32 Bit    ||    32 Bit    |
    // | PulseLength || PortOutValue |
    //--------------------------------
    HalfPeriod  = (UINT)round(5.0 * 32.0);   // Period = 5 µs, PixelFrequency = 200 kHz
    PulseLength = (UINT)round(0.5 * 64.0);   // PulseLength = 0.5 µs

    jump_abs(PixelLineStartPosX, PixelLineStartPosY);
    set_pixel_line(Channel, HalfPeriod, dx, dy);
    set_n_pixel(1 * PulseLength, PixelArray[1], Number);        // pixel 1
    set_n_pixel(2 * PulseLength, PixelArray[2], Number);        // pixel 2
    set_n_pixel(3 * PulseLength, PixelArray[3], Number);        // pixel 3
    set_n_pixel(4 * PulseLength, PixelArray[4], Number * 2);    // (pixel 4)*2

}
break;

case ENHANCED:
{
    // enhanced Pixel Output Mode up to 800 kHz (Mode = 16)
    // 1 value of 32 bit per pixel
    // 2 pixel per set_pixel command
    //----------------------------------
    // |    pixel1     ||    pixel2     |
    // |    32 Bit     ||    32 Bit     |
    // | PortOutValue1 || PortOutValue2 |
    //----------------------------------
    HalfPeriod  = (UINT)round(2.0 * 32.0);   // Period = 2 µs, PixelFrequency = 500 kHz
    PulseLength = (UINT)round(0.5 * 64.0);   // PulseLength = 0.5 µs
    set_laser_pulses(HalfPeriod, PulseLength);

    jump_abs(PixelLineStartPosX, PixelLineStartPosY);
    set_pixel_line(Channel, HalfPeriod, dx, dy);
    set_n_pixel(PixelArray[1], PixelArray[2], Number);        // pixel 1, pixel 2
    set_n_pixel(PixelArray[3], PixelArray[4], Number);        // pixel 3, pixel 4
    set_n_pixel(PixelArray[5], PixelArray[6], Number * 2);    // (pixel 5, pixel 6)*2
    //--> pixel output: pixel 5, pixel 6, pixel 5, pixel 6

}
break;
```

```
case FAST:
{
    // fast Pixel Output Mode up to 1.6 MHz (Mode = 32) - needs UFPM-Option
    // 1 value of 16 bit per pixel
    // 4 pixel per set_pixel command
    //------------------------------------------------------
    // |   pixel2    |   pixel1    ||   pixel4    |   pixel3    |
    // |   16 Bit    |   16 Bit    ||   16 Bit    |   16 Bit    |
    // |      PortOutValue1        ||      PortOutValue2        |
    //------------------------------------------------------
    HalfPeriod  = (UINT)round(1.0 * 32.0);   // Period = 1 µs, PixelFrequency = 1 MHz
    PulseLength = (UINT)round(0.5 * 64.0);   // PulseLength = 0.5 µs
    set_laser_pulses(HalfPeriod, PulseLength);

    jump_abs(PixelLineStartPosX, PixelLineStartPosY);
    set_pixel_line(Channel, HalfPeriod, dx, dy);
    // low 16 bit - first pixel, high 16 bit - second pixel
    PortOutValue1 = PixelArray[1] | (PixelArray[2] << 16);
    PortOutValue2 = PixelArray[3] | (PixelArray[4] << 16);
    set_n_pixel(PortOutValue1, PortOutValue2, Number);        // pixel 2 + pixel 1, pixel 4 + pixel 3

    PortOutValue1 = PixelArray[5] | (PixelArray[6] << 16);
    PortOutValue2 = PixelArray[7] | (PixelArray[8] << 16);
    set_n_pixel(PortOutValue1, PortOutValue2, Number * 2);    // (pixel 6 + pixel 5, pixel 8 + pixel 7)*2
    //--> pixel output: pixel 5, pixel 6, pixel 7, pixel 8, pixel 5, pixel 6, pixel 7, pixel 8

}
break;
case ULTRAFAST:
{
    // ultra fast Pixel Output Mode up to 3.2 MHz (Mode = 64) - needs UFPM-Option
    // 1 value of 8 bit per pixel
    // 8 pixel per set_pixel command
    //----------------------------------------------------------------------------------------------------
    //|   pixel4    |   pixel3    |   pixel2    |   pixel1    ||   pixel8    |   pixel7    |   pixel6    |   pixel5    |
    //|    8 Bit    |    8 Bit    |    8 Bit    |    8 Bit    ||    8 Bit    |    8 Bit    |    8 Bit    |    8 Bit    |
    //|                 PortOutValue1                          ||                  PortOutValue2                         |
    //----------------------------------------------------------------------------------------------------
    HalfPeriod  = (UINT)round(0.5  * 32.0);   // Period = 0.5 µs, PixelFrequency = 2 MHz
    PulseLength = (UINT)round(0.25 * 64.0);   // PulseLength = 0.25 µs
    set_laser_pulses(HalfPeriod, PulseLength);

    jump_abs(PixelLineStartPosX, PixelLineStartPosY);
    set_pixel_line(Channel, HalfPeriod, dx, dy);

    // lowest 8 bit - first pixel, highest 8 bit - fourth  pixel
    PortOutValue1 = PixelArray[1] | (PixelArray[2] << 8) | (PixelArray[3] << 16) | (PixelArray[4] << 24);
    PortOutValue2 = PixelArray[5] | (PixelArray[6] << 8) | (PixelArray[7] << 16) | (PixelArray[8] << 24);

    // pixel 4 + pixel 3 + pixel 2 + pixel 1, pixel 8 + pixel 7 + pixel 6 + pixel 5
    set_n_pixel(PortOutValue1, PortOutValue2, Number);

    PortOutValue1 = PixelArray[ 9] | (PixelArray[10] << 8) | (PixelArray[11] << 16) | (PixelArray[12] << 24);
    PortOutValue2 = PixelArray[13] | (PixelArray[14] << 8) | (PixelArray[15] << 16) | (PixelArray[16] << 24);

    // (pixel 12 + pixel 11 + pixel 10 + pixel 9, pixel 16 + pixel 15 + pixel 14 + pixel 13)*2
    set_n_pixel(PortOutValue1, PortOutValue2, Number * 2);
    // --> pixel output: pixel 9, pixel 10, pixel 11, pixel 12, pixel 13, pixel 14, pixel 15, pixel 16,
    //                   pixel 9, pixel 10, pixel 11, pixel 12, pixel 13, pixel 14, pixel 15, pixel 16

}
break;

default:
break;

}

set_end_of_list();
```

8.8 micro_vector[*] Commands

micro_vector[*] Commands move the galvanometer scanners directly to the specified position by a "Hard Jump"

- in 2D Image Field:
  - **micro_vector_abs**
  - **micro_vector_rel**
- in 3D Image Field:
  - **micro_vector_abs_3d**
  - **micro_vector_rel_3d**

The Signals for "Laser Active" Operation can be switched on and off individually for each micro_vector[*] Command by the specified Laser Delays (parameters `LasOn` and `LasOff`). The Laser Delays defined by **set_laser_delays** are not overwritten by that. These remain valid for normal Jump Commands and [*]mark[*] Commands.

The micro_vector[*] Commands can be used to mark trajectories (see Glossary entry on page 31) of arbitrary shape and at variable speed, as long as the dynamic limits of the scan system are not exceeded.

Notes

- micro_vector[*] Commands wait for a preceding Scanner Delay. They never trigger new Scanner Delays.
- Users themselves are responsible for appropriately parameterizing micro_vector[*] Commands:
  - Complying with the dynamic limits of the scan system
  - Avoiding cross over and take over with LaserON and LaserOff, see Chapter 7.2.3 "Notes on Optimizing the Delays", page 158
- A Wobbel motion specified by **set_wobbel** or **set_wobbel_mode**, see Chapter 8.4 "Wobbel Mode", page 241 is not taken into account (but also not deactivated).
- The Sky Writing mode is not taken into account (but also not deactivated).
- The output values are calculated according Chapter 7.3.6 "Output Values to the Scan System", page 186 without 1 and 2.

8.9 Timed Commands

"Normal" Vector Commands and "Arc" Commands are executed in such a way that the laser focus moves with a defined speed[1]. This is fine for most laser marking and laser material processing applications.

At the beginning of a jump, the Signals for "Laser Active" Operation are switched off. The Signals for "Laser Standby" Operation are switched on depending on the settings, see Chapter 7.4.1 "Enabling, Activating and Switching Laser Control Signals", page 189.

However, some applications require that each Vector Command or "Arc" Command consumes exactly the same amount of time, regardless of its spacial length.

In this case, it is necessary to specify a jump *duration* or mark process *duration* (rather than the jump speed or mark speed).

Timed commands allow specification of the duration of the Vector Command or "Arc" Command with an accuracy of 10 μs (the output period of the Microsteps) and in the range from 10 μs to 167,772,160 μs ($\approx$ 2.8 min).

A vector or arc is split-up into the specified (`T` Parameter) number of Microsteps.

For $T \geq 5$, the following applies:

$$\text{Length } \Delta s \text{ of a Microstep} = L / t$$
$$\text{with } t = \text{integer}(\,(T + 5) / 10\,)$$ [2]

Thus, jump speed and mark speed speed automatically depend on the length of the vectors or arcs.

The following timed commands are available:

- Vectors and arcs
  - **timed_jump_abs**
  - **timed_jump_rel**
  - **timed_mark_abs**
  - **timed_mark_rel**
  - **timed_arc_abs**
  - **timed_arc_rel**
- Parametrized vectors
  - **timed_para_jump_abs**
  - **timed_para_jump_rel**
  - **timed_para_mark_abs**
  - **timed_para_mark_rel**
- 3D vectors
  - **timed_jump_abs_3d**
  - **timed_jump_rel_3d**
  - **timed_mark_abs_3d**
  - **timed_mark_rel_3d**
- Parametrized 3D vectors
  - **timed_para_jump_abs_3d**
  - **timed_para_jump_rel_3d**
  - **timed_para_mark_abs_3d**
  - **timed_para_mark_rel_3d**

Notes

- After a timed command, a "normal" Jump Delay, Mark Delay or Polygon Delay is inserted.
- Ellipses are fundamentally not available as timed commands.
- With $T < 5$ μs, a timed command ist synonym to its "normal" (= without **[timed_*]**) command.
- The total execution time of a timed command is the sum of the specified time and the associated delay, see also Chapter 7.2.2 "Scanner Delays", page 150.

(1) Either jump speed or mark speed.

(2) For $T < 5$, the command is carried out as non-timed command, see Chapter 7.1.2 "Microstepping", page 143.

8.10 Automatic Self-Calibration

- This functionality does not apply to current SCANLAB scan systems
- Previously, some SCANLAB scan systems have been equipped with an additional internal sensor system for automatic self-calibration (ASC sensor system, home-In sensors) to meet higher requirements for long-term repeatability

8.10.1 Use for Drift Compensation

Long-term repeatability is very important in many applications, for example, for rapid prototyping in which the processing operation can span several hours. For such laser applications, the galvanometer scanner's long-term drift and temperature drift, which manifest as a shift (offset drift) and increase or decrease in the size (gain drift) of the working Image Field, can exceed the allowed tolerances.

In such applications, it is helpful to start up the application only after the galvanometer scanners have reached their operating temperature. In addition, the magnitudes of environmental fluctuations (for example, operating temperature changes to which the scan system is exposed) should be kept as small as possible and the scan system preferably operated with a constant load.

The ASC sensor system (see above) allows the position detectors of the galvanometer scanners to be calibrated and gain drift and offset drifts to be reliably compensated. The positioning accuracy is thus maintained over long periods of time. Remaining long-term drift effects are of the same order of magnitude as short-term repeatability accuracies.

8.10.2 How it Works

By **auto_cal**, a measurement routine can be started for determining the exact control values for reference positions (Home-In positions) defined by the internal sensor system.

For drift *measurement*, the routine should be executed:

- When setting up the equipment – to determine reference values for the Home-In positions, see Chapter 8.10.3 "Determining Reference Values", page 286
- During the application at appropriate time intervals – to determine if and how the Home-In positions have changed, see Chapter 8.10.4 "Calibration During the Application", page 286

For drift *compensation*, appropriate gain values and offset values can be calculated and set separately for each galvanometer scanner based on the determined deviations between the current Home-In position and the reference value. See also #11 in Chapter 7.3.6 "Output Values to the Scan System", page 186.

The setting can also be made manually by **set_hi**, if customer-specific measuring procedures are to be used, see Section "Customer-Specific Calibration", page 287.

Notes

- Prior to performing a measurement routine, **auto_cal**(`Command` = 4) can be used to check if:
  - the attached scan system in fact has an internal sensor system for automatic self-calibration (Home-In sensors)
  - the sensor system is functioning properly

This ASC hardware check also occurs automatically by **auto_cal**(Command = 0) and if required, for **auto_cal**(Command = 1 and 3), see also **auto_cal** command description and **get_auto_cal**.

- During execution of the measurement routine determining the Home-In positions:
  - the laser should be switched off
  - no other commands should be sent to the scan system
- For Gain/Offset Correction:
  See #11 in Chapter 7.3.6 "Output Values to the Scan System", page 186.
  After an initialization by **load_program_file** or **auto_cal**(Command = 2), the following is set:
  - Gain = 1.0
  - Offset = 0
- For iDRIVE Scan Systems, the following applies in addition:
  - At the beginning of a measurement routine, the XY2-100 status word is automatically set as to-be-returned by the scan system. At the end of the measurement routine, the previously set data type is restored.
  - **auto_cal** aborts with an error result value 7 if
    - **auto_cal** is to be executed for the Connector for First Scan Head, and
    - an "Automatic Laser Control" in Mode = 2 (basic mode) has been activated, see Section "Speed-Dependent Laser Control", page 209.

    The "Automatic Laser Control" must be deactivated before performing automatic self-calibration.
  - **auto_cal** also aborts (result value 1, 10 or 11), if the scaling factor has been set by **control_command**(Data = $12xx_H$) to a value < 1. This is because the sensor positions then are not reachable, see also **control_command**(Data = $053F_H$). The scaling factor must have been set to 1 (default setting) by prior to automatic self-calibration
    - **control_command**(Data = 1283_H) and
    - **control_command**(Data = 1200_H).

8.10.3 Determining Reference Values

The reference values are determined by **auto_cal** (Command = 0). This starts the measurement routine for determining the current Home-In positions. These are then the reference values for later calibrations with **auto_cal**(Command = 1 or 3). Immediately after determination, they can be read out by **get_hi_pos** and are available for customer-specific calibration, see Section "Customer-Specific Calibration", page 287. The reference values are stored in the Flash Memory of the RTC6 board. This guarantees that the reference values are available even after a restart.

Notes

- After **auto_cal**(Command = 0), the galvanometer scanners are in the same real Image Field position as before the command.
- The reference values should be determined when the machine is set up, for example, when the scan system is installed or exchanged. When exchanging the scan system, for which reference values have already been determined earlier and read out by **get_hi_pos**, these reference values can be transferred directly to the RTC6 board by **write_hi_pos**.
- Reference values should be determined under conditions (ambient temperature, load) typical for the application and after the overall system has fully attained its operating temperature. The reference values should always be determined only after a warm-up time of more than 20 minutes and not before the TempOK signal has been activated.
- Execution of **auto_cal**(Command = 0) typically lasts up to 10 seconds.
- Storing the reference values takes several ms and interrupts the 10 μs clock cycle of the DSP. For scan systems with clock cycle monitoring (interlock), this should be temporarily deactivated.

8.10.4 Calibration During the Application

Automatic Self-Calibration

Automatic self-calibration of the scan system during an application can be executed with **auto_cal**(Command = 1).

This starts a measurement routine for determining the current Home-In positions. From the deviations from the stored reference values (see Chapter 8.10.3 "Determining Reference Values", page 286) determined in the process, gain values and offset values are calculated and set separately for the x axis and y axis.

This drift compensation is effective immediately. It is valid until:

- **auto_cal**(Command = 1) is called again
- it is switched off
  - by **auto_cal**(Command = 2)
  - after **load_program_file**

Notes

- After **auto_cal**(Command = 1) the galvanometer scanners are in the same position in the real Image Field as before the command.
- The calibration routine started with **auto_cal**(Command = 1) typically lasts 1…2 seconds (depending on the strength of the drift). An ASC hardware check is automatically performed, if **auto_cal**(Command = 0 or 4) has not been executed prior to the first time call of **auto_cal**(Command = 1). This extends the execution of **auto_cal** by a few seconds.

Customer-Specific Calibration

Automatic self-calibration is midpoint centered, that is, the Image Field center remains stable.

If an alternative is preferred (for example, if the left edge should remain stable, size is irrelevant), then the calibration can also be externally calculated and set.

For such a customer-specific calibration, the Home-In positions determined by **auto_cal**(`Command` = 0) can be read out by **get_hi_pos**. From this, compensating gain values and offset values can be calculated and set by **set_hi**. The drift compensation set in this way is effective immediately.

Notes

- **get_hi_pos** returns the last measured Home-In positions. These are the stored Home-In reference values immediately after
  - **auto_cal**(`Command` = 0)
  - initialization by **load_program_file**
- After **set_hi**, a correction of the current position occurs at jump speed.
- Gain and offset correction factors can also be set by **set_hi** for systems *without* Home-In sensors.
- After **auto_cal**(`Command` = 3) the galvanometer scanners are in exactly the same position as before the command.

Supplemental Information about Calibration

- The accuracy of fit of the set drift compensation can decrease with increasing time. Therefore, calibration should be repeated after appropriate time intervals. The shorter the time interval between individual calibrations, the higher the attained long-term repeatability. Time intervals are typically in the range of minutes. Events such as workpiece changes or line feeds are ideal opportunities for conducting a new calibration.
- The accuracy of fit of the calibration, and the thereby attained long-term repeatability, are further enhanced by steady environmental and load conditions.
- The measurement routine determines the Home-In positions by several measurements. If deviations between the individual measurements are too large (maximum – minimum > 96 bits), the measurement routine aborts and an error 2 is returned. In this case, no reference values are stored for the affected axis (`Command` = 0) and the gain and offset factors remain unchanged with (`Command` = 1)(1). Then **get_hi_pos** returns 0 instead of the faulty values.
- An error within an individual measurement cycle can be caused by a brief mechanical or electrical disturbance. If significant spreading occurs within an individual measurement cycle, we recommend the following:
  - either a further measurement cycle is immediately conducted or
  - the error is initially ignored while using the current gain values and offset values until new correction values are determined by the next (successful) measurement cycle.

  Significant spreading across several measurement cycles might indicate a defect in the internal sensor system or another part of the scan system. But because continuous (mechanical or electrical) external disturbances or contamination can also impair automatic self-calibration, the scan system and its environment should in such cases be appropriately inspected to assess overall functionality.
- Certain hardware error states (for example, signal faulty or not found) get permanently stored. After correcting the error, you must call **auto_cal** with `Command` = 0 or 4 to clear the error state. Until this time, correction with `Command` = 1 or 3 is not possible.

(1) (`Command` = 0) always sets gain = 1.0 and offset = 0.

8.11 Camming

camming produces a marking that simulates the classic camshaft action of moving a valve tappet – or more generally a cam disk moving a lever. An example for a Camming process is shown in Figure 67.

The galvanometer scanner motion here is a lever motion defined as a 2D curve. It is written in a list as a closed point-by-point sequence of **mark_abs** commands.

The entire curve must fit within a contiguous list region, see **config_list**. Though it is not possible to switch among lists to load further portions of the curve.

"Propulsion" is furnished either by external fed in encoder pulses or by internally simulated ones.

Each outputted point is derived from the xy coordinate of a **mark_abs** at the position `FirstPos` + `Index`, whereby `Index = Round((EncoderCurrent - EncoderStart) × Scale)`. `FirstPos` is the first **mark_abs** command's list position and `Scale` is a freely selectable scaling factor. `EncoderStart` gets automatically determined when **camming** is called. There is no automatic encoder reset. The first outputted point is always `Index = 0`.

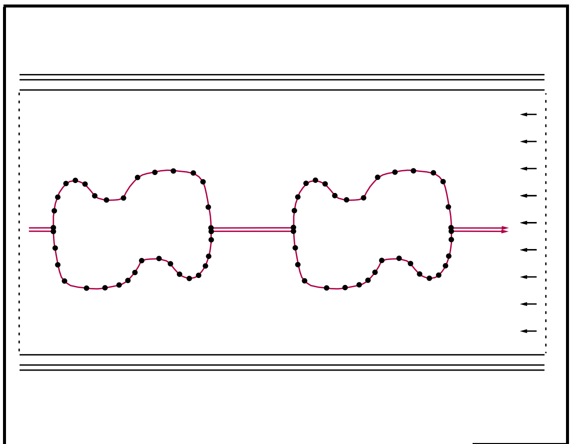
The individual points are executed every 10 μs as a "Hard Jump" without Scanner Delays.

The distance between two points (= "resolution") is freely selectable.

`Scale` determines how precisely the curve is sampled. The larger `Scale` is, the coarser is the piecewise linear approximation of the curve.

The number of encoder pulses per 10 μs clock cycle and the spacing of the points determine the actual mark speed.

"Resolution", `Scale` and encoder speed should suit the dynamic characteristics of the connected scan system.



67

Camming process example. A transport system moves a continuous workpiece. Two *scan heads* team up to mark the contours. Schematic depiction.

The Camming process can be controlled in various ways, see **camming** command description:

- `Ctrl` > 0
  The laser is controlled externally (with **laser_signal_on_list** before **camming** and **laser_signal_off_list** after **camming**)
- `Ctrl` = 0
  The laser is controlled RTC6 board-internally automatically (as with a normal Polyline with consideration of Laser Delays)

The curve can be executed once and then automatically ended (`Ctrl = 0` or `Ctrl = 1`). The list then continues by executing the next command that follows the end of the point list (the length of the point list is defined by `NPos`).

In accordance with encoder direction point lists can also run backward. Then, the curve is terminated with Index = 0.

The curve can be repeated indefinitely (`Ctrl = 2` or `Ctrl = 3`). To avoid "Hard Jumps" at the start of a repeat, the point list should represent a closed curve. Indefinite repeating must be canceled by **stop_execution** or /STOP.

At any time, the curve can be restarted at the address defined by **set_extstartpos** or **set_extstartpos_list** (typically but not necessarily `FirstPos`) by a /START (independently of the current index, possibly hard-jumped to the first marking position).

Notes

- A /START during list execution is:
  - Suppressed during normal operation
  - Not suppressed during a Camming process with indefinite repeating

If `Ctrl = 2`, then the Camming process waits at the end of the point list for a new start. If `Ctrl = 3`, then processing automatically starts again from the beginning (the point list is processed as a ring buffer).

Furthermore, the Camming process can be combined with Processing-on-the-fly (which can apply its own scaling if different from **camming** parameter `Scale`).

The laser control and Vector-Defined Laser Control can be combined (**set_vector_control**). This requires to use **para_mark_abs**[*] commands instead of **mark_abs[*]** commands. The value defined there is outputted immediately, thus allowing systematic variation of laser power along the curve. If Vector-Defined Laser Control is not enabled, then no laser power is outputted by the **para_mark_abs**[*] calls.

In addition, a combination with Encoder-Speed-Dependent Laser Control is possible.

Alternatively, **mark_abs_3d** or **para_mark_abs_3d** as well as **timed_mark_abs_3d** can be used.
However, the z coordinate and `T` parameter is ignored during outputting. Other commands within point lists are likewise ignored.
Point output then remains unchanged for one clock cycle.

Notes for Testing

- If a curve point list is finished by **set_end_of_list** and does not cross the boundary between List 1 and List 2 (see Chapter 6.4 "List Handling", page 108), then the curve shape can be tested using a normal execution start, for example, by `execute_at_pointer( Pos )`
  (to some extend as a Polyline, but with a predefined mark speed and using the currently defined "Variable Polygon Delay").
- The point list should be closed with **list_return**, if it is part of a subroutine and a **sub_call** call is used for testing.
- If the test should include the "Hard Jump", then **timed_mark_abs_3d** (with `Time` = 10 μs) can be used as well, but not **timed_para_mark_abs_3d**.

8.12 Time Measurements

8.12.1 RTC6 Timer

RTC6 PCIe Boards are equipped with an integrated timer, which is referred to as the "RTC6 Timer" in the following. The RTC6 Timer only counts clock cycles of list commands. Counting is paused during interruptions by **set_wait** or **pause_list**.

In order to measure the marking time consumed by any particular marking process, **save_and_restart_timer** is called before and then after the marking process. **save_and_restart_timer** saves the present RTC6 Timer value and resets it to 0. The elapsed time can then be read by **get_time**, which returns the RTC6 Timer value saved during the most recent call of **save_and_restart_timer**.

The present RTC6 Timer value can be read out by **get_lap_time**. It returns the elapsed time since the last call of **save_and_restart_timer** but without resetting the RTC6 Timer to zero. In this way the interim execution time of lengthy marking processes can be monitored.

8.12.2 Timestamps

32-bit "Timestamp Counter"

There is also a 32-bit "Timestamp Counter" on RTC6 boards. It starts counting at 0 with **load_program_file** and counts uninterruptible all 10 μs clock periods. Using the 32-bit "Timestamp Counter", an absolute reference to the individual time periods can be established, even if the RTC6 Timer has been reset by **save_and_restart_timer**. The 32-bit "Timestamp Counter" can be recorded by **set_trigger[*]** (signal 52) and is read out by **get_value**(52).

store_timestamp_counter / **store_timestamp_counter_list** reads out the 32-bit "Timestamp Counter" from the RTC6 board and buffers it there as the time reference `TimeStampStorage`.

Subsequently, **wait_for_timestamp_counter** or **wait_for_timestamp_counter_mode** can be used to wait in a list for a fixed time difference to the stored 32-bit "Timestamp Counter". Overflows of the 32-bit "Timestamp Counter" are detected and corrected.

About 12 hours is the longest waiting time that can be specified with the 32-bit offset. Waiting times beyond this must be broken down into shorter blocks and processed sequentially with sequences of **store_timestamp_counter_list** and **wait_for_timestamp_counter** / **wait_for_timestamp_counter_mode**. This is not always practicable with long process times.

64-bit "Timestamp Counter"

As of DLL 624, OUT 624 **get_timestamp_long** with 64-bit "Timestamp Counter" and **wait_for_timestamp_counter_long** with 64-bit offset are available as an alternative. **get_timestamp_long** reads out the 64-bit "Timestamp Counter" (= `TimeStampCounterLong`) from the RTC6 board. Its lower part (= `returned parameter value TimeStampL`) and the 32-bit "Timestamp Counter" are identical.

wait_for_timestamp_counter_long waits for an absolute 64-bit "Timestamp Counter" value(1) (64-bit offset). The user program must control the synchronization.

Notes

- **get_time** and **get_lap_time** only take list execution times into account.
- To compare the RTC6 board-internal **save_and_restart_timer** time measurement to an external time measurement via the BUSY Pin, you should insert a **list_nop** between **save_and_restart_timer** and **set_end_of_list**. This ensures that a Scanner Delay is processed before **set_end_of_list**. Without **list_nop**, **save_and_restart_timer** includes the Scanner Delay in its measurement even though it completes only after **set_end_of_list** (and therefore the BUSY Pin is already LOW).

(1) Analogously to **wait_for_timestamp_counter_mode**, however, **store_timestamp_counter** / **store_timestamp_counter_list** is not used.

8.13 *SCAN*ahead Functionality

Due to the SCANahead Servo Control, SCANahead Systems exhibit a different dynamic behavior than Tracking Error Systems. Refer to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual.

When synchronously controlling SCANahead System, laser and peripheral devices, the different timing must be taken into account.

For this purpose, the RTC6 command set contains dedicated [*]scanahead[*] Commands for switching on the SCANahead Functionality on the RTC6 board. These allow existing RTC6 user programs for Tracking Error Systems to be adapted very easily for SCANahead Systems.

To do this, you essentially only need to add some lines of code:

```
set_scanahead_params( 1, 1, 1, 0, 0, 0 );
```

```
activate_scanahead_autodelays( 1 );
```

Optional in addition:

```
set_scanahead_line_params(n, n, n);
```

However, the RTC6 board must be equipped with the Option "SCANa" to execute this program code.

8.13.1 RTC6 *SCAN*ahead Functions

SCANahead Systems internally calculate a set trajectory with limited, constant accelerations, whereby the acceleration time depends on speed changes, see "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual. As a consequence, Scanner Delays and Laser Delays needed for taking acceleration time into account depend on speed. The RTC6 board can automatically calculate the required Scanner Delays and Laser Delays.

For this, the SCANahead Functionality must be switched on on the RTC6 board by 2 RTC6 command and thus the control suitable for SCANahead System.

(1) **set_scanahead_params**(`Mode =1`)

The RTC6 board queries relevant information from the SCANahead System, processes it (correction file) and holds it for further usage. Additionally, the RTC6 board laser control section gets prepared for taking `PreviewTime` into account. This affects LASER DIGITAL OUT1/2 etc., see Notes on Controlling Peripherals, page 301.

(2) **activate_scanahead_autodelays**`(1)`

The RTC6 board is instructed:

- To use the results of (1)
- To ignore delays set in source code by **set_scanner_delays** and **set_laser_delays**
- To automatically calculate (for high contour fidelity) optimal speed-dependent Scanner Delays and Laser Delays[(1)].
- To also set these dynamically in real time during list execution.
- With Sky Writing, all required parameters are automatically calculated as well (for example, run-in motions and run-out motions), see also Notes on Sky Writing, page 301.

(1) With all RTC6 commands that require delays: Jump Commands, Mark Commands, "Arc" Commands.

Thus, markings with high contour fidelity[1] can be effortlessly and immediately created (without the need to determine and optimize Scanner Delays and Laser Delays by yourself).

Of course, the marking result can be further influenced by means of corresponding RTC6 commands:

- Scanner Delays are affected by the `CornerScale` and `EndScale` parameters[2] of **set_scanahead_line_params** and **set_scanahead_line_params_list**.
  The effects of these parameters are shown in Figure 68.
  You can choose between optimal contour fidelity and optimal process speed at the cost of contour fidelity (*process optimization*).
- Laser Delays are affected by the `AccScale`[3] parameter of **set_scanahead_line_params** and **set_scanahead_line_params_list**.
  The effect of this parameter is shown in Figure 69 which demonstrates a line marked using three different parameter values: the energy deposition at line ends is influenced by partially or wholly show/hide acceleration phases.
- Available for *fine-tuning* Laser Delays are **set_scanahead_laser_shifts** and **set_scanahead_laser_shifts_list**. They let you set a temporal offset for the laser switching time points, for example, to compensate signal propagation times[4].

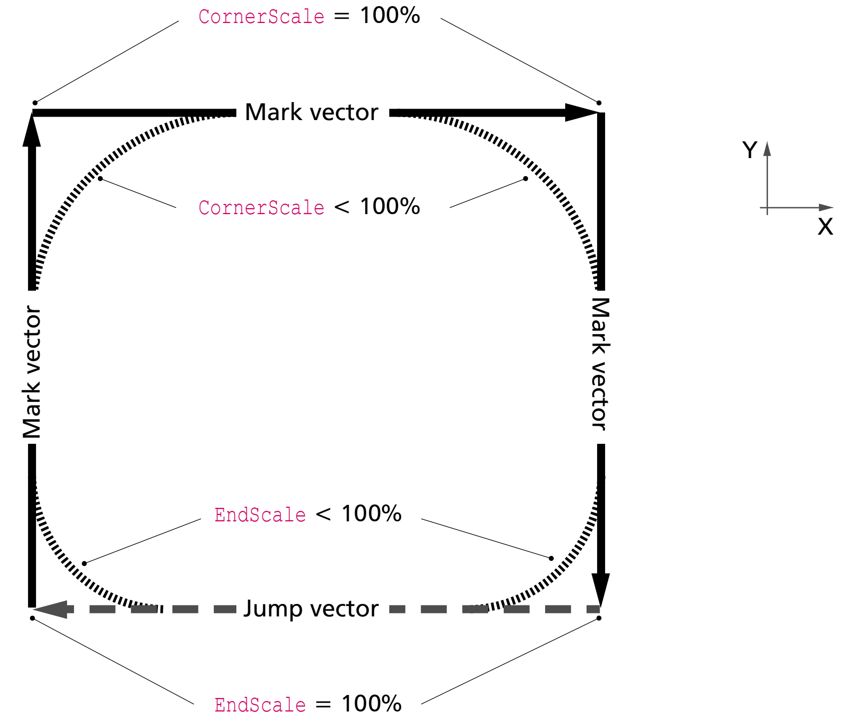
(1) Provided that the specified mark speed makes sense. The jump speed, on the other hand, is adapted to the jump distance.

(2) These parameters only affect the galvanometer scanner positioning. They *do not* affect points in time when the laser is to be switched on and off.

(3) This parameter only affects points in time when the laser is to be switched on and off. It *does not* affect the galvanometer scanner positioning!

(4) The following applies for ≥ DLL 605: the transport delay from the RTC6 board to the scan system (20 μs) is automatically taken into account.

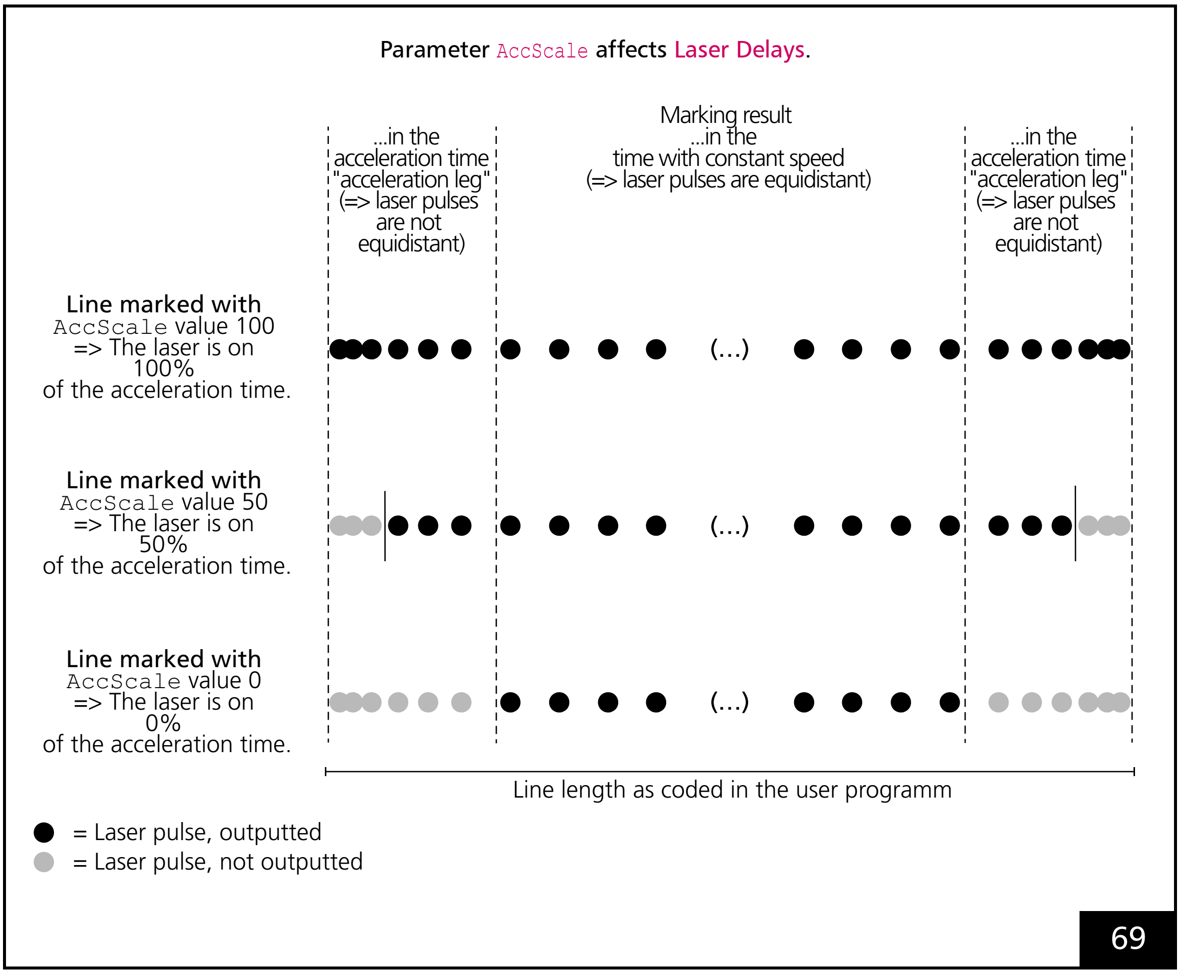
Parameter CornerScale and EndScale affect Scanner Delays.



Parameter CornerScale	Controls the marking accuracy at mark vector to mark vector alternations (“rounding of corners”). 100% = sharp corners.
Parameter EndScale	Controls the marking accuracy at mark vector to jump vector alternations (”line end”) and at jump vector to mark vector alternations (“line start”). 100% = straight line ends/line starts.

68

set_scanahead_line_params and **set_scanahead_line_params_list**: the parameters CornerScale and EndScale affect the Scanner Delays.



set_scanahead_line_params and **set_scanahead_line_params_list**: the parameter `AccScale` affects Laser Delays.

With `AccScale` values < 100 the marked line lengths are shorter than they are actually coded in the user program. With these values the laser is not always on during acceleration times. That is, portions of the acceleration legs are not marked. The acceleration leg lengths are proportional to the square of the acceleration time.
The portion of each acceleration leg which is *not marked* corresponds to (1 – `AccScale` value / 100)² × 100%.
For example, 25% of each acceleration leg is not marked with an `AccScale` value 50.

Note: To achieve the line lengths as coded in the user program *completely and with equidistant laser pulse spacing*, you must use Sky Writing.

8.13.2 Quick Start for Advanced Users

In this Chapter:

Example Code (C++)

The following example code snippets illustrate basic elements of an RTC6 user program for a SCANahead System to initialize the RTC6DLL.dll and RTC6 board (that is, it is not a complete program code).

Because this example uses implicit linking, the files RTC6impl.h and RTC6DLL.lib are required. Additionally, the program needs to be able to load RTC6DLL.dll. If the operating system can not find RTC6DLL.dll during program start, then it responds with an error message and terminates the program.

```
// SAMPLE SOURCE CODE SNIPPETS: TO MARK A SQUARE.
//
// RTC6 header file for implicit linking to RTC6DLL.dll.
// also link to Visual C++ import library RTC6DLL.lib,
// to compile an executable
#include "RTC6impl.h"

// scanahead line params
// polyline: 100%: max precision, 0% min execution time
CornerScale = 100;
// mark, jump: 100%: max precision, 0% min execution time
EndScale = 100;
// 100%: laser is on during acceleration and deceleration
//   0%: laser is off during acceleration and deceleration
AccScale = 100;

// enable/disable variable Polygon Delay or Sky Writing
VariablePolygonDelay = 0;
SkyWriting = 0;
int main()
{

// initialize
init_rtc6_dll();
// load RTC firmware
if( 0 != load_program_file( NULL ) ) { return -1; }
```

```
if( 0 != load_correction_file( NULL, 1, 2 ) ) { return -2; }
set_mark_speed_ctrl( 25000 );
set_jump_speed_ctrl( 50000 );

// laser control
// high active laser signals
set_laser_control( 0 );
// mode 1 supports YAG
set_laser_mode( 1 );
set_firstpulse_killer( 640 );
set_laser_pulses_ctrl( 320, 128 );

// SCANahead: calculate delays automatically.
// SCANahead System must be connected and powered.
//
// place anywhere after load_correction_file
set_scanahead_params( 1, 1, 1, 0, 0, 0 );
activate_scanahead_autodelays( 1 );
set_scanahead_line_params( CornerScale, EndScale, AccScale );

// fine tuning LaserON and LaserOFF. 1 bit = 1/64 µs
set_scanahead_laser_shifts( 1600, 1600 );

// Sky Writing and variable Polygon Delay
set_delay_mode( VariablePolygonDelay, 0, 2^30, 0, 0 );
set_sky_writing_para( SkyWriting, 0, 0, 0 );
set_sky_writing_mode( 3 );
set_sky_writing_limit( 0.5 );

// create and execute list
set_start_list( 1 );
jump_abs( 50000, -50000 );
mark_abs( 50000, 50000 );
mark_abs( -50000, 50000 );
mark_abs( -50000, -50000 );
mark_abs( 50000, -50000 );
jump_abs( 0, 0 );
set_end_of_list();
execute_list( 1 );
}
// END OF SAMPLE SOURCE CODE SNIPPETS.
```

Modifying Existing User Programs for SCANahead Systems – Basic Steps

It is assumed that it is a complete RTC6 user program or RTC5 user program for controlling a Tracking Error System, for example, intelli*SCAN* III.

These modifications should already suffice for successful execution of your user program.

Any source code sections for "Variable Polygon Delays" and Sky Writing should thereby also be runnable.

(1) After **load_correction_file** at the beginning of your user program, be sure to insert a **set_scanahead_params** call (for parameters, see the command description).

This activates the SCANahead Functionality of the RTC6 board (incl. laser control). `Mode = 1` initiates direct querying of the SCANahead System information on `PreviewTime`, `Vmax` and `Amax`. Therefore, a SCANahead System must be connected to the RTC6 board and already and switched-on during user program runtime.

(2) Insert **activate_scanahead_autodelays**`(1)`. This switches on automatic calculation of Scanner Delays and Laser Delays.
All calls of **set_scanner_delays** and **set_laser_delays** in subsequent code lines have no effect, see also Comments, page 351 for **activate_scanahead_autodelays**.

(3) If the original program is an RTC5 user program, then also make the following changes:

– Change `init_rtc5_dll` to `init_rtc6_dll`
– Change `rtc5_count_cards` to `rtc6_count_cards`
– Change `free_rtc5_dll` to `free_rtc6_dll`
– *Important*
  • *DO NOT* change `set_rtc4_mode` to `set_rtc6_mode`
  • *DO NOT* change `set_rtc5_mode` to `set_rtc6_mode`

If neither **set_rtc4_mode** nor **set_rtc5_mode** is present, then the default state with RTC5 boards is **set_rtc5_mode**. Insert **set_rtc5_mode** to the RTC6 user program, if certain parameter values are to be automatically converted from RTC5 units to RTC6 units, see Section "RTC5 Compatibility Mode", page 173.

(4) Check whether your code is affected by the restrictions mentioned in Chapter "Modifying Existing User Programs for SCANahead Systems – Observing Restrictions", page 298.

Modifying Existing User Programs for SCANahead Systems – Observing Restrictions

When further adapting the user program for the SCANahead System, you must take the following points into account.

- Non-Usable Functionalities of the RTC6 Board and RTC6 DLL, page 298
- Non-Usable iDRIVE Functionalities, page 298

Non-Usable Functionalities of the RTC6 Board and RTC6 DLL

- *After activation of* SCANahead Functionality *functionality by* **set_scanahead_params**, the following RTC6 functionalities are *not* usable (status ≤ DLL 615):
  - **load_varpolydelay**
  - **laser_on_pulses_list** cannot be used for SCANahead Systems, if external signal pulses are required at DIGITAL IN1 (**laser_on_list** can be used with SCANahead Systems)

Non-Usable i*DRIVE* Functionalities

- The following i*DRIVE* functions are not available for SCANahead Systems:
  - Changing the effective calibration (**SetPositionScale**)
    `control_command( Data = 12nn`$_H$`)`
- The following i*DRIVE* functions are not available for SCANahead Systems with firmware version ≤ 5.05.4 (see Notes, page 24):
  - Switching between tunings (**SelectControlDefinition**)
    `control_command( Data = 11nn`$_H$`)`

Modifying Existing User Programs – Further Steps, Background Info on SCANahead System-related Functionalities

While continuing to modify your user program for the SCANahead System, keep in mind the following:

- Notes on Using RTC6 commands for Scanner Delays and Laser Delays, page 299
- Notes on [*]arc[*] Commands, page 300
- Notes on Variable Polygon Delays, page 300
- Notes on Sky Writing, page 301
- Notes on Coordinate Transformations, page 301
- Notes on Controlling Peripherals, page 301
- Notes on Transmitted Data Signals, page 302

Notes on Using RTC6 commands for Scanner Delays and Laser Delays

When using RTC6 commands with the SCANahead System to synchronize scan motions and Laser Control Signals, observe the following:

- All relevant *SCAN*ahead parameters (article-number-specific) are stored on the SCANahead System during manufacturing. Although **set_scanahead_params**`( Mode = 2 )` allows other values, we recommend only using the values queried from the SCANahead System. `Mode = 2` is primarily intended for software development without a SCANahead System attached.
- For SCANahead Systems, the actual position of the scan motion lags the control values by `PreviewTime`, see Figure 1[(1)]. For synchronous laser control, the Laser Control Signals also need to take `PreviewTime` into account. This could be done manually by appropriately modifying parameters of **set_scanner_delays** and **set_laser_delays**. But here, you would have to take the speed-dependent acceleration time of the SCANahead System into account.
  The preferable alternative is to simply use **activate_scanahead_autodelays** for automatic calculation of Scanner Delays and Laser Delays. Then, the following applies:
  – The existing parameters for **set_scanner_delays** and **set_laser_delays** are invalidated by **`activate_scanahead_autodelays`**`( Mode = 1 )` and not modified. They become immediately effective again, if automatic delay calculation gets deactivated by **`activate_scanahead_autodelays`**`( Mode = 0 )`.
  – `set_scanahead_params( Mode = 1 or 2, HeadNo, TableNo, PreviewTime, Vmax, Amax )` defines parameters for automatic calculation of Scanner Delays and Laser Delays together with **`activate_scanahead_autodelays`**`( Mode = 1)`.
- If needed, throughput of laser-marked workpieces can be increased, if lower quality is acceptable (“process optimization”): see `set_scanahead_line_params( CornerScale, EndScale, AccScale )`.
  See also Figure 68 and Figure 69.
- For *fine-tuning* Laser Delays (for example, to compensate signal propagation times to the laser or to take laser system switching behavior into account), see `set_scanahead_laser_shifts( dLasOn, dLasOff )`.

(1) This temporal offset differs quantitatively and qualitatively from Tracking Error in Tracking Error Systems. Refer to “excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards” Manual.

Notes on [*]arc[*] Commands

- SCANahead Systems have a nominal Tracking Error of 0(1). Tracking-related artifacts (for example, necking during small, quick circular motions) do not occur as long as the maximum acceleration of the SCANahead System is not exceeded (for example, when in the source code a too large speed in relation to the circular diameter has been specified).
- To estimate maximum acceleration within the Image Field, the maximum acceleration in programming bits (Image Field coordinates) can be queried by `get_scanahead_params( HeadNo = 256 )`. For conversion to m/s^2, see Comments, page 458.
- The allowed velocity range for a given circle radius can be estimated by:
  $v^2/r <$ `Amax`(2)(3)(4)
  - Note that **arc_abs**, **arc_abs_3d**, **arc_rel**, **arc_rel_3d** apply rounding upon Microstepping. Thus, the speed which the RTC6 commands to the SCANahead System may differ from the mark speed (specified in **set_mark_speed**). In cases where the acceleration (resulting from the commanded speed) exceeds the maximum acceleration `Amax` of the SCANahead System, then circles are smaller as programmed.
  - With **timed_arc_abs** and **timed_arc_rel** – to prevent rounding – you must specify the execution time in multiples of 10 µs.

Notes on Variable Polygon Delays

- Variable Polygon Delays have the same effect for SCANahead Systems as for Tracking Error Systems.
  To activate variable polygon Scanner Delays, you must call **set_delay_mode**(`VarPoly` > 0) as well. Its parameters `EdgeLevel`, `MinJumpDelay` and `JumpLengthLimit` have no effect, if automatic delay calculation (**activate_scanahead_autodelays**) is switched on. But they become immediately effective if automatic delay calculation gets switched off. In contrast, the `DirectMove3D` parameter is always effective.
- **load_varpolydelay** has no effect with a SCANahead System.
- No user-defined "Variable Polygon Delay" tables are available.

(1) The output delayed by `PreviewTime` is not Tracking Error in the usual sense.

(2) Query `Amax` by `get_scanahead_params`(`HeadNo` = 256). Observe units .

(3) Applies to frequencies ≤ 6,25 kHz (excelli*SCAN* 14) as well as ≤ 4,00 kHz (excelli*SCAN* 20, excelli*SCAN* 30); $f = v / (2\pi \times r)$.

(4) Valid for excelli*SCAN* with any objective. Insert either control bits or programming bits into the formula). See also second list bullet.

Notes on Sky Writing

- Sky Writing has the same effect for SCANahead Systems as for Tracking Error Systems. All 3 modes are available. To switch on Sky Writing, you need to call **set_sky_writing_para**(`Timelag` > 1/4 µs) or **set_sky_writing**(`Timelag` > 1/4 µs) as well. If automatic delay calculation (**activate_scanahead_autodelays**) is switched on, then `Timelag` (except for the actual activation of Sky Writing) and the parameters `Nprev` and `Npost` have no effect. But they immediately become effective, as soon as automatic delay calculation is switched off. In contrast, parameter `LaserOnShift` is always effective.
- If Sky-Writing is switched on, only the parameter `EndScale`(1) is effective.
- The other parameters(1) `CornerScale` and `AccScale` are not applied – except there is *no* Sky Writing motion:
  - If Sky Writing Mode 3 or Sky Writing Mode 4 is activated and the angular limit is undercut, `CornerScale` is effective.

Notes on Coordinate Transformations

When using RTC6 commands to transform coordinates, observe the following:

- A coordinate transformation should *not* change the shape and size(2) of the marking, that is, the (2 x 2) total matrix M (see Chapter 8.2 "Coordinate Transformations", page 233) is ideally a pure rotation matrix.

Notes on Controlling Peripherals

When using RTC6 commands to control peripherals (pins for ANALOG OUT1 and ANALOG OUT2 on the LASER Connector, EXTENSION 1 Socket Connector, EXTENSION 2 Socket Connector), observe the following.

If the SCANahead Functionality of the RTC6 board has been activated by **set_scanahead_params** (`PreviewTime` > 0) the following applies:

(1) The control commands **write_io_port**, **write_8bit_port**, **write_da_x**, **write_da_1** and **write_da_2** are immediately executed ("asynchronous")

(2) The list commands **write_io_port_list**, **write_8bit_port_list**, **write_da_x_list**, **write_da_1_list** and **write_da_2_list** are executed delayed by the `PreviewTime`, that is, simultaneous with the galvanometer scanners ("axissynchronous")

(3) For a "laser synchronous" output of a laser power control, you must use **set_laser_power** instead of the list commands from (2).

If the SCANahead Functionality of the RTC6 board has not been activated (`PreviewTime` = 0), the following applies:

- asynchronous execution (1) and axissynchronous execution (2) are identical

With the RTC6 laser control mode (**set_dsp_mode**`( 3 ))`, LaserOn Delays during a series of short vectors can extend across one or several subsequent vectors. The RTC5-compatible synthetic Scanner Delays in DSP mode 2 are omitted. Output is then synchronous with the LaserOn Delay assigned to the respective vector.

With the RTC5-compatible laser control (**set_dsp_mode**`( 2 ))` axissynchronous execution and lasersynchronous execution are identical.

(1) See **set_scanahead_line_params**, **set_scanahead_line_params_list**.

(2) Quality losses are likely, if size changes more than 5%.

Notes on Transmitted Data Signals

When adapting the user program, also note that some data signals sent by SCANahead System differ content-wise from that of Tracking Error Systems (for example, intelli*SCAN* III).

- **TrAck Signal**
  (instead of **PosAck Signal**)
- **Trajectory Error**
  (instead of Position error)

9 Programming Peripheral Interfaces

Scan systems are often used in equipment that needs to synchronize processing by the laser and scan system with other process steps (for example, workpiece placement, robotic motion, process monitoring etc.).

For this purpose, the RTC6 PCIe Board provides a variety of peripheral interfaces, see Chapter 4.6 "Interfaces for the Laser and Peripheral Equipment", page 75.

With the commands for programming these interfaces, you can supplementally and/or synchronously control the following in addition to lasers and scan systems:

- Signals transmitted for peripheral control
- Querying and evaluation of peripheral signals
- Control and synchronization of laser scan processes and peripheral control by external control signals

9.1 Signal Output

For peripheral control (for example, controlling a workpiece transport system or a shutter), appropriate signals can be outputted by the interfaces described below.

The output values can be changed at any time by control commands or – during processing of a list – by list commands.

9.1.1 16-Bit Digital Output Port

The EXTENSION 1 Socket Connector provides a buffered 16-Bit Digital Output Port (DIGITAL OUT 0…DIGITAL OUT 15). The level of its output signals must be configured by a jumper, see Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81.

16-bit digital values are assigned to the output port by **write_io_port_list**, **write_io_port**, **write_io_port_mask_list**, **write_io_port_mask** or **write_port_list**. The output is in high-impedance mode until an initial value is assigned to it.

In addition, **set_port_default** (`Port` = 3) can be used to specify the value to which the output port is set as soon as processing of a list is canceled by **stop_execution** or an external stop signal.

The default value also takes effect:

- With Position-Dependent Laser Control and Speed-Dependent Laser Control (see **set_port_default**)
- Upon terminating Pixel Mode

If "Automatic Laser Control" is activated with `Ctrl=6` from **set_auto_laser_control**, then the value at the 16-Bit Digital Output Port automatically gets adjusted, see Chapter 7.4.9 ""Automatic Laser Control"", page 205. This can be recorded by **set_trigger[*]** (signal 24).

When the output value is outputted, a LATCH signal is outputted at the EXTENSION 1 Socket Connector as a trigger signal for synchronization of data transmission.

By **get_io_status**, the current values of the 16-Bit Digital Output Port can be read out.

9.1.2 8-Bit Digital Output Port

The EXTENSION 2 Socket Connector provides a (jumper-configurable) buffered 8-Bit Digital Output Port, see Section "8-Bit Digital Output Port", page 84.

Its output values can be set by **write_8bit_port**, **write_8bit_port_list** or **write_port_list**. The output is in high-impedance mode until an initial value is assigned to it. In addition, **set_port_default** (`Port` = 2) or **set_laser_off_default** can be used to define the value to be outputted at the 8-Bit Digital Output Port, as soon as processing of a list has ended with **stop_execution** or by an external stop signal.

The default value also takes effect:

- With Position-Dependent Laser Control and Speed-Dependent Laser Control (see **set_port_default**)
- Upon terminating Pixel Mode

If "Automatic Laser Control" is activated with `Ctrl` = 3 from **set_auto_laser_control**, then the value at the 8-Bit Digital Output Port automatically gets adjusted, see Chapter 7.4.9 ""Automatic Laser Control"", page 205. This can be recorded by **set_trigger[*]** (signal 24).

When the output value is outputted, a LATCH signal is outputted at the EXTENSION 2 Socket Connector as a trigger signal for synchronization of data transmission (provided that pin (17) has been correspondingly configured by the jumper setting, see Section "8-Bit Digital Output Port", page 84.

9.1.3 2-Bit Digital Output Port

The LASER Connector provides a buffered 2-Bit Digital Output Port, see Section "2-Bit Digital Output Port", page 77.

By **set_laser_pin_out**, **set_laser_pin_out_list** or **write_port_list**, values are assigned to this output port.
The value is 0 (pins are LOW) until an initial value is assigned to it.

In addition, **set_port_default** (`Port` = 4) can be used to define the value to be outputted at the 2-bit digital output port, as soon as processing of a list is canceled either by **stop_execution** or an external stop signal.

9.1.4 12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2

The LASER Connector provides the 12-Bit Analog Output Port 1 ANALOG OUT1 and 12-Bit Analog Output Port 2 ANALOG OUT2, see Section "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 77.

12-Bit Analog Output Port 2 ANALOG OUT2 is also available by the MARKING ON THE FLY Socket Connector, see Section "Analog Output Port", page 85.

The output values of 12-Bit Analog Output Port 1 ANALOG OUT1 and 12-Bit Analog Output Port 2 ANALOG OUT2 can be separately set by

- **write_da_x**
- **write_da_x_list**
- **write_port_list**

The values are 1 (corresponds to approx. 0 V) until initial values are assigned to it.

In addition, **set_port_default** (`Port` = 0 or 1) or **set_laser_off_default** can be used to define the values to be outputted at 12-Bit Analog Output Port 1 ANALOG OUT1 and 12-Bit Analog Output Port 2 ANALOG OUT2, as soon as processing of a list has ended with **stop_execution** or by an external stop signal.

The default value also takes effect together with Position-Dependent Laser Control and Speed-Dependent Laser Control (see **set_port_default**). If accordingly preset by corresponding commands, the values at 12-Bit Analog Output Port 1 ANALOG OUT1 and 12-Bit Analog Output Port 2 ANALOG OUT2 are changed as well:

- By a softstart, see Chapter 7.4.7 "Softstart Mode (not yet implemented)", page 202
- In Pixel Output Mode, see Chapter 8.7 "Pixel Output Mode", page 272

If "Automatic Laser Control" is activated with `Ctrl` = 1 or `Ctrl` = 2 from **set_auto_laser_control**, then the value at 12-Bit Analog Output Port 1 or 12-Bit Analog Output Port 2 automatically gets adjusted, see Chapter 7.4.9 ""Automatic Laser Control"", page 205. This can be recorded by **set_trigger[*]** (signal 24).

9.1.5 Controlling Stepper Motors

Output Signals

The signals (ENABLE, DIRECTION and CLOCK) for controlling up to two stepper motors are outputted at the STEPPER MOTOR Socket Connector:

- You can appropriately change the ENABLE signal (to switch motor current on or off) during initialization by **stepper_init** and afterward by **stepper_enable** or **stepper_enable_list**.
- The RTC6 PCIe Board generates periodic CLOCK signal pulses (during reference runs by **stepper_init** and set-position motions by **stepper_abs** etc.). With each CLOCK pulse, the stepper motor executes a single step. You can adjust the CLOCK signal's pulse period (and thereby the speed of stepper motor motion) during initialization by **stepper_init** and afterward by **stepper_control** or **stepper_control_list** (the period is specified in units of 10 μs cycles).
- You can explicitly set the DIRECTION signal (and thereby the direction of stepper motor motion) during reference runs by **stepper_init**. In contrast, the DIRECTION signal is internally controlled during set-position motions by **stepper_abs** etc.: the signal gets set (to HIGH) if the next CLOCK pulse (in accordance with the defined set-position value) would increase the internal position variable. The DIRECTION signal also remains set for the cycles between two clock cycles and even when the stepper motor has reached its set position. Only upon an actual change of direction does the DIRECTION signal correspondingly change in place of a CLOCK pulse. Here, output of the next CLOCK pulse is delayed by a full CLOCK pulse period (undefined truncation of CLOCK pulse periods never occurs).

Notes

- For signal specifications, see Chapter 4.6.7 "STEPPER MOTOR Socket Connector", page 90.
- For querying signals, see Section "Querying Signals and Status Values", page 308.
- Stepper motor signals are outputted independently of any executing lists. A **set_end_of_list**, **pause_list**, **set_wait**, **stop_execution** or external stops do not terminate or pause a forwarding motion.
- For changes of direction or pulse period, the new values do not become active until an already-begun period is complete. Thus, pulse intervals are never be shorter than the currently defined value. For change of direction, an additional empty period (without CLOCK pulse) gets inserted.

Reference Runs and Position Initialization

With **stepper_init**, you can initiate reference runs to limit switches. Here, the desired direction can be specified by the `Dir` parameter. To ensure that, despite mechanical play or long signal transit times, the reached end position still does not lie beyond the limit switch, you can define a tolerance value `Tol` that moves the stepper motor in the opposite direction after reaching the limit switch.

SCANLAB recommends executing a (fast) reference run with a short CLOCK pulse period `Period` first and then a further (shorter but slower) reference run with a longer CLOCK pulse period.

Once the reference run has been successfully completed, the position variable (for the current position) is set to the value defined by parameter `Pos` as the reference for subsequent set-position motions. The reached reference position is offset from the limit switch position by ±`Tol` (direction dependent).

During reference run, the status is "Init" (Bit #5 = 1), see Section "Querying Signals and Status Values", page 308. The limit switch position is traversed 4 times and a mean limit switch position is calculated from this. Then the stepper motor moves away from the limit switch by `Tol` steps in the opposite direction to **stepper_init** `Dir`. `Tol` must be large enough to overcome a possible hysteresis. During this set-position motion, see Section "Set-Position Motions", page 307, the status is no longer "Init" but "Busy" (Bit #4 = 1). If you select `Pos` = `Tol` with **stepper_init**, the average position of the limit switch is 0. The only resulting inaccuracy is the fluctuation of the limit switch position measurement when the limit switch position is passed over 4 times.

If `Period` = 0 and/or `Dir` < 0, no reference run is carried out. Instead, the current stepper motor position becomes the new reference position (with the value newly defined in `Pos` as the position variable).

Because **stepper_init** always stops a previously begun stepper motor motion, you could also use **stepper_init** as an emergency stop for the stepper motor.

Parallel execution of reference runs for both stepper motors is also possible, but cannot be simultaneously started through a single command.

Set-Position Motions

Set-position motions can be initiated by **stepper_abs**, **stepper_rel**, **stepper_abs_no** and **stepper_rel_no** or the corresponding list commands.

Specify absolute set-position values for **_abs** commands and relative values for **_rel** commands (always as CLOCK pulse units). **_no** commands only produce set-position motions for *one* stepper motor output, the other set-position commands do so for both stepper motor output ports simultaneously.

With **stepper_wait**, you can interrupt further execution of a list until a started stepper motor motion is completed.

The list commands for set-position motions are short list commands. Therefore, a stepper motor motion can also execute synchronously with a galvanometer scanner motion.

If the limit switch activates during a set-position motion, then the motion immediately aborts and cannot be resumed as a normal set-position motion. You either have to request a reference run or mechanically or via the software, see below, deactivate the limit switch. If a stepper motor motion aborts once (for example, also by `Period` = 0), then the existing set position gets overwritten by the current position value. Therefore the stepper motor motion to the original set position cannot be resumed by eliminating the cause of interruption (limit switch or CLOCK pulse period = 0). Instead, it needs to be newly retriggered.

To work around this behavior, the consideration of the limit switch can be deactivated by **stepper_disable_switch**. Then, the "release" can occur without carrying-out an initialization motion once again, even if a limit switch is present. The deactivation is especially useful, if a continuously rotating rotary axis is controlled by the stepper motor. During an initialization motion a possible deactivation of a limit switch is not considered.

Querying Signals and Status Values

The current status of stepper motor signals (ENABLE, DIRECTION, CLOCK and SWITCH), the currently defined CLOCK pulse period and the current values of internal position variables for both stepper motors can be queried by **get_stepper_status**.

The **get_stepper_status** command also returns the Busy and Init statuses of both stepper motors. The Init status is set during a reference run and the Busy status during set-position motions.
As long as the Init status is set, no set-position commands (**stepper_abs**, **stepper_rel**, etc.) are permitted; control commands (except **stepper_init**) are denied with a **get_error** return code of `RTC6_BUSY` and list commands wait until the Init status gets reset. In some circumstances, the list itself or the motion process must be aborted.

Terminating Infinite Motions

Depending on chosen parameters, very long or even infinite stepper motor motions can be initiated by **stepper_init**, **stepper_abs**, for example, if no limit switch exists in the specified direction or if a very large set-position value is combined with a long pulse period.

- If an infinite motion is started by a control command (for example, **stepper_abs**), then this control command completes at the latest when the positive time (in seconds) supplied for the `WaitTime` parameter has expired. However, the stepper motor's infinite motion itself continues and you need to separately abort it by setting the CLOCK pulse period (for example, by the control command **stepper_control**) to 0 (emergency stop) or by defining an appropriate new set position.
- If you start an infinite motion by a list command (for example, **stepper_abs_list**) and wait for its completion by **stepper_wait**, then further list execution is blocked as long as the infinite motion is not aborted by a control command such as **stepper_control** with `Period` = 0 or an appropriate new set position. You could also abort the list by **stop_execution** or an external stop. But here, too, the stepper motor's infinite motion needs to be separately stopped with **stepper_control**(`Period` = 0).

WaitTime Parameter

The `WaitTime` parameter of the control commands can be used to set them to return to the calling program after the specified time (in seconds) has elapsed, regardless of whether the stepper motor motion is complete or not.

With `WaitTime` = 0, the command returns immediately. In this case, users must ensure that no unallowed commands are called. In particular, initialization should not be canceled before it is complete. Otherwise, this leads to incorrect reference positions.

9.1.6 RS-232 Interface

The RS232 Socket Connector provides an RS-232 interface, see Chapter 4.6.5 "RS232 Socket Connector", page 86.

For configuring the RS-232 interface, see Chapter 4.6.5 "RS232 Socket Connector", page 86.

rs232_write_data can be used to send single data words (bytes) to the RS-232 interface. Texts can be sent to the interface by **rs232_write_text** or **rs232_write_text_list**.

9.1.7 McBSP Interface

At the McBSP interface, see Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87, a 32-bit data word every 10 µs at DX0 pin (07) is permanently outputted.

The **set_mcbsp_out** and **set_mcbsp_out_ptr** commands allow selection of the signal types (analogously to **set_trigger**) to be outputted there:

- **set_mcbsp_out** lets you specify two signal types for simultaneous output at the McBSP interface. A 16-bit portion of the first signal type is packed along with a 16-bit portion of the second signal type into a 32-bit data word for output every 10 µs. For a detailed description, see **set_mcbsp_out**
- **set_mcbsp_out_ptr** lets you define a list of up to 8 signal types. The signals are outputted sequentially in the specified order. For every 10 µs clock cycle, the lower 24 bits of the corresponding data signal and the associated signal type number (8 bits) are packed into a 32-bit data word and outputted at the McBSP interface. For a detailed description, see **set_mcbsp_out_ptr**.

Notes

- For signals and operating conditions, see Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87.
- In the default setting, the McBSP interface always outputs Bit #4…Bit #19 of the Cartesian control values for the x axis and y axis (SampleX and SampleY) in a common 32-bit data word. This is equivalent to specifying `set_mcbsp_out(7, 8)`.
- Signals specified by **set_mcbsp_out** or are outputted (with **set_mcbsp_out_ptr** sequentially) until you call one of these two commands again.
- OIE Output Mode, page 758 can be set at the McBSP interface by **set_mcbsp_out_oie_ctrl** and **set_mcbsp_out_oie_list**.

9.2 Signal Input

Signals of peripherals (for example, signals of a transport system, workpiece recognition system or process monitoring camera) can be queried by the interfaces described below through control commands at any desired time or – during processing of a list – with list commands.

9.2.1 16-Bit Digital Input Port

For reading in 16-bit digital values, the EXTENSION 1 Socket Connector provides a protected 16-Bit Digital Input Port (DIGITAL IN 0…DIGITAL IN 15), see Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81.

For synchronization of data transmission, the EXTENSION 1 Socket Connector outputs a SYNC signal.

The **read_io_port** or **read_io_port_buffer** and **read_io_port_list** commands can be used to read the current value of the digital input port.

Further commands are provided for conditional command execution, see Chapter 9.3.2 "Execution of Conditional Commands", page 317.

9.2.2 2-Bit Digital Input Port

For querying 2-bit digital values, the LASER Connector provides a 2-Bit Digital Input Port, see Section "2-Bit Digital Input Port", page 77.

The input port can be read by **get_laser_pin_in**.

Further commands are provided for conditional command execution, see Chapter 9.3.2 "Execution of Conditional Commands", page 317.

9.2.3 RS-232 Interface

The RS232 Socket Connector provides an RS-232 interface for reading signals, see Chapter 4.6.5 "RS232 Socket Connector", page 86.

For configuring the RS-232 interface, see Chapter 4.6.5 "RS232 Socket Connector", page 86.

Data can be read in by **rs232_read_data**.

9.2.4 McBSP Interface

At the McBSP interface, see Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87, the 32-bit data word most recently fully transmitted to the specified memory location can be queried with **read_mcbsp**.

The interpretation as one 32 bit data word or two 16 bit data words is the responsibility of the user.

Signals (position values) received by the McBSP interface can also be integrated directly into Processing-on-the-fly correction of workpiece or scan-system motion (see Chapter 8.6 "Processing-on-the-fly", page 251 and Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 322) or can be used for an Online Positioning, see Chapter 8.3.1 ""Local Online Positioning"", page 237 and Chapter 8.3.2 ""Global Online Positioning"", page 240.

Notes

- For signals and operating conditions, see Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87.

9.3 Control by External Signals

The previously described input and output of peripheral signals can be synchronized with scan system control and laser control, as follows:

- The related list commands can be inserted in command lists at appropriate locations.
- Execution of related control commands can be made dependent on the current status of list execution. For this, the list status can be requested by **read_status**, see Chapter 6.4.2 "List Status", page 110 and the list execution status by **get_status**, see Chapter 6.4.3 "List Execution Status", page 111.
- In addition, the BUSY list execution status List Execution Status is provided by the BUSY OUT signal:
  - at the LASER Connector, see Section "BUSY List Execution Status", page 76
  - at the EXTENSION 1 Socket Connector, see Section "BUSY List Execution Status", page 82
  - at the MARKING ON THE FLY Socket Connector, see Section "BUSY List Execution Status", page 85

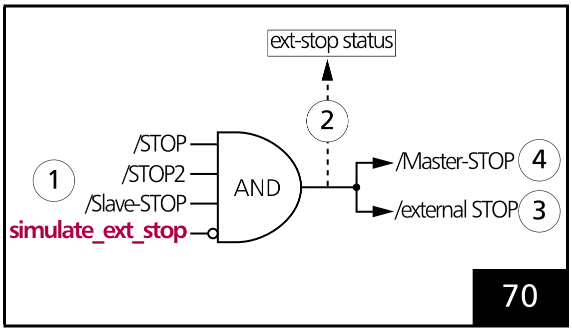
Moreover, the RTC6 PCIe Board provides commands and interfaces (described in the following sections) that allow external control signals (for example, from a light-barrier or from an encoder) to directly control and synchronize execution of command lists or individual commands (including the output of peripheral signals).

Moreover, the RTC6 PCIe Board provides interfaces to control and synchronize list execution directly with external signals.

9.3.1 Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization

External Stop

By a signal at the input ports /STOP, /STOP2 or /Slave STOP, or by **simulate_ext_stop**, an External Stop can be initiated, see (1) and (3) in Figure 70.



External Stop. See text for description.

This:

- immediately cancels the currently executed list
- switches off the Signals for "Laser Active" Operation (but does not deactivate them)

Like after calling **stop_execution** (internal stop), the following output ports are then set to the previously defined (by **set_port_default**) stop-case values given these have not been defined as ≠ "–1":

- 16-Bit Digital Output Port (EXTENSION 1 Socket Connector)
- 8-Bit Digital Output Port (EXTENSION 2 Socket Connector)
- 2-Bit Digital Output Port (LASER Connector)
- 12-Bit Analog Output Port 1 ANALOG OUT1 (LASER Connector)
- 12-Bit Analog Output Port 2 ANALOG OUT2 (LASER Connector)

The input ports for external stop signals (1) are always unlocked so that an external stop can occur at *any* time (emergency stop).

The /STOP input port is accessible by the LASER Connector, see Section "External Control Signals", page 76, the /STOP2 input port at the MARKING ON THE FLY Socket Connector, see Section "External Control Signals", page 85. Both signal input ports are internally connected to +3.3 V by pull-up resistors (4.7 kΩ). Both input ports are TTL active-LOW and level sensitive. A list abort is triggered as soon as at least one of the two input ports is set to LOW (that is, to 0 V or ground) for at least 10 µs.

If an RTC6 PCIe Board is connected to other boards by the Master or Slave connector, see Figure 7, then external *stops* from the board receiving the signal are passed on to *all* boards connected to it given the respective master/slave interface has been configured (by **master_slave_config**) accordingly. See also Chapter 6.6.3 "Master/Slave Operation", page 127.

Stops triggered by **stop_execution** are *not* passed on. In contrast, external stops triggered by **simulate_ext_stop** are passed through.

By **get_startstop_info** the current stop status (that is, whether one of the input ports is currently set to LOW) (2) can be queried and whether or not a new External Stop has occurred since the last query.

External Start

By a signal at the input ports /START, /START2 or /Slave-START, or by **simulate_ext_start** or **simulate_ext_start_ctrl**, an External Start can be initiated (see (1), (5) and (7) in Figure 71).

This starts execution at the beginning of "List 1". But the commands **set_extstartpos** or **set_extstartpos_list** also allow pre-selection of another absolute start address. A list is only started if neither the BUSY list execution status (as during list execution) nor the INTERNAL-BUSY list execution status (as for example, with **goto_xy**) nor the PAUSED list execution status (after **pause_list**, **stop_list** or **set_wait**) is set at the moment.

Before the /START, /START2 or /Slave-START input ports (1) can be used, they must be enabled by **set_control_mode** (3).

The control command **simulate_ext_start_ctrl** can be deactivated by **set_control_mode**(`Bit #4` = 1). The list command **simulate_ext_start** still remains active.

The /START input port is accessible by the LASER Connector, see Section "External Control Signals", page 76, the /START2 input port at the MARKING ON THE FLY Socket Connector, see Section "External Control Signals", page 85. Both signal input ports are internally connected to +3.3 V by pull-up resistors (4.7 kΩ). Both input ports are TTL active-LOW and edge sensitive (HIGH to LOW level transition). A start is triggered – after activation by **set_control_mode** – as soon as one of the three input signals changes from HIGH to LOW (that is, to 0 V or ground).

If an RTC6 PCIe Board is connected to other boards by the Master or Slave connector, see Figure 7, then external *starts* from the board receiving the signal are passed on to *all* boards connected to it given the respective master/slave interface has been configured (by **master_slave_config**) accordingly. See also Chapter 6.6.3 "Master/Slave Operation", page 127.

Internal starts triggered by **execute_list** or **execute_at_pointer** are *not* passed on.

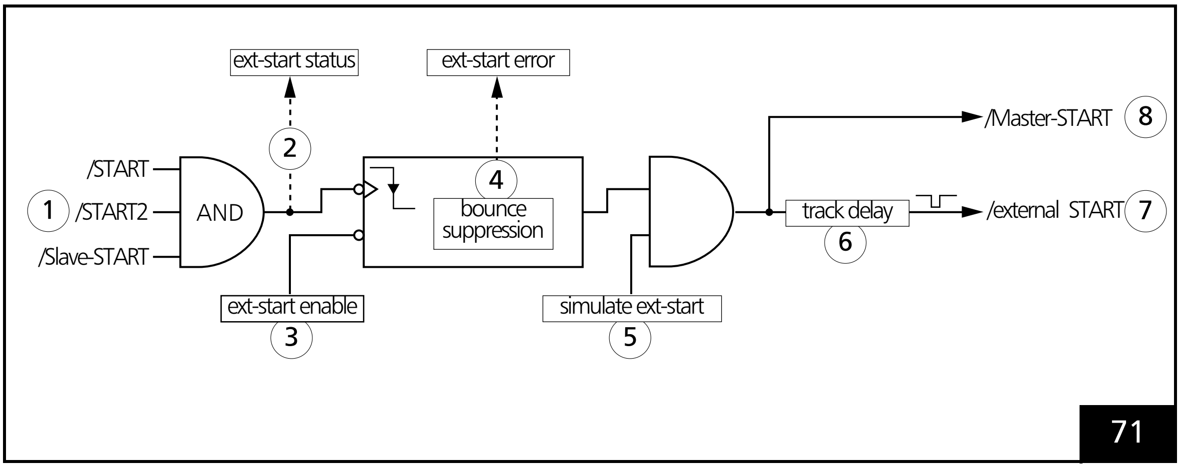
External Starts triggered by **simulate_ext_start** or **simulate_ext_start_ctrl** are passed on, see also Chapter 6.6.3 "Master/Slave Operation", page 127.

get_startstop_info queries the current start status (that is, whether one of the input ports is set to LOW) (2) and whether a list has successfully started since the last query.

bounce_supp enables debouncing of start signals received at the /START, /START2 or /Slave-START input ports (4). Start signals occurring within the defined debouncing time after a successful start signal are thereby suppressed. **get_marking_info** queries whether a start signal has been suppressed (4).

simulate_ext_start_ctrl can be used to start by control command a synchronous start of master/slave-synchronized boards.

The list command **simulate_ext_start** can be used to trigger further starts at defined intervals after the successful one-time External Start (see below).



External Start. See text for description.

External Start with Track Delay

For many applications (for example, if a workpiece must be initially transported from the light barrier to the scan system), the start must be delayed with reference to the triggering start signal.

For this purpose, **set_ext_start_delay**, **set_ext_start_delay_list** or **simulate_ext_start** allow configuring a track delay (see (6) in Figure 71) that postpones execution of a start relative to the triggering input signal or corresponding command. The track delay is specified in counting units of an internal encoder (encoder-counter) that itself can be triggered by an external or simulated encoder signal, see Chapter 9.3.3 "Synchronization by Encoder Signals", page 320.

External Starts triggered by an external start signal or by **simulate_ext_start** or **simulate_ext_start_ctrl** that do not execute immediately because of the track delay setting are held in a queue that can accommodate up to 8 starts (each start trigger is automatically generated when the delay has expired). This can be useful, for instance, when processing multiple workpieces transported to the scan system (even) at irregular intervals: here, up to 8 workpieces can simultaneously reside within the track delay (distance between the light barrier and scan system). If more External Starts are triggered than can be simultaneously held in the 8-start wait loop, then an error bit is set, which can be queried by **get_startstop_info** (`Bit #11`). If a track delay is set, then any previous queue is canceled (see **set_ext_start_delay** and **simulate_ext_start**).

By **set_control_mode** (`Bit #1` = 1) it can be set that the queue with the external start entries get explicitly canceled upon an External Stop. With **set_control_mode** (`Bit #1` = 0) the queue remains existing after an External Stop.

Notes

- /START, /START2 and /Slave-START input ports are *edge* sensitive (HIGH to LOW level *transition*).
- /STOP, /STOP2 and /Slave-STOP input ports are *level* sensitive.
- A **stop_execution** call disables the /START, /START2 and /Slave-START input ports. An external stop signal also (at least temporarily) disables these input ports, that is, as long as one of the input ports /STOP, /STOP2 or /Slave-STOP is LOW. **set_control_mode** can be used to define whether or not the /START, /START2 and /Slave-START input ports also stay disabled when the external stop signal is no longer active.
- **set_control_mode** additionally allows activation or deactivation of the input ports /START, /START2 or /Slave-START and deactivation of track delay.
- External Starts are also be suppressed after **pause_list**, **stop_list** or **set_wait** (PAUSED list execution status is set). **restart_list**, **stop_execution**, **release_wait** or an External Stop ends suppression of the start.
- If list input ports are not yet finished, a buffer flush should be initiated before an External Start, for example, by `set_input_pointer( get_input_pointer() )`, so that any still buffered list commands are fully transferred to RTC6 List Memory, see Chapter 6.4.1 "Loading Lists", page 108.
- If a master board is started internally (for example, by **execute_list_pos**) and subsequently a slave board by **simulate_ext_start**, then the master and slave boards do not run synchronously if a home jump has been previously activated by **home_position** or **home_position_xyz**: the home return executes on the master board *before* **simulate_ext_start** starts the slave board, but executes on the slave board *afterward*. While the home return executes on the slave board, the master board continues running. This asynchronicity does not occur if all boards are started by an external start signal (or by **simulate_ext_start** or **simulate_ext_start_ctrl**) or if no home jump is activated.

Regular (Periodic) External Starts

By **set_control_mode** and **set_control_mode_list** (Bit #10), equidistant External Starts can be created that are independent of the time point of the start trigger as long as they occur within the specified track delay.

This strongly periodic list processing is – independently of a list's actual duration of execution and the exact time point of the External Start – exactly synchronized to the 10 µs clock of the RTC6 PCIe Board.

If desired, set Bit #10 = 1 (`Mode|Bit #10`) to configure the internal encoder-counter's processing so that the track delay of an External Start is *not* counted only beginning with the time point of the triggering external start signal or **simulate_ext_start** (Bit #10 = 0) but already beginning with the most recently executed External Start (also executed by an external start signal or **simulate_ext_start**), see Figure 72. This makes the distance between consecutive External Starts (in encoder pulses) constant.

For activation of this mode, an External Start must have successfully occurred (only one-time) in mode Bit #10=0 (`Mode &~Bit #10`). Each subsequent External Start must be requested within the specified track delay.

Example in Pascal of a typical command sequence without use of an external start signal:

```
set_control_mode(Mode &~Bit #10);
// (one-time) reset (disable) Bit #10
// (initialization)
set_start_list_pos(ListNo, Pos);
// open some list
// afterward: some commands
simulate_ext_start(Delay,EncoderNo);
// first time start in mode Bit #10 = 0,
// otherwise in mode Bit #10 = 1
set_control_mode_list(Mode|Bit #10);
// set Bit #10 = 1
// afterward: further commands
set_end_of_list;
// close the list
execute_list_pos(ListNo,Pos);
// (one-time) start the list
```

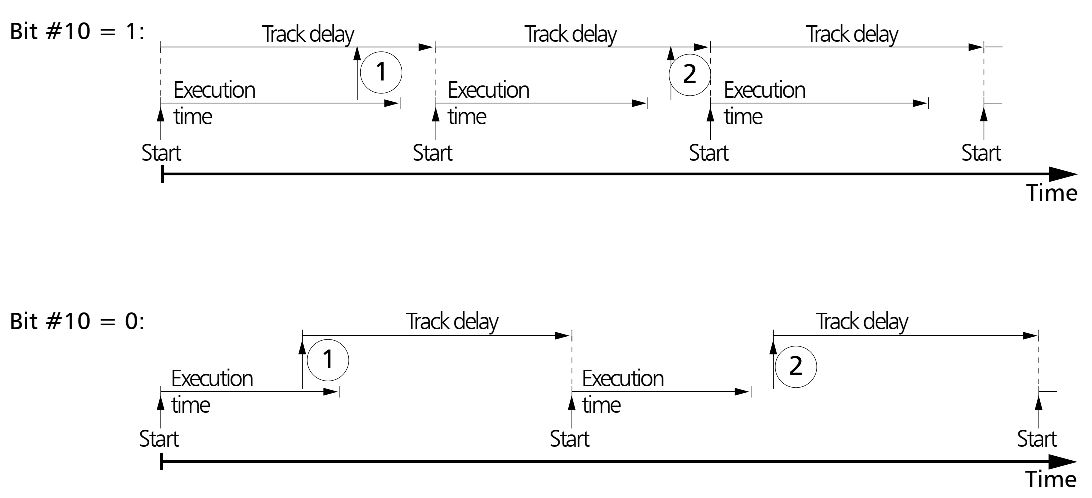
If the first start is to be triggered externally (for example, by /START or by **simulate_ext_start_ctrl**) rather than by an **execute_list_pos** command, but all subsequent starts triggered by **simulate_ext_start**, then **set_control_mode_list** in the above example must be called before **simulate_ext_start**.

After setting **set_control_mode**(`Bit #1` = 1), the external start queue entries get explicitly canceled upon an External Stop (thereby, External Starts can be permanently stopped by an External Stop).

For **set_control_mode**(`Bit #1` = 0) (default setting) – after an otherwise infinitely repetitive series has been stopped (for example, by **set_control_mode**(`Bit #0` = 0) – you should deactivate the track delay and cancel the queue of not-yet-executed External Starts by **set_ext_start_delay**(`Delay` = 0). Otherwise, the next “equidistant” External Start does not have the correct gap. **set_control_mode**(`Bit #2` = 1) alone is not sufficient for termination, because the track delay is reactivated by any not-yet-executed **simulate_ext_start** calls.

If a further External Start is missed within the track delay, then you should delete the wait loop (otherwise the encoder counter needs to run through a full 31-bit sequence before a start can again be successfully triggered). Deletion can be accomplished by resetting the track delay with **set_ext_start_delay**.

In any case, Bit #10 needs to first be reset (disabled) for initialization and then a (one-time) External Start must be triggered (for external start signals Bit #0 must be set) before Bit #10 can again be set (see example). Otherwise, the first track delay in the wait loop is undefined.



Legend

1 **simulate_ext_start**.

2 /START or /START2 or /Slave-START.

72

Regular and irregular External Starts (see text for description).

9.3.2 Execution of Conditional Commands

The so-called conditional commands allow the execution of individual list commands to be made dependent on external control signals.

The conditional commands read out the current value at the 16-Bit Digital Input Port at the EXTENSION 1 Socket Connector, see Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81, and their execution *depend on the read out value*:

- Conditional Jumps

  list_jump_pos_cond (synonymous with **list_jump_cond**) and **list_jump_rel_cond** result in either a jump within a RTC6 List Memory area or no jump. The thereby specified jump addresses must fulfill the same conditions as with **list_jump_pos** and **list_jump_rel**
- Variable-distance jump

  switch_ioport executes a relative list jump
- Conditional Calls of Non-Indexed Subroutines

  list_call_cond and **list_call_abs_cond** either call or do not call a non-indexed subroutine at a specified memory address
- Conditional Calls of Indexed Subroutines

  sub_call_cond and **sub_call_abs_cond** either call or do not call – depending on the queried value – an indexed subroutine with a specified index
- Conditional output of peripheral signals

  set_io_cond_list and **clear_io_cond_list** associate the output value of the 16-Bit Digital Output Port at the EXTENSION 1 Socket Connector, see Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81, directly with the signals at the digital input port: individual bits of the output port are set or cleared
- Conditional execution of any desired list commands:

  if_cond and **if_not_cond** have no effect if the condition for the queried value is fulfilled or not. Otherwise, they result in skipping the next list command. In this way, all list commands can be executed conditionally.

  Example: the command sequence

  `if_cond(…)`
  `list_call(…)`

  is functionally identical to

  `list_call_cond(…).`

The execution of any desired list command can also be made dependent on the current value at the 2-Bit Digital Input Port of the LASER Connector. For this, **if_pin_cond** and **if_not_pin_cond** are available. These are functionally similar to the commands **if_cond** and **if_not_cond**. Reliable functioning of these conditional commands requires that the signals at the 2-Bit Digital Input Port remain unchanged for at least 10 µs.

A condition for the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector can be defined by the control command **set_pause_list_cond**. If this condition is met, a currently executed list is paused by an automatic call of set **pause_list**. The list can only be resumed by **restart_list**. The condition is checked once per 10 µs clock cycle. A conditional **pause_list** takes precedence over a simultaneously present /STOP signal. **set_pause_list_not_cond** does the same as **set_pause_list_cond**, if the specified condition is not fulfilled.

Example Code (Pascal)

(1) Confirm a signal:

```
set_start_list(1);
   …
   // set Bit #0 of the 16-Bit Digital Output Port
   set_io_cond_list(0, 0, 1);
   // loop until the signal is confirmed (that is, Bit #0 of the digital input turns HIGH)
   list_jump_rel_cond(0, 1, 0);
   // clear Bit #0 of the 16-bit output
   clear_io_cond_list(0, 0, 1);
   // loop until the signal is confirmed
   list_jump_rel_cond(1, 0, 0);
   …
   set_end_of_list;
execute_list(1);
```

(2) If the lower 4 bits of the digital input have the value (`0110`), set Bit #1 of the 16-Bit Digital Output Port. Otherwise clear Bit #1:

```
set_start_list(2);
   …
   // RTC4 style: list_jump_cond($0006, $0009, get_input_pointer + 3);
   // this command uses absolute addresses and is not relocatable
   // the following RTC6 command uses relative addresses and is relocatable:
   // skip the next two commands, if the state
   // of the 16-bit input port is (xxxx xxxx xxxx 0110)
   // list_jump_rel_cond($0006, $0009, 3);

   // clear Bit #1 of the 16-bit output port and…
   clear_io_cond_list(0, 0, 2);
   //RTC4 style: set_list_jump(get_input_pointer + 2);
   //this command uses absolute addresses and is not relocatable
   //the following RTC6 command uses relative addresses and is relocatable
   //…skip the next command
   list_jump_rel(2);

   // set Bit #1 of the 16-bit output port
   set_io_cond_list(0, 0, 2);

// (continue)
   …
   set_end_of_list;
execute_list(2);
…
bit1 := (get_io_status AND $0002)        // returns the current state of Bit #1
```

(3) Choose between 15 small subroutines at defined memory addresses:

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

(4) Choose between 15 indexed subroutines:

```
…
for i := 1 to 15 do
     // call subroutine with index i, if [Bit #3…Bit #0] (binary) = i
     sub_call_cond(i, 15-i, i);
…
```

9.3.3 Synchronization by Encoder Signals

Intended Use

When processing moving workpieces, the laser scan processes need to be adapted to the current workpiece position.

To incorporate the current workpiece position, the RTC6 can evaluate signals of up to two user-supplied incremental encoders. Though incremental encoders do not register the current workpiece position, they register the motion of the transport system (conveyor belt, rotating plate, etc.)(1): For each transport motion, they provide signals (depending on the direction of motion) to the RTC6 which can result in incrementing or decrementing of its two internal encoder counters(2). The states of the RTC6's encoder counters thereby correspond directly to the position of the workpiece(3).

If workpieces are always processed at a constant speed and an encoder is therefore not mandatory, then the encoder signals can also be simulated by **simulate_encoder**, so that the encoder counters are incremented with a constant counting rate of 1 MHz.

The current counts of both encoder counters can be queried by the control command **get_encoder**. Alternatively, they can be stored in a buffer by the list command **store_encoder** and then retrieved from there by the control command **read_encoder**.

In addition, the RTC6 automatically evaluates the current counts if execution of the laser scan processes is controlled as follows:

- For Processing-on-the-fly-applications (see Chapter 8.6 "Processing-on-the-fly", page 251), the coordinate values of all Vector Commands and "Arc" Commands are transformed in accordance with the current encoder counts.
- By the list command **wait_for_encoder_mode**, further execution of a list can be postponed until the selected encoder counter has overstepped or understepped a predefined value.
- With 2D motions, **wait_for_encoder_in_range** waits until the encoder values are within a given rectangle.
- For External Starts, a track delay can be defined by **simulate_ext_start**, **set_ext_start_delay** or **set_ext_start_delay_list** for postponing execution of a start relative to the triggering input signal or corresponding command, see Section "External Start", page 312.
- With Encoder-Speed-Dependent Laser Control, a laser control signal parameter can be controlled according to the current encoder speed resent (counter pulses of the latest 10 μs interval) of an encoder counter.

(1) The actual workpiece position can also be forwarded by the McBSP interface to the RTC6 PCIe Board (see Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 322).

(2) See Notice!, page 59.

(3) The encoder counters are signed 32-bit counters. Upon reaching the maximum (minimum) counter value, counting continues with the minimum (maximum) value. A counter reset only occurs if triggered by Processing-on-the-fly-commands (see **set_fly_x**, **set_fly_y**, **set_fly_rot**).
set_control_mode(Bit #9 = 1) can be used to precisely synchronize the encoder reset with external start signals.

Input Ports for External Encoder Signals

For receiving encoder signals, the MARKING ON THE FLY Socket Connector provides 2 encoder input ports, see Section "Encoder Input Ports", page 85:

- ENCODER X
- ENCODER Y

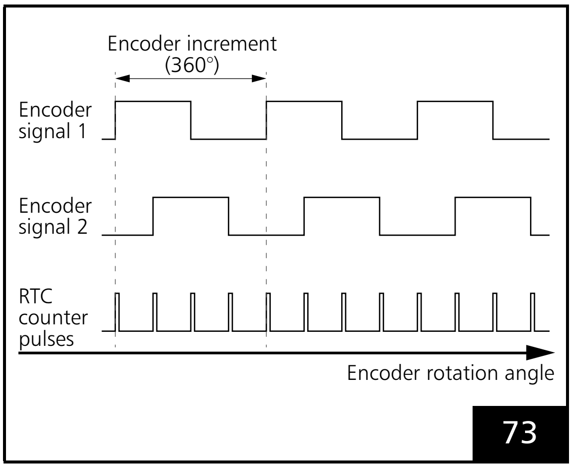
For linear workpiece motions, up to 2 user-supplied incremental encoders (that define, independently from each other, the workpiece motion in the x direction and y direction) can be connected to them.

For rotational motions only *one* incremental encoder is necessary. For Processing-on-the-fly applications it must be connected to ENCODER X.

ENCODER X and ENCODER Y are each designed for a pair (1, 2) of standardized differential input signals (RS-422).

Technical specifications see page 953 (RTC6 PCIe Board) and page 1042 (RTC6 Ethernet Board).

The timing diagram of a typical encoder signal pair shows Figure 73. The second encoder signal is usually phase-shifted by 90° relative to the first signal. The corresponding RTC6-internal encoder counter[1] is triggered at each edge of both signals, that is, one encoder increment results in 4 counter pulses (counts). The relative 90° phase-shift of the two signals allows the RTC6 PCIe Board to detect the motion direction of the workpiece as well. Depending on the direction of motion, the counter value is increased or decreased.



Timing diagram. Typical encoder signal pair and corresponding RTC6 counter pulses.

The signals at encoder input port:

- ENCODER X trigger encoder counter "Encoder0"
- ENCODER Y trigger encoder counter "Encoder1"

The encoder signals should not exceed the maximum allowed frequency of 4 MHz (16 encoder signal edges / µs). See `set_encoder_filter_ctrl( Mode = 1 )` (Encoder Input Filter).

Noise in encoder signals can lead to unwanted effects in marking results. By **set_encoder_filter_ctrl**, these can be reduced or even eliminated: then the RTC6 PCIe Board smooths encoder speeds with a moving average filter. Its length can be configured with the parameter `Length`. A higher `Length` value improves the filter effect at the expense of a certain delay in the encoder speeds used.

Encoder Simulation

The encoder simulation can be activated (deactivated) by **simulate_encoder**. The RTC6 PCIe Board provides a 1 MHz clock signal (counter pulses), which replaces the signals of an external incremental encoder.

(1) See Notice!, page 59.

9.3.4 Synchronization and Online Positioning by McBSP Signals

Instead of incremental encoders, the motion between the workpiece and the scan system can be synchronized using absolute position data via the McBSP interface.

Alternatively, this interface also allows the alignment of the workpiece relative to the (stationary) scan system ("Online Positioning").

The input value most recently fully transmitted at the McBSP interface can be queried by **read_mcbsp**. In addition, the RTC6 PCIe Board automatically evaluates the current input value if execution of the laser scan processes is controlled as follows:

- For Processing-on-the-fly-applications (see Chapter 8.6 "Processing-on-the-fly", page 251), the coordinate values of all Vector Command and "Arc" Commands are transformed in accordance with the current input value.
- By the list command **wait_for_mcbsp**, further execution of a list can be postponed until the input value has reached, overstepped or understepped a predefined value.
- By Online Positioning (see Chapter 8.3.1 ""Local Online Positioning"", page 237 and Chapter 8.3.2 ""Global Online Positioning"", page 240), the scan system is aligned in accordance with the current input values.

Notes

- To compensate linear motion, Cartesian coordinates can be forwarded by the McBSP interface. Compensation of rotational motion, on the other hand, requires transmission of angle positions. During evaluation, full revolutions are suppressed as long as the input values do not exceed the allowed value range (20 full circles, see **set_mcbsp_rot**).
- The signals at the McBSP interface have no impact on the track delay, which can be defined for External Starts (by **simulate_ext_start**, **set_ext_start_delay** or **set_ext_start_delay_list**).

9.4 Periodical I/O Signals

By **periodic_toggle** and **periodic_toggle_list**, periodically repeated signals can be outputted at a selectable IO port:

- ANALOG OUT1 output port, see Section "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 77
- ANALOG OUT2 output port, see Section "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 77
- 8-Bit Digital Output Port, see Section "8-Bit Digital Output Port", page 84
- 16-Bit Digital Output Port, see Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81
- 2-Bit Digital Output Port, see Section "2-Bit Digital Output Port", page 77

Thereby, for example, external peripheral equipment can be triggered synchronously to list execution.

Notes

- With **set_end_of_list**, **stop_execution** or /STOP the periodical signals continue, even if they have been activated by the list command **periodic_toggle_list**.
- **periodic_toggle** and **periodic_toggle_list** toggle endless with `Count` = 2^{32}–1.

10 RTC6 Commands

10.1 Overview

10.1.1 Control Commands

- Meaning(s):

(n_)	Corresponding multi-board command, see Multi-board Com'd Name, page 340.

Initialization of the RTC6 DLL

Using Several RTC6 PCIe Boards in One PC

List Memory Commands

Board Initialization and Image Field Correction

Laser Mode and Parameters

(1) Only with Option "3D".

Setting the Galvanometer Scanner Parameters

Coordinate Transformations

Online Positioning

Status Monitoring and Diagnostics

***iDRIVE* Commands**

Pixel Output Mode

I/O Commands

(1) Limited functionality if no Option "3D".

Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization

List Handling and List Status

Input Pointer Commands

Output Pointer Commands

Subroutine Commands

Direct Laser and Scan Head Control

Version Commands

Error Commands

Date, Time, Serial Numbers

(1) Limited functionality if no Option "3D".

“Classic” Processing-on-the-fly Control Commands

Controlling Stepper Motors

Jump Mode

Control Commands only for SCANahead Systems

Control Commands for RTC6 Ethernet Boards

(1) Limited functionality if no Option Processing-on-the-fly.

Standalone Functionality for RTC6 Ethernet Boards

Control Commands for Preprocessing Systems

Control Commands for a Restricted User Group only[1]

Other Control Commands

(1) These users have been informed separately in how to handle these RTC6 commands. Therefore, only the command table is available in this manual and further information is omitted.

(2) Limited functionality if no Option "3D".

10.1.2 List Commands

- Meaning(s):

(n_)	Corresponding multi-board command, see Multi-board Com'd Name, page 340.
nor	Normal List Command, page 335
us	Undelayed Short List Command, page 336
ds	Delayed Short List Command, page 336
var	Variable List Command, page 337
mul	Multiple List Command, page 337

Board Initialization and Image Field Correction

2D Jump Commands

3D Jump Commands(1)

micro_vector[*] Commands

2D Mark Commands

3D Mark Commands(1)

Text Commands

(1) Limited functionality if no Option "3D".

Date, Time, Serial Numbers

Status Monitoring and Diagnostics

Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization

List Handling and Structured Programming

Subroutine Commands

Setting the Laser Parameters

Setting the Galvanometer Scanner Parameters

(1) Only with Option "3D".

(2) Limited functionality if no Option "3D".

"Classic" Processing-on-the-fly List Commands

"Fly Extension" List Commands

Controlling Stepper Motors

Jump Mode

Camming

Wobbel Mode

(1) Only with Option Processing-on-the-fly.

(2) Limited functionality if no Option Processing-on-the-fly.

List Commands only for SCANahead Systems

List Commands for a Restricted User Group only(1)

Other List Commands

(1) These users have been informed separately in how to handle these RTC6 commands. Therefore, only the command table is available in this manual and further information is omitted.

Normal List Command, Short List Command, Variable List Command and Multiple List Command

The list commands of the RTC6 command set vary somewhat in their length of command execution. “normal”, “short”, “variable” and “multi“.

Accordingly, in the command description (in Chapter 10.2 ”RTC6 Command Set”, page 338), they are designated differently:

Normal List Command, page 335
Short List Command, page 335 • Undelayed Short List Command, page 336 • Delayed Short List Command, page 336
Variable List Command, page 337
Multiple List Command, page 337

Normal List Command

A Normal List Command requires a full 10 µs clock cycle for command execution.

Short List Command

A Short List Command requires less than a full 10 µs clock cycle for command execution.

Therefore, a Short List Command can be executed along with the next list command, one directly after the other within a single 10 µs clock cycle.

In contrast, a Short List Command that directly follows a Normal List Command executes in the subsequent 10 µs clock cycle.

The quicker execution of Short List Commands reduces total list processing time.

In addition, during a Polyline, the laser power can, for instance, be varied or the IO ports can be addressed (see **write_da_x_list**) between the Polyline’s individual Mark Commands (see **set_laser_pulses**), all without interrupting the Polyline (the laser remains on).
In contrast, if a specific time behavior is desired (10 µs clock cycle), you can insert an additional **list_nop** call or **list_continue** call after any short list command to ensure that the next command only executes in the following 10 µs clock cycle. Insertion of **list_nop** (but not insertion of **list_continue**) results in the interruption of the Polyline (the “laser active” Laser Control Signals are switched off).
Up to 8 short list commands per 10 µs clock cycle are possible. However, the maximum number can be lower, depending on the workload of the RTC6 board and the DSP mode. Short list commands that alter the output pointer (for example, **sub_call**, **list_return** or **list_jump_pos**) count as 2 commands. If the maximum number is exceeded, a 10 µs clock cycle is inserted (equivalent to an additionally inserted **list_continue**, during the Polyline the laser remains on).

A maximum of 2 Short List Commands per 10 μs clock cycle are allowed before a Normal List Command. If a Normal List Command succeeds more than 2 Short List Commands, then the Short List Commands execute immediately and the Normal List Command execute delayed by a 10 μs clock cycle.

The maximum number of up to 8 Short List Commands may change in the future. For fully future-safe applications, only one Short List Command should precede a Normal List Command. If necessary, you should explicitly insert a **list_continue** call or **list_nop** call (**list_nop** interrupts the Polyline).

Notes

- The total execution time of Normal List Commands and Variable List Commands equals the sum of the command execution time and the execution time of the process initiated by the command. A subsequent list command only runs after this total execution time has completed.

  Example: the total execution time of the Normal List Command **mark_abs** is 10 μs (command execution time) + the execution time of the marking process. The latter is dependent on the settings of such parameters as marking length, mark speed, and delays etc.

Undelayed Short List Command, Delayed Short List Command

Most short list commands (for example, **list_jump_pos**, **list_call**, **sub_call** or **list_return**) execute before the next list command and prior to a possible Scanner Delay (Jump Delay, Mark Delay, Polygon Delay). In the corresponding command descriptions (in Chapter 10.2 "RTC6 Command Set", page 338), these are designated as:

- Undelayed Short List Command

However, some short list commands only execute after the respective Scanner Delay (that is, directly before the next command). In the corresponding command descriptions (in Chapter 10.2 "RTC6 Command Set", page 338), these are designated as:

- Delayed Short List Command

These include commands that:

- Immediately affect output or laser power (for example, **set_rot_center_list**, **set_wobbel_mode**, **write_da_x_list**, **set_laser_pulses**, **set_standby_list**, **set_mark_speed**, **set_encoder_speed**)
- Affect data acquisition or time measurement (for example, **set_trigger** or **save_and_restart_timer**)

Without this delayed execution, such commands would, for example, result in a laser power change already being effective at the end of a Mark Command (that is, during a still-active Mark Delay or Polygon Delay) and not at the beginning of the following Mark Command.

If such a power changing command, such as a peripheral output or **set_laser_power**, is called immediately before a **set_end_of_list**, it is not executed because **set_end_of_list** "clears up" the laser control. Add a **list_nop** to be sure that the command is actually executed.

set_trigger and **save_and_restart_timer** would erroneously take into account the delay of a preceding command instead of the delay of the subsequent command.

If these commands are directly before a **set_end_of_list**, add a **list_nop** call so that they are still executed within the list.

With several consecutive short list commands, even a Delayed Short List Command is only executed with a delay, if no further Delayed Short List Command follows immediately.

With several Delayed Short List Commands in a sequence of Short List Commands, only the last “delayed” command can actually be executed delayed. All others before that are executed immediately (yet before a Scanner Delay).

When you sequence the commands, make sure to place the most important “delayed” command at the end, especially within a Polyline.

Alternatively, you can also explicitly initiate processing of the Scanner Delay (for example, by **list_nop**), so that all subsequent short list commands in fact always execute “delayed”.

If a Normal List Command succeeds more than 2 short list commands, then it is executed delayed by a 10 µs clock cycle.

Variable List Command

Command execution length of a Variable List Command is dependent on additional parameters or user program, see corresponding command descriptions.

Notes

- The total execution time of Normal List Commands and Variable List Commands equals the sum of the command execution time and the execution time of the process initiated by the command. A subsequent list command only runs after this total execution time has completed.

Multiple List Command

A Multiple List Command has multiple components that accordingly occupy multiple list storage positions.

The initial components are always Undelayed Short List Commands. The final component is always a Short List Command, Normal List Command or Variable List Command. All components are immediately executed successively. Any still-pending Delayed Short List Commands are executed beforehand.

Currently, the RTC6 command set only contains two-component multiple list commands, for example, **wait_for_encoder_in_range** or **set_pixel_line_3d**.

10.2 RTC6 Command Set

In this Chapter:

10.2.1 General Structure of the Command Descriptions

The general structure of the command tables is as follows[1]:

(1) A program language-neutral form is used. Each real programming language has its own individual naming.

Category of the command	example_command_name_one	
Function	Short description describing the purpose of the RTC6 command.	
Restriction	For example, states the behavior of the RTC6 command, if a certain option is not enabled, see Chapter 2.6 "Options", page 39. • Without Option Processing-on-the-fly, commands to activate a Processing-on-the-fly correction have no effect • Without Option "3D", 3D vector commands are executed, however, no z axis signals are outputted • Without Option "Second Scan Head Control", commands for which the scan head connector can be explicitly specified have not effect on the Connector for Second Scan Head (for example, **set_matrix**)	

Call	Shows the correct spellings and the sequence of the parameters. Note, there is no semicolon at the end of the line. A '&' (address operator) is only used in this table row and indicates a pointer. Examples:
	`example_command_name_one( parameter_A, &parameter_B, parameter_C )`
	`example_variable = example_command_name_one( parameter_A )`	

Parameters	`A`	Short text. Data type. In some C/C++ code descriptions, Parameters are labeled with `__out`.
	`C`	Short text. Data type.
Returned Parameter Values	`B`	Short text. Data type. In some C/C++ code descriptions, Returned Parameter Values are labeled with `__out`.
Result	If implemented: mentions the returned result value and data type in generic form (Example: error code #. As an unsigned 32-bit value.). A value range is here only given, if the actual usable one is smaller than the value range of the data type. If not implemented: "None." List Commands generally have no return values.	

Category of the command	example_command_name_one
Multi-board Com'd Name	• If available: "**n_** + single-board command name", here, for example, **n_example_command_name_one** • If not available: "Not available as a multi-board command." See also Chapter 6.6.1 "Multi-Board Programming", page 126 and Chapter 6.6 "Using Several RTC6 PCIe Boards in One PC", page 126.
Comments	• Additional information on this RTC6 command. • References to other chapters and publications.
RTC4→RTC6	States the differences to the RTC4 command (of the same name). See also RTC4 Compatibility Mode and Chapter 2.10 "Notes for RTC4 Users", page 49 as well as Chapter 10.4 "Unsupported RTC4 Commands", page 941.
RTC5→RTC6	States the differences to the RTC5 command (of the same name). See also RTC5 Compatibility Mode and Chapter 2.11 "Notes for RTC5 Users", page 60 as well as Chapter 10.5 "Unsupported RTC5 Commands", page 943.
Version Info	For example, states the minimum versions of DLL, RBF, OUT which are required to use the RTC6 command, latest change in version. Refer also to `RTC6_RevisionHistory_<SW-V>.pdf`.
References	Links to related RTC6 commands: **command_name_two**, **command_name_three**

10.2.2 Data Types

The following table defines the formats and ranges of the different data types used by the RTC6 commands:

Data Format	Range	Pascal	C, C++	C#
unsigned 32-bit value	$[0; (2^{32}–1)]$	longword	unsigned long	uint
signed 32-bit value	$[–2^{31}; +(2^{31}–1)]$	longint	long	int
64-bit IEEE floating point value		double	double	double
pointer to a \0-terminated ANSI string (1 byte per char)	4 Byte for Win32-user programs 8 Byte for Win64-user programs	pchar	char*	string

10.2.3 Pointers to Computer Memory Areas

Some RTC6 commands (for example, **get_transform**/**get_transform_offset**, **get_values**, **get_waveform**, **transform** or **upload_transform**) have pointers to locations in the PC memory as parameters. In C# and Pascal, appropriate pointer data types are herefore used (see import declarations). In C and C++, the data type ULONG_PTR is used for this pointer parameters. The ULONG_PTR data type is defined in the C and C++ import declarations as follows (ULONG_PTR = unsigned 32-bit value for Win32-user programs, ULONG_PTR = unsigned 64-bit value for Win64-user programs):

```
#if !defined(ULONG_PTR)
   #if !defined(_WIN64)
      #define ULONG_PTR UINT
   #else
      #define ULONG_PTR UINT64
   #endif // !defined(_WIN64)
#endif // !defined(ULONG_PTR)
```

Usually, the data type ULONG_PTR is also appropriately defined in the Windows header file `BaseTsd.h`.

10.2.4 Command Descriptions

The RTC6 commands are in alphabetical order.

Ctrl Command	acquire_rtc
Function	Acquires the specified RTC6 board for a user program.
Call	`NoOfAcquiredCard = acquire_rtc( CardNo )`
Parameters	`CardNo` RTC6 DLL-internal number (RTC6 board management index) of the desired board. As an unsigned 32-bit value.
Result	The return value is: • `CardNo`, if the acquisition has been successful • 0, if the board is currently acquired by another user program or the version check detects an error As an unsigned 32-bit value.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• **acquire_rtc** is also useful for single-board systems which need to coordinate the use of a RTC6 board by different user programs. • **acquire_rtc** has no effect (return value 0, **get_last_error** return code `RTC6_PARAM_ERROR`), if: – `CardNo` > number of RTC6 PCIe Boardns found during initialization (see **rtc6_count_cards**) and no RTC6 Ethernet Board is entered there – `CardNo` = 0 (real boards begin with 1) • Access rights to existing boards are granted exclusively (always to only *one* user program at a time). Therefore, **acquire_rtc** has no effect if the specified board is already acquired by another user program (return value 0, **get_last_error** return code `RTC6_ACCESS_DENIED`). This must explicitly release the RTC6 board (by **release_rtc** or **free_rtc6_dll**) before the RTC6 board can be acquired by another user program (with **acquire_rtc**). On the other hand, if the board has been already freed prior to initialization of a user program, then initialization by **init_rtc6_dll** result in RTC6 board management assigning board access rights for the user program. In this case, an explicit **acquire_rtc** call is not needed and has no effect. Nevertheless, the return value is `CardNo`. • Assorted versions of the RTC6 DLL and the files `RTC6OUT.out`, `RTC6RBF.rbf` and `RTC6DAT.dat` cannot be arbitrarily combined with another. **acquire_rtc** performs a version compatibility check. If `RTC6OUT.out`, `RTC6RBF.rbf` and `RTC6DAT.dat` are not yet loaded, then this check cannot be explicitly executed (**get_last_error** return code `RTC6_TIMEOUT`), but the check still regarded as successful and acquisition is not hindered. If `RTC6OUT.out`, `RTC6RBF.rbf` and `RTC6DAT.dat` are loaded and the version check determines an error, then access is denied (return value 0, **get_last_error** return code `RTC6_ACCESS_DENIED` | `RTC6_VERSION_MISMATCH`). With RTC6 Ethernet Boards, `RTC6OUT.out` must be replaced by `RTC6ETH.out`. **init_rtc6_dll** does not automatically acquire RTC6 Ethernet Boards.

Ctrl Command	acquire_rtc
Comments (cont'd)	• With RTC6 Ethernet Boards, **acquire_rtc** can take up to 1 s. • A board successfully acquired by **acquire_rtc** does not automatically become the "active" board. Activation of a board is only achieved by **select_rtc** or **init_rtc6_dll**. • Running boards are neither halted nor initialized by **acquire_rtc**. • **acquire_rtc** is available even without explicit access rights to a particular RTC6 board. • See also Chapter 6.7.1 "Notes on Board Acquisition by a User Program", page 131.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**init_rtc6_dll**, **select_rtc**, **free_rtc6_dll**, **release_rtc**, **rtc6_count_cards**

Variable List Command	activate_fly_1_axis
Function	"Fly Extension" Command: Activates a 1 Axisn-Processing-on-the-fly application.
Restriction	If the Option Processing-on-the-fly is not enabled, then **activate_fly_1_axis** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`activate_fly_1_axis( Axis, Mode, Scale, Offset )`
Parameters	`Axis` Axis from Table 3, page 268. As an unsigned 32-bit value. Allowed values: `1`…`2`. `Mode` Mode from Table 4, page 270. As an unsigned 32-bit value. Allowed values: 1…4. `Scale` Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: • 1/256 ≤ |`Scale`| ≤ 16.000,0 with linear axis (`1` or `2`) `Scale` can be + or –. Only the absolute value is restricted. `Offset` Offset to the encoder value. As a signed 32-bit value.
Multi-board Com'd Name	**n_activate_fly_1_axis**
Comments	• Being an "Fly Extension" Command, **activate_fly_1_axis** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • **activate_fly_1_axis** occupies two RTC6 List Memory positions. • **activate_fly_1_axis** requires two 10 µs clock cycles for execution. • At the specified Axis`1`, neither a Processing-on-the-fly correction nor a rotation correction must be active. However, the Axis can be combined as a linear axis with the third linear axis. • An xy positioning stage compensation is automatically executed, if after the successful **activate_fly_1_axis** execution Axis 1 and Axis 2 are activated with two physically different encoder modes (for example, Mode 1 and 3 mean the same physical encoder) and xy positioning stage compensation is defined and activated (see **load_fly_2d_table**), then xy travel table compensation is automatically executed. • With an unallowed parameter value, **activate_fly_1_axis** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • See also comments on **activate_fly_2_axes**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**activate_fly_2_axes**

Variable List Command	activate_fly_2_axes
Function	"Fly Extension" Command: Activates a 2-Axes-Processing-on-the-fly application.
Restriction	If the Option Processing-on-the-fly is not enabled, then **activate_fly_2_axes** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`activate_fly_2_axes( ModeX, ScaleX, OffsetX, ModeY, ScaleY, OffsetY)`
Parameters	`ModeX` Mode from Table 4, page 270. As an unsigned 32-bit value. `ScaleX` Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: • 1/256 ≤ |`Scale`| ≤ 16.000,0 with linear axis (`1` or `2`) `Scale` can be + or –. Only the absolute value is restricted. `OffsetX` Offset to the encoder value. As a signed 32-bit value. `ModeY` Like `ModeX`. `ScaleY` Like `ScaleX`. `OffsetY` Like `OffsetX`.
Multi-board Com'd Name	**n_activate_fly_2_axes**
Comments	• Being an "Fly Extension" Command, **activate_fly_2_axes** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • The Axes are automatically `1` and `2`. The modes need to be two physically different encoder modes (Mode 1 and 3 or 2 and 4 are not allowed). • **activate_fly_2_axes** occupies two RTC6 List Memory positions. • **activate_fly_2_axes** requires two 10 µs clock cycles for execution. • At Axis `1` and `2`, neither a Processing-on-the-fly correction nor a rotation correction must be active. However, the Axes can be combined as linear axes with the third linear axis. • With an unallowed parameter value, **activate_fly_2_axes** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • If an xy positioning stage compensation is defined and active (see **load_fly_2d_table**), then it is carried out automatically. See also comments on **activate_fly_2d** (encoder reset, error bit, **get_marking_info**).

Variable List Command	activate_fly_2_axes

Comments (cont'd)	• The following command calls are executed in the same way:
	– `activate_fly_2_axes( 1, ScaleX, 0, 2, ScaleY, 0 ) =`
	`activate_fly_2d( ScaleX, ScaleY )`
	or
	`activate_fly_xy( ScaleX, ScaleY )`
	– `activate_fly_2_axes( 1, ScaleX, EncX, 2, ScaleY, EncY ) =`
	or
	`activate_fly_xy_encoder( ScaleX, ScaleY, EncX, EncY )`
	"xy" and "2d" are distinguished with `wait_for_1_axis` and `wait_for_2_axes`!
	– `activate_fly_2_axes( 1, ScaleX, OffsetX, 2, ScaleY, OffsetY ) =`
	`{`
	`activate_fly_1_axis( 1, 1, ScaleX, OffsetX );`
	`activate_fly_1_axis( 2, 2, ScaleY, OffsetY );`
	`}`

RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**activate_fly_1_axis**

Variable List Command	**activate_fly_2d**
Function	Activates a **set_fly_2d** Processing-on-the-fly application without encoder resets.
Restriction	If the Option Processing-on-the-fly is not enabled, then **activate_fly_2d** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`activate_fly_2d( ScaleX, ScaleY )`
Parameters	`ScaleX` Scaling factor as for **activate_fly_2d_encoder**. `ScaleY` Like `ScaleX`.
Multi-board Com'd Name	**n_activate_fly_2d**
Comments	• If no Processing-on-the-fly correction is active, then **activate_fly_2d** activates **set_fly_2d** Processing-on-the-fly correction, see Chapter 8.6.4 "Compensating 2D Motions", page 258. Unlike **set_fly_2d**, **activate_fly_2d** does not thereby reset the encoders, but instead calculates coordinate values such that the Processing-on-the-fly-corrected output matches the current output. If the then-current recalculated coordinate values would have fallen outside the 29-bit virtual Image Field, then Processing-on-the-fly correction is not activated. Then an error bit is set that can be queried by **get_marking_info** (`Bit #9`). • If Processing-on-the-fly correction is *active*, then **activate_fly_2d** is a Short List Command without further effect and merely sets an error bit queryable by **get_marking_info** (`Bit #9`). Therefore, **activate_fly_2d** cannot be used to modify the Processing-on-the-fly mode itself or to modify the scaling factors of the same mode. • You can also query the error bit by the short list command **if_not_activated** in order to jump to an appropriate error handling routine. • Successful activation by **activate_fly_2d** *does not* reset an error bit. It remains set for **get_marking_info** until **get_marking_info** is called. • If unallowed parameter values are supplied (for example, for `ScaleX` = 0), then **activate_fly_2d** is (already during loading) replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • **activate_fly_2d** does not affect the Laser Control Signals (Signals for "Laser Active" Operation remain on/off if they are on/off).
RTC4→RTC6	New command. RTC4 Compatibility Mode: see **set_fly_2d**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_fly_2d**, **activate_fly_2d_encoder**, **activate_fly_xy**

Multiple List Command	**activate_fly_2d_encoder**
Function	Activates a **set_fly_2d** Processing-on-the-fly application with encoder reset and encoder offsets.
Restriction	If the Option Processing-on-the-fly is not enabled, then **activate_fly_2d_encoder** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`activate_fly_2d_encoder( ScaleX, ScaleY, EncX, EncY )`
Parameters	`ScaleX` Scaling factor as for **set_fly_2d**. `ScaleY` Like `ScaleX`. `EncX` Encoder offset. As a signed 32-bit value. `EncY` Like `EncX`.
Multi-board Com'd Name	**n_activate_fly_2d_encoder**
Comments	• **activate_fly_2d_encoder** occupies two RTC6 List Memory positions and also needs two 10 μs clock cycles to execute (the first part of **activate_fly_2d_encoder** is *not* a Short List Command). • **activate_fly_2d_encoder** is a combination of **set_fly_2d** (encoder reset) and **activate_fly_2d** (see also comments there). However, in **activate_fly_2d_encoder** the current (reset) encoder values are not used to calculate the Processing-on-the-fly-uncorrected virtual Image Field coordinates, but the parameter values `EncX` and `EncY`. For the error handling, see comments of **activate_fly_2d**. • Because of this combination, **activate_fly_2d_encoder** saves the positioning stage motion (which is often long but may be necessary for the Processing-on-the-fly activation without this command) from the initialization position (for example, at the lower left corner) to the center and back again. • Subsequently all encoder values are offset with `EncX` and `EncY`, before the Processing-on-the-fly correction is applied. All other encoder related commands refer to the actual encoder values and behave as before. • If the value of `EncX` or `EncY` is not allowed (that is, the Processing-on-the-fly-corrected virtual Image Field coordinates are outside the virtual Image Field limits), then **activate_fly_2d_encoder** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • If the value of `ScaleX` or `ScaleY` is not allowed (see **set_fly_2d**), the first part is transferred to the board (and thus executes the encoder reset). However, the second part is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • With *active* Processing-on-the-fly correction, **activate_fly_2d_encoder** is a Short List Command without further effect and merely sets an error bit queryable by **get_marking_info** (`Bit #9`). Therefore, **activate_fly_2d_encoder** cannot be used to modify the Processing-on-the-fly mode itself nor the scaling factors or encoder offsets of the same mode.

Multiple List Command	**activate_fly_2d_encoder**
RTC4→RTC6	New command. RTC4 Compatibility Mode: see **set_fly_2d**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 610, OUT 610, RBF 615.
References	**set_fly_2d**, **activate_fly_2d**

Variable List Command	activate_fly_xy
Function	Activates a set_fly_x/set_fly_y Processing-on-the-fly application without encoder resets.
Restriction	If the Option Processing-on-the-fly is not enabled, then **activate_fly_xy** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`activate_fly_xy( ScaleX, ScaleY )`
Parameters	`ScaleX` Scaling factor as for **set_fly_x**. `ScaleY` Scaling factor as for **set_fly_y**.
Multi-board Com'd Name	**n_activate_fly_xy**
Comments	• Like **activate_fly_2d** with the difference that a **set_fly_x**/**set_fly_y** Processing-on-the-fly Session is activated.
RTC4→RTC6	New command. RTC4 Compatibility Mode: see **set_fly_x**/**set_fly_y**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_fly_x**, **set_fly_y**, **activate_fly_xy_encoder**

Multiple List Command	activate_fly_xy_encoder
Function	Activates a set_fly_x/set_fly_y Processing-on-the-fly application with encoder reset and encoder offsets.
Restriction	If the Option Processing-on-the-fly is not enabled, then **activate_fly_xy_encoder** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`activate_fly_xy_encoder( ScaleX, ScaleY, EncX, EncY )`
Parameters	`ScaleX` Scaling factor as for **set_fly_x**. `ScaleY` Scaling factor as for **set_fly_y**. `EncX` Encoder offset. As a signed 32-bit value. `EncY` Encoder offset. As a signed 32-bit value.
Multi-board Com'd Name	**n_activate_fly_xy_encoder**
Comments	• Like **activate_fly_2d_encoder** with the difference that a **set_fly_x**/**set_fly_y** Processing-on-the-fly Session is activated.
RTC4→RTC6	New command. RTC4 Compatibility Mode: see **set_fly_2d**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 610, OUT 610, RBF 615.
References	**set_fly_x**, **set_fly_y**, **activate_fly_xy**, **activate_fly_2d_encoder**

Ctrl Command	activate_scanahead_autodelays
Function	Only for SCANahead Systems. Switches on or off the automatic (dynamic) calculation of Scanner Delays and Laser Delays. Returns the current mode after **activate_scanahead_autodelays** has been executed.
Call	`CurrentMode = activate_scanahead_autodelays( Mode )`
Parameters	`Mode` As a signed 32-bit value. `-1`: Only returns the current mode. The current mode is not changed. `0`: Switches off automatic calculation. `1`: Switches on automatic calculation (`Mode = 1`) if **set_scanahead_params** has been called previously. Otherwise, `Mode` remains 0 after the command has been executed.
Result	Current mode. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_activate_scanahead_autodelays**
Comments	• If `Mode = 0`, then normal Scanner Delays and Laser Delays are used (see **set_scanner_delays** and **set_laser_delays**). If `Mode = 1`, then Scanner Delays and Laser Delays are calculated and set automatically. Values for `Mode = 0` do not get overwritten. After switch-off, they remain available in `Mode = 0` for usage (as with non-SCANahead Systems). • **activate_scanahead_autodelays** also affects Sky Writing: the Sky Writing command parameters `Timelag` (except for the actual activation of Sky Writing), `Nprev` and `Npost` have no effect. However, they are going to have immediate effect once the automatic delay calculation is switched off. In contrast, the parameter `LaserOnShift` is always effective.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of version DLL 600, OUT 600.
References	**activate_scanahead_autodelays_list**, **set_scanner_delays**, **set_laser_delays**

Undelayed Short List Command	**activate_scanahead_autodelays_list**
Function	Only for SCANahead Systems. As **activate_scanahead_autodelays**, but a list command.
Call	`CurrentMode = ` **`activate_scanahead_autodelays_list`**`( Mode )`
Parameters	`Mode` As an unsigned 32-bit value. For values, see **activate_scanahead_autodelays**.
Multi-board Com'd Name	**n_activate_scanahead_autodelays_list**
Comments	• **activate_scanahead_autodelays_list** takes effect upon the next to-be-calculated delay. • `Mode = -1` is not available with **activate_scanahead_autodelays_list**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of version DLL 600, OUT 600.
References	**activate_scanahead_autodelays**

Ctrl Command	apply_mcbsp		
Function	Queries the most recent values fully transmitted over the McBSP interface for “Local Online Positioning” and defines offset and/or rotation matrix M_R or general transformation matrix M_T for subsequent coordinate transformations.		
Call	`apply_mcbsp( HeadNo, at_once )`		
Parameters	`HeadNo`	Number of the scan head connector. As an unsigned 32-bit value. = 1: The definition only affects the Connector for First Scan Head. = 2: The definition only affects the Connector for Second Scan Head. = 0, 3: The definition affects *both* scan head connectors. Only the two least significant bits are evaluated.	
	`at_once`	Determines when the defined transformation becomes effective. As an unsigned 32-bit value. = 0: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next list command is executed. = 1: The new total transformation is calculated immediately (or before the next list command if currently BUSY list execution status or INTERNAL-BUSY list execution status is set) and applied to the current position. = 2: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next **jump_abs**, **jump_rel**, **goto_xy** or **goto_xyz** is executed. > 2: Like `at_once` = 2.	
Multi-board Com’d Name	**n_apply_mcbsp**		
Comments	• Data acquisition by the McBSP interface for “Local Online Positioning” must be activated in advance by **set_mcbsp_x**, **set_mcbsp_y** and/or **set_mcbsp_rot** or **set_mcbsp_matrix** (or with the corresponding list commands). Depending on the configuration, **apply_mcbsp** only defines (as with **set_offset**) an x offset and/or y offset or also (as with **set_angle**) a rotation matrix or (as with **set_matrix**) a general matrix operation (see Chapter 8.3.1 ““Local Online Positioning“”, page 237). As with the commands described in Chapter 8.2 “Coordinate Transformations”, page 233, the parameter `at_once` determines when the newly defined total transformation becomes effective. • Transformations previously defined by **set_angle**, **set_offset** or **set_matrix** get overwritten by **apply_mcbsp**. In contrast, transformations and focus shifts previously defined by **set_scale** or **set_defocus** and z offsets defined by **set_offset_xyz** is continued to be taken into account, when the total transformation gets recalculated. • Any new definitions made with **set_angle**, **set_offset** or **set_matrix** overwrite coordinate transformations defined by the McBSP interface. • The McBSP interface ignores the first FrameSync signal after a **load_program_file** or **mcbsp_init**. That is, data provided is not transmitted, see Section “RTC6 PCIe Board as Receiver”, page 88.		

Ctrl Command	apply_mcbsp
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**apply_mcbsp_list**, **set_mcbsp_x**, **set_mcbsp_y**, **set_mcbsp_rot**, **set_mcbsp_matrix**

Normal List Command	apply_mcbsp_list
Function	Like **apply_mcbsp**, but a list command.
Call	`apply_mcbsp_list( HeadNo, at_once )`
Parameters	`HeadNo` Like **apply_mcbsp**.
	`at_once` Determines when the defined transformation becomes effective. As an unsigned 32-bit value. = 0: The transformation settings are only accumulated and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (for example, by a list command with `at_once` = 1 or a corresponding control command). = 1: The transformation is immediately calculated (including all transformation settings that were accumulated until then) and processed prior to the next list command. = 2: The transformation settings are only accumulated and intermediately stored (as with `at_once` = 0). However, The transformation is immediately calculated (including all transformation settings that were accumulated and intermediately stored until then) and applied to the current position when the next **jump_abs** or **jump_rel** (only if no list is currently being executed: also **goto_xy** or **goto_xyz**) is executed. > 2: Like `at_once` = 2.
Multi-board Com'd Name	**n_apply_mcbsp_list**
Comments	• See **apply_mcbsp**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**apply_mcbsp**

Normal List Command	arc_abs	
Function	Moves the laser focus from the current position at mark speed along an arc with the specified angle and center point (absolute coordinate values) within a 2D Image Field.	
Call	`arc_abs( X, Y, Angle )`	
Parameters	`X`	Absolute x coordinate of the arc center. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values.
	`Y`	Like `X` (analogously).
	`Angle`	Arc angle. In degrees. As a 64-bit IEEE floating point value. A positive sign means “clockwise”. Allowed value range: [–3,600.0°…+3,600.0°] (±10 full circles). Out-of-range values are clipped to the boundary values.
Multi-board Com’d Name	**n_arc_abs**	
Comments	• If the mark speed has not been previously explicitly set by **set_mark_speed** or **set_mark_speed_ctrl**, then the marking is executed at a predefined mark speed of 1,000 *bits/ms*. • The Signals for “Laser Active” Operation are automatically turned on at the beginning of the marking (or remain on after a directly preceding [*]mark[*] Command or “Arc” Command). The defined Scanner Delays and Laser Delays are thereby taken into account, see Chapter 7.2 “Delay Settings – Coordinating Scan Head Control and Laser Control”, page 148. Note that other delays are executed in Sky Writing mode. Exception: zero-length “Arc” Commands, see Section “Notes”, page 152.	
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `X` and `Y` by 16. The allowed value ranges decrease accordingly.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_mark_speed**, **set_scanner_delays**, **arc_rel**, **timed_arc_abs**, **mark_abs**, **arc_abs_3d**, **mark_ellipse_abs**	

Normal List Command	**arc_abs_3d**	
Function	Moves the laser focus at mark speed from the current position helical around an axis parallel to the z axis. The x and y components thereby characterize an arc with the specified angle around the specified axis, while the z component characterizes a linear motion from the current position to the specified end point. The position of the helical axis and the z end coordinate are specifiable as absolute coordinate values.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **arc_abs_3d** has the same effect as **arc_abs**.	
Call	`arc_abs_3d( X, Y, Z, Angle )`	
Parameters	`X`	Position of the helical axis (parallel to the z axis) as absolute x coordinate. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values.
	`Y`	Like `X` (analogously).
	`Z`	Absolute z end coordinate. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
	`Angle`	Arc angle. In degrees. As a 64-bit IEEE floating point value. A positive sign means "clockwise". Allowed value range: [–3,600.0°…+3,600.0°] (±10 full circles). Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_arc_abs_3d**	
Comments	• Except for the additional motion in the third dimension, **arc_abs_3d** functions similarly to **arc_abs** (see comments there). • The z motion is not taken into account during calculation of the number of Microsteps.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the values specified for `X`, `Y` and `Z` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range. In RTC5 Compatibility Mode, the RTC6 multiplies the values specified for `Z` by 16. The allowed value range decreases accordingly.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**arc_abs**, **arc_rel_3d**	

Normal List Command	**arc_rel**	
Function	Moves the laser focus from the current position at mark speed along an arc with the specified angle and center point (relative coordinate values) within a 2D Image Field.	
Call	`arc_rel( dX, dY, Angle )`	
Parameters	`dX`	Relative x coordinate of the arc center. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values.
	`dY`	Like `dX` (analogously).
	`Angle`	Arc angle. In degrees. As a 64-bit IEEE floating point value. A positive sign means “clockwise”. Allowed value range: [–3,600.0°…+3,600.0°] (±10 full circles). Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_arc_rel**	
Comments	• The coordinates for the arc center are to be supplied as relative coordinates with respect to the current position. Otherwise, **arc_rel** is identical to **arc_abs** (see comments there).	
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `dX` and `dY` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_mark_speed**, **set_scanner_delays**, **arc_abs**, **timed_arc_rel**, **mark_rel**, **arc_rel_3d**, **mark_ellipse_rel**	

Normal List Command	**arc_rel_3d**	
Function	Moves the laser focus at mark speed from the current position helical around an axis parallel to the z axis. The x and y components thereby characterize an arc with the specified angle around the specified axis, while the z component characterizes a linear motion from the current position to the specified end point. The position of the helical axis and the z end coordinate are specifiable as relative coordinate values.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **arc_rel_3d** has the same effect as **arc_rel**.	
Call	`arc_rel_3d( dX, dY, dZ, Angle )`	
Parameters	`dX`	Position of the helical axis (parallel to the z axis) as relative x coordinate. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values.
	`dY`	Like `dX` (analogously).
	`dZ`	Relative z end coordinate. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
	`Angle`	Arc angle. In degrees. As a 64-bit IEEE floating point value. A positive sign means "clockwise". Allowed value range: [–3,600.0°…+3,600.0°] (±10 full circles). Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_arc_rel_3d**	
Comments	• The position of the helical axis (`dX`, `dY`) and the z end coordinate (`dZ`) are to be specified as relative coordinates with respect to the current position. Otherwise, **arc_rel_3d** is identical to **arc_abs_3d** (see comments there).	
RTC4→RTC6	New command. RTC4 Compatibility Mode: see **arc_abs_3d**.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**arc_abs_3d**, **arc_rel**	

Ctrl Command	auto_cal
Function	Only for some discontinued systems. Controls the functions for (automatic self-) calibration of the scan system attached to the specified scan head connector.
Call	`ErrorCode = auto_cal( HeadNo, Command )`
Parameters	`HeadNo` Number of the scan head connector. As an unsigned 32-bit value. Allowed values: = 1: Connector for First Scan Head. = 2: Connector for Second Scan Head. Requires Option "Second Scan Head Control".
	`Command` Control parameter. As an unsigned 32-bit value. Allowed value range: [0…4]. = 0: The RTC6 detects the current Home-In positions, stores them in the RTC6 DLL and in the Flash Memory as Home-In reference values and initializes the gain values and offset values (Gain = 1.0, Offset = 0). = 1: The RTC6 detects the current Home-In positions, calculates and sets the new gain values and offset values and thereby activates drift compensation. = 2: The RTC6 deactivates drift compensation by initializing the gain and offset values (Gain = 1.0, Offset = 0). = 3: The RTC6 detects the current Home-In positions (but – in comparison to `Command` = 1 – leaves the gain and offset values unchanged and that is, does not activate drift compensation = 4: The RTC6 checks the ASC hardware whether a scan system attached to the specified scan head connector is equipped with an internal sensor system for automatic self-calibration – Home-In sensors) and returns the type and status of the detected sensor system. The detected type is also stored in the Flash Memory.
Result	Error code or type of sensor system. As an unsigned 32-bit value. 3 **auto_cal** cannot be executed because the BUSY list execution status or INTERNAL-BUSY list execution status is currently set. 6 Parameter error. The following error codes are only returned after `Command` = 0…3: 0 No error. 1, 10, 11 Home-In sensor not found (this could also mean a Home-In sensor is defective) (1: for x axis (galvanometer scanner 2) 10: for y axis (galvanometer scanner 1) 11: for both axes). 2, 20, 22 The spread in measured values during a measurement cycle is too high (2: for x axis / 20: for y axis / 22: for both axes).

Ctrl Command	auto_cal	
Result (cont'd)	4, 40, 44	Reference data not found (only for `Command` = 1 and 3) (4: for x axis / 40: for y axis / 44: for both axes).
	5, 50, 55	Calibration error (Error during calibration or error in reference data) (5: for x axis / 50: for y axis / 55: for both axes).
	9, 90	Sensor for x axis or y axis is defective. Only returned after `Command` = 0, 1 or 3.
	The following error code is only returned after `Command` = 0 or 4, and even then only if no other errors occurred:	
	8	Download error. The values have possibly not been saved. For this error, the **get_last_error** return code `RTC6_FLASH_ERROR` is always generated.
	The following values are only returned after `Command` = 4:	
	100	For both axes: a sensor system of type1 is included and is functioning.
	200	For both axes: a sensor system of type2 is included and is functioning.
	19	x axis (galvanometer scanner 2): a sensor system of type1 is included and is functioning. y axis (galvanometer scanner 1): sensor system is defective.
	29	x axis (galvanometer scanner 2): a sensor system of type2 is included and is functioning. y axis (galvanometer scanner 1): sensor system is defective.
	91	x axis (galvanometer scanner 2): sensor system is defective. y axis (galvanometer scanner 1): a sensor system of type1 is included and is functioning.
	92	x axis (galvanometer scanner 2): sensor system is defective. y axis (galvanometer scanner 1): a sensor system of type2 is included and is functioning.
	99	For both axes: sensor system is defective.
	255	For both axes: there is no sensor system included in the scan system.
Multi-board Com'd Name	**n_auto_cal**	
Comments	• For usage of **auto_cal**, see Chapter 8.10 "Automatic Self-Calibration", page 284. • At the end of **auto_cal** execution, the RTC6 board always moves the galvanometer scanners back to the position held prior to the call, possibly with corrections attributable to changed gain values and offset values. • During determination of the current Home-In positions with **auto_cal**(`Command` = 0, 1 or 3), the current gain and offset corrections of the galvanometer scanners are not taken into account; neither are any head corrections or the current positions of the galvanometer scanners. • After first-time or renewed connecting a scan system, a reference value determination should be performed by **auto_cal**(`Command` = 0). Otherwise, the RTC6 uses the stored reference values of a previously operated (possibly different) scan system when later performing an automatic self-calibration.	

Ctrl Command	auto_cal
Comments (cont'd)	• After initialization of the RTC6, drift compensation is turned off (gain = 1.0, offset = 0). However, previously determined reference values are still available. • If no appropriate Home-In reference values were stored or the scan system is not equipped with Home-In sensors, then the measurement routine for **auto_cal**(Command = 1) (axis-specific) automatically aborts and restores the prior state. • **auto_cal** is not executed, if a HeadNo or Command value is invalid (return value 6, **get_last_error** return code RTC6_PARAM_ERROR). This also applies for HeadNo = 2 if the Option "Second Scan Head Control" is not enabled (return value 6). • **auto_cal** is not executed (return value 3, **get_last_error** return code RTC6_BUSY), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **auto_cal** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • For RTC6_PARAM_ERROR, the BUSY list execution status is not checked; therefore return codes RTC6_BUSY and RTC6_PARAM_ERROR do not occur simultaneously. • For Command = 0, 1 or 2, a valid correction table must be loaded and assigned. Otherwise, unexpected jumps can occur when gain/offset values are updated. • Gain and offset correction can also be directly set by **set_hi** (even for systems *without* Home-In sensors). • Reference values determined and stored by **auto_cal**(Command = 0, 1 or 3) can be queried by **get_hi_pos**. • ASC hardware checks are performed not just by **auto_cal**(Command = 4), but also automatically for – **auto_cal**(Command = 0) and – the first call of **auto_cal**(Command = 1) and **auto_cal**(Command = 3) if neither **auto_cal**(Command = 0) nor **auto_cal**(Command = 4) were previously executed. In each case, the detected ASC hardware type gets stored in the Flash Memory and can be subsequently queried by **get_auto_cal**. For automatic Command = 4 execution, however, no corresponding return value is generated. The return value is instead simply that of the primary Command call (see above). When an error occurs, a corresponding error code gets saved to the Flash Memory (therefore, **get_auto_cal** does not return 100 or 200). As soon as hardware functionality is restored, you can clear the error from the Flash Memory by explicitly calling **auto_cal**(Command = 0) or **auto_cal**(Command = 4). **auto_cal**(Command = 1) and **auto_cal**(Command = 3) cannot be executed as long as the error is still in the Flash Memory. The error is *not* cleared by these calls from the Flash Memory.

Ctrl Command	auto_cal
RTC4→RTC6	The functions for automatic self-calibration (`Command` = 0…2) are (largely) unchanged. New: `Command` = 3 and `Command` = 4.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_hi**, **get_hi_pos**, **get_auto_cal**, **write_hi_pos**

Ctrl Command	auto_change
Function	Activates a one-time automatic list change.
Call	`auto_change()`
Multi-board Com’d Name	**n_auto_change**
Comments	• **auto_change** is synonymous with `auto_change_pos( 0 )`. This starts the subsequent list at its beginning. • See also comments for **auto_change_pos**.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**auto_change_pos**, **get_status**, **read_status**

Ctrl Command	auto_change_pos
Function	Activates a one-time automatic list change and simultaneously defines the list position at which execution continues.
Call	`auto_change_pos( Pos )`
Parameters	`Pos` Start position (RTC6 List Memory address) as an offset referenced to the beginning of the list to be started by the automatic list change. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_auto_change_pos**
Comments	• **auto_change_pos** triggers a subsequent *one-time-only* list change or a list new start. For further list changes or list new starts, **auto_change_pos** must be called again. • **auto_change_pos** can be called at any time. • If the automatic list change is activated during processing of a list, then upon reaching **set_end_of_list** execution continues without delay at the supplied start position of the other list. If there is only *one* list (`Mem2` = 0, see **config_list**), then upon reaching **set_end_of_list** execution continues at the supplied start position of this list. • During processing of a list, the other list (and also the current list) can be newly loaded, see Chapter 6.4.6 "Changing Lists Automatically", page 113. • So that **auto_change_pos** can function at all, any already active list must absolutely be finalized by **set_end_of_list**; the new list should already be loaded and the input pointer should be sufficiently ahead of the output pointer (otherwise, "old" commands are executed). If, during list execution, the end of the list is reached without encountering a **set_end_of_list**, then execution automatically continues at the beginning of the current list. • If an automatic list change is activated when no list is currently being processed, then checking takes place as to whether a list has been already processed and the other list has been started (at the supplied start position). If no list has been previously executed, then "List 1" is regarded as already executed (initialization) and "List 2" is started. • If a RTC6 List Memory address outside the corresponding list area is supplied (depending on which list should be started: `Pos` ≥ `Mem1` or `Pos` ≥ `Mem2`), then the start position is set to the beginning of the list (`Pos` = 0). • If, during processing of a list, the **auto_change_pos**(`Pos` > 0) *and* **start_loop** are called, then upon the next **set_end_of_list** the command **auto_change_pos**(`Pos` > 0) is executed; and at the next one the **start_loop** command is executed.

Ctrl Command	auto_change_pos
Comments (cont'd)	• The current List Status values can be queried by **read_status**. • The current List Execution Status values can be queried by **get_status**. • **auto_change_pos** triggers a flush of the buffered list input, see Chapter 6.4.1 "Loading Lists", page 108.
RTC4→RTC6	Basically unchanged functionality. However: The RTC6 List Memory address (`Pos`) is supplied to the RTC6 as a relative memory address referenced to the beginning of the respective list, whereas the RTC4 is supplied an absolute memory address (0…7999). If no RTC6 List Memory area is assigned to "List 2" by **config_list** (`Mem2` = 0), then the RTC6 command behaves like the RTC4 command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**auto_change**, **get_status**, **read_status**

Ctrl Command	bounce_supp
Function	Debounces the external start signal.
Call	`bounce_supp( Length )`
Parameters	`Length` Debouncing time. In ms. As an unsigned 32-bit value. Allowed value range: [0…1023]. The 22 bits with higher significance are ignored.
Multi-board Com'd Name	**n_bounce_supp**
Comments	• **bounce_supp** enables debouncing of start signals received at the /START, /START2 or /Slave-START input ports, see Section "External Start", page 312. Start signals occurring within the defined debouncing time after a successful start signal are thereby suppressed. • Recommended procedure: Start a list, which operates more than one second. If /START, /START2 or /Slave-START bounces, then an additional trigger error signal is generated, which can be detected by **get_marking_info** (Bit #8). Increase the debouncing time until this additional signal is no longer detected. • The debouncing time default value is 0 ms. • **bounce_supp** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_startstop_info**

Normal List Command	camming	
Function	Enables Camming functionality.	
Call	`camming ( FirstPos, NPos, EncoderNo, Ctrl, Scale, Code )`	
Parameters	`FirstPos`	Starting address of the Camming command list (absolute address in RTC6 List Memory). As an unsigned 32-bit value. Allowed values: [0…(2^{23}–1)].
	`NPos`	Length of the Camming command list (without terminating **set_end_of_list** or **list_return**). As an unsigned 32-bit value. Allowed values: [1…(2^{23}–`FirstPos`)].
	`EncoderNo`	Number of the encoder counter, whose pulses is evaluated for controlling the Camming process. As an unsigned 32-bit value. Allowed values: = 0: Encoder counter “Encoder0”. = 1: Encoder counter “Encoder1”. Bits with higher significance are ignored.
	`Ctrl`	Camming control mode. As an unsigned 32-bit value. Allowed values: = 0: RTC6 board controls laser as with a [*]mark[*] Command. **camming** terminates automatically. = 1: User controls laser. **camming** terminates automatically. = 2: User controls laser. **camming** does *not* terminate automatically. The Camming command list is executed until the end point (0 or `NPos`–1) and then waits for a new /START. = 3: User controls laser. **camming** does *not* terminate automatically. The Camming command list is continuously processed in circulation (output index modulus `NPos`).
	`Scale`	Translation (conversion factor) between the encoder-counter value and the command index. As a 64-bit IEEE floating point value. Allowed values: 2^{-60} < |`Scale`| < 2^{60}.
	`Code`	Is not used. As an unsigned 32-bit value.
Multi-board Com’d Name	**n_camming**	

Normal List Command	camming
Comments	• See also Chapter 8.11 "Camming", page 288. • If there are unallowed parameter values, **camming** is replaced by a **list_nop** already during loading (**get_last_error** return code RTC6_PARAM_ERROR). • Each time **camming** is called, the current encoder-counter value is ascertained for use as the new reference value. At a later time point, the corresponding command index of the Camming command list is calculated for the then-current encoder-counter value (using the reference value and the Scale parameter). For each call of **camming**, the first command in the Camming command list (index = 0) is always the first command to be processed. • **camming** waits for a Scanner Delay but sets no delay itself. • If Ctrl = 0, then the laser is (as with a normal [*]mark[*] Command) switched on at the beginning of the Camming process and switched off after it terminates (laser delay settings are taken into account). If Ctrl > 0, then the state of the laser is not changed; its control is then the full responsibility of the user. • If Ctrl = 0 or 1, then **camming** terminates automatically (in the next cycle) as soon as the index first undershoots 0 or overshoots NPos–1 (the final Camming command to be executed is then be the one with an index of 0 or NPos–1). For these two modes (as always, if a list is BUSY list execution status), no External Starts are allowed as long as **camming** has not yet terminated. • In contrast, control modes Ctrl = 2 and 3 are endless. Here, the Camming process can only be terminated by **stop_execution** or an External Stop. However, in these two modes (as an exception) External Starts are allowed if the list (or **camming**) is still active. If Ctrl = 2, then the index is executed until the end point (0 or NPos–1) and then waits for a new /START. If Ctrl = 3, then the index is set to modulus NPos (whereby the encoder speed is supposed not to be so high that a complete rotation is skipped). Thus, Ctrl = 3 works like a ring buffer. • **camming** is a Normal List Command, but with a variable execution period. • **camming** functions even if the Option Processing-on-the-fly is not activated. • While **camming** is executing, **get_out_pointer** always provides the position of **camming**, not the position of the current index.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_encoder_speed**, **set_auto_laser_control**

Undelayed Short List Command	**clear_fly_overflow**
Function	Resets the specified error bits (from **get_marking_info**) for customer-defined monitoring of Processing-on-the-fly applications.
Call	`clear_fly_overflow( Mode )`
Parameters	`Mode` The error bits to be reset. As an unsigned 32-bit value. Bit #0 = 1: error bit Bit #4 (underflow X). Bit #1 = 1: error bit Bit #5 (overflow X). Bit #2 = 1: error bit Bit #6 (underflow Y). Bit #3 = 1: error bit Bit #7 (overflow Y). Bit #4 = 1: error bit Bit #24 (underflow Z). Bit #5 = 1: error bit Bit #25 (overflow Z). `Bit #0…Bit #5` can be combined as desired. Higher-order bits are ignored.
Multi-board Com'd Name	**n_clear_fly_overflow**
Comments	• For usage of **clear_fly_overflow**, see Section "Customer-Defined Monitoring Area", page 265. • All 6 error bits are reset for: – `Mode` = 0 – `Mode` = 63
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_marking_info**, **clear_fly_overflow_ctrl**

Ctrl Command	**clear_fly_overflow_ctrl**
Function	Like **clear_fly_overflow**, but a control command.
Call	`clear_fly_overflow_ctrl( Mode )`
Parameters	`Mode` Like **clear_fly_overflow**.
Multi-board Com'd Name	**n_clear_fly_overflow_ctrl**
Comments	• See **clear_fly_overflow**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 628, OUT 629.
References	**clear_fly_overflow**

Undelayed Short List Command	**clear_io_cond_list**
Function	Clears the bits of the 16-Bit Digital Output Port on the EXTENSION 1 Socket Connector that are set in the parameter `MaskClear`, if the current `IOvalue` at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector meets the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are `1` and the bits specified in `Mask0` are `0`).
Call	`clear_io_cond_list( Mask1, Mask0, MaskClear )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.
	`Mask0` See `Mask1`.
	`MaskClear` See `Mask1`.
Multi-board Com'd Name	**n_clear_io_cond_list**
Comments	• **clear_io_cond_list** clears only those bits of the digital output port that are set in the parameter `MaskClear` and leaves the other bits unchanged. • See also Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81 and Chapter 9.3.2 "Execution of Conditional Commands", page 317.
Examples (Pascal)	• Clear Bit #4 of the output port (DIGITAL OUT4), if Bit #0 of the input port (DIGITAL IN0) is set and Bit #1 to Bit #3 (DIGITAL IN1…3) of the input port are not set: `clear_io_cond_list`($0001, $000E, $0010) • Always clear Bit #15 of the output port (and leave the other bits unchanged): `clear_io_cond_list`(0, 0, $8000)
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_io_cond_list**, **write_io_port**, **write_io_port_mask**, **get_io_status**, **read_io_port**

Ctrl Command	config_laser_signals
Function	Configures the laser output signal types to be outputted on pin (01) (LASER1), pin (02) (LASERON) and pin (09) (LASER2) of the LASER Connector.
Call	`config_laser_signals( Config )`
Parameters	`Config` Desired signal configuration. As an unsigned 32-bit value. The following bits configure: Bit #0…Bit #1: Pin (02) (LASERON channel) Bit #2…Bit #3: Pin (01) (LASER1 channel) Bit #4…Bit #5: Pin (09) (LASER2 channel) Bit #6: Ignored … Bit #31: Ignored Meanings: = 0 = 00_b: LASERON signal = 1 = 01_b: LASER1 signal = 2 = 10_b: LASER2 signal = 3 = 11_b: FirstPulseKiller signal (relevant for YAG Mode, Laser Mode 4) The default setting (after **load_program_file**) is: `Config` = 100100_b = 0x24 = 36.
Multi-board Com'd Name	**n_config_laser_signals**
Comments	• The specified configuration takes effect the next time the laser is switched on. Therefore, **config_laser_signals** should not be called during an active marking procedure. • See also **set_laser_control**, Bit #3 and Bit #4. • **config_laser_signals** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
Examples	• `Config` = 000111_b: FirstPulseKiller signal on LASERON channel, LASER1 signal on LASER1 channel, LASERON signal on LASER2 channel. In this configuration, LASER1 signals synchronously generated with the LASERON signal can be immediately outputted, whereas the laser itself switches on only after a delay by the FirstPulseKiller signal. • `Config` = 100100_b (default setting): LASERON signal on the LASERON channel, LASER1 signal on the LASER1 channel, LASER2 signal on the LASER2 channel. In this configuration, LASER1 signals can only be switched on after the laser, not before it (here the LASERON signal is the laser start signal; LASER1 signals cannot be outputted before the laser start, because the RTC6 only generates LASER1 signals if the LASERON signal is on).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**config_laser_signals_list**, **set_laser_control**

Delayed Short List Command	config_laser_signals_list
Function	Like **config_laser_signals**, but a list command.
Call	`config_laser_signals_list( Config )`
Parameters	`Config` Like **config_laser_signals**.
Multi-board Com'd Name	**n_config_laser_signals_list**
Comments	• See **config_laser_signals**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**config_laser_signals**

Ctrl Command	config_list
Function	Configures the RTC6 List Memory, that is, assigns specific memory locations to the RTC6 List Memory areas.
Call	`config_list( Mem1, Mem2 )`
Parameters	`Mem1` Storage positions for RTC6 List Memory area "List 1". As an unsigned 32-bit value.
	`Mem2` Storage positions for RTC6 List Memory area "List 2". As an unsigned 32-bit value.
Multi-board Com'd Name	**n_config_list**
Comments	• The RTC6 List Memory features 8,388,608 (= 2^{23}) storage positions in total. They can be divided into three areas ("lists") by **config_list**. The sizes of "List 1" and "List 2" are specified by the parameters `Mem1` and `Mem2`. All remaining storage positions not assigned to "List 1" or "List 2" are automatically assigned by **config_list** to the protected RTC6 List Memory area "List 3". • The following rules apply to the parameters `Mem1` and `Mem2` (invalid values are automatically corrected in the specified order): – `Mem1` > 0 ("List 1" must not be empty). `Mem1` = 0 is corrected to `Mem1` = 1. – `Mem1` ≤ 2^{23} ("List 1" can contain a maximum of 2^{23} storage positions). `Mem1` > 2^{23} is corrected to `Mem1` = 2^{23}. – `Mem1` = "–1" is interpreted as `Mem1` = (2^{32}–1) and corrected to `Mem1` = 2^{23}. Example: With `config_list( -1, x )` where `x` is any value (also `x` = –1), "List 1" is automatically assigned the entire RTC6 List Memory (`Mem1` = 2^{23}, `Mem2` = 0, no memory for "List 3"). – `Mem2` ≤ 2^{23} – `Mem1` ("List 2" can maximally receive the "rest" of RTC6 List Memory). `Mem2` = 0 is allowed. `Mem2` > 2^{23} – `Mem1` is corrected to `Mem2` = 2^{23} – `Mem1`. – `Mem2` = "–1" is interpreted as `Mem2` = (2^{32}–1) and corrected to `Mem2` = 2^{23} – `Mem1`. Example: With `config_list( Mem1, -1 )`, "List 2" is automatically assigned with the "rest" of RTC6 List Memory (`Mem2` = 2^{23} – `Mem1`, no memory for "List 3"). – Storage positions for "List 3": 2^{23} – `Mem1` – `Mem2`. • The RTC6 List Memory features 8,388,608 (= 2^{23}) storage positions in total. By default, it is preconfigured so that "List 1" and "List 2" can each accept 4,194,304 (= 2^{22}) list commands (`Mem1` = `Mem2` = 4,194,304). The protected RTC6 List Memory area "List 3" owns no storage positions, because 2^{23} – `Mem1` – `Mem2` = 0. • **config_list** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – a list has been paused by **set_wait** (PAUSED list execution status set)

Ctrl Command	config_list
Comments (cont'd)	• Configuration by **config_list** does not alter the contents of RTC6 List Memory. Repeating the call with differing parameters is therefore nondestructive. However, after a configuration change, previously loaded list commands are processed in accordance with the new configuration. Moreover, a configuration change could in some circumstances affect the input pointer (if, prior to the reconfiguration, it pointed to a memory position which has been assigned to "List 3" by the configuration change, then it is shifted to the beginning of "List 1") or affect a previously started automatic list change. This should be taken into account when further loading or executing command lists. Also observe the notes in Chapter 6.3.2 "Configuring the RTC6 List Memory", page 106. • If you do not know the current configuration data for RTC6 List Memory (`Mem1` and `Mem2`), you can find out after **`load_list( ListNo, 0 )`** or `set_start_list_pos( ListNo, 0 )` by using **get_list_space** (with "ownership" changes, **get_config_list** must be called in advance). • **config_list** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. In addition: increased value range and changed initialization values.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_config_list**

Ctrl Command	control_command	
Function	Sends a control command. Whether and how the addressed scan system reacts depends on its properties (among other things, the scan system firmware). See also Comments, page 374.	
Call	`control_command( Head, Axis, Data )`	
Parameters	`Head`	Scan head connector number of the RTC board. As an unsigned 32-bit value. Allowed values: = 1: Connector for First Scan Head. = 2: Connector for Second Scan Head.
	`Axis`	Number of the axis. As an unsigned 32-bit value. Allowed values: = 1: x axis (STATUS channel, Galvanometer scanner 2). = 2: y axis (STATUS1 channel, Galvanometer scanner 1).
	`Data`	Command code with optional parameter. See Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards", page 1057. As an unsigned 32-bit value. The upper part (Bit #16…Bit #31) is *not* evaluated. The lower part (Bit #0…Bit #15) is evaluated as follows: • $Code_{HIGH}$ The more significant data byte (Bit #8…Bit 15). Represents a command code. Presented as hexadecimal number "(hex)" in Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards", page 1057. • $Code_{LOW}$ The less significant data byte (Bit #0…Bit #7). Represents an optional parameter. Presented as hexadecimal number "(hex)" in Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards", page 1057. Example: By `Data` = 0501_{H}, that is, $Code_{HIGH}$ = 05 (**SetMode**) and $Code_{LOW}$ = 01 the actual position is set as the data type to be transmitted.
Multi-board Com'd Name	**n_control_command**	

Ctrl Command	control_command
Comments	• **control_command** can only be used with iDRIVE Scan Systems. • **control_command** is not evaluated by scan systems without i*DRIVE* technology. • **control_command** is not executed with unallowed values of `Head` and/or `Axis` (**get_last_error** return code `RTC6_PARAM_ERROR`). • Command code `Data` is passed to the scan system instead of the usual position data. Therefore, the corresponding galvanometer scanner Microstep is omitted, if **control_command** is called during execution of a list. • Under some circumstances, **control_command** might be unavailable at the Connector for First Scan Head if Speed-Dependent Laser Control has been activated by `set_auto_laser_control`(`Mode` = 2). • **control_command** is allowed in Boot Phase 2, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	Unchanged functionality. Note that all data selected to be transmitted by **control_command** are always in the 20-bit range, even in RTC4 Compatibility Mode (see also comments on **get_value**).
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_value**, **get_values**, **get_head_status**, **set_trigger**, **set_trigger4**, **set_trigger8**, **get_waveform**

Ctrl Command	**copy_dst_src**
Function	Creates entries in the internal management table for an indexed character, text string or subroutine with the specified index (`Dst`) by copying the table entries of another index (`Src`).
Call	`copy_dst_src( Dst, Src, Mode )`
Parameters	`Dst` Index of the indexed character, text string or subroutine whose entries should be copied from `Src`. As an unsigned 32-bit value. Allowed value range: [0…1023] for indexed characters or subroutines, [1024+0…1024+41] for indexed text strings.
	`Src` Index of the indexed character, text string or subroutine whose entries should be copied to `Dst`. As an unsigned 32-bit value. Allowed value range: [0…1023] for indexed characters or subroutines, [1024+0…1024+41] for indexed text strings.
	`Mode` Determines which management tables are to be changed. As an unsigned 32-bit value. Bit #0 = 0: `Dst` is the index of an indexed character or the index of an indexed text string. Bit #0 = 1: `Dst` is the index of an indexed subroutine. Bit #1 = 0: `Src` is the index of an indexed character or the index of an indexed text string. Bit #1 = 1: `Src` is the index of an indexed subroutine. Bit #2…Bit #31: Not evaluated.
Multi-board Com'd Name	**n_copy_dst_src**
Comments	• If an index value (`Dst` and/or `Src`) is invalid, then **copy_dst_src** is ignored (**get_last_error** return code `RTC6_PARAM_ERROR`). • **copy_dst_src** creates an additional reference (index) to an indexed character, text string or subroutine (that can also be called with this new index). **copy_dst_src** only alters the corresponding entry in the internal management table and does not modify the content of the RTC6 List Memory. This allows copying, renumbering or converting between indexed characters, text strings or subroutines without having to reload each time. • A real copy of an indexed character, text string or subroutine in the protected RTC6 List Memory area "List 3" can be created (subsequent to **copy_dst_src**) with **save_disk**/**load_disk**. Characters, text strings and/or subroutines with multiple references are thereby written several times to the RTC6 List Memory. Keep this in mind in order to prevent unintended memory overflow of the protected RTC6 List Memory area "List 3". • See also Section "Managing Indexed Characters and Text Strings", page 123.

Ctrl Command	copy_dst_src
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**save_disk**, **load_disk**

<table>
<tr><th>Ctrl Command</th><th>create_dat_file</th></tr>
<tr><td>Function</td><td>Generates a new RTC6DAT.dat file of the current version.</td></tr>
<tr><td>Call</td><td><code>Version = create_dat_file (Flag)</code></td></tr>
<tr><td>Parameters</td><td><code>Flag</code> As a signed 32-bit value.
1: Generates a new RTC6DAT.dat file and returns the current version.
–1: Only returns the current version.</td></tr>
<tr><td>Result</td><td><code>Version</code> As an unsigned 32-bit value.
> 600: Current version.
0: create_dat_file cannot be executed (get_last_error return code RTC6_BUSY or no program is loaded on the RTC6).
1: Not enough Windows memory (get_last_error return code RTC6_OUT_OF_MEMORY).
2: The output file cannot be opened (get_last_error return code RTC6_PARAM_ERROR). Make sure that you have write permissions granted for the output path containing the RTC6DAT.dat.</td></tr>
<tr><td>Multi-board Com’d Name</td><td>n_create_dat_file</td></tr>
<tr><td>Comments</td><td>• Earlier RTC6DAT.dat versions cannot be used anymore. Otherwise, load_program_file returns error code 7 (get_last_error return code RTC6_VERSION_MISMATCH).
• RTC6DAT.dat version information is available only after load_program_file has also been executed with the current RTC6 DLL instance.
• RTC6DAT.dat version information is not taken over when acquiring an RTC6 board (no matter if it is already initialized or not).
• A newly generated RTC6DAT.dat (of a specific version) can be used at any time for all RTC6 boards that require the same version.
• Each newly generated RTC6DAT.dat contains (besides static initializations) also:
– user definable tables which have been loaded earlier by load_varpolydelay, load_auto_laser_control, load_jump_table, load_jump_table_offset, load_position_control
– a “freely definable wobbel shape” which have been defined earlier by set_wobbel_vector
The user definable tables are automatically available after the next load_program_file whereas the “freely definable wobbel shape” requires a set_wobbel_mode(Mode > 1) call in addition.
• Note that create_dat_file overwrites an already existing RTC6DAT.dat without warning. Keep a copy of your original RTC6DAT.dat in a safe place.
• create_dat_file cannot be executed, while a list is being processed (get_last_error return code RTC6_BUSY).</td></tr>
</table>

Ctrl Command	create_dat_file
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 610, OUT 610, RBF 615. Last change DAT 603.
References	**load_program_file**

Ctrl Command	disable_laser
Function	Disables the Signals for "Laser Active" Operation.
Call	`disable_laser()`
Multi-board Com'd Name	**n_disable_laser**
Comments	• **disable_laser** disables the Signals for "Laser Active" Operation at the output ports LASER1, LASER2 and LASERON (the signals are set to their respective "Off" level). Pin (02) of the LASER Connector is then at a fixed level. However, standby signals activated with **set_standby** or **set_standby_list** continue to be outputted by the LASER1 and LASER2 output ports. If the standby signals are deactivated, also pin (01) and pin (09) of the LASER Connector and pins (19) and (22) of the EXTENSION 2 Socket Connector are at a fixed level after **disable_laser**. • The Signals for "Laser Active" Operation can also be disabled by **set_laser_control** and reenabled by **set_laser_control** or **enable_laser**. • After initialization of the RTC6 with **load_program_file**, the laser control is *deactivated* and absolutely requires **set_laser_control** for activation. • If **disable_laser** results in an `RTC6_TIMEOUT` error (for example, if no program file has been loaded), the LASER1, LASER2 and LASERON output ports can only be reactivated with **set_laser_control** after a **load_program_file** command. • By **get_startstop_info** (`Bit #9`), the current status of the Laser Control Signals (globally enabled, yes or no) can be queried. • By **get_startstop_info** (`Bit #14`), the status "laser enabled" (yes or no) can be queried.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change: DLL 617, OUT 617: **get_startstop_info** (`Bit #14`).
References	**enable_laser**, **set_laser_control**, **get_startstop_info**

Ctrl Command	enable_laser
Function	Enables the Signals for "Laser Active" Operation.
Call	`enable_laser()`
Multi-board Com'd Name	**n_enable_laser**
Comments	• After a hardware reset (and after **load_program_file**), **set_laser_control** must be called to activate the LASER1, LASER2 and LASERON output ports and define the signal level, see Chapter 7.4 "Laser Control", page 189. Otherwise, **enable_laser** has no effect. • After initialization of the RTC6 with **load_program_file**, the Laser Control Signals are *deactivated*. For first-time activation, **set_laser_control** must be called (see above). After a subsequent disabling by **disable_laser** (or **set_laser_control**), the **enable_laser** (or **set_laser_control**) command can be used for reenabling. • Even if the Laser Control Signals have been enabled with **enable_laser** or **set_laser_control**, they are not outputted without further commands, see Chapter 7.4 "Laser Control", page 189. • By **get_startstop_info** (`Bit #9`) the current status of the Laser Control Signals (globally enabled, yes or no) can be queried. • By **get_startstop_info** (`Bit #14`), the status "laser enabled" can be queried.
RTC4→RTC6	Basically unchanged functionality. In some circumstances, **set_laser_control** must be called before **enable_laser** (see above).
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**disable_laser**, **set_laser_control**, **get_startstop_info**, **set_laser_mode**

Ctrl Command	eth_assign_card
Function	Enters the RTC6 Ethernet Board with index SearchNo from the search result list at the RTC6 board management index CardNo.
Call	`Result = eth_assign_card( SearchNo, CardNo )`
Parameters	SearchNo: Index of the search result list. As an unsigned 32-bit value. Allowed value range: [1…(Result value > 0 of **eth_found_cards**]. CardNo: Index of the RTC6 board management, at which the RTC6 Ethernet Board is to be entered. As an unsigned 32-bit value. Allowed value range: [(**rtc6_count_cards** + 1)…255] or 0.
Result	Error code. As a signed 32-bit value. –2: Error: the entry cannot be made. At this index, already an RTC6 Ethernet Board is entered. –1: Error: the entry cannot be made. At this index, already an RTC6 PCIe Board is entered. 0: Error: the entry cannot be made. SearchNo is invalid (0, > **eth_found_cards**). Or: CardNo is invalid (1 ≤ CardNo ≤ **rtc6_count_cards**, > 255). n (= CardNo): Success: the card number entry has been made.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• To create a search result list, **eth_search_cards** and **eth_search_cards_range** are used. See also Chapter 16.5.2 "About Searching for RTC6 Ethernet Boards", page 984. • No RTC6 Ethernet Board or RTC6 PCIe Board must already be entered at the specified index CardNo. An RTC6 Ethernet Board must be explicitly removed by **eth_remove_card** in advance. RTC6 PCIe Boards cannot be removed. • When successful, **eth_count_cards** is automatically increased by 1. • With CardNo = 0 the RTC6 Ethernet Board is entered at **rtc6_count_cards** + **eth_count_cards** + 1. Thus, a continuous card numbering without gaps can automatically be created (see also **eth_remove_card**). CardNo = 0 and freely chosen assignment of RTC6 Ethernet Boards to indexes should not be concurrently used. • With all errors, the **get_last_error** return code RTC6_PARAM_ERROR is generated. • See also Chapter 16.5.3 "About the RTC6 Board Management", page 985.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_assign_card_ip**, **eth_count_cards**, **eth_found_cards**, **eth_max_card**, **eth_search_cards**, **eth_search_cards_range**, **eth_remove_card**, **rtc6_count_cards**, **eth_set_com_timeouts_auto**

Ctrl Command	eth_assign_card_ip
Function	Enters an RTC6 Ethernet Board with the IP address `Ip` at the index `CardNo` in the RTC6 board management.
Call	`Result = eth_assign_card_ip( Ip, CardNo )`
Parameters	`Ip` IP address for the RTC6 Ethernet Board. As an unsigned 32-bit value. `CardNo` Index of the RTC6 board management, at which the RTC6 Ethernet Board is to be entered. As an unsigned 32-bit value. Allowed value range: [(**rtc6_count_cards** + 1)…255] or 0.
Result	Error code. As a signed 32-bit value. –2 Error: the entry cannot be made. At this index, already an RTC6 Ethernet Board is entered. –1 Error: the entry cannot be made. At this index, already an RTC6 PCIe Board is entered. 0 Error: the entry cannot be made. `CardNo` is invalid (1 ≤ `CardNo` ≤ **rtc6_count_cards**, > 255). n (= `CardNo`) Success: the entry has been made.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• No RTC6 Ethernet Board or RTC6 PCIe Board must already be entered at the specified index `CardNo`. An RTC6 Ethernet Board must be explicitly removed by **eth_remove_card** in advance. RTC6 PCIe Boards cannot be removed. • When successful, **eth_count_cards** is automatically increased by 1. • With `CardNo` = 0 the RTC6 Ethernet Board is entered at **rtc6_count_cards** + **eth_count_cards** + 1. Thus, a continuous card numbering without gaps can automatically be created (see also **eth_remove_card**). `CardNo` = 0 and freely chosen assignment of RTC6 Ethernet Boards to indexes should not be concurrently used. • With all errors, the **get_last_error** return code `RTC6_PARAM_ERROR` is generated. • The RTC6 Ethernet Board is entered in the RTC6 board management, even if there is an Ethernet error during its initialization. Before trying to initialize once again, this RTC6 Ethernet Board must be removed explicitly from the RTC6 board management. • See also Chapter 16.5.3 "About the RTC6 Board Management", page 985.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_assign_card**, **eth_count_cards**, **eth_remove_card**, **eth_set_com_timeouts_auto**

Ctrl Command	eth_boot_dcmd
Function	Standalone Functionality: Defines the next following control command as a boot command for Standalone Operation Mode.
Prerequisite	RTC6 Software Package ≥ V1.7.0 and BIOS-ETH ≥ 26.
Call	`eth_boot_dcmd()`
Parameters	None.
Result	None.
Multi-board Com'd Name	**n_eth_boot_dcmd**
Comments	• The definition refers to the subsequent control command, that is, list commands in between do not matter. • The boot definition is omitted without replacement, if the defined control command is not permitted for booting, see Chapter 16.7.5 "Control Commands Allowed for Automatic Booting", page 992. • **eth_boot_dcmd** is only allowed with RTC6 Ethernet Boards. Otherwise, a **get_last_error** return code `RTC6_TYPE_REJECTED` is generated. • See Chapter 16.7 "Standalone Functionality", page 988.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 618, OUT 618, RBF 623.
References	**eth_boot_timeout**

Ctrl Command	eth_boot_timeout
Function	Standalone Functionality: Sets the waiting time for the automatic transition from Boot Phase 1 to Boot Phase 2.
Prerequisite	RTC6 Software Package ≥ V1.7.0 and BIOS-ETH ≥ 26.
Call	`eth_boot_timeout( TimeOut )`
Parameters	`TimeOut` Waiting time. In s. As an unsigned 32-bit value.
Result	None.
Multi-board Com'd Name	**n_eth_boot_timeout**
Comments	• **eth_boot_timeout** is only allowed with RTC6 Ethernet Boards. Otherwise, a **get_last_error** return code `RTC6_TYPE_REJECTED` is generated. • `TimeOut` = 0 deactivates the waiting time. The boot process waits endlessly for a /START. • Boot Phase 2 starts immediately when a /START is triggered within the waiting time. • See Chapter 16.7 "Standalone Functionality", page 988. • **eth_boot_timeout** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	New command.

Ctrl Command	eth_boot_timeout
Version Info	Available as of DLL 618, OUT 618, RBF 623.
References	**eth_boot_dcmd**

Ctrl Command	eth_check_connection
Function	Checks whether the RTC6 Ethernet Board responds.
Call	`CheckOK = eth_check_connection()`
Parameters	None.
Result	Response behavior. As an unsigned 32-bit value. 0: Not OK. The RTC6 Ethernet Board does not have an Ethernet connection, it does not respond or it is not an RTC6 Ethernet Board. 1: OK. The RTC6 Ethernet Board has an Ethernet connection and it responds.
Multi-board Com'd Name	**n_eth_check_connection**
Comments	• When `CheckOK = 0`, the possible error cause can be queried by **get_last_error**, **eth_get_last_error** and **eth_get_error**. • If the board is not an RTC6 Ethernet Board, then: – **eth_check_connection** is not executed – **get_last_error** return code `RTC6_TYPE_REJECTED` is set • See also Chapter 16.5.4 "Checking the Connection to the RTC6 Ethernet Board", page 986.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 608, OUT 608, RBF 611.
References	**eth_get_error**, **eth_get_last_error**, **get_last_error**

Ctrl Command	eth_config_waveform_streaming_ctrl
Function	Relevant for Data Streaming: Configures the interface of the RTC6 Ethernet Board for Data Streaming.
Prerequisite	RTC6 Software Package ≥ 1.16.3 and BIOS-ETH ≥ 36.
Call	`eth_config_waveform_streaming_ctrl( size, flags )`
Parameters	`size` Payload size in an individual Data Streaming data packet. In bytes. Allowed values: 256, 512, 1024, 2048, 4096, 8192, 16384, 32768. As an unsigned 32-bit value.
	`flags` Bit #31: Reserved. … Bit #01: Reserved. Bit #0 = 0: Data Streaming is switched off. An established connection from StreamParser-DLL with RTC6 Ethernet Board is disconnected immediately. Bit #0 = 1: Data Streaming is switched on. An StreamParser-DLL may establish a connection with RTC6 Ethernet Board. As an unsigned 32-bit value.
Result	None.
Multi-board Com'd Name	**n_eth_config_waveform_streaming_ctrl**
Comments	• **eth_config_waveform_streaming_ctrl** is not executed with an unallowed `size` value (**get_last_error** return code `RTC6_PARAM_ERROR`). • By default, 1024 bytes are preset. • New meta data is sent with each Data Streaming data packet. A smaller `size` value means: – More frequent update of meta data – Higher network load • See StreamParser-DLL – Application Programming Interface Manual. • **eth_config_waveform_streaming_ctrl** uses the same recording period as **set_trigger[*]**. The RTC6 Ethernet Board sends meta data with each Data Streaming data packet: accordingly the meta data rate results from recording period and packet size. The meta data sent is fixed and cannot be configured by users. The RTC6 Ethernet Board sends the Data Streaming data packets to the IP address from which the StreamParser-DLL connects. The RTC6 Ethernet Board listens rigidly to port 63751. The RTC6 Ethernet Board supports only Data Streaming over TCP (the protocol is detected automatically).
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 635, OUT 636.
References	–

Ctrl Command	eth_configure_link_loss
Function	Sets the behavior of the RTC6 Ethernet Board in case of an Ethernet Link Loss.
Prerequisite	RTC6 Software Package ≥ V1.7.6 and BIOS-ETH ≥ 28.
Call	`eth_configure_link_loss( Mode )`
Parameters	`Mode` Action to take. As an unsigned 32-bit value. 0: The Signals for "Laser Active" Operation are suppressed immediately. List execution continues. 1: The Signals for "Laser Active" Operation are switched-off immediately. List execution is stopped immediately (as with **stop_execution** or /STOP). 2: A **simulate_ext_stop** is passed on to all slave boards. 3: Ethernet Link Loss detection switched off. Default after **load_program_file**.
Result	None.
Multi-board Com'd Name	**n_eth_configure_link_loss**
Comments	• For `Mode` > 3, the **get_last_error** return code `RTC6_PARAM_ERROR` is set and **eth_configure_link_loss** is not executed. • The RTC6 Ethernet Board detects an Ethernet Link Loss after 100 ms at the latest. • Whether an Ethernet Link Loss has been detected can be queried by **get_startstop_info**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 622, OUT 622.
References	**get_startstop_info**

Ctrl Command	eth_convert_ip_to_string
Function	Converts an IP address in big-endian byte order to the usual dotted decimal notation (for example, "192.168.250.1").
Call	`eth_convert_ip_to_string( Ip, IpString )`
Parameters	`Ip` To-be converted IP address in big-endian byte order. As an unsigned 32-bit value. `IpString` Converted IP address. As a pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 64-bit value) to an array of 4 unsigned 32-bit values, where the converted IP address as a string in usual dotted decimal notation can be found (synonymously to a character array of length 16, low byte first, with concluding \0).
Result	None.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• For storage of the converted IP address, the user program must provide a memory area of 4 × 4 bytes at the address specified by `IpString`. • Example: `Ip` = 33204416, `IpString`[0] = "192.", `IpString`[1] = "168.", `IpString`[2] = "250.", `IpString`[3] = "1\0xx". • The use of **eth_convert_ip_to_string** is useful, for example, for **eth_get_static_ip**. • See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 984.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_convert_string_to_ip**, **eth_get_static_ip**

Ctrl Command	eth_convert_string_to_ip
Function	Converts an IP address in usual dotted decimal notation (for example, "192.168.250.1") to the big-endian byte order.
Call	`ResultIp = eth_convert_string_to_ip( &IpString )`
Parameters	`IpString` To-be converted IP address in usual dotted decimal notation. As a string (char array). As a pointer to a \0-terminated ANSI string. 16 bytes max.
Result	IP address. As an unsigned 32-bit value. 0: `IpString` is faulty. > 0: Converted IP address in big-endian byte order.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• Example: With parameter `IpString` = "192.168.250.1" the result value is `IpString` = 33204416. See also comment at **eth_search_cards**. • The use of **eth_convert_string_to_ip** is useful, for example, for **eth_search_cards**, **eth_search_cards_range** and **eth_set_static_ip** as well as with **eth_get_card_info** and **eth_get_card_info_search**. • See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 984.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_convert_ip_to_string**, **eth_get_card_info**, **eth_get_card_info_search**, **eth_search_cards**, **eth_search_cards_range**, **eth_set_static_ip**

Ctrl Command	eth_count_cards
Function	Returns the number of RTC6 Ethernet Boards entered in the *RTC6 board management*.
Call	`Result = eth_count_cards()`
Parameters	None.
Result	Number of RTC6 Ethernet Boards recorded in the RTC6 board management. As an unsigned 32-bit value.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• Unlike **rtc6_count_cards**, you cannot deduce in each case the highest index of all RTC6 boards from the result value (see **eth_assign_card** and **eth_remove_card** with `CardNo` = 0, see also **eth_max_card**). • It is only ensured that each RTC6 Ethernet Board number is greater than the highest RTC6 PCIe Board number, see **rtc6_count_cards**. • Certainly **eth_count_cards** can be greater than **eth_found_cards**. This is the case, if RTC6 Ethernet Boards have been entered explicitly by its IP addresses into the RTC6 board management, see **eth_assign_card_ip**. • After successful execution of **eth_assign_card**, **eth_count_cards** is automatically increased by 1. • After successful execution of **eth_remove_card**, **eth_count_cards** is automatically decreased by 1. • See also Chapter 16.5.3 "About the RTC6 Board Management", page 985.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_assign_card**, **eth_assign_card_ip**, **eth_found_cards**, **eth_max_card**, **eth_remove_card**, **rtc6_count_cards**

Ctrl Command	eth_found_cards
Function	Returns the number of RTC6 Ethernet Boards in the search result list.
Call	`Result = eth_found_cards()`
Parameters	None.
Result	Number of RTC6 Ethernet Boards in the search result list. As an unsigned 32-bit value. 0: There are no RTC6 Ethernet Boards in the search result list. Possibly the search has not been carried-out yet. n: Number of RTC6 Ethernet Boards contained in the search result list.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• **eth_found_cards** does not necessarily match the – RTC6 Ethernet Board number in the RTC6 board management – total number of RTC6 Ethernet Boards and RTC6 PCIe Boards. • To create the search result list, use either **eth_search_cards** or **eth_search_cards_range**. Their result value is the same as the one of **eth_found_cards**. See also Chapter 16.5.2 "About Searching for RTC6 Ethernet Boards", page 984. • Information about a particular RTC6 Ethernet Board in the search result list can be queried by **eth_get_card_info_search**. • See also Chapter 16.5.3 "About the RTC6 Board Management", page 985.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_get_card_info_search**, **eth_search_cards**, **eth_search_cards_range**

Ctrl Command	eth_get_card_info
Function	Returns information about a particular RTC6 Ethernet Board enlisted in the RTC6 board management and has been previously acquired once or is currently acquired.
Call	`eth_get_card_info( CardNo, Ptr )`
Parameters	`CardNo` Index of the RTC6 Ethernet Board in the RTC6 board management. As an unsigned 32-bit value. `Ptr` Board information. As a pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 64-bit value) to an array of 16 unsigned 32-bit values.
Result	None.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• For storage of the card information, the user program must provide (at the address specified by `Ptr`) a memory area of 16 × 4 bytes. • Board information [0] = Firmware version of the Ethernet communication software [1] = Serial number [2] = IP address (Big Endian format) [3] = MAC address (lower 4 bytes) [4] = MAC address (upper 2 bytes) [5] = Is acquired (1 or 0) [6] = IP address of the connected PC (Big Endian format) [7] = Force DHCP (1 or 0) [8] = Static IP address (Big Endian format) [9] = Static net mask (Big Endian format) [10] = Static gateway (Big Endian format) [11] = UDP port for board searches [12] = UDP port for exclusive connections [13] = TCP port for an exclusive connection [14] = Reserved [15] = Reserved • See also Chapter 16.5.3 "About the RTC6 Board Management", page 985.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_count_cards**, **eth_get_card_info_search**, **eth_max_card**, **get_card_type**

Ctrl Command	eth_get_card_info_search
Function	Returns information about a particular RTC6 Ethernet Board in the search result list.
Call	`eth_get_card_info_search( SearchNo, Ptr )`
Parameters	`SearchNo` Index of the search result list. As an unsigned 32-bit value. `Ptr` Card information. As a pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 64-bit value) to an array of 16 unsigned 32-bit values.
Result	None.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• To create the search result list, use either **eth_search_cards** or **eth_search_cards_range**. Their result value is the same as the one of **eth_found_cards**. See also Chapter 16.5.2 "About Searching for RTC6 Ethernet Boards", page 984. • For `SearchNo` = 0 and `SearchNo` > **eth_found_cards** a default information is returned. Furthermore, the **get_last_error** return code `RTC6_PARAM_ERROR` is generated. • If an search has not yet been carried-out then automatically the following applies: `SearchNo` > **eth_found_cards**. • For storage of the card information, the user program must provide (at the address specified by `Ptr`) a memory area of 16 × 4 bytes. • Card information: Same as **eth_get_card_info**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_found_cards**, **eth_get_card_info**, **eth_search_cards**, **eth_search_cards_range**

Ctrl Command	eth_get_com_timeouts
Function	Deprecated. Returns timing information for an RTC6 Ethernet Board that is entered in the RTC6 board management.
Call	`eth_get_com_timeouts( &AcquireTimeout, &AcquireMaxRetries, &SendRecvTimeout, &SendRecvMaxRetries, &KeepAlive, &KeepInterval )`
Result	None.
Returned Parameter Values	`AcquireTimeout` As a pointer to an unsigned 32-bit value. `AcquireMaxRetries` See `AcquireTimeout`. `SendRecvTimeout` See `AcquireTimeout`. `SendRecvMaxRetries` See `AcquireTimeout`. `KeepAlive` See `AcquireTimeout`. `KeepInterval` See `AcquireTimeout`. All values are each 0, if the board is not an RTC6 Ethernet Board. All time specifications in μs.
Multi-board Com'd Name	**n_eth_get_com_timeouts**
Comments	• See also Chapter 16.5.3 "About the RTC6 Board Management", page 985. • By **eth_set_com_timeouts**, the settings can be changed (usually only in the context of a problem clarification and only on request by SCANLAB). • The parameters `AcquireTimeout`, `AcquireMaxRetries`, `SendRecvTimeout` and `SendRecvMaxRetries` refer only to the RTC6 DLL. Therefore, they can be read out even without granted access right to an RTC6 Ethernet Board. • The returned parameters `KeepAlive` and `KeepInterval` have the value 0, if for the addressed RTC6 Ethernet Board – there is no access right granted (≤ DLL 618:**get_last_error** return code `RTC6_ACCESS_DENIED` and `RTC6_ETH_ERROR`; ≥ DLL 619: no error message any more) – there is a granted access right, but no connection is established (**get_last_error** return code `RTC6_ETH_ERROR`)
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611. Last change DLL 619: no error message any more, if no access right granted.
References	**eth_set_com_timeouts**, **eth_set_com_timeouts_auto**

Ctrl Command	eth_get_com_timeouts_auto	
Function	Returns timing information (for an automatic mechanism) for an RTC6 Ethernet Board that is entered in the RTC6 board management.	
Call	`eth_get_com_timeouts_auto( &InitialTimeout, &MaxTimeout, &Multiplier, &Mode )`	
Result	None.	
Returned Parameter Values	`InitialTimeout`	The timeout to start with. In ms. As a pointer to a 64-bit IEEE floating point value.
	`MaxTimeout`	Maximum timeout (= sum of the timeouts of the individual attempts). After `MaxTimeout` has elapsed, the network connection is considered to be faulty (`udp_recv_timeout` error, see **eth_get_last_error**). In ms. As a pointer to a 64-bit IEEE floating point value.
	`Multiplier`	Multiplication factor by which the timeout is increased for each attempt. As a pointer to a 64-bit IEEE floating point value.
	`Mode`	Reserved. As a pointer to an unsigned 32-bit value.
Multi-board Com'd Name	**n_eth_get_com_timeouts_auto**	
Comments	• See also Chapter 16.5.3 "About the RTC6 Board Management", page 985. • By **eth_set_com_timeouts_auto**, the settings can be changed. • **eth_get_com_timeouts_auto** returns the values of the automatic mechanism (for setting appropriate timeout values). • All time specifications in ms, not µs. • Default values after **eth_assign_card** or **eth_assign_card_ip**: `InitialTimeout` 0.75, `MaxTimeout` 20.0, `Multiplier` 1.3 • The parameter `Mode` is reserved for future extensions. Returns 1. • **eth_get_com_timeouts_auto** affects solely the RTC6 DLL. There is no need for a connection to the RTC6 Ethernet Board.	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 620, OUT 620.	
References	**eth_set_com_timeouts_auto**, **eth_get_com_timeouts**	

Ctrl Command	eth_get_error		
Function	Returns an accumulated error code. It contains all errors which occurred with the most recently executed RTC6 Ethernet Board commands.		
Call	`EthAccError = eth_get_error()`		
Parameters	None.		
Result	Error code. Identical to **eth_get_last_error**. As an unsigned 32-bit value. If multiple errors occurred simultaneously, then multiple bits are set.		
	`no_error = 0. No error.`	Error	Numeric value ($2^{[Bit]}$)
	`wsasstartup_failed`	Bit 0	1
	`wrong_winsock_version`	Bit 1	2
	`Reserved.`	Bit 2	4
	`udp_create_socket_error`	Bit 3	8
	`udp_bind_socket_error`	Bit 4	16
	`udp_connect_socket_error`	Bit 5	32
	`udp_excl_add_use_error`	Bit 6	64
	`udp_bc_ena_error`	Bit 7	128
	`Reserved.`	Bit 8	256
	`tcp_create_socket_error`	Bit 9	512
	`tcp_excl_add_use_error`	Bit 10	1,024
	`tcp_bind_socket_error`	Bit 11	2,048
	`Reserved.`	Bit 12	4,096
	`tcp_connect_socket_error`	Bit 13	8,192
	`tcp_connect_sel_error`	Bit 14	16,384
	`tcp_connect_fds_error`	Bit 15	32,768
	`tcp_nodelay_error`	Bit 16	65,536
	`create_thread_failed`	Bit 17	131,072
	`udp_recv_error`	Bit 18	262,144
	`udp_send_error`	Bit 19	524,288
	`udp_recv_timeout`	Bit 20	1,048,576
	`already_acquired`	Bit 21	2,097,152
	`not_acquired`	Bit 22	4,194,304
	`access_denied`	Bit 23	8,388,608
	`send_tgm_timeout`	Bit 24	16,777,216
	`card_not_found`	Bit 25	33,554,432
	`core1_timeout`	Bit 26	67,108,864
	`bootmode_set_failed`	Bit 27	134,217,728
Multi-board Com'd Name	**n_eth_get_error**		

Ctrl Command	eth_get_error
Comments	• For error handling, see Chapter 6.8 "Error Handling", page 133. • **eth_get_error** and **n_eth_get_error** are available even without explicit access rights to a specific RTC6 Ethernet Board. • The board-specific error variables `LastError` and `AccError` (see Chapter 6.8 "Error Handling", page 133) are neither generated nor altered by **eth_get_error**. • If with an RTC6 Ethernet Board command call (internally) several errors occur successively, **eth_get_last_error** probably returns only the very last error (for example, only "not_acquired"). The actual error causes can be identified by **eth_get_error** because it returns the accumulated errors. • The accumulated error is reset automatically with each acquiring of the RTC6 Ethernet Board.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 608, OUT 608, RBF 611.
References	**acquire_rtc**, **eth_get_last_error**

Ctrl Command	eth_get_ip
Function	Returns the pertaining IP address for a specified RTC6 board management index.
Call	`IP = eth_get_ip( CardNo )`
Parameters	`CardNo` Index of the RTC6 Ethernet Board in the RTC6 board management. As an unsigned 32-bit value.
Result	IP address. In Big Endian byte order. The value is 0, if the board is not an RTC6 Ethernet Board.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• To convert from Big Endian byte order to the usual dotted decimal notation, **eth_convert_ip_to_string** can be used. See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 984. • **eth_get_ip** allows an faster IP address query from the RTC6 board management than by **eth_get_card_info**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_convert_ip_to_string**, **eth_get_card_info**

Ctrl Command	eth_get_ip_search
Function	Returns the pertaining IP address for a specified RTC6 Ethernet Board in the search result list.
Call	`IP = eth_get_ip_search( SearchNo )`
Parameters	`SearchNo` Index in the search result list. As an unsigned 32-bit value.
Result	IP address. In Big Endian byte order.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• To create the search result list, use either **eth_search_cards** or **eth_search_cards_range**. Their result value is the same as the one of **eth_found_cards**. See also Chapter 16.5.2 "About Searching for RTC6 Ethernet Boards", page 984. • The result value is 0 for `SearchNo` = 0 and `SearchNo` > **eth_found_cards**. At the same time the **get_last_error** return code `RTC6_PARAM_ERROR` is generated. • If an search has not yet been carried-out then automatically the following applies: `SearchNo` > **eth_found_cards**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 608, OUT 608, RBF 611.
References	**eth_get_ip**

Ctrl Command	eth_get_last_error		
Function	Returns an error code which contains only the errors that have occurred during execution of the most recent RTC6 Ethernet Board command.		
Call	`EthLastError = eth_get_last_error()`		
Parameters	None.		
Result	Error code. Identical to **eth_get_error**. As an unsigned 32-bit value. If several errors occurred at the same time, several bits are set.		
	`no_error` = 0. No error.	Error	Numeric value ($2^{[Bit]}$)
	`wsasstartup_failed`	Bit 0	1
	`wrong_winsock_version`	Bit 1	2
	`Reserved.`	Bit 2	4
	`udp_create_socket_error`	Bit 3	8
	`udp_bind_socket_error`	Bit 4	16
	`udp_connect_socket_error`	Bit 5	32
	`udp_excl_add_use_error`	Bit 6	64
	`udp_bc_ena_error`	Bit 7	128
	`Reserved.`	Bit 8	256
	`tcp_create_socket_error`	Bit 9	512
	`tcp_excl_add_use_error`	Bit 10	1,024
	`tcp_bind_socket_error`	Bit 11	2,048
	`Reserved.`	Bit 12	4,096
	`tcp_connect_socket_error`	Bit 13	8,192
	`tcp_connect_sel_error`	Bit 14	16,384
	`tcp_connect_fds_error`	Bit 15	32,768
	`tcp_nodelay_error`	Bit 16	65,536
	`create_thread_failed`	Bit 17	131,072
	`udp_recv_error`	Bit 18	262,144
	`udp_send_error`	Bit 19	524,288
	`udp_recv_timeout`	Bit 20	1,048,576
	`already_acquired`	Bit 21	2,097,152
	`not_acquired`	Bit 22	4,194,304
	`access_denied`	Bit 23	8,388,608
	`send_tgm_timeout`	Bit 24	16,777,216
	`card_not_found`	Bit 25	33,554,432
	`core1_timeout`	Bit 26	67,108,864
	`bootmode_set_failed`	Bit 27	134,217,728
Multi-board Com’d Name	**n_eth_get_last_error**		

Ctrl Command	eth_get_last_error
Comments	• For error handling, Chapter 6.8 "Error Handling", page 133. • **eth_get_last_error** and **n_eth_get_last_error** are available even without explicit access rights to a specific RTC6 Ethernet Board. • The board-specific error variables `LastError` and `AccError` (Chapter 6.8 "Error Handling", page 133) are neither generated nor altered by **eth_get_last_error**. • Each **eth_get_last_error** error also leads to a **get_last_error** error `RTC6_ETH_ERROR`. • If with an RTC6 Ethernet Board command call (internally) several errors occur successively, **eth_get_last_error** probably returns only the very last error (for example, only "not_acquired"). The actual error causes can be identified by **eth_get_error** because it returns the accumulated errors.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_get_error**, **get_error**, **reset_error**, **set_verify**

Ctrl Command	eth_get_port_numbers
Function	Returns the UDP ports for board searches (`UDPsearch`) and exclusive connections (`UDPexcl`) as well as the TCP port of an RTC6 Ethernet Board.
Call	`Result = eth_get_port_numbers( &UDPsearch, &UDPexcl, &TCP )`
Result	Error code. As an unsigned 32-bit value. 0: Success: OK. 1: Error: Not an RTC6 Ethernet Board.
Returned Parameter Values	`UDPsearch` UDP port for board searches. As a pointer to an unsigned 32-bit value. `UDPexcl` UDP port for exclusive connections. As a pointer to an unsigned 32-bit value. `TCP` TCP port for exclusive connections. As a pointer to an unsigned 32-bit value. All values are each 0, if the board is not an RTC6 Ethernet Board.
Multi-board Com'd Name	**n_eth_get_port_numbers**
Comments	• The values can also be queried by **eth_get_card_info**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_get_card_info**, **eth_set_port_numbers**

Ctrl Command	eth_get_serial_search
Function	Returns the pertaining serial number for a specified RTC6 Ethernet Board in the search result list.
Call	`Serialnumber = eth_get_serial_search( SearchNo )`
Parameters	`SearchNo` Index in the search result list. As an unsigned 32-bit value.
Result	Serial number. As an unsigned 32-bit value.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• To create the search result list, use either **eth_search_cards** or **eth_search_cards_range**. Their result value is the same as the one of **eth_found_cards**. See also Chapter 16.5.2 "About Searching for RTC6 Ethernet Boards", page 984. • The result value is 0 for `SearchNo` = 0 and `SearchNo` > **eth_found_cards**. At the same time the **get_last_error** return code `RTC6_PARAM_ERROR` is generated. • If an search has not yet been carried-out then automatically the following applies: `SearchNo` > **eth_found_cards**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 608, OUT 608, RBF 611.
References	**get_serial_number**, **eth_get_card_info_search**

Ctrl Command	eth_get_standalone_status
Function	Returns Standalone Operation Moderelated information about the RTC6 Ethernet Board.
Prerequisite	RTC6 Software Package ≥ V1.9.0 and BIOS-ETH ≥ 30.
Call	`eth_get_standalone_status( &Status, &Error, &Mode )`
Result	None.
Parameters	None.
Returned Parameter Values	`Status` Status of automatic booting, see also Chapter 16.7.6 ”Automatic Booting – Process in Detail”, page 994. As a pointer to an unsigned 32-bit value. = 0: Finished. = 1: In progress. `Error` Errors during automatic booting. As a pointer to an unsigned 32-bit value. = 0: No. = 1: Yes. `Mode` Target state after automatic booting (after a Hardware Reset). As a pointer to an unsigned 32-bit value. = 0: “Normal PC Operation State“. = 1: “Standalone Basic State“. = 2: “Standalone Full State“.
Multi-board Com’d Name	**n_eth_get_standalone_status**
Comments	• **eth_get_standalone_status** requires access rights for the RTC6 Ethernet Board. Otherwise, **eth_get_standalone_status** has no effect (**get_last_error** return code `RTC6_ACCESS_DENIED`). • **eth_get_standalone_status** has no effect, if the board is not an RTC6 Ethernet Board (**get_last_error** return code `RTC6_TYPE_REJECTED`). • Even during automatic booting, access rights can be requested by **acquire_rtc**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 628, OUT 629.
References	–

Ctrl Command	**eth_get_static_ip**
Function	Returns static IP address, subnet mask and gateway address which are saved on the RTC6 Ethernet Board.
Call	`Result = eth_get_static_ip( &Ip, &NetMask, &Gateway )`
Result	Error code. As an unsigned 32-bit value. 0: Success: OK. 1: Error. Not an RTC6 Ethernet Board.
Parameters	None.
Returned Parameter Values	`Ip` Static IP address in Big Endian byte order. As a pointer to an unsigned 32-bit value. `NetMask` Sub net mask in Big Endian byte order. As a pointer to an unsigned 32-bit value. `Gateway` Address of the gateway in Big Endian byte order. As a pointer to an unsigned 32-bit value. All values are each 0, if the board is not an RTC6 Ethernet Board or a static IP address has not been programmed yet.
Multi-board Com'd Name	**n_eth_get_static_ip**
Comments	• To convert from Big Endian byte order to the usual dotted decimal notation, **eth_convert_ip_to_string** can be used. See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 984. • The values can also be queried by **eth_get_card_info**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_convert_ip_to_string**, **eth_get_card_info**, **eth_set_static_ip**

Ctrl Command	eth_max_card
Function	Returns the highest index in the RTC6 board management, where an RTC6 Ethernet Board is entered.
Call	`Result = eth_max_card()`
Parameters	None.
Result	Highest index of an RTC6 Ethernet Board in the RTC6 board management. As an unsigned 32-bit value. 0: No RTC6 Ethernet Board recorded in the RTC6 board management. n: Highest index in the RTC6 board management, where an RTC6 Ethernet Board is entered.
Multi-board Com'd Name	Not available as a multi-board command.

Comments	• **eth_max_card** does not return the total number of RTC6 Ethernet Boards in the RTC6 board management. For this purpose, **eth_count_cards** is available.
	• With **eth_max_card**, all RTC6 Ethernet Boards can be removed from the RTC6 board management by a simple loop:
	`while ( eth_max_card() )`
	`{ release_rtc( eth_max_card() ); eth_remove_card( eth_max_card() ); }`
	• See also Chapter 16.5.3 "About the RTC6 Board Management", page 985.

RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_count_cards**, **eth_remove_card**

Ctrl Command	eth_remove_card
Function	Deletes the RTC6 Ethernet Board entry from the RTC6 board management at the specified index.
Call	`Result = eth_remove_card( CardNo )`
Parameters	`CardNo` Index of the RTC6 Ethernet Board in the RTC6 board management. As an unsigned 32-bit value.
Result	Error code. As a signed 32-bit value. –2 Error: the entry cannot be deleted. At this index an RTC6 Ethernet Board is entered which is still acquired yet. –1 Error: the entry cannot be deleted. At this index an RTC6 PCIe Board is entered. 0 Error: the entry cannot be deleted. At this index "No card" is entered or `CardNo` is invalid (> 255). n (= `CardNo`) Success: the entry has been deleted.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• With all errors, the **get_last_error** return code `RTC6_PARAM_ERROR` is generated. • Entries for "No card"s and RTC6 PCIe Boards cannot be deleted. • Before deleting an RTC6 Ethernet Board entry, an acquired board must be explicitly released by **release_rtc**. • With `CardNo` = 0, the RTC6 Ethernet Board entry at index (**rtc6_count_cards** + **eth_count_cards**) is deleted (see also continuous card numbering without gaps using **eth_assign_card** or **eth_assign_card_ip** and `CardNo` = 0). `CardNo` = 0 and freely chosen assignment of RTC6 Ethernet Boards to indexes should not be concurrently used. • All RTC6 Ethernet Board entries can be deleted from the RTC6 board management by a simple loop (provided that all RTC6 Ethernet Boards are *not* acquired): `while ( eth_remove_card(0)> 0 );` • See also Chapter 16.5.3 "About the RTC6 Board Management", page 985.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_assign_card**, **eth_assign_card_ip**, **eth_count_cards**, **release_rtc**, **rtc6_count_cards**

Ctrl Command	eth_search_cards
Function	Executes a search for RTC6 boards in form of a broadcast and returns the number of RTC6 Ethernet Boards which have answered in the specified address range.
Call	`Result = eth_search_cards( Ip, Netmask )`
Parameters	`Ip` IP address in Big Endian byte order. As an unsigned 32-bit value. `Netmask` Subnet mask in Big Endian byte order. As an unsigned 32-bit value.
Result	Number of RTC6 Ethernet Boards in the specified address range which have answered (within TimeOut). As an unsigned 32-bit value.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• To convert from the usual dotted decimal notation to Big Endian byte order, **eth_convert_string_to_ip** can be used. See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 984. • The timeout value can be set with **eth_set_search_cards_timeout**. • A broadcast only reaches reliably addresses within the specified network segment. For other use cases a board search by IP scan (**eth_search_cards_range**) can be carried out. • The number of found RTC6 Ethernet Boards can also be queried by **eth_found_cards** at a later time.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_convert_string_to_ip**, **eth_found_cards**, **eth_search_cards_range**, **eth_set_search_cards_timeout**

Ctrl Command	eth_search_cards_range
Function	Executes a search within a precisely specified IP address range and returns the number of RTC6 Ethernet Boards which have answered.
Call	`Result = eth_search_cards_range( StartIP, EndIp )`
Parameters	`StartIP` Start IP address in Big Endian byte order. As an unsigned 32-bit value. `EndIp` End IP address in Big Endian byte order. As an unsigned 32-bit value.
Result	Number of RTC6 Ethernet Boards which have answered (within TimeOut). As an unsigned 32-bit value.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• To convert from the usual dotted decimal notation to Big Endian byte order, **eth_convert_string_to_ip** can be used. See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 984. • **eth_search_cards_range** executes the search by sending UDP packets to each IP address within the specified address range. It reliably covers the specified address range (compare to broadcast with **eth_search_cards**). • The TimeOut value can be set with **eth_set_search_cards_timeout**. • The number of found RTC6 Ethernet Boards can also be queried by **eth_found_cards** at a later time.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_convert_string_to_ip**, **eth_found_cards**, **eth_search_cards**, **eth_set_search_cards_timeout**

Ctrl Command	**eth_set_com_timeouts**
Function	Deprecated. Sets timing information (for Ethernet communication) for an RTC6 Ethernet Board that is entered in the RTC6 board management (and needs to be acquired, if `KeepAlive` and `KeepInterval` are to be changed).
Call	`eth_set_com_timeouts( AcquireTimeout, AcquireMaxRetries, SendRecvTimeout, SendRecvMaxRetries, KeepAlive, KeepInterval )`
Parameters	`AcquireTimeout` — As an unsigned 32-bit value. With 0 the parameter is not changed. `AcquireMaxRetries` — See `AcquireTimeout`. `SendRecvTimeout` — See `AcquireTimeout`. `SendRecvMaxRetries` — See `AcquireTimeout`. `KeepAlive` — See `AcquireTimeout`. `KeepInterval` — See `AcquireTimeout`.
Result	None.
Multi-board Com'd Name	**n_eth_set_com_timeouts**
Comments	• See also Chapter 16.5.3 "About the RTC6 Board Management", page 985. • The settings can be queried by **eth_get_com_timeouts**. • All time specifications in µs. • As a rule, it is not required to change these parameter values. In case of timing problems contact SCANLAB in order to clarify your special local characteristics. • `AcquireTimeout`, `AcquireMaxRetries`, `SendRecvTimeout` and `SendRecvMaxRetries` refer only to the RTC6 DLL. Therefore, they can be set even without granted access right to an RTC6 Ethernet Board. • `KeepAlive` and `KeepInterval` are not set, if for the addressed RTC6 Ethernet Board – there is no access right granted (**get_last_error** return code `RTC6_ACCESS_DENIED` and `RTC6_ETH_ERROR`) – there is a granted access right, but no connection is established (**get_last_error** return code `RTC6_ETH_ERROR`) • ≥ DLL 619: No **get_last_error** return code is set with `KeepAlive` = `KeepInterval` = 0 because a connection to an RTC6 Ethernet Board is not necessary. • **eth_set_com_timeouts_auto** sets an automatic mechanism, see there.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611. Last change DLL 619: No longer an error message, if there is no RTC6 Ethernet Board connection.
References	**eth_get_com_timeouts**, **eth_set_com_timeouts_auto**

Ctrl Command	**eth_set_com_timeouts_auto**	
Function	Sets timing information (for Ethernet communication) for an RTC6 Ethernet Board that is entered in the RTC6 board management.	
Call	`eth_set_com_timeouts_auto( InitialTimeout, MaxTimeout, Multiplier, Mode )`	
Parameters	`InitialTimeout`	The timeout to start with. In ms. As a 64-bit IEEE floating point value.
	`MaxTimeout`	Maximum timeout (= sum of the timeouts of the individual attempts). After `MaxTimeout` has elapsed, the network connection is considered to be faulty (`udp_recv_timeout` error, see **eth_get_last_error**). In ms. As a 64-bit IEEE floating point value.
	`Multiplier`	Multiplication factor by which the timeout is increased for each attempt. As a 64-bit IEEE floating point value.
	`Mode`	Reserved. As an unsigned 32-bit value.
Result	None.	
Multi-board Com'd Name	**n_eth_set_com_timeouts_auto**	
Comments	• See also Chapter 16.5.3 "About the RTC6 Board Management", page 985. • Each control command and list command is sent to the RTC6 Ethernet Board in the form of network packets. For each packet sent, the response packet must be received within a certain timeout. If this timeout is exceeded, the process is repeated. To avoid overload, the next timeout value is increased by the factor `Multiplier` with each further attempt. • The settings can be queried by **eth_get_com_timeouts_auto**. • All time specifications in ms, not μs. • If 0 is specified for a parameter, its current value is not changed. Default parameter values (after **eth_assign_card** and **eth_assign_card_ip**): see **eth_get_com_timeouts_auto**. • The parameter `Mode` is reserved for future extensions. • **eth_set_com_timeouts_auto** affects solely the RTC6 DLL. There is no need for a connection to the RTC6 Ethernet Board. • **eth_set_com_timeouts** sets a static mechanism.	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 620, OUT 620.	
References	**eth_get_com_timeouts_auto**, **eth_set_com_timeouts**	

Ctrl Command	eth_set_high_performance_mode
Function	Switches on (off) the "High Performance Mode" for an RTC6 Ethernet Board.
Prerequisite	`Minimum requirement: RTC6 Software Package V1.14.1 and BIOS-ETH 35.`
Call	`ErrorCode = eth_set_high_performance_mode( Mode )`
Parameters	`Mode` Mode. As an unsigned 32-bit value. = 0: Switches the "High Performance Mode" off. = 1: Switches the "High Performance Mode" on.
Result	Error code. As an unsigned 32-bit value. 0: Success: OK. 1: Error: Mode not allowed. **get_last_error** return code `RTC6_PARAM_ERROR`. 2: Error: BIOS not supported (< BIOS-ETH 35). **get_last_error**-return code `RTC6_UNSUPPORTED_BIOS`. 3: Error: No access to the RTC6 Ethernet Board. **get_last_error** return code `RTC6_ACCESS_DENIED`. 4: Error: Not an RTC6 Ethernet Board. **get_last_error** return code `RTC6_TYPE_REJECTED`.
Multi-board Com'd Name	**n_eth_set_high_performance_mode**
Comments	• In "High Performance Mode", the network load can increase significantly. – For compensation, **eth_set_high_performance_mode** calls internally `eth_set_com_timeouts_auto( 0.75, 1000.0, 1.3, 1 )`. – If **eth_set_com_timeouts** has been previously called with larger values, they are not be changed. • Default setting after **eth_assign_card** and **eth_assign_card_ip**: – `Mode` = 0 • **load_program_file** does not reset the setting. • See also Chapter 16.8 ""High Performance Mode"", page 996.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 633.
References	–

Ctrl Command	eth_set_port_numbers
Function	Saves the UDP ports for board searches (UDPsearch) and exclusive connections (UDPexcl) as well as the TCP port into the Flash Memory of the RTC6 Ethernet Board.
Call	`Result = eth_set_port_numbers( UDPsearch, UDPexcl, TCP )`
Parameters	UDPsearch: UDP port for board searches. As an unsigned 32-bit value. Allowed value range: [0…65,535]. Out-of-range values are clipped. UDPexcl: UDP port for exclusive connections. As an unsigned 32-bit value. Allowed value range: [0…65,535]. Out-of-range values are clipped. TCP: TCP port for exclusive connections. As an unsigned 32-bit value. Allowed value range: [0…65,535]. Out-of-range values are clipped.
Result	Error code. As an unsigned 32-bit value. 0: Success: OK. 1: Error: No access to the RTC6 Ethernet Board. **get_last_error** return code RTC6_ACCESS_DENIED. 2: Error: No RTC6 Ethernet Board. **get_last_error** return code RTC6_TYPE_REJECTED. 3: Error: Programming the Flash Memory is not possible. The RTC6 Ethernet Board is possibly busy. **get_last_error** return code RTC6_BUSY or RTC6_FLASH_ERROR.
Multi-board Com'd Name	**n_eth_set_port_numbers**
Comments	• Prerequisites for **eth_set_port_numbers**: – No list must currently be executed on the RTC6 Ethernet Board. – The RTC6 Ethernet Board must be entered in the RTC6 board management. – The RTC6 Ethernet Board must have been acquired. • If 0 is specified for a parameter, it is not overwritten. • Saved parameters are not used until the RTC6 Ethernet Board has been restarted.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_get_port_numbers**

Ctrl Command	**eth_set_remote_tgm_format**
Function	Switches Remote Interface Mode on (off).
Prerequisite	`Minimum requirement: RTC6 Software Package V1.16.0 and BIOS-ETH 38.`
Call	`eth_set_remote_tgm_format( Format )`
Parameters	`Format` Telegram format. As an unsigned 32-bit value. = 0: Remote Interface Mode is not enabled. = 1: Remote Interface Mode is enabled. > 1: Reserved.
Result	None.
Multi-board Com'd Name	**n_eth_set_remote_tgm_format**
Comments	• **eth_set_remote_tgm_format** is only allowed with RTC6 Ethernet Boards. Otherwise, a **get_last_error** return code `RTC6_TYPE_REJECTED` is generated. • Invalid `Format` value: see `Format`= 0. • **eth_set_remote_tgm_format** is not effective in normal operation. • **eth_set_remote_tgm_format** is only effective in Boot Phase 1, see step 3 in Chapter 16.7.3 "Preparing the "Standalone Full State"", page 990. • **eth_set_remote_tgm_format** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 640, OUT 639.
References	–

Ctrl Command	eth_set_search_cards_timeout
Function	Sets a timeout value.
Call	`eth_set_search_cards_timeout( TimeOut )`
Parameters	`TimeOut` The timeout value. In µs. As an unsigned 32-bit value.
Result	None.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• The default timeout value is 5 ms. • The timeout value is relevant for **eth_search_cards** and **eth_search_cards_range**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_search_cards**, **eth_search_cards_range**

Ctrl Command	eth_set_static_ip
Function	Writes a static IP address, a subnet mask and also a gateway address onto the RTC6 Ethernet Board.
Call	`Result = eth_set_static_ip( Ip, NetMask, Gateway )`
Parameters	`Ip` Static IP address in Big Endian byte order. As an unsigned 32-bit value. `NetMask` Subnet mask in Big Endian byte order. As an unsigned 32-bit value. `Gateway` Address of the gateway in Big Endian byte order. As an unsigned 32-bit value.
Result	Error code. As an unsigned 32-bit value. 0: Success: OK. 1: Error: No access to the RTC6 Ethernet Board. **get_last_error** return code `RTC6_ACCESS_DENIED`. 2: Error: No RTC6 Ethernet Board. **get_last_error** return code `RTC6_TYPE_REJECTED`. 3: Error: Programming the Flash Memory is not possible. The RTC6 Ethernet Board is possibly busy. **get_last_error** return code `RTC6_BUSY` or `RTC6_FLASH_ERROR`.
Multi-board Com'd Name	**n_eth_set_static_ip**
Comments	• Prerequisites for **eth_set_static_ip**: – No list must currently be executed on the RTC6 Ethernet Board. – The RTC6 Ethernet Board must be entered in the RTC6 board management. – The RTC6 Ethernet Board must have been acquired. • If no gateway is to be used, then `Gateway` must be set to 0. • To convert from usual dotted decimal notation to Big Endian byte order, **eth_convert_string_to_ip** can be used. See also Chapter 16.5.1 "Notes on Working with IP Addresses", page 984.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_convert_string_to_ip**, **eth_get_static_ip**

Ctrl Command	**execute_at_pointer**
Function	Starts list execution (“List 1” or “List 2”) at the specified address in the RTC6 List Memory.
Call	`execute_at_pointer( Pos )`
Parameters	`Pos` Absolute address of the first list command to be executed. As an unsigned 32-bit value. Allowed value range: [0…(2^{23}–1)].
Multi-board Com’d Name	**n_execute_at_pointer**
Comments	• **execute_at_pointer** essentially functions like **execute_list_pos** (see comments there). However, **execute_at_pointer** requires a start address specified as an *absolute* memory address, whereas **execute_list_pos** requires specification of the list number and a *relative* memory address. • For `Pos` ≥ `Mem1` + `Mem2` (see **config_list**), `Pos` is set to 0.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**execute_list_pos**, **get_out_pointer**

Ctrl Command	execute_list
Function	Starts execution at the beginning of the specified list (“List 1” or “List 2”).
Call	`execute_list( ListNo )`
Parameters	`ListNo` Number of the list to be executed. As an unsigned 32-bit value. Allowed values: [uneven: “List 1“, even: “List 2“].
Multi-board Com’d Name	**n_execute_list**
Comments	• **execute_list** is synonymous with **execute_list_pos** with `Pos` = 0. • **execute_list_1** and **execute_list_2** (with no parameters) can be used alternatively.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_status**, **execute_at_pointer**, **execute_list_pos**

Ctrl Command	execute_list_1
Function	See **execute_list**.
Call	`execute_list_1()`
Multi-board Com’d Name	**n_execute_list_1**
Comments	• –
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**execute_list**

Ctrl Command	execute_list_2
Function	See **execute_list**.
Call	`execute_list_2()`
Multi-board Com’d Name	**n_execute_list_2**
Comments	• –
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**execute_list**

Ctrl Command	execute_list_pos	
Function	Starts list execution ("List 1" or "List 2") at the specified position.	
Call	`execute_list_pos( ListNo, Pos )`	
Parameters	`ListNo`	Number of the list to be executed. As an unsigned 32-bit value. Allowed values: [uneven: "List 1", even: "List 2"].
	`Pos`	Address of the first list command to be executed (offset relative to the start of the respective list). As an unsigned 32-bit value. Allowed value range: [0…(2^{23}–1)].
Multi-board Com'd Name	**n_execute_list_pos**	
Comments	• **execute_list_pos** is not executed (**get_error** return value = `RTC6_BUSY`), if: – the BUSY list execution status is set – a list has been paused by **set_wait** (PAUSED list execution status set) • If the INTERNAL-BUSY list execution status is set, **execute_list_pos** is only executed with a delay (after INTERNAL-BUSY list execution status has been reset again). No checks are performed to determine if a list is currently being loaded. During list processing, the other list (or even the same list) can be simultaneously reloaded (see also Chapter 6.4 "List Handling", page 108). • Programs in the protected RTC6 List Memory area "List 3" cannot be directly executed by **execute_list_pos**. They can only be called from a list ("List 1" or "List 2") as a subroutine. Alternatively, the corresponding area can be assigned by **config_list** to "List 1" or "List 2". • Uneven `ListNo` values cause "List 1" to be executed; otherwise "List 2" is executed. This allows automatically generated continuous list changing by an incremented count. • If "List 2" has not been assigned memory (`Mem2` = 0, see **config_list**) then "List 1" is opened. • If `Pos` is specified as being larger than the RTC6 List Memory area of the respective list (`Pos` > `Mem1` or `Pos` > `Mem2`), then `Pos` is set to 0. • The BUSY list status of the selected list is set and the BUSY list status of the other corresponding list is reset (see **read_status**). The BUSY list execution status-List Execution Status (see **get_status**) is set. • Execution stops when a **set_end_of_list** is encountered. If the end of a list area is reached without encountering a **set_end_of_list**, then execution continues at the beginning of the same list area instead of with the next list. The output pointer remains in the active list area unless a **set_end_of_list** has been encountered and an **auto_change_pos** or **start_loop** has previously called. For both lists to be treated as a single list, you must set the configuration appropriately: for example, `config_list( Mem1+Mem2, 0 )`.	

Ctrl Command	**execute_list_pos**
Comments (cont'd)	• If a home jump (defined with **home_position** or **home_position_xyz**) has been executed by **set_end_of_list**, then **execute_list_pos** leads to a corresponding home return (the INTERNAL-BUSY list execution status is set while the home return is executed). • **execute_list_pos** triggers a flush of the buffered list input (see Chapter 6.4.1 "Loading Lists", page 108), even if the start has been unsuccessful. • **execute_list_pos** also covers the specialized variants **execute_list_1**, **execute_list_2**, **execute_list** and **execute_at_pointer**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**execute_list**, **execute_at_pointer**, **set_start_list_pos**

Normal List Command	**fly_disable_list**
Function	Deactivates the previously set Processing-on-the-fly correction and caches its settings (scaling factors and mode) for an Encoder-Speed-Dependent Laser Control.
Call	`fly_disable_list( )`
Parameters	–
Multi-board Com'd Name	**n_fly_disable_list**
Comments	• **fly_disable_list** does not execute a jump unlike **fly_return**. • The previously set Processing-on-the-fly correction is cached and used for Encoder-Speed-Dependent Laser Control (**set_auto_laser_control** with `Mode` = 2 extension M = +4, page 661). See also comment, page 662. • If no Processing-on-the-fly correction is currently active, a **fly_disable_list** call deletes the cached settings. • See also Chapter 8.6 "Processing-on-the-fly", page 251.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 644, OUT 647.
References	**fly_return**

Undelayed Short List Command	**fly_prediction**	
Function	Category: List Commands for a Restricted User Group only. Sets the Processing-on-the-fly total prediction values.	
Call	`fly_prediction( PredictionX, PredictionY )`	
Parameters	`PredictionX`	Processing-on-the-fly total prediction value for x. In [10/16 µs]. As an unsigned 32-bit value. Allowed value range: [0…4,095].
	`PredictionY`	Like `PredictionX` (analogously).
Multi-board Com'd Name	**n_fly_prediction**	
Comments	• See **set_fly_tracking_error**. • The following applies for List Commands for a Restricted User Group only: These users have been informed separately in how to handle these RTC6 commands. Therefore, only the command table is available in this manual and further information is omitted.	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 646, OUT 649, RBF 641.	
References	**set_fly_tracking_error**, **set_controlpreview_compensation_ctrl**	

Normal List Command	fly_return
Function	Deactivates the previously set Processing-on-the-fly correction and subsequently executes a jump to the defined new output position.
Restriction	If the Option Processing-on-the-fly is not enabled, **fly_return** only executes the jump to the specified new output position.
Call	`fly_return( X, Y )`
Parameters	`X` Absolute x coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
	`Y` Like `X` (analogously).
Multi-board Com'd Name	**n_fly_return**
Comments	• The jump to the new output position is executed as with **jump_abs** (see comments there). • If Processing-on-the-fly-correction has been activated within a subroutine called by an "AbsCall" and subsequently gets deactivated by **fly_return**, then the coordinate values specified with **fly_return** receive an offset (based on the current coordinates at the time of the call, see also Section ""AbsCalls"", page 117). • See also Chapter 8.6 "Processing-on-the-fly", page 251.
RTC4→RTC6	Unchanged functionality. In addition: increased value range, and "AbsCall", see above. In RTC4 Compatibility Mode, the RTC6 multiplies the values specified for `X` and `Y` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_fly_x**, **set_fly_y**, **set_fly_rot**, **set_fly_x_pos**, **set_fly_y_pos**, **set_fly_rot_pos**, **fly_disable_list**

Variable List Command	**fly_return_1_axis**
Function	"Fly Extension" Command: Deactivates 1 Axis of a Processing-on-the-fly application and subsequently carries-out a jump to the specified new output position.
Restriction	If the Option Processing-on-the-fly is not enabled, then **fly_return_1_axis** only carries-out the jump to the specified new output position.
Call	`fly_return_1_axis( Axis, RetPos1 )`
Parameters	`Axis` Axis from Table 3, page 268. As an unsigned 32-bit value. `RetPos1` Absolute axis coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_fly_return_1_axis**
Comments	• Being an "Fly Extension" Command, **fly_return_1_axis** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • `Axis` needs to be a linear axis (`1`, `2` or `3`). **fly_return_2_axes** is to be used to deactivate a Rotary axis. • **fly_return_1_axis** only deactivates 1 Axis of a Processing-on-the-fly application, if it has been activated by an "Fly Extension" Command. • If no Processing-on-the-fly functionality is active at the specified Axis, only the jump is carried out. • With an unallowed parameter value, **fly_return_1_axis** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • See also comments on **fly_return**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**fly_return_2_axes**, **fly_return_3_axes**

Variable List Command	fly_return_2_axes
Function	"Fly Extension" Command: Deactivates 2 Axes of a Processing-on-the-fly application and subsequently carries-out a jump to the specified new output position.
Restriction	If the Option Processing-on-the-fly is not enabled, then **fly_return_2_axes** only carries-out the jump to the specified new output position.
Call	`fly_return_2_axes( Axis1, RetPos1, Axis2, RetPos2)`
Parameters	`Axis1` Axis from Table 3, page 268. As an unsigned 32-bit value. `RetPos1` Absolute axis coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values. `Axis2` Like `Axis1`. `RetPos2` Like `RetPos1`.
Multi-board Com'd Name	**n_fly_return_2_axes**
Comments	• Being an "Fly Extension" Command, **fly_return_2_axes** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • `Axis1` and `Axis2` need to be 2 different linear axes (`1`, `2` or `3`) or both the Rotary axis (`4`). In the latter case `RetPos1`/`RetPos2` mean the return coordinates of the Axis `1`/`2`. • **fly_return_2_axes** only deactivates Axes of a Processing-on-the-fly application, if these have been activated by an "Fly Extension" Command. • If no Processing-on-the-fly functionality is active at one of the specified Axes, only the jump is carried out. • With an unallowed parameter value, **fly_return_2_axes** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • See also comments on **fly_return**. • The following command calls are executed in the same way: – `fly_return_2_axes( 1, X, 2, Y ) = fly_return( X, Y )` – `fly_return_2_axes( 4, X, 4, Y ) = fly_return( X, Y )`
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**fly_return_1_axis**, **fly_return_3_axes**

Variable List Command	**fly_return_3_axes**
Function	"Fly Extension" Command: Deactivates all 3 Axes of a Processing-on-the-fly application and subsequently carries-out a jump to the specified new output position.
Restriction	If the Option Processing-on-the-fly is not enabled, then **fly_return_3_axes** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`fly_return_3_axes( RetPosX, RetPosY, RetPosZ )`
Parameters	`RetPosX` Absolute axis coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values. `RetPosY` Like `RetPosX`. `RetPosZ` Like `RetPosX`.
Multi-board Com'd Name	**n_fly_return_3_axes**
Comments	• Being an "Fly Extension" Command, **fly_return_3_axes** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • **fly_return_3_axes** only deactivates Axes of a Processing-on-the-fly application, if these have been activated by an "Fly Extension" Command. • If no Processing-on-the-fly functionality is active at one of the specified Axes, only the jump is carried out. • With an unallowed parameter value, **fly_return_3_axes** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • See also comments on **fly_return**. • The following command calls are executed in the same way: – `fly_return_3_axes( X, Y, Z )` = `fly_return_z( X, Y, Z )`
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**fly_return_1_axis**, **fly_return_2_axes**

Normal List Command	fly_return_z	
Function	Deactivates the previously set Processing-on-the-fly correction. Subsequently executes a jump to the defined position.	
Restriction	If the Option Processing-on-the-fly is not enabled, **fly_return_z** only executes the jump to the defined new output position.	
Call	`fly_return_z( X, Y, Z )`	
Parameters	`X`	Absolute x coordinate of the new output position. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
	`Y`	Like `X` (analogously).
	`Z`	Like `X` (analogously).
Multi-board Com'd Name	**n_fly_return_z**	
Comments	• Like **fly_return**, however, with additional `Z` output position.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `X`, `Y` and `Z` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the specified value for `Z` by 16. The allowed value range decreases accordingly.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_fly_z**	

Ctrl Command	free_rtc6_dll
Function	Frees up all resources allocated by the RTC6 DLL for a user program.
Call	`free_rtc6_dll()`
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• RTC6 DLL-allocated resources particularly include memory area in the RTC6 DLL allocated for the RTC6 board management (which is created by **init_rtc6_dll**). **free_rtc6_dll** deletes the RTC6 board management of the RTC6 DLL. Afterward, the user program has no access to boards (**get_last_error** return code `RTC6_ACCESS_DENIED`). • The calling of **free_rtc6_dll** is not absolutely necessary, because RTC6 DLL-assigned resources are automatically freed up when the user program terminates and the RTC6 DLL is thereby unloaded by Microsoft Windows. However, some user program development environments (in debug mode) issue "memory leaks detected" warnings even though the RTC6 DLL is unloaded. The calling of **free_rtc6_dll** eliminates this annoyance.
RTC4→RTC6	New command.
RTC5→RTC6	New command. The functionalities of **free_rtc6_dll** and the RTC5 command **free_rtc5_dll** are identical.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**init_rtc6_dll**, **release_rtc**

Ctrl Command	get_auto_cal
Function	Only for some discontinued systems. Returns the type of ASC hardware integrated in the attached scan system previously detected by **auto_cal**.
Call	`ASCtype = get_auto_cal( HeadNo )`
Parameters	`HeadNo` Number of the scan head connector. As an unsigned 32-bit value. Allowed values: = 1: Connector for First Scan Head. = 2: Connector for Second Scan Head.
Result	ASC hardware type. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_auto_cal**
Comments	• If the ASC hardware type has been previously detected by **auto_cal**, then **get_auto_cal** returns the same value as **auto_cal**(`Command` = 4), see comments there. • If the ASC hardware type has *not* been previously detected by **auto_cal**, then **get_auto_cal** returns the value 255 (initialized value).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**auto_cal**

Ctrl Command	get_bios_version
Function	Returns the BIOS version number of the RTC6 Board.
Call	`BiosVersion = get_bios_version()`
Result	BCD-coded BIOS version number. As an unsigned 32-bit value. Example: `BiosVersion = 33` = 0x21 means BIOS version 21.
Parameters	None.
Multi-board Com'd Name	**n_get_bios_version**
Comments	• **get_bios_version** returns reliable results only as of BIOS version 21, otherwise 0.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**get_dll_version**, **get_hex_version**, **get_rtc_version**

Ctrl Command	get_card_type
Function	Returns the RTC6 board type.
Call	`Result = get_card_type()`
Result	Board type. As an unsigned 32-bit value. 0: "No card". 1: RTC6 PCIe Board. 2: RTC6 Ethernet Board.
Parameters	None.
Multi-board Com'd Name	**n_get_card_type**
Comments	• See also Chapter 16.5.3 "About the RTC6 Board Management", page 985.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 606, OUT 606, RBF 611.
References	**eth_assign_card**, **eth_assign_card_ip**

Ctrl Command	get_char_pointer
Function	Returns the absolute start address of an indexed character.
Call	`CharPointer = get_char_pointer( Char )`
Parameters	`Char` Index of the indexed character. As an unsigned 32-bit value. Allowed value range: [0…1023].
Result	Absolute start address. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_char_pointer**
Comments	• **get_char_pointer** reads from the internal management table the start address of the indexed character with the specified index. Whether the read address resides in a protected or the unprotected RTC6 List Memory area depends on whether the character has been loaded into the protected RTC6 List Memory area "List 3" or an unprotected subroutine has been only subsequently referenced. • If `Index` > 1023 or if no character has been referenced with the specified index, then **get_char_pointer** returns the value "–1" (for example, 2^{32}–1). • **get_char_pointer** is useful for checking if a character has already been defined or for calling an indexed character by an absolute memory address as if it were a non-indexed subroutine, for example, for conditional execution with **list_call_cond**. Be aware, though, that a subsequent **save_disk**/**load_disk** might alter the absolute memory address. And you should ensure that **get_char_pointer** does not return "–1"; otherwise **list_call_cond** is ignored.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_sub_pointer**, **get_text_table_pointer**

Ctrl Command	get_config_list
Function	Passes the parameters of the current RTC6 List Memory configuration (`Mem1, Mem2`) to the RTC6 board management of the RTC6 DLL and initializes it as if **config_list** would have been called.
Call	`get_config_list()`
Result	–
Multi-board Com'd Name	**n_get_config_list**
Comments	• The **get_config_list** command is useful when a board changes "ownership" and the new RTC6 board management is not aware of the memory configuration (at the start of each user program, the board and board management each independently initialize `Mem1 =` 4,194,304 and `Mem2 =` 4,194,304; the board by **load_program_file** and board management when starting the corresponding user program). See also Chapter 6.7.1 "Notes on Board Acquisition by a User Program", page 131. • **get_config_list** does not return a value to the user program. The user program can, however, read the RTC6 List Memory configuration data after `load_list( ListNo, 0 )` or `set_start_list_pos( ListNo, 0 )` by using **get_list_space**. • **get_config_list** is executed regardless of the BUSY list execution status.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**config_list**

Ctrl Command	get_counts
Function	Reads the current number of successful External Starts.
Call	`Counts = get_counts()`
Result	Number of successful External Starts. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_counts**
Comments	• The number is read from an internal counter, which is incremented each time a list is started by an external start signal. This counter can be reset to 0 by **set_control_mode**.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_max_counts**, **set_control_mode**, **get_startstop_info**

Ctrl Command	get_dll_version
Function	Returns the version number of the RTC6 DLL.
Call	`DLLVersion = get_dll_version()`
Result	Version number. As an unsigned 32-bit value.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• The RTC6 DLL version numbers are in the range 600…699. • **get_dll_version** is available even without explicit access rights to a specific RTC6 board. • The board-specific error variables `LastError` and `AccError`, see Chapter 6.8 "Error Handling", page 133, are neither generated nor altered by **get_dll_version**.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_hex_version**, **get_rtc_version**

Ctrl Command	get_encoder
Function	Returns the current counts of the two internal encoder counters.
Call	`get_encoder( &Encoder0, &Encoder1 )`
Returned Parameter Values	`Encoder0` Current count of encoder counter "Encoder0". As a pointer to a signed 32-bit value. `Encoder1` Current count of encoder counter "Encoder1". As a pointer to a signed 32-bit value.
Multi-board Com'd Name	**n_get_encoder**
Comments	• For usage of **get_encoder**, see Chapter 8.6 "Processing-on-the-fly", page 251 and Chapter 9.3.3 "Synchronization by Encoder Signals", page 320. • If the workpiece motion is registered with an incremental encoder: – Encoder counter "Encoder0" is triggered by the signals at encoder input port ENCODER X – Encoder counter "Encoder1" is triggered by the signals at encoder input port ENCODER Y • If an encoder simulation has been started by **simulate_encoder**, the encoder counters are triggered by an internal periodic 1 MHz clock signal.
RTC4→RTC6	Unchanged functionality. In addition: increased value range.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**store_encoder**, **read_encoder**, **set_fly_x**, **set_fly_y**, **set_fly_rot**, **wait_for_encoder**

Ctrl Command	get_error
Function	Returns the cumulative error code. It corresponds to a list of error types occurring since the last reset or error reset.
Call	`AccError = get_error()`
Result	Error code. As an unsigned 32-bit value. If multiple errors occurred, then multiple bits are set. For the specific errors the error constants should be predefined as mentioned below.

Bit	Error type	Error constant	
–	No error.	`RTC6_NO_ERROR`	= 0
Bit #0 (LSB)	= 1: No RTC6 PCIe Board found. This error can only occur with **init_rtc6_dll**.	`RTC6_NO_PCIE_CARD_FOUND`	= 1
Bit #1	= 1: Access denied. This error can occur by **init_rtc6_dll**, **select_rtc**, **acquire_rtc** or all multi-board commands.	`RTC6_ACCESS_DENIED`	= 2
Bit #2	= 1: Command not forwarded. This error implies an internal, RTC6 board driver error or PCI error, for example, caused by a hardware defect or an incorrect connection.	`RTC6_SEND_ERROR`	= 4
Bit #3	= 1: No response from board. It is likely that no program has been loaded onto the RTC6. This error can especially occur in connection with control commands that expect a response, for example, **get_head_para**.	`RTC6_TIMEOUT`	= 8
Bit #4	= 1: Invalid parameter. This error can occur through all commands for which invalid parameters are not automatically corrected to valid values, for example, parameters with limited choices such as **get_head_para**. If this error occurs for a list command, it is replaced by **list_nop**. If this error occurs for a control command, it is not executed.	`RTC6_PARAM_ERROR`	= 16

Ctrl Command	get_error			
Result (cont'd)	Bit #5	= 1: List processing is (not) active. Examples: for **execute_list**, if a list is currently being processed; for **stop_execution**, if no list is currently being processed; for **restart_list**, if **pause_list** has not been previously called.	`RTC6_BUSY`	= 32
	Bit #6	= 1: List command rejected, invalid input pointer. For example, for any list command directly after **load_char** + **list_return**: the list command is then not loaded.	`RTC6_REJECTED`	= 64
	Bit #7	= 1: List command has been converted to a **list_nop**. For example, **set_end_of_list** in a protected subroutine.	`RTC6_IGNORED`	= 128
	Bit #8	= 1: Version error: RTC6 DLL version, `RTC6RBF.rbf` version and `RTC6OUT.out`/`RTC6ETH.out` version are not compatible with each other. See also **load_program_file**.	`RTC6_VERSION_MISMATCH`	= 256
	Bit #9	= 1: Verify error: The download verification has detected an incorrect download. See also Chapter 6.8.1 "Download Verification", page 134.	`RTC6_VERIFY_ERROR`	= 512
	Bit #10	= 1: For example, an **[*]eth[*]** command has been sent to an RTC6 PCIe Board.	`RTC6_TYPE_REJECTED`	= 1024
	Bit #11	= 1: A RTC6 DLL-internal Windows memory request failed.	`RTC6_OUT_OF_MEMORY`	= 2048
	Bit #12	= 1: Download error. The values have possibly not been saved. May occur with **auto_cal** and **write_hi_pos**.	`RTC6_FLASH_ERROR`	= 4096

Ctrl Command	get_error			
Result (cont'd)	Bit #13	= 1: General Ethernet error. May occur, for example, when trying to switch by **select_rtc** or **acquire_rtc** from RTC6 PCIe Board to RTC6 Ethernet Board. Further information is provided by **n_eth_get_error** on the addressed RTC6 Ethernet Board.	`RTC6_ETH_ERROR`	= 8192
	Bit #14	Reserved.	–	
	Bit #15	= 1: Unsupported Windows version. May only occur during **init_rtc6_dll**.	`RTC6_UNSUPPORTED_WINDOWS`	= 32768
	Bit #16	Reserved.	–	
	Bit #17	= 1: Unsupported BIOS version.	`RTC6_UNSUPPORTED_BIOS`	= 131072
	Bit #18	Reserved.	–	
	…	…		
	Bit #30	Reserved.		
	Bit #31	= 1: Can occur exclusively in Remote Interface Mode: Invalid Command Telegram. See Section "Error Codes for `LastError` Occurring in Remote Interface Mode Only", page 1003.	`RTC6_REMOTE_ERROR`	= 2147483648
Multi-board Com'd Name	**n_get_error**			
Comments	• For error handling see Chapter 6.8 "Error Handling", page 133. • **get_error** and **n_get_error** are available even without explicit access rights to a specific RTC6 board. • The board-specific error variables `LastError` and `AccError` (see Chapter 6.8 "Error Handling", page 133) are neither generated nor altered by **get_error**. • Bit #0 = 1 (error constant: `RTC6_NO_PCIE_CARD_FOUND = 1`) means that no RTC6 PCIe Board has been found (**init_rtc6_dll** does not search for RTC6 Ethernet Boards).			

Ctrl Command	get_error

Example (C/C++)	Creates an array for specifying which board has no existing access rights and resets the cumulative error code.
	`UINT NoAccess[MaxCount+1]; // MaxCount is a user-defined constant`
	`UINT Error = init_rtc6_dll();// Searches for all installed RTC6 boards`
	`if (Error & RTC6_ACCESS_DENIED) { // at least one board is inaccessible`
	`   UINT Count = rtc6_count_cards(); // number of boards found`
	`   for ( UINT Num = 1; Num <= Count; Num++ ) {`
	`      NoAccess[Num] = n_get_last_error(Num) & RTC6_ACCESS_DENIED;`
	`      n_reset_error(Num, RTC6_ACCESS_DENIED);`
	`   }`
	`}`

RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_last_error**, **reset_error**, **set_verify**

Ctrl Command	get_fly_2d_offset
Function	Returns the current reference values (offset values) for 2D encoder compensation.
Call	`get_fly_2d_offset( &OffsetX, &OffsetY )`
Returned Parameter Values	`OffsetX` x reference value. As a pointer to a signed 32-bit value. `OffsetY` y reference value. As a pointer to a signed 32-bit value.
Multi-board Com'd Name	**n_get_fly_2d_offset**
Comments	• For 2D encoder compensation, see Section "2D Encoder Compensation for xy Positioning Stages", page 258.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**init_fly_2d**, **set_fly_2d**

Ctrl Command	get_free_variable
Function	Returns the current value of a free variable.
Call	`VariableValue = get_free_variable( No )`
Parameters	`No` Number of the free variable to be queried. As an unsigned 32-bit value. Allowed value range: [0…7]. Only the 3 least significant bits are evaluated.
Result	The value currently stored in the free variable `No`. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_free_variable**
Comments	• See also Chapter 6.9.1 "Free Variables", page 138.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_free_variable**, **set_free_variable_list**

Ctrl Command	get_galvo_controls
Function	Returns the corresponding control values for given input values.
Restriction	**get_galvo_controls** can only be executed, if no list is currently being processed (**get_last_error** return code `RTC6_BUSY`).
Call	`get_galvo_controls( InPtr, OutPtr )`
Parameters	`InPtr` Pointer (data type ULONG_PTR in C and C++, an unsigned 32-bit or unsigned 64-bit value) to an array of five unsigned 32-bit values, where the to-be-outputted settings are specified: X, Y, Z, Defocus, Zoom. Out-of-range values are clipped to the boundary values. `OutPtr` Pointer (data type ULONG_PTR in C and C++, an unsigned 32-bit or unsigned 64-bit value) to an array of 4 unsigned 32-bit values, where the corresponding control values are to be stored: XA, YA, XB, YB. Value range in each case [–524,288…+524,287].
Returned Parameter Values	XA, YA, XB, YB denote the control values for the x axis/y axis of scan head A/B.
Multi-board Com'd Name	**n_get_galvo_controls**
Comments	• **get_galvo_controls** carries-out a virtual jump to the specified coordinates. Though the galvanometer scanners are *not* moved. • All other settings which are not specified within **get_galvo_controls** (for example, matrix, offset, angle, etc.; also stretch table) are considered as they are set at the moment. Make sure to apply these by `at_once > 0` before calling **get_galvo_controls**!. The settings are not used, if they just only have been saved by `at_once = 0`. • Control values are calculated only for channels where a correction table has been previously assigned by **select_cor_table**, see the following examples. `select_cor_table(0,0)`: XA = YA = XB = YB = 0 2D correction files: inputs Z, Defocus, Zoom are ignored `select_cor_table(1,0)`: XA, YA calculated, XB, YB = 0 `select_cor_table(0,1)`: XA, YA = 0, XB, YB calculated `select_cor_table(1,1)`: XA, YA, XB, YB calculated (Standard-)3D correction file: input Zoom is ignored `select_cor_table(1,0)`: XA, YA calculated, XB = YB = Zout calculated `select_cor_table(0,1)`: XA= YA = Zout calculated, XB, YB calculated `select_cor_table(1,1)`: same as `select_cor_table(0,0)` 3D zoom correction file (only for intelli*WELD* II with zoom axis): `select_cor_table(1,0)`: XA, YA calculated, XB = Zout, YB = ZoomOut calculated `select_cor_table(0,1)`: XA= Zout, YA = ZoomOut calculated, XB, YB calculated • The return values are 0, if a **get_last_error** return code `RTC6_BUSY` has been generated.

Ctrl Command	get_galvo_controls
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for x, y, z and Defocus by 16. The allowed value range decreases accordingly. Even in RTC4 Compatibility Mode, all returned values are in the RTC6 20-bit range.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the specified values for z and Defocus by 16. The allowed value range decreases accordingly. Even in RTC5 Compatibility Mode, all returned values are in the RTC6 20-bit range.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**select_cor_table**

Ctrl Command	get_head_para
Function	Returns the value of the requested parameter in the correction table assigned to the specified scan head.
Call	`HeadPara = get_head_para( HeadNo, ParaNo )`
Parameters	`HeadNo` Number of the scan head connector. As an unsigned 32-bit value. Allowed values: = 1: Connector for First Scan Head. = 2: Connector for Second Scan Head.
	`ParaNo` Number of the parameter. As an unsigned 32-bit value. Allowed values: 0…15. Mapping: see Section "ct5 Correction File Header", page 183.
Result	Parameter value, see Section "ct5 Correction File Header", page 183. As a 64-bit IEEE floating point value.
Multi-board Com'd Name	**n_get_head_para**
Comments	• The parameter values can be read out by **get_table_para** from a currently loaded correction table and by **get_head_para** from an assigned correction table and thus directly incorporated into a user program, see Section "ct5 Correction File Header", page 183. • If the parameters `HeadNo` and `ParaNo` are out of range, then the return value is 0 (**get_last_error** return code `RTC6_PARAM_ERROR`). The return value is also 0 (no **get_last_error** return code) if no correction table has been assigned to the specified head (for example, for `HeadNo` = 2 if the Option "Second Scan Head Control" has not been enabled) and no 3D correction table has been assigned to the other head. • If a 3D correction table has been assigned to a head, then this 3D correction table's parameter is returned regardless of `HeadNo` (two 3D correction tables cannot be simultaneously assigned). `HeadNo` must nevertheless be 1 or 2 (see preceding comment).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_table_para**

Ctrl Command	get_head_status
Function	Returns the XY2-100 status word from the specified scan head connector.
Call	`get_head_status( Head )`
Parameters	`Head` = 1: Returns the status of Connector for First Scan Head (Byte #1 = Byte #0). = 2: Returns the status of the Connector for Second Scan Head (Byte #1 = Byte #0). Else: Returns the status of Connector for First Scan Head (Byte #0) *and* Connector for Second Scan Head (Byte #1).
Result	XY2-100 status word. As an unsigned 32-bit value. Byte #0 (LSB) — Bit #0 (LSB): 1. Bit #1: 0. Bit #2: 1 (reserved). Bit #3: Position Acknowledge of x axis, 1 = OK. Bit #4: Position Acknowledge of y axis, 1 = OK. Bit #5: 1 (reserved). Bit #6: Temperature Status, 1 = OK. See comment, page 440. Bit #7: Power Status, 1 = OK. See comment, page 440. Byte #1 — Bit #8 … Bit #15: Bit assignments as with Byte #0. Byte #2 … Byte #3 — Bit #16 … Bit #31: 0.
Multi-board Com'd Name	**n_get_head_status**
Comments	• **get_head_status** is available even with the SL2-100 protocol, if the data type is not set to the 20-bit status word itself, see Section "Status Information Returned from the Scan System", page 188. • The status bits are returned by **get_head_status** in Bit #3…Bit #7 and/or Bit #11…Bit #15. Independently of the scan system's current state, Bit #0 and Bit #8 are returned by **get_head_status** as 1, while Bit #1 and Bit #9 are returned as 0. • If no scan system is currently connected or is not switched on, then the value 0 is returned. • The PowerOK signal is an electronically generated signal. It means: The scan system servo control is ready for input data ("PowerOK", actually corresponds to a "ServoOK"). Therefore, it cannot be used to check whether a connected scan head is switched on at all. **get_startstop_info** (Bit #17 and/or Bit #25) can be used for distinguishing.

Ctrl Command	get_head_status
Comments (cont'd)	• The Power Status and Temperature Status signals deliver combined status information of both axes. • In any case also obey the status signal information described in the manual of your scan system. • With iDRIVE Scan Systems – Without the SL2-100 interface, **get_head_status** only returns meaningful return values, if the XY2-100 status word has been selected for return transmission. – the Position Acknowledge signals of the x- and y axis are logically AND-connected and only returned as a common signal (for example, at Bit #3,4). – after a reset or power-up of the scan system, it can take around 5 seconds for data to be returned from the scan system. • Status signals can also be read out by: – **get_value** – **get_values** – **set_trigger[*]** • See also Chapter 8.5 "Controlling 2D Scan Systems and 3D Scan Systems", page 245 for information about using two *scan heads*.
RTC4→RTC6	Basically unchanged functionality. However: • The RTC6 allows the simultaneous reading of both scan head connectors' status words, and with iDRIVE Scan Systems with SL2-100 interface even independently of the signal to be returned (set by **control_command**). • With iDRIVE Scan Systems, Bit #0 and Bit #1 do not return information about the operational readiness of the scan system (see **get_value**).
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_value**, **get_values**, **set_trigger**, **set_trigger4**, **set_trigger8**, **get_waveform**

Ctrl Command	**get_hex_version**
Function	Returns the version number of the DSP program file `RTC6OUT.out`, which is currently loaded on the RTC6.
Call	`HexVersion = get_hex_version()`
Result	Version number. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_hex_version**
Comments	• The version numbers of program files are in the range 600…699. **get_hex_version** returns the following values: – if the Option "3D" is *not* enabled values in the range 2600…2699 (version number + 2000) – if the Option "3D" is enabled values in the range 3600…3699 (version number + 3000) • The file name extension for RTC6 DSP program files is `*.out`. See also **load_program_file**. • The software version number can also be returned after an `RTC6_VERSION_MISMATCH` or `RTC6_ACCESS_DENIED` error. The return value is 0 if no program has yet been loaded. • The board-specific error variables `LastError` and `AccError`, see Chapter 6.8 "Error Handling", page 133, are neither generated nor altered by **get_hex_version**.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_dll_version**, **get_rtc_version**

Ctrl Command	get_hi_data	
Function	Only for some discontinued systems. Returns the Home-In positions, last determined by **auto_cal** of the scan system attached to the Connector for First Scan Head.	
Call	`get_hi_data( &X1, &X2, &Y1, &Y2)`	
Returned Parameter Values	`X1`	x1 coordinate of the currently stored (most recently measured) Home-In positions. In bits. As a pointer to a signed 32-bit value.
	`X2`	Like `X1` (analogously).
	`Y1`	Like `X1` (analogously).
	`Y2`	Like `X1` (analogously).
Multi-board Com'd Name	**n_get_hi_data**	
Comments	• **get_hi_data** is synonymous with **get_hi_pos** with `HeadNo` = 1 (see comments there).	
RTC4→RTC6	Unchanged functionality. In addition: increased value range. The returned values are in the 20-bit value range.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**get_hi_pos**, **write_hi_pos**	

Ctrl Command	get_hi_pos	
Function	Only for some discontinued systems. Returns the Home-In positions, last determined (by **auto_cal**) of the scan system attached to the specified scan head connector.	
Call	`get_hi_pos( HeadNo, &X1, &X2, &Y1, &Y2 )`	
Parameters	`HeadNo`	Number of the scan head connector. As an unsigned 32-bit value. Allowed values. = 1: Connector for First Scan Head. = 2: Connector for Second Scan Head.
Returned Parameter Values	`X1`	x1 coordinate of the currently stored (most recently measured) Home-In positions. In bits. As a pointer to a signed 32-bit value.
	`X2`	Like `X1` (analogously).
	`Y1`	Like `X1` (analogously).
	`Y2`	Like `X1` (analogously).
Multi-board Com'd Name	**n_get_hi_pos**	
Comments	• For information on using **get_hi_pos**, see Section "Customer-Specific Calibration", page 287. • Make sure that the scan system currently attached to the specified scan head connector is the same scan system which has been used to determine the returned Home-In positions. For determination of Home-In position values, this scan system should be equipped with an internal sensor system for automatic self-calibration (Home-In sensors). • The returned values are 0, if: – no scan system equipped with automatic self-calibration (Home-In sensors) is attached to the specified scan head connector – an error has occurred during determination of the Home-In values for such a system • Directly after initialization (**init_rtc6_dll**), particularly prior to a first call of **auto_cal**(`Command` = 0, 1 or 3), the returned values are the Home-In reference values stored in the Flash Memory. If such reference values have not been successfully determined at least once by **auto_cal**(`Command` = 0), **get_hi_pos** returns 0. • **get_hi_pos** is available even without (current) explicit access rights to a specific RTC6 board. However, the return values are 0 as long as no access to the addressed board has been successful at least once before. • If parameter values are invalid, then all returned coordinates are 0 (**get_last_error** return code `RTC6_PARAM_ERROR`).	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**get_hi_data**, **auto_cal**, **set_hi**, **write_hi_pos**	

Ctrl Command	get_input_pointer
Function	Returns the present absolute position of the input pointer.
Call	`InputPointer = get_input_pointer()`
Result	Position of the input pointer [0…(2^{23}–1)]. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_input_pointer**
Comments	• The position of the input pointer corresponds to the position in RTC6 List Memory (also in the protected "List 3" area), where the next list command is stored. The number of still-available storage positions there can be queried by **get_list_space**. • **get_input_pointer** returns the absolute RTC6 List Memory address (offset relative to the start of "List 1"). The relative position referenced to the start of the respective list area can be queried by **get_list_pointer**. • Before loading a non-indexed subroutine or character set, you should use **get_input_pointer** to obtain the start address if subsequent referencing is to be performed by **set_sub_pointer** or **set_char_pointer**. • The absolute position of the output pointer can be queried by **get_status** or **get_out_pointer**. • The board-specific error variables `LastError` and `AccError`, see Section "Error Handling", page 133, are neither generated nor altered by **get_input_pointer**.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_list_pointer**, **set_input_pointer**, **get_list_space**, **get_status**, **get_out_pointer**

Ctrl Command	get_io_status
Function	Returns the current state of the 16-Bit Digital Output Port on the EXTENSION 1 Socket Connector.
Call	`IOStatus = get_io_status()`
Result	16-bit value (DIGITAL OUT0…DIGITAL OUT15). As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_io_status**
Comments	• **get_io_status** is designed for use in combination with **set_io_cond_list** and **clear_io_cond_list**. See also Section "Example Code (Pascal)", page 318. • See also Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_io_port**, **write_io_port_mask**, **set_io_cond_list**, **clear_io_cond_list**

Ctrl Command	get_jump_table
Function	Reads out the Jump Delay table which is currently stored on the board. Then copies the 1024 corresponding unsigned 16-bit values to the specified PC address.
Call	`ErrorCode = get_jump_table( Addr )`
Parameters	`Addr` PC address for the 2048 byte memory area.
Result	Error code. As an unsigned 32-bit value. 0 No error. 11 RTC6 board driver error.
Multi-board Com'd Name	**n_get_jump_table**
Comments	• Do not call **get_jump_table** during processing of a list. • The data format is "1024 16-bit values" representing the delay values for a piecewise linear interpolation at the sampling points (=jump lengths) N × 1024 with 0 ≤ N < 1024. The values can thus also be directly generated or modified by users.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
Reference	**set_jump_table**, **load_jump_table_offset**

Ctrl Command	get_lap_time
Function	Returns the *current* RTC6 Timer value (without resetting it to zero).
Call	`TimerValue = get_lap_time()`
Result	RTC6 Timer value in seconds since the last call of **save_and_restart_timer**. In seconds. As a 64-bit IEEE floating point value.
Multi-board Com'd Name	**n_get_lap_time**
Comments	• **get_lap_time** serves to query the elapsed time of a time consuming marking during processing.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_time**, **save_and_restart_timer**

Ctrl Command	**get_laser_pin_in**
Function	Returns the current status of the 2-Bit Digital Input Port at the LASER Connector, see also Section "2-Bit Digital Input Port", page 77.
Call	`LaserPinIn = get_laser_pin_in()`
Result	As an unsigned 32-bit value. Bit #0 (LSB) DIGITAL IN1. Bit #1 DIGITAL IN2. Bit #2 Reserved. … … Bit 31 Reserved.
Multi-board Com'd Name	**n_get_laser_pin_in**
Comments	• –
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_laser_pin_out**

Ctrl Command	**get_last_error**
Function	Returns an error code listing any errors which occurred during execution of the most recent command.
Call	`LastError = get_last_error()`
Result	Error code. As an unsigned 32-bit value. If multiple errors occurred simultaneously, then multiple bits are set. The meanings of bit numbers, error types and error constants is identical to those for **get_error**.
Multi-board Com'd Name	**n_get_last_error**
Comments	• For error handling see Chapter 6.8 "Error Handling", page 133. • **get_last_error** and **n_get_last_error** are available even without explicit access rights to a specific RTC6 board. • The board-specific error variables `LastError` and `AccError`, see Chapter 6.8 "Error Handling", page 133, are neither generated nor altered by **get_last_error**. • Each **eth_get_last_error** error also leads to a **get_last_error** error `RTC6_ETH_ERROR`.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**eth_get_last_error**, **get_error**, **reset_error**, **set_verify**

Ctrl Command	get_list_pointer
Function	Returns the current relative position of the input pointer as well as the list number.
Call	`get_list_pointer( &ListNo, &Pos )`
Returned Parameter Values	`ListNo` Number of the list in which the input pointer is currently located. As a pointer to an unsigned 32-bit value. [1…3].
	`Pos` Current position of the input pointer (offset relative to the start of the respective list). As a pointer to an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_list_pointer**
Comments	• The absolute RTC6 List Memory address (offset relative to the start of "List 1") of the input pointer can be queried by **get_input_pointer** (see also comments there). • The number of list positions until the end of the respective list (from the input pointer) can be queried by **get_list_space**. • The board-specific error variables `LastError` and `AccError`, see Chapter 6.8 "Error Handling", page 133, are neither generated nor altered by **get_list_pointer**. • If the input pointer is invalid when **get_list_pointer** is called (for example, after a **list_return**), the return values are `ListNo` = 0 and `Pos` = 0xFFFFFF.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_input_pointer**, **get_list_space**

Ctrl Command	get_list_serial
Function	Returns the number of the serial-number-set most recently selected by **select_serial_set_list** (or of serial-number-set 0 after **load_program_file**) and the current serial number of this serial-number-set.
Call	`LastMarkedSerialNo = get_list_serial( &Set )`
Result	Serial number. As a 64-bit IEEE floating point value.
Returned Parameter Values	`Set` Number of the selected serial-number-set. As a pointer to an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_list_serial**
Comments	• The serial number queried by **get_list_serial** is typically the one most recently marked by **mark_serial** or **mark_serial_abs**. • For usage of **get_list_serial**, see Chapter 7.5.2 "Marking Serial Numbers", page 219.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**select_serial_set_list**, **get_serial**

Ctrl Command	get_list_space
Function	Returns the amount of free RTC6 List Memory, hence the number of list commands that can still be loaded from the input pointer's current position to the last position in the respective list.
Call	`ListSpace = get_list_space()`
Result	Number of free list positions. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_list_space**
Comments	• If an indexed subroutine or indexed character set is currently being loaded into the protected RTC6 List Memory area "List 3", then **get_list_space** returns the amount of still-available protected memory (otherwise the input pointer is not located in the protected area).
RTC4→RTC6	**get_list_space** has been made available on the RTC4 to support the RTC4-Circular Queue Mode and returns the distance between the input pointer and output pointer. The RTC6 does not support the RTC4-Circular Queue Mode. The input pointer position can be queried by **get_input_pointer** or **get_list_pointer** and the output pointer position can be queried by **get_status** or **get_out_pointer**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_input_pointer**, **get_status**, **get_out_pointer**, **get_list_pointer**

Ctrl Command	get_marking_info
Function	Returns information about any boundary exceedances during Processing-on-the-fly correction as well as improper encoder signals. **get_marking_info** also returns the error bits of automatic suppression of Laser Control Signals.
Call	`MarkingInfo = get_marking_info()`
Result	Error code. As an unsigned 32-bit value. Bit #0 (LSB) = 1: Processing-on-the-fly underflow in x direction (X < –524,288). Bit #1 = 1: Processing-on-the-fly overflow in x direction (X > +524,287). Bit #2 = 1: Processing-on-the-fly underflow in y direction (Y < –524,288). Bit #3 = 1: Processing-on-the-fly overflow in y direction (Y > +524,287). Bit #4 = 1: Processing-on-the-fly underflow in x direction (X < Xmin). Bit #5 = 1: Processing-on-the-fly overflow in x direction (X > Xmax). Bit #6 = 1: Processing-on-the-fly underflow in y direction (Y < Ymin). Bit #7 = 1: Processing-on-the-fly overflow in y direction (Y > Ymax). Bit #8 = 1: TriggerError: an enabled external trigger or simulated trigger occurred during execution of a list. Bit #9 = 1: An error has occurred during activation of Processing-on-the-fly correction by **activate_fly_2d**, **activate_fly_2d_encoder**, **activate_fly_xy** or **activate_fly_xy_encoder** ("ActivateFlyError"). See also Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 261, Chapter 8.6.8 "Encoder Resets", page 263 and Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 264. Bit #10 PosAck error bit of scan system A, x axis. Bit #11 TempOK error bit of scan system A, x axis. Bit #12 PowerOK error bit of scan system A, x axis. Bit #13 PosAck error bit of scan system A, y axis. Bit #14 TempOK error bit of scan system A, y axis. Bit #15 PowerOK error bit of scan system A, y axis. Bit #16 = 1: The distance between the edges in signal 1 of encoder input port ENCODER X is too short. Bit #17 = 1: The distance between the edges in signal 1 of encoder input port ENCODER Y is too short. Bit #18 = 1: The distance between the edges in signal 2 of encoder input port ENCODER X is too short. Bit #19 = 1: The distance between the edges in signal 2 of encoder input port ENCODER Y is too short.

Ctrl Command	get_marking_info
Result (cont'd)	Bit #20 = 1: Improper signal sequence at encoder input port ENCODER X. Bit #21 = 1: Improper signal sequence at encoder input port ENCODER Y. Bit #25 = 1: Processing-on-the-fly overflow in z direction (Z > Zmax). Bit #22 = 1: Processing-on-the-fly underflow in z direction (Z < –524,288). Bit #23 = 1: Processing-on-the-fly overflow in z direction (Z > +524,287). Bit #24 = 1: Processing-on-the-fly underflow in z direction (Z < Zmin). Bit #26 PosAck error bit of scan system B, x axis. Bit #27 TempOK error bit of scan system B, x axis. Bit #28 PowerOK error bit of scan system B, x axis. Bit #29 PosAck error bit of scan system B, y axis. Bit #30 TempOK error bit of scan system B, y axis. Bit #31 PowerOK error bit of scan system B, y axis.
Multi-board Com'd Name	**n_get_marking_info**
Comments	• For usage of **get_marking_info** and of the error bits Bit #0…Bit #7, see Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 264. • The limits for the customer-defined monitoring range are determined for: – Xmin, Xmax, Ymin, Ymax (Bit #4…Bit #7) by **set_fly_limits** – Zmin, Zmax (Bit #24…Bit #25) by **set_fly_limits_z** • The error bits Bit #4…Bit #7 and Bit #24…Bit #25: – Are *not* reset by **get_marking_info** – Get implicitly reset by the conditional commands – Can also be explicitly reset by **clear_fly_overflow** and **clear_fly_overflow_ctrl** • Encoder-signal spacing could be too short if interfering signals are present, a rapid directional change occurs or the frequency is essentially too high. • An improper encoder signal sequence occurs if both signals 1 and 2 change simultaneously, thus hindering determination of the counting direction. • The error bits Bit #10…Bit #15 and Bit #26…Bit #31 are only set in case of an error if automatic suppression of Laser Control Signals has been activated, see Section "Automatic Suppression of Laser Control Signals", page 192. Any time an error occurs, all error bits corresponding to an error-indicating status signal are (cumulatively) set. If applicable, even error bits are set, which have not been selected to be used for automatic suppression of Laser Control Signals by **set_laser_control**. All error bits are reset by **get_marking_info**.

Ctrl Command	get_marking_info
Comments (cont'd)	• All error bits are reset during initialization (by **load_program_file**). • All error bits (except Bit #4…Bit #7 and Bit #24…Bit #25): – Are reset when reading out by **get_marking_info** – However, can be reset at any time as long as the error condition still persists
RTC4→RTC6	Unchanged functionality for info bits that are also used on the RTC4.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_fly_x**, **set_fly_y**, **set_fly_z**, **set_fly_rot**, **set_fly_x_pos**, **set_fly_y_pos**, **set_fly_rot_pos**, **set_fly_limits**, **set_fly_limits_z**, **clear_fly_overflow**, **clear_fly_overflow_ctrl**, **if_not_activated**

Ctrl Command	get_master_slave
Function	Returns the master/slave status of the addressed RTC6 board.
Call	`MasterSlaveStatus = get_master_slave()`
Result	Master/slave status. As an unsigned 32-bit value. Information whether a further RTC6 board is connected to the Master or Slave connector of the addressed RTC6 board: Bit #0 (LSB) = 1: A RTC6 board is connected to the Slave connector. Bit #1 = 1: A RTC6 board is connected to the Master connector. Bit #2 = 0. … … Bit #31 = 0. Information, whether the addressed board is operated as a master, slave or single board: = 0 Single board. = 1 Slave without any further downstream slave board. = 2 Master. = 3 Slave together with a downstream slave board.
Multi-board Com'd Name	**n_get_master_slave**
Comments	• See Chapter 6.6.3 "Master/Slave Operation", page 127. • **get_master_slave** only returns the status, if the addressed RTC6 board has been previously initialized by **load_program_file**. Otherwise, 0 is returned.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**sync_slaves**

Ctrl Command	get_mcbsp
Function	Returns the most recent input value that has been fully transferred by the McBSP interface to the memory location for Processing-on-the-fly applications.
Call	`mcbsp_value = get_mcbsp()`
Result	Input value. As a signed 32-bit value.
Multi-board Com'd Name	**n_get_mcbsp**
Comments	• **get_mcbsp** is equivalent to **read_mcbsp**(0) (see notes there).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**read_mcbsp**

Undelayed Short List Command	get_mcbsp_list
Function	No function.
Call	`get_mcbsp_list()`
Multi-board Com'd Name	**n_get_mcbsp_list**
Comments	• **get_mcbsp_list** has no effect on the user program.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_mcbsp**

Ctrl Command	get_out_pointer	
Function	Returns the current (or most recent) position of the output pointer as offset relative to the start of the respective list and the list number.	
Call	`get_out_pointer( &ListNo, &Pos )`	
Returned Parameter Values	`ListNo`	Number of the list ("List 1" or "List 2") in which the output pointer is currently located (or in which it most recently resided). As a pointer to an unsigned 32-bit value.
	`Pos`	Current (or most recent) position of the output pointer (relative memory address). As a pointer to an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_out_pointer**	
Comments	• **get_out_pointer** calls **get_status** (see the comments there, particularly with respect to "List 3") and uses the command's returned absolute output pointer position to determine the list number and the relative position within the list. The relative position and list number returned by **get_out_pointer** simplify comparing the output pointer position to the current input pointer position (particularly with respect to list consistency) during an alternative list change.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**get_status**, **get_input_pointer**, **get_list_pointer**	

Ctrl Command	get_overrun
Function	Returns the number of overruns of the 10 µs clock cycle since the last call and resets the overrun counter.
Call	`NumberOfOverruns = get_overrun()`
Result	Number of overruns of the 10 µs clock cycle since the last call of **get_overrun**. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_overrun**
Comments	• See Section "Clock Overruns", page 187. • See also Sky Writing Mode 4.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Ctrl Command	get_rtc_mode
Function	Returns the currently set operation mode of the RTC6 DLL.
Call	`DLLMode = get_rtc_mode()`
Result	DLL operation mode as a 32-bit value. = 4: RTC4 Compatibility Mode. = 5: RTC5 Compatibility Mode. = 6: RTC6 Standard Mode (default setting).
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• The RTC6 DLL operation mode can be set by **set_rtc4_mode**, **set_rtc5_mode** or **set_rtc6_mode**. The default setting is RTC6 Standard Mode. • **get_rtc_mode** is available even without explicit access rights to a particular RTC6 board. • The board-specific error variables `LastError` and `AccError`, see Chapter 6.8 "Error Handling", page 133, are neither generated nor altered by **get_rtc_mode**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_rtc4_mode**, **set_rtc5_mode**, **set_rtc6_mode**

Ctrl Command	get_rtc_version
Function	Returns the version number of the FPGA firmware (`RTC6RBF.rbf`) and information about enabled options of the RTC6 board.
Call	`RTCVersion = get_rtc_version()`
Result	Version number of the FPGA firmware and additional information. As an unsigned 32-bit value. Bit #0 (LSB) Version number of the FPGA firmware (`RTC6RBF.rbf`). … Bit #7 Bit #8 = 1: Option Processing-on-the-fly is enabled. See also Chapter 8.6 "Processing-on-the-fly", page 251. Bit #9 = 1: Option "Second Scan Head Control" is enabled. See also Section "2. SCANHEAD Socket Connector", page 70. Bit #10 = 1: Option "3D" is enabled. See also Chapter 8.5.2 "3D Scan Systems", page 246. Bit #11 = 1: Option "LDSA" is enabled (as of DLL 628). Bit #12 = 1: Option "SCANa" is enabled (as of DLL 605). Bit #13 = 1: Option "UFPM" is enabled (as of DLL 605). Bit #14 = 1: Option "syncA" is enabled (as of DLL 607). Bit #15 Reserved. Bit #16 DSP version number. … Bit #23 Bit #24 Subversion number of the FPGA firmware (`RTC6RBF.rbf`). … Bit #31
Multi-board Com'd Name	**n_get_rtc_version**
Comments	• The FPGA firmware version numbers are in the range 600…699. • **get_rtc_version**`( Bit #0…Bit #7 )` returns values in the range 0…99 (version number – 600). • The current DSP version number is 3. • The current FPGA firmware subversion is 0. • The FPGA firmware version can even be returned after an `RTC6_VERSION_MISMATCH` or `RTC6_ACCESS_DENIED` error. The return value is 0, if no program has yet been loaded.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_hex_version**, **get_dll_version**

Ctrl Command	get_scanahead_params
Function	Only for SCANahead Systems. Queries certain parameters from the specified SCANahead System.
Restriction	**get_scanahead_params** can be executed only, if no list is currently being processed. Otherwise, the **get_last_error** return code gets set to RTC6_BUSY and error code 5 is returned.
Call	`Error = get_scanahead_params( HeadNo, &PreviewTime, &Vmax, &Amax )`
Parameters	`HeadNo` Scan head connector number. As an unsigned 32-bit value. Allowed values: = 1: Connector for First Scan Head. = 2: Connector for Second Scan Head. Option "Second Scan Head Control" required. = 256: RTC6 board (no scan head connector). `Vmax` and `Amax` are returned converted to Image Field coordinates.
Returned Parameter Values	`PreviewTime` Precalculation time for galvanometer scanner control. In 10 µs. As a pointer to an unsigned 32-bit value. `Vmax` For `HeadNo` 1, 2: Velocity limit as control bits (galvanometer scanner rotation angle). In bits/10 µs. For `HeadNo` 256: Velocity limit as programming bits (Image Field coordinates). In bits/10 µs. As a pointer to an unsigned 32-bit value. `Amax` For `HeadNo` 1, 2: Acceleration limit as control bits (galvanometer scanner rotation angle). In bits/(10 µs)². For `HeadNo` 256: Acceleration limit as programming bits (Image Field coordinates). In bits/(10 µs)². As a pointer to a 64-bit IEEE floating point value.
Result	Error code. As an unsigned 32-bit value. 3 No SCANahead System is attached or its SCANahead System tuning is not active (**get_last_error** return code RTC6_PARAM_ERROR). 5 A list is currently active (**get_last_error** return code: RTC6_BUSY). 6 `HeadNo = 0` or `HeadNo` >2 has been specified as parameter (**get_last_error** return code: RTC6_PARAM_ERROR). 8 The RTC6 board is not responding. Probably a program has not been loaded yet (**get_last_error** return code: RTC6_TIMEOUT). 11 A PCI error occurred (**get_last_error** return code: RTC6_SEND_ERROR).
Multi-board Com'd Name	**n_get_scanahead_params**

Ctrl Command	get_scanahead_params
Comments	• The returned parameter values are also used by **set_scanahead_params**. • `Vmax` is the maximum angular velocity of the galvanometers. `Vmax` *is not* the mark speed in the image plane. For this reason, the unit is "In bits/10 µs." and *not* "In Bits/ms.". Distortion by the correction file is not yet taken into account in the returned parameter value (see **set_scanahead_params**). • To convert maximum speed `Vmax` from programming bits/clock cycle to m/s: (`Vmax` [bits/clock cycle] / K [bits/mm]) × ((0.001 [m/mm] / 0.00001 [s/clock cycle]) • To convert maximum acceleration `Amax` from programming bits/clock cycle² to m/s²: (`Amax` [bits/clock cycle²] / K [bits/mm]) × ((0.001 [m/mm] / (0.00001)² [s²/clock cycle²]) • `HeadNo` = `256` only returns meaningful values if **set_scanahead_params** was previously called. • The values returned by `HeadNo` = `256` are scaled. Do not use them with `set_scanahead_params( Mode = 2 )` because then they would be scaled again. • For dual SCANahead Systems (for example, 2 excelli*SCAN* scan heads; requires Option "Second Scan Head Control") it is recommended that the single SCANahead Systems used have as similar properties as possible (aperture, lens, correction file): For automatic calculation of the Scanner Delays and Laser Delays, among other things, `Amax` of SCANahead System `HeadNo` is required. The delays are only calculated for this SCANahead System, but are also used unchanged for the second one. If the two SCANahead Systems differ too much for example, in `Amax`, unwanted deviations may occur between the markings.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of version DLL 600, OUT 600. Last change with version DLL 605: data type of `Amax`.
References	**set_scanahead_params**, **get_last_error**

Ctrl Command	get_serial
Function	Returns the current serial number of the serial-number-set selected by **select_serial_set** (or of serial-number-set 0 after **load_program_file**).
Call	`CurrentSerialNo = get_serial()`
Result	Serial number. As a 64-bit IEEE floating point value.
Multi-board Com'd Name	**n_get_serial**
Comments	• For usage of **get_serial**, see Chapter 7.5.2 "Marking Serial Numbers", page 219. • **get_serial** should not be confused with **get_serial_number**, which returns the product serial number of the RTC6 board.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**select_serial_set**

Ctrl Command	**get_serial_number**
Function	Returns the individual serial number of the active RTC6 board.
Call	`RTCSerialNumber = get_serial_number()`
Result	RTC6 serial number. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_serial_number**
Comments	• Serial numbers of installed boards are ascertained by **init_rtc6_dll** and cached in the RTC6 DLL, from where you can query them by **get_serial_number**. • **get_serial_number** is helpful when using several RTC6 boards in one computer, see Chapter 6.6 "Using Several RTC6 PCIe Boards in One PC", page 126. The associated multi-board command **n_get_serial_number** can be used for determining the relationship between the installed boards and the RTC6 DLL-internal numbers assigned to them during initialization. The RTC6 DLL-internal numbers are newly assigned during each initialization of a user program (see **init_rtc6_dll**) and must be supplied for a variety of commands (in particular, all multi-board commands). The number of boards found during initialization can be queried by **rtc6_count_cards**. • **get_serial_number** and **n_get_serial_number** is available even without (current) explicit access rights to a specific RTC6 board. However, the return values are 0 as long as no access to the addressed board has been successful at least once before. • The board-specific error variables `LastError` and `AccError` (see Chapter 6.8 "Error Handling", page 133) are neither generated nor altered by **get_serial_number**.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**rtc6_count_cards**

Ctrl Command	get_standby
Function	Returns the currently set standby parameters.
Call	`get_standby( &HalfPeriod, &PulseLength )`
Returned Parameter Values	`HalfPeriod` *Half* of the currently set standby output period of the standby pulses. As a pointer to an unsigned 32-bit value. 1 bit equals 1/64 μs.
	`PulseLength` Currently set pulse length of the standby pulses. As a pointer to an unsigned 32-bit value. 1 bit equals 1/64 μs.
Multi-board Com'd Name	**n_get_standby**
Comments	• For usage of **get_standby**, see Section "Signals for "Laser Standby" Operation", page 191.
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 divides the values for `HalfPeriod` and `PulseLength` by 8.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_standby**

Ctrl Command	get_startstop_info
Function	Provides information about internal and External Starts and External Stops since the last call of **get_startstop_info**. Also provided are the current External Start and External Stop levels, the status and signal level of the Laser Control Signals, and possible transmission errors to and from the attached scan system.
Call	`StartStopInfo = get_startstop_info()`
Result	Info signal. As an unsigned 32-bit value. Bit #0 (LSB) = 1: An internal start has been executed (by **execute_list** or similar) since the last call of **get_startstop_info**. Bit #1 = 1: An External Start has been executed (by /START, /START2, /Slave-START, **simulate_ext_start** or **simulate_ext_start_ctrl**) since the last call of **get_startstop_info**. Bit #2 = 1: An internal stop has been executed (by **stop_execution**) since the last call of **get_startstop_info**. Bit #3 = 1: An External Stop has been executed (by /STOP, /STOP2, /Slave-STOP or **simulate_ext_stop**) since the last call of **get_startstop_info**. Bit #4 Ext-stop status (= logical AND operation of the signals /STOP, /STOP2, /Slave-STOP and **simulate_ext_stop**, see Figure 70): = 1: *No* stop signals are currently present at the input ports or the input ports are not connected. = 0: There is a stop signal at least at one of the input ports. Bit #5 ≥ DLL 619: = 1: The timeout with **wait_for_timestamp_counter_mode** and **wait_for_timestamp_counter_long**. Bit #6 Reserved. Bit #7 Reserved. Bit #8 Reserved. Bit #9 = 1: The Laser Control Signalsare globally enabled, see Chapter 7.4.1 "Enabling, Activating and Switching Laser Control Signals", page 189. See also **set_laser_control**. Bit #10 = 1: The TTL Laser Control Signals at the LASER1 and LASER2 output ports are active-LOW (the signal level can be defined by **set_laser_control**). Bit #11 = 1: Since the last call of **get_startstop_info**, at least one External Start has failed (more External Starts were triggered than could be simultaneously held in the 8-start wait loop). Bit #12 Ext-Start status (= logical AND operation of the signals /START, /START2 and /Slave-START, see Figure 70): = 1: *No* start signals are currently present at the input ports or the input ports are not connected. = 0: A start signal is present at least at one of the input ports. Bit #13 = 1: The TTL laser control signal at the LASERON output port is active-LOW (the signal level can be defined by **set_laser_control**).

Ctrl Command	get_startstop_info
Result (cont'd)	Bit #14 = 1: The Laser Control Signals are enabled (**enable_laser**). = 0: The Laser Control Signals are disabled (**disable_laser**). Bit #15 = 1: If previously activated by **eth_configure_link_loss**: An Ethernet Link Loss has been detected since the last call of **get_startstop_info**. Bit #16 … Bit #31 The error bits. Get set when an error occurs during data transmission from the scan system: Bit #16…Bit #23 for the Connector for First Scan Head Bit #24…Bit #31 for the Connector for Second Scan Head Bit #16, Bit #24: Incorrect number of frames within a data block. Bit #17, Bit #25: Incorrect pulse length of signal received from scan system, maybe no scan system is connected. Bit #18, Bit #26: Preamble sequence incorrect. Bit #19, Bit #27: Bit count within a subframe incorrect. Bit #20, Bit #28: Parity error when reading data received from scan system. Bit #21, Bit #29: The present data is invalid (old). Bit #22, Bit #30: Reserved. Bit #23, Bit #31: Reserved.
Multi-board Com'd Name	**n_get_startstop_info**
Comments	• After **get_startstop_info** has been executed, reset are: – The info bits Bit #0…Bit #3, Bit #15 – The error bits Bit #5, Bit #16…Bit #31 • See also Section "External Stop", page 311 and Section "External Start", page 312.
RTC4→RTC6	Unchanged functionality for info bits that are also used on the RTC4.
RTC5→RTC6	Unchanged functionality for info bits that are also used on the RTC5.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 622, OUT 622: Bit #15.
References	**get_counts**, **get_status**, **set_control_mode**

<table>
<tr><th>Ctrl Command</th><th colspan="2">get_status</th></tr>
<tr><td>Function</td><td colspan="2">Returns the current List Execution Status values and the current (or most recent) position of the output pointer.</td></tr>
<tr><td>Call</td><td colspan="2"><code>get_status(&Status, &Pos)</code></td></tr>
<tr><td rowspan="2">Returned Parameter Values</td><td><code>Status</code></td><td>Status value.
As a pointer to an unsigned 32-bit value.
Bit #0 (LSB) = 1: BUSY list execution status set.
Bit #1 Reserved.
… …
Bit #6 Reserved.
Bit #7 = 1: INTERNAL-BUSY list execution status set.
Bit #8 Reserved.
… …
Bit #14 Reserved.
Bit #15 = 1: PAUSED list execution status set.
Bit #16 Reserved.
… …
Bit #22 Reserved.
Bit #23 = 1: HEAD BUSY list execution status set. Operating status for SCANahead Systems only.
Bit #24 Reserved.
… …
Bit #31 Reserved.</td></tr>
<tr><td><code>Pos</code></td><td>Current (or most recent) position of the output pointer (absolute memory address). As a pointer to an unsigned 32-bit value.</td></tr>
<tr><td>Multi-board Com’d Name</td><td colspan="2">n_get_status</td></tr>
<tr><td>Comments</td><td colspan="2">• For a description of when the BUSY list execution status, INTERNAL-BUSY list execution status or PAUSED list execution status values are set or not set, see Chapter 6.4.3 “List Execution Status”, page 111.
• (BUSY list execution status and PAUSED list execution status set) requires restart_list for continuation, (BUSY list execution status not set and PAUSED list execution status set) requires release_wait and (both BUSY list execution status and PAUSED list execution status not set) requires execute_list_pos. “Continuation” is not allowed with (BUSY list execution status set and PAUSED list execution status not set) and a currently running list. An improper continuation generates the get_last_error return code <code>RTC6_BUSY</code>. With (INTERNAL-BUSY list execution status set), release_wait and execute_list_pos are only executed with a delay (after INTERNAL-BUSY list execution status has been reset again).</td></tr>
</table>

Ctrl Command	get_status
Comments (cont'd)	• The output pointer points to the command (in "List 1" or "List 2") currently being executed or most recently executed. If, during processing of a subroutine in the protected RTC6 List Memory area "List 3", the output pointer's position `Pos` is queried, then the position is returned of *the* list command in the RTC6 List Memory area ("List 1" or "List 2") in which the output pointer most recently resided (typically from where the subroutine has been called (for example, with **list_call**). **pause_list** and **set_wait** leave `Pos` unchanged. • **get_status** returns the output pointer position as an absolute memory address (offset relative to the start of "List 1"). The relative position referenced to the start of the respective list area can be queried by **get_out_pointer**. • The current input pointer position can be queried by **get_input_pointer** or **get_list_pointer**. • List Status values for individual lists can be queried by **read_status**. • **get_status**, **get_input_pointer** and **read_status** can be used during loading of a list to ensure that no list is overwritten that has still not been processed (see also the **load_list**). • As long as no program has been loaded (by **load_program_file**), **get_status** returns undefined values. • **get_head_status** is available for querying the status signals of the scan heads.
RTC4→RTC6	Basically unchanged functionality. However: The parameter `Status` returns additionally the INTERNAL-BUSY list execution status and PAUSED list execution status with an RTC6.
RTC5→RTC6	Changed functionality. • HEAD BUSY list execution status is returned by `Bit #23`.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**read_status**, **get_input_pointer**, **get_list_pointer**, **get_out_pointer**

Ctrl Command	get_stepper_status
Function	Returns the following status information for both stepper motor output ports: the current statuses of the stepper motor signals, the currently defined CLOCK pulse period, status "Busy" and status "Init", and the current values of the internal position variables.
Call	`get_stepper_status( &Status1, &Pos1, &Status2, &Pos2 )`
Returned Parameter Values	`Status1` Current status of stepper motor output 1. As a pointer to an unsigned 32-bit value. Bit #0 (LSB) ENABLE signal. Bit #1 DIRECTION signal. Bit #2 CLOCK signal. Bit #3 SWITCH signal = limit switch signal. Bit #4 Status "Busy". Bit #5 Status "Init". Bit #6 Reserved. Bit #7 Reserved. Bit #8 … Bit #31 CLOCK pulse period (24-bit value).
	`Pos1` Current value of the internal position variable for stepper motor output port 1. As a pointer to a signed 32-bit value.
	`Status2` Current status of stepper motor output port 1. Otherwise, like `Status1`.
	`Pos2` Like `Pos1`.
Multi-board Com'd Name	**n_get_stepper_status**
Comments	• For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 306. • If the SWITCH signal (limit switch signal) is set to (pin is LOW), no more CLOCK pulses are generated. • Status "Busy" indicates that a previously initiated (by **stepper_abs**, **stepper_rel**, etc.) set-position motion has not yet completed. • Status "Init" indicates that a previously initiated (by **stepper_init**) reference run has not yet completed.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Ctrl Command	get_sub_pointer
Function	Returns the absolute start address of an indexed subroutine.
Call	`SubPointer = get_sub_pointer( Index )`
Parameters	`Index` Index of the indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0…1023].
Result	Absolute start address. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_sub_pointer**
Comments	• The **get_sub_pointer** command reads from the internal management table the start address of the indexed subroutine with the specified index. Whether the read address resides in a protected or the unprotected RTC6 List Memory area "List 3" depends on whether the subroutine has been loaded into the protected RTC6 List Memory area "List 3" or an unprotected subroutine has been only subsequently referenced. • If `Index` > 1023 or if no subroutine has been referenced with the specified index, then **get_sub_pointer** returns the value "–1" (for example, 2^{32}–1). • This command is useful for checking if a subroutine has already been defined or for calling an indexed subroutine by an absolute memory address as if it were a non-indexed subroutine. Be aware, though, that a subsequent **save_disk**/**load_disk** might alter the absolute memory address.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_char_pointer**, **get_text_table_pointer**

Ctrl Command	get_sync_status		
Function	Returns the master/slave synchronization information on the addressed RTC6 board.		
Call	`MasterSlaveSyncStatus = get_sync_status()`		
Result	Master/slave synchronization information. As an unsigned 32-bit value.		
	Bit #0 … Bit #9	Master/slave synchronization status [0…640]. As unsigned 10 bit value.	
		< 4:	The addressed board is synchronous to the master board (or to its preceding board in the master/slave chain).
		640:	The addressed board is operated as master.
	Bit #10	= 0:	Bit #0…Bit #9 contain valid values.
		= 1:	Bit #0…Bit #9 do not contain valid values. No External Start has been received from the master board yet.
	Bit #11	Reserved.	
	Bit #12	= 1:	A master board has been detected. However, no /Slave-START or /Slave-STOP can be received. See **master_slave_config**.
	Bit #13	= 1:	A Slave board has been detected. However, no /Slave-START or /Slave-STOP can be received. See **master_slave_config**.
	Bit #14	= 1:	Since the last call of **get_sync_status**, the connection to the master has been lost.
	Bit #15	= 1:	Since the last call of **get_sync_status**, the connection to the slave has been lost.
	Bit #16 … Bit #20	Reserved.	
	Bit #21 … Bit #30	Exact propagation time in 1/64 µs clock cycles between two RTC6 boards (outbound and return). As unsigned 10 bit value.	
	Bit #31	= 1:	The propagation time Bit #21…Bit #30) could not be measured successfully.
Multi-board Com'd Name	**n_get_sync_status**		

Ctrl Command	get_sync_status
Comments	• For usage of **get_sync_status**, see Chapter 6.6.3 "Master/Slave Operation", page 127. • If the addressed board has not been initialized previously by **load_program_file**, the **get_sync_status** returns 0. • As of RBF 619: – Synchronization is automatic. **sync_slaves** is no longer required. The master/slave synchronization state is measured automatically. – The exact propagation time is returned by Bit #21…Bit #30, if Bit #31 does not indicate an error. • Up to RBF 618: – Prior to **get_sync_status**, the boards of the master/slave chain should have been synchronized by **sync_slaves**. – To ensure that **get_sync_status** actually returns the current master/slave synchronization status, you should first have already triggered an External Start for the master board by an external start signal or by **simulate_ext_start_ctrl**, see Section "External Start", page 312. For all slave boards for which External Starts and External Stops are not suppressed (see **master_slave_config**), this start then triggers a measurement of the time difference between the respective /Slave-START pulse and respective 10 μs clock cycle. This time difference gets stored on each board as the master/slave synchronization status (in units of 1/64 μs) and remains stored there until the a new External Start is triggered for the master board. This gives you the flexibility to query the synchronization status by **get_sync_status** even at a later point in time. – In the synchronized state, measured time differences are shorter than the transit time difference for synchronization between the boards themselves, see Chapter 6.6.3 "Master/Slave Operation", page 127: master/slave synchronization status < 3. – In a not-explicitly-synchronized state (prior to **sync_slaves**), the master/slave synchronization status might coincidentally be < 3. If so, then the boards behave as if synchronized. – If Bit #12 or Bit #13 is set, the affected cards cannot be synchronized with **sync_slaves**. – If Bit #14 or Bit #15 is set, /Slave STARTs or /Slave STOPs may not have been forwarded. In this case it is recommended to adjust **master_slave_config** and to execute **sync_slaves** again. – If one or more of Bit #12…Bit #15 are sporadically 1, this may indicate electromagnetic disturbances that interfere with communication between the boards.
RTC4→RTC6	New command.
RTC5→RTC6	Basically unchanged functionality. However, more details are available.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 614, OUT 614, RBF 619: internal improvements and additional results.
References	**sync_slaves**, **get_master_slave**, **master_slave_config**

Ctrl Command	get_table_para
Function	Returns the value of the specified parameter from a currently loaded correction table.
Call	`TablePara = get_table_para( TableNo, ParaNo )`
Parameters	`TableNo` Number of the currently loaded correction table. As an unsigned 32-bit value. Allowed values: [1…8]. See also **number_of_correction_tables**.
	`ParaNo` Number of the parameter. As an unsigned 32-bit value. Allowed values: 0…15. Assignment see Section "ct5 Correction File Header", page 183.
Result	Parameter value, see Section "ct5 Correction File Header", page 183. As a 64-bit IEEE floating point value.
Multi-board Com'd Name	**n_get_table_para**
Comments	• The parameter values can be read out by **get_table_para** from a currently loaded correction table and by **get_head_para** from an assigned correction table and thus directly incorporated into a user program, see Section "ct5 Correction File Header", page 183. • If the parameters `TableNo` and `ParaNo` are out of range, then the return value is 0 (**get_last_error** return code `RTC6_PARAM_ERROR`). • If no correction table with the specified number has been loaded, then the parameter values are undefined.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Ctrl Command	get_temperature
Function	Returns the current temperature of the RTC6 board.
Call	`Temp = get_temperature()`
Parameters	None.
Result	Temperature of the RTC6 board. In degrees Celsius. As a 64-bit IEEE floating point value.
Multi-board Com'd Name	**n_get_temperature**
Comments	• Execution of **get_temperature** takes about 1 ms. • Continuous readout of the analog voltage (after a **read_analog_in** call) is delayed by **get_temperature** by about 0.1 ms, see Section "Analog Input Ports", page 89.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 631, OUT 632.
References	–

Ctrl Command	get_text_table_pointer
Function	Returns the absolute start address of an indexed text string.
Call	`TextTablePointer = get_text_table_pointer( Index )`
Parameters	`Index` Index of the indexed text string. As an unsigned 32-bit value. Allowed value range: [0…41].
Result	Absolute start address. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_text_table_pointer**
Comments	• **get_text_table_pointer** reads from the internal management table the start address of the indexed text string with the specified index. Whether the read address resides in a protected or the protected RTC6 List Memory area "List 3" depends on whether the text string has been loaded into the protected RTC6 List Memory area "List 3" or an unprotected subroutine has been only subsequently referenced. • If `Index` > 41 or if no text string has been referenced with the specified index, then **get_text_table_pointer** returns the value "–1" (for example, 2^{32}–1). • This command is useful for checking if a text string has already been defined or for calling an indexed text string by an absolute memory address as if it were a non-indexed subroutine, for example, for conditional execution with **list_call_cond**. Be aware, though, that a subsequent **save_disk**/**load_disk** might alter the absolute memory address. And you should ensure that **get_text_table_pointer** does not return "–1"; otherwise **list_call_cond** is ignored.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_char_pointer**, **get_sub_pointer**

Ctrl Command	get_time
Function	Returns the RTC6 Timer value (without resetting it to zero). It has been stored during the most recent call of **save_and_restart_timer**.
Call	`TimerValue = get_time()`
Result	RTC6 Timer value. In seconds. As a 64-bit IEEE floating point value.
Multi-board Com'd Name	**n_get_time**
Comments	• See **save_and_restart_timer**. • The number of elapsed list-command clock cycles since the reset to zero can be queried by **get_lap_time**.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_lap_time**, **save_and_restart_timer**

Ctrl Command	get_timestamp_long
Function	Returns the current value of the 64-bit "Timestamp Counter".
Call	`get_timestamp_long( &TimeStampL, &TimeStampH )`
Returned Parameter Values	`TimeStampL` Lower 64-bit "Timestamp Counter" value. As an unsigned 32-bit value.
	`TimeStampH` Upper 64-bit "Timestamp Counter" value. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_timestamp_long**
Comments	• The RTC6 board-internal 64-bit "Timestamp Counter" `TimeStampCounterLong` is: – Initialized with 0 at **load_program_file** and then keeps running until the next **load_program_file** – Used with **wait_for_timestamp_counter_long** • The lower part of `TimeStampCounterLong` (= `returned parameter value TimeStampL`) and the 32-bit "Timestamp Counter" are identical. • See Chapter 8.12 "Time Measurements", page 290.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 624, OUT 624.
References	**wait_for_timestamp_counter_long**

Ctrl Command	get_transform	
Function	Like **get_transform_offset**, but without parameter Channel1, Channel2, Offset.	
Call	`get_transform( Number, Ptr1, Ptr2, Ptr, Code )`	
Parameters	Number	Number of to-be-backward-transformed position values. As an unsigned 32-bit value. Allowed value range: [1…2^{24}]. The measured values with indices 0 to (Number–1) are backward transformed.
	Ptr1	Like **get_transform_offset**.
	Ptr2	Like **get_transform_offset**.
	Ptr	Like **get_transform_offset**.
	Code	Like **get_transform_offset**.
Multi-board Com'd Name	**n_get_transform**	
Comments	• `get_transform( Number, Ptr1, Ptr2, Ptr, Code )` – Transfers the data pairs recorded with **set_trigger[*]** to the PC with • **get_waveform**(1,Number, Ptr1) and • **get_waveform**(2,Number, Ptr2) **Overwrites the data pairwise with transform(Sig1, Sig2, Ptr, Code)** • Comments, page 479 on **get_transform_offset** apply analogously.	
RTC4→RTC6	New command. Even in the RTC4 Compatibility Mode, all coordinate values transferred to the PC are in the RTC6 20-bit range. The backward transformed coordinate values must, if necessary, be divided by 16 by users themselves.	
RTC5→RTC6	Basically unchanged functionality. However: Allowed value range of Number remains [1…2^{23}].	
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 635, OUT 636: Number. Allowed value range extended to [1…2^{24}] (previously: [1…2^{23}]).	
References	**upload_transform**, **transform**, **set_trigger**, **set_trigger4**, **set_trigger8**, **get_waveform**, **get_transform_offset**	

Ctrl Command	get_transform_offset	
Function	Transfers to the PC the position values that has been recorded by **set_trigger[*]** and stored on the RTC6, and applies backward transformation to them.	
Call	`get_transform_offset( Channel1, Channel2, Offset, Number, Ptr1, Ptr2, Ptr, Code )`	
Parameters	`Channel1`	Measurement channel. As an unsigned 32-bit value. Allowed values: [1...8].
	`Channel2`	Like `Channel1`.
	`Offset`	Start position. As an unsigned 32-bit value. Allowed value range: [1…2^{24}].
	`Number`	Number of to-be-backward-transformed position values. As an unsigned 32-bit value. The measured values from position `Offset`…[(`Offset` + (`Number` – 1)] are backward transformed.
	`Ptr1`	Pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 64-bit value) to the area of PC main memory to which the backward transformed values are to be transferred (see also `Code`).
	`Ptr2`	Like `Ptr1`.
	`Ptr`	Pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 64-bit value) to the area of PC main memory to which the correction and transformation settings for backward transformation were previously transferred by **upload_transform**.
	`Code`	Aspects of backward transformation, particularly which partial transformations are to be performed. If a partial transformation should *not* be performed, then the corresponding Bit must be set to 1 (Bit #2…Bit #5, Bit #2…Bit #4). `Code` has the same meaning as `Code` of **transform** (`Ptr1` corresponds to `Sig1`, `Ptr2` corresponds to `Sig2`). As an unsigned 32-bit value.

Ctrl Command	get_transform_offset			
Parameters (cont'd)	Code (cont'd)	For Bit #0 = 0, the measurement value pairs recorded by **set_trigger[*]** are backward transformed as xy coordinates:		
		Bit #1	= 0:	Values recorded by measurement channel 1 (`Signal1`) are transferred to the PC using `Ptr1` and then backward transformed as x coordinates. Values recorded by measurement channel 2 (`Signal2`) are transferred to the PC using `Ptr2` and then backward transformed as y coordinates.
			= 1:	Values recorded by measurement channel 1 (`Signal1`) are transferred to the PC using `Ptr1` and then backward transformed as y coordinates. Values recorded by measurement channel 2 (`Signal2`) are transferred to the PC using `Ptr2` and then backward transformed as x coordinates.
		Bit #2	= 0:	Gain/offset correction of automatic self-calibration is backward transformed.
		Bit #3	= 0:	Image Field correction is backward transformed.
		Bit #4	= 0:	Offset of the defined coordinate transformation is backward transformed.
		Bit #5	= 0:	Total matrix of the defined coordinate transformation is backward transformed.
		Bit #6	Reserved.	
		…	…	
		Bit #31	Reserved.	

Ctrl Command	get_transform_offset			
Parameters (cont'd)	`Code` (cont'd)	If Bit #0 = 1, then (only) the values recorded by **set_trigger[*]** by one of the two measurement channels (either channel 1 or 2) are backward transformed as z coordinates:		
		Bit #1	= 0:	Values recorded by measurement channel 1 (`Signal1`) are transferred to the PC using `Ptr1` and then backward transformed as z coordinates. Values recorded by measurement channel 2 (`Signal2`) are transferred untransformed to the PC using `Ptr2`.
			= 1:	Values recorded by measurement channel 2 (`Signal2`) are transferred to the PC using `Ptr1` and then backward transformed as z coordinates. Values recorded by measurement channel 1 (`Signal1`) are transferred untransformed to the PC using `Ptr2`.
		Bit #2	= 0:	Offset to the focal length defined by **set_defocus** or **set_defocus_list** is backward transformed.
		Bit #3	= 0:	ABC correction is backward transformed.
		Bit #4	= 0:	Offset to the z coordinate defined by **set_offset_xyz** or **set_offset_xyz_list** is backward transformed.
		Bit #5	Reserved.	
		…	…	
		Bit #31	Reserved.	
Multi-board Com'd Name	**n_get_transform_offset**			

Ctrl Command	get_transform_offset

Comments	• `get_transform_offset( Channel1, Channel2, Offset, Number, Ptr1, Ptr2, Ptr, Code )`
	– Transfers the data pairs recorded with **set_trigger[*]** to the PC with
	  • **get_waveform_offset**(Channel1, Offset, Number, Ptr1) and
	  • **get_waveform_offset**(Channel2, Offset, Number, Ptr2)
	– Overwrites the data pairwise with **transform**(`Sig1`, `Sig2`, `Ptr`, `Code`)
	• Prior to a **get_transform_offset** call:
	– **upload_transform** must have been executed
	– Position values should have been recorded by **set_trigger[*]**
	• Prior to a **get_transform_offset call** or **get_waveform_offset** call, it can be checked by **measurement_status** whether a recording is currently running that has been started with **set_trigger**/**set_trigger4**/**set_trigger8**.
	In addition, **measurement_status** can be used to read out the number `Pos` of data pairs recorded during the last (or current) measurement. Read out at most `Pos`+1 data elements. All other data elements are from previous recordings or initialization.
	• You must provide the working memory areas under `Ptr1` and `Ptr2` with sufficient size (`Number` × 4 bytes per channel).
	• For backward transformation of position values, see Chapter 8.1.3 "Monitoring the Positioning", page 223.
	• If only Z position values are to be backward transformed (`Code` Bit #0 = 1), then you can set the pointer (`Ptr1` or `Ptr2`) for the unused measurement channel to NULL. For this measurement channel, no date are transferred to the PC and backward transformed. In this case, make sure that `Code` Bit #1 is appropriately set.
	• For both measurement channels, no data are transferred to the PC and no backward transformation is performed (**get_last_error** return code `RTC6_PARAM_ERROR`) with:
	– **get_transform_offset**(`Ptr1` = NULL, `Ptr2` = NULL)
	– `Number` = 0
	– `Number` > 2^{24}
	• With **get_transform_offset**(`Ptr` = NULL), then no backward transformation is performed. The data recorded by **set_trigger[*]** is then transferred untransformed to the PC, as with **get_waveform**(1, `Number`, `Ptr1`) and **get_waveform**(2, `Number`, `Ptr2`).
	The same applies for **get_transform_offset**(`Ptr` ≠ NULL), if an error occurred during execution of **get_transform_offset** (for example, data referenced by `Ptr` are invalid or erroneous, z axis inversion not possible). In this case, a **get_last_error** return code `RTC6_PARAM_ERROR` is generated in addition.

Ctrl Command	get_transform_offset
Comments (cont'd)	• If needed, the values recorded with **set_trigger** and backward-transformed transferred to the PC by **get_transform_offset** can additionally be untransformed transferred to the PC by **get_waveform**. • If backward transformation of z position values is requested (`Code` Bit #0 = 1), but only a 2D correction table has been assigned at the timepoint of the prior successful call to **upload_transform**, then the offsets to the focal length and z coordinates are initialized with 0 and the values A, B, C with are initialized 0, 1, 0 (1-to-1 backward transformation). • For backward transformation of xy position values (`Code` Bit #0 = 0), only the z = 0 plane are transformed. xy stretching and Z defocus due to z deviations (particularly with non-F-Theta systems) are not taken into account. • PCI transmission errors generate the **get_last_error** return code `RTC6_SEND_ERROR`.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 635, OUT 636.
References	**get_transform**, **upload_transform**, **transform**, **set_trigger**, **set_trigger4**, **set_trigger8**, **get_waveform**, **get_transform**

Ctrl Command	get_value
Function	Returns the current value of the specified signal.
Call	`Value = get_value( Signal )`
Parameters	`Signal` Desired signal type. As an unsigned 32-bit value.
Result	Current value of the specified signal. As a signed 32-bit value.
Multi-board Com'd Name	**n_get_value**
Comments	• The selectable signal types are identical to those of **set_trigger** (refer to the comments there for the allowed value range, signal types and other information). If the value for `Signal` is unallowed, then **get_value** does not read out a signal and returns 0 (**get_last_error** return code `RTC6_PARAM_ERROR`). • To observe the specified `Signal` over a long time period, use **set_trigger** to start a corresponding measurement session. • After a reset or power-up of iDRIVE Scan Systems it can take around 5 seconds before it transmits valid data. • For `Signal` = 0, **get_value** returns the current laser status (LASERON signal) even when list execution has already been finished. • When you query data returned as status signals from the scan system to the RTC6 (`Status<AX…BY>`), then be mindful of the returned data type's value range when evaluating it (see **control_command**): – Data types originally generated in the scan system as unsigned 16-bit values (for example, the XY2-100 status word or the serial number) and returned to the RTC6 as unsigned 20-bit values (whereby Bit #0…Bit #3 = 0) contain the relevant information in Bit #4…Bit #19. Here, only Bit #4…Bit #19 should be evaluated (see code example below). For Bit #20…Bit #31 of this data type, **get_value** returns to the PC not only zero, but (depending on Bit #19 of the underlying 16-bit status value) even the value one. So only evaluate Bit #4…Bit #19. – In contrast, data types returned to the RTC6 as signed 20-bit values (for example, actual positions or actual speeds) can be evaluated as a complete signed 32-bit value returned by **get_value** (see also code example below).

Ctrl Command	get_value

Example (C/C++)	Querying diverse data types (Connector for First Scan Head, x axis):
	a) XY2-100 status word, PowerOK status
	`UINT statusword, powerOK;`
	`control_command (1, 1, 0x0500); // only applicable for iDRIVE Scan Systems`
	`statusword = (get_value(1) & 0x000FFFF0) >> 4;`
	`powerOK = (statusword & 0x00000080);`
	
	b) Only with iDRIVE Scan Systems: serial number
	`UINT SN_low, SN_high, SN;`
	`// the serial number's lower 16 bits are selected for return`
	`// and queried by get_value:`
	`control_command (1, 1, 0x051E);`
	`SN_low = (get_value(1) & 0x000FFFF0)>>4;`
	`// the serial number's upper 16 bits are selected for return`
	`// and queried by get_value:`
	`control_command (1, 1, 0x051F);`
	`SN_high = (get_value(1) & 0x000FFFF0)>>4;`
	`//Complete serial number:`
	`SN = (SN_high << 16) + SN_low;`
	
	c) Only with iDRIVE Scan Systems: actual position
	`long real_position;`
	`control_command (1, 1, 0x0501);`
	`real_position = get_value(1);`

RTC4→RTC6	Basically unchanged functionality. However: Even in RTC4 Compatibility Mode, all returned values are in the RTC6 20-bit range, but are transferred to the PC as 32-bit values (see above).
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_values**, **set_trigger**, **get_waveform**, **get_head_status**

Ctrl Command	get_values
Function	Returns the current values of up to 4 specified signals.
Call	`get_values( SignalPtr, ResultPtr )`
Parameters	`SignalPtr` Pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 64-bit value) to an array of 4 unsigned 32-bit values, where the to-be-outputted signal types are specified.
	`ResultPtr` Pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 64-bit value) to an array of 4 signed 32-bit values, where the current values of the up to 4 specified signals are to be stored.
Multi-board Com'd Name	**n_get_values**
Comments	• Up to 4 desired signals can be simultaneously queried. The selectable signal types are identical to those of **set_trigger** (refer to the comments there for the allowed value range, signal types and other information). The to-be-outputted signal types must be specified by `SignalPtr`. The corresponding signal values are then stored by `ResultPtr`. For storage of each queried data set, the user program must make available (at the address specified by `ResultPtr`) 4 × 4 bytes of PC memory. • **get_values** functions similarly to **get_value** (see comments there). **get_values** returns 0 and performs no query on channels for which an invalid signal type has been specified by `SignalPtr`. A **get_last_error** return code `RTC6_PARAM_ERROR` is only generated if all 4 specified signal types are invalid. • If any of the pointer parameters are NULL, then **get_values** is not executed and a **get_last_error** return code of `RTC6_PARAM_ERROR` is generated.
RTC4→RTC6	New command. Even in RTC4 Compatibility Mode, the 4 returned values are in the RTC6 20-bit range, but are transferred to the PC as 32-bit values (see comments for **get_value**).
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_value**, **set_trigger**, **transform**

Ctrl Command	get_wait_status
Function	Returns the wait state of the RTC6 board.
Call	`WaitStatus = get_wait_status()`
Result	Wait state. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_get_wait_status**
Comments	• If list processing has been stopped at a break point ("wait marker"), then **get_wait_status** returns the corresponding number. See **set_wait**. • If no break point has been encountered, **get_wait_status** returns the value zero. • The list processing is resumed by calling **release_wait**.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_wait**, **release_wait**

Ctrl Command	get_waveform	
Function	Like **get_waveform_offset**, but without parameter Offset.	
Call	get_waveform(Channel, Number, Ptr)	
Parameters	Channel	Like **get_waveform_offset**.
	Number	Number of measured values to be transferred. Allowed value range: [0…max. channel size, see **set_trigger[*]**]. The measured values from position 0 to (Number–1) are transferred. As an unsigned 32-bit value.
	Ptr	Like **get_waveform_offset**.
Multi-board Com'd Name	**n_get_waveform**	
Comments	• No data are transferred (**get_last_error** return code RTC6_PARAM_ERROR): – For Number = 0 – For Number > 2^{22}, if the recording has been started by **set_trigger8** – For Number > 2^{23} × 4 and Channel = 1 – For Number > 2^{23} × 2 and Channel = 2 – For Number > 2^{23} × 3 and Channel = 3 – For Number > 2^{23} × 1 and Channel = 4 – For Ptr = NULL. • **get_waveform** is synonymous with: – **get_waveform_offset**(Channel, 0, Number, Ptr) • Comments, page 479 on **get_waveform_offset** apply analogously.	
RTC4→RTC6	Basically unchanged functionality. However: Even in RTC4 Compatibility Mode, all values are in the RTC6 20-bit range, but are transferred to the PC as 32-bit values (see comments for **get_value**).	
RTC5→RTC6	Basically unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 635, OUT 636: Channel. Allowed values extended to 8 (previously: 4).	
References	**set_trigger**, **set_trigger4**, **set_trigger8**, **get_value**, **get_values**, **get_waveform_offset**	

Ctrl Command	get_waveform_offset
Function	Transfers to the PC the data that has been measured and stored onto the RTC6 by **set_trigger[*]**.
Call	`get_waveform_offset( Channel, Offset, Number, Ptr )`
Parameters	`Channel` Measurement channel. Allowed values: 1…2 with recordings started by **set_trigger**. 1…4 with recordings started by **set_trigger4**. 1…8 with recordings started by **set_trigger8**. As an unsigned 32-bit value.
	`Offset` Start position. As an unsigned 32-bit value. Allowed value range: [0…max. channel size, see **set_trigger[*]**].
	`Number` Number of measured values to be transferred. The measured values from position `Offset` to (`Offset` + (`Number`–1)) are transferred. As an unsigned 32-bit value.
	`Ptr` Pointer (data type ULONG_PTR in C and C++, an unsigned 32-bit or unsigned 64-bit value) to a location in the PC memory to where the measured values are to be transferred.
Multi-board Com'd Name	**n_get_waveform_offset**
Comments	• No data are transferred (**get_last_error** return code `RTC6_PARAM_ERROR`): – For `Number` = 0. – For `Number` > 2^{22}, if the recording has been started by **set_trigger8** – For `Offset` + `Number` > max. channel size – For `Ptr` = NULL • PCI transmission errors generate the **get_last_error** return code `RTC6_SEND_ERROR`. • See also **set_trigger[*]** on configuration of the channel. • Prior to a **get_transform_offset call** or **get_waveform_offset** call, it can be checked by **measurement_status** whether a recording is currently running that has been started by **set_trigger[*]**. In addition, **measurement_status** can be used to read out the number `Pos` of data pairs recorded during the last (or current) measurement. Read out at most `Pos`+1 data elements. All other data elements are from previous recordings or initialization.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 610, OUT 610, RBF 614. Last change DLL 635, OUT 636: `Channel`. Allowed values extended to 8 (previously: 4).
References	**get_waveform**, **set_trigger**, **set_trigger4**, **set_trigger8**, **get_value**, **get_values**

Ctrl Command	get_z_distance
Function	Returns the focus length value l for the specified point within the 3D Image Field.
Restriction	If the Option "3D" has not been enabled or if no 3D correction table has been assigned (see **select_cor_table**), then **get_z_distance** returns 0 and otherwise has no effect.
Call	`ZDistance = get_z_distance( X, Y, Z )`
Parameters	`X` Absolute coordinates of the point (x|y|z) in the 3D Image Field. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
	`Y` Like `X` (analogously).
	`Z` Like `X` (analogously).
Result	Focus length value. [–524,288…+524,287]. As a signed 32-bit value.
Multi-board Com'd Name	**n_get_z_distance**
Comments	• **get_z_distance** is only needed for re-calibrating the z axis in a 3-axis scan system, see Section "Checking the z axis Calibration", page 175. • The focus length value l: – Has no dimension – Corresponds to the focus length difference between the specified point (x|y|z) and the point (0|0|0) – Can be positive or negative • With the RTC6, `ZDistance` is always in 20-bit range [–524,288…+524,287]: – In RTC4 Compatibility Mode – In RTC5 Compatibility Mode – In RTC6 Standard Mode Important: If you do not use **load_z_table_20b** or **load_z_table_no_20b**, you must divide `ZDistance` by 16 and insert the result into the parabolic function $z_{out} = A + Bl + Cl^2$. • **get_z_distance** first performs a (virtual) jump to the point (x|y|z) and then returns the focus length value. • If a list is currently executed, then **get_z_distance** has no effect and returns 0 (**get_last_error** return code `RTC6_BUSY`). • **get_z_distance** is not executed and returns 0 (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **get_z_distance** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • For 3D Image Field calibration, see Chapter "3D Commands", page 247.

Ctrl Command	get_z_distance
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode, the RTC6 multiplies the values specified for `X`, `Y` and `Z` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the value specified for `Z` by 16. The allowed value range decreases accordingly.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**load_z_table**, **load_z_table_no**, **load_z_table_20b**, **load_z_table_no_20b**

Ctrl Command	goto_xy
Function	Moves the output point (of the laser focus) along a 2D vector at jump speed from the current position to the specified position (absolute coordinate values) within a 2D Image Field.
Call	`goto_xy( X, Y )`
Parameters	`X` Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
	`Y` Like `X` (analogously).
Multi-board Com'd Name	**n_goto_xy**
Comments	• If the jump speed has not been previously explicitly set by **set_jump_speed** or **set_jump_speed_ctrl**, then the jump is executed at a predefined jump speed of 10000 *bits/ms*. • **goto_xy** (unlike the list commands **jump_abs** and **jump_rel**) has no effect on the Laser Control Signals and also does not set a Jump Delay. • Previously accumulated coordinate transformations become effective by **goto_xy**, see list item "With `at_once` = 2 …", page 235. • **goto_xy** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set – an /STOP, /STOP2 or /Slave STOP is present It can also be generated by automatic monitoring, see **set_laser_control** and **range_checking**. • **goto_xy** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • The INTERNAL-BUSY list execution status is set while **goto_xy** is executed. • **goto_xy** only returns to the user program when the motion has been completed.
RTC4→RTC6	• Increased value range. • **goto_xy** returns only after motion has been executed (see comments above). • In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `X` and `Y` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_jump_speed**, **jump_abs**, **jump_rel**, **goto_xyz**, **get_status**

Ctrl Command	goto_xyz	
Function	Moves the output point (of the laser focus) along a 3D vector at jump speed from the current position to the specified position (absolute coordinate values) within the 3D Image Field.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **goto_xyz** has the same effect as **goto_xy**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.	
Call	`goto_xyz( X, Y, Z )`	
Parameters	`X`	Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
	`Y`	Like `X` (analogously).
	`Z`	Like `X` (analogously).
Multi-board Com'd Name	**n_goto_xyz**	
Comments	• Except for the additional motion in the third dimension, **goto_xyz** functions similarly to **goto_xy** (see comments there). • The `DirectMove3D` parameter of **set_delay_mode** determines the type of z axis motion (linear or with stepwise correction). • See also Chapter 7.3.6 "Output Values to the Scan System", page 186.	
RTC4→RTC6	• Increased value range. • **goto_xyz** returns only after the motion has completed (see comments for **goto_xy**). • RTC4 Compatibility Mode: see **goto_xy**.	
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the value specified for `Z` by 16. The allowed value range decreases accordingly.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**goto_xy**, **jump_abs_3d**, **jump_rel_3d**	

Ctrl Command	home_position
Function	Activates the home jump mode (for the x axis and y axis) and defines the home position.
Call	`home_position( XHome, YHome )`
Parameters	`XHome` Absolute x coordinate of the home position. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Larger values are clipped.
	`YHome` Like `XHome` (analogously).
Multi-board Com'd Name	**n_home_position**
Comments	• **home_position** defines the coordinates of a home jump to be executed, for example, upon reaching the end of a list. Accordingly, a home return to the last valid position is then executed again at start. The home jump and home return are executed at jump speed. • **home_position** is intended for laser systems that do not allow fast switching of the laser. After calling **home_position**, the laser focus moves to the specified home position whenever no list is executing or when a list has been paused by **set_wait**. A home jump is also executed, if list execution is stopped by **stop_execution** or by an external stop signal. The home jump itself (in contrary to a home return) cannot be stopped. • While a home jump or a home return is executed, the INTERNAL-BUSY list execution status is set. • A *beam dump* should be placed in the home position. • The home jump mode is deactivated by `home_position(0, 0)`, even if it has been activated by **home_position_xyz**. • **home_position** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `XHome` and `YHome` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**home_position_xyz**

Ctrl Command	home_position_xyz	
Function	Activates the home jump mode (for the x, y and z axis) and defines the home position.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **home_position_xyz** has the same effect as **home_position**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.	
Call	`home_position_xyz( XHome, YHome, ZHome )`	
Parameters	`XHome`	Absolute x coordinate of the home position. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
	`YHome`	Like `XHome` (analogously).
	`ZHome`	Like `XHome` (analogously).
Multi-board Com'd Name	**n_home_position_xyz**	
Comments	• Except for the additional motion in the third dimension, **home_position_xyz** functions similarly to **home_position** (see comments there). • The home jump mode is deactivated by: – `home_position( 0, 0 )` – `home_position_xyz( 0, 0, 0 )` • The `DirectMove3D` parameter of **set_delay_mode** determines the type of z axis motion (linear or with stepwise correction). • **home_position_xyz** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `XHome`, `YHome` and `ZHome` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the specified value for `ZHome` by 16. The allowed value range decreases accordingly.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**home_position**	

Undelayed Short List Command	**if_cond**
Function	*Conditional command execution*: **if_cond** immediately executes the directly following list command, if the current `IOvalue` at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector meets the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are `1` and the bits specified in `Mask0` are `0`). Otherwise, this list command is skipped.
Call	`if_cond( Mask1, Mask0 )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.
	`Mask0` See `Mask1`.
Multi-board Com'd Name	**n_if_cond**
Comments	• See also Chapter 9.3.2 "Execution of Conditional Commands", page 317.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**if_not_cond**, **if_pin_cond**, **if_not_pin_cond**

Undelayed Short List Command	**if_fly_x_overflow**
Function	*Conditional command execution for Processing-on-the-fly applications*: **if_fly_x_overflow** immediately executes the directly subsequent list command, if the condition in accordance with `Mode` for the x axis has been fulfilled. Otherwise, this list command is skipped.
Call	`if_fly_x_overflow( Mode )`
Parameters	`Mode` To-be-evaluated condition. As a signed 32-bit value. = 0: Some kind of boundary exceedance occurred (error bit Bit #4 = 1 or error bit Bit #5 = 1). > 0: An overflow occurred (error bit Bit #5 = 1). < 0: An underflow occurred (error bit Bit #4 = 1).
Multi-board Com'd Name	**n_if_fly_x_overflow**
Comments	• For usage of **if_fly_x_overflow**, see Section "Customer-Defined Monitoring Area", page 265. • The error bits queried in accordance with `Mode` (error bit Bit #4 and/or error bit Bit #5) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_marking_info**, **set_fly_limits**, **clear_fly_overflow**, **clear_fly_overflow_ctrl**, **if_not_fly_x_overflow**

Undelayed Short List Command	**if_fly_y_overflow**
Function	*Conditional command execution for Processing-on-the-fly applications*: **if_fly_y_overflow** immediately executes the directly subsequent list command, if the condition in accordance with `Mode` for the y axis has been fulfilled. Otherwise, this list command is skipped.
Call	`if_fly_y_overflow( Mode )`
Parameters	`Mode` To-be-evaluated condition. As a signed 32-bit value. = 0: Some kind of boundary exceedance occurred (error bit Bit #6 = 1 or error bit Bit #7 = 1). > 0: An overflow occurred (error bit Bit #7 = 1). < 0: An underflow occurred (error bit Bit #6 = 1).
Multi-board Com'd Name	**n_if_fly_y_overflow**
Comments	• For usage of **if_fly_y_overflow**, see Section "Customer-Defined Monitoring Area", page 265. • The error bits queried in accordance with `Mode` (error bit Bit #6 and/or error bit Bit #7) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_marking_info**, **set_fly_limits**, **clear_fly_overflow**, **clear_fly_overflow_ctrl**, **if_not_fly_y_overflow**

Undelayed Short List Command	**if_fly_z_overflow**
Function	*Conditional command execution for Processing-on-the-fly applications:* **if_fly_z_overflow** immediately executes the directly subsequent list command, if the condition in accordance with `Mode` for the z axis has been fulfilled. Otherwise, this list command is skipped.
Call	`if_fly_z_overflow( Mode )`
Parameters	`Mode` To-be-evaluated condition. As a signed 32-bit value. = 0: Some kind of boundary exceedance occurred (error bit Bit #24 = 1 or error bit Bit #25 = 1). > 0: An overflow occurred (error bit Bit #25 = 1). < 0: An underflow occurred (error bit Bit #24 = 1).
Multi-board Com'd Name	**n_if_fly_z_overflow**
Comments	• For usage of **if_fly_z_overflow**, see also Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 264. • The error bits queried in accordance with `Mode` (error bit Bit #24 and/or error bit Bit #25) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_marking_info**, **set_fly_limits_z**, **clear_fly_overflow**, **clear_fly_overflow_ctrl**, **if_not_fly_z_overflow**

Undelayed Short List Command	**if_not_activated**
Function	Conditional command execution due to an error bit from **activate_fly_xy**/**activate_fly_xy_encoder** or **activate_fly_2d**/**activate_fly_2d_encoder**: If the error bit is set, then the next list command is executed immediately, otherwise it is skipped.
Call	`if_not_activated()`
Multi-board Com'd Name	**n_if_not_activated**
Comments	• See also comments at **activate_fly_2d**/**activate_fly_2d_encoder** and **activate_fly_xy**/**activate_fly_xy_encoder**. • It is useful to insert a list jump or subroutine call in the list directly after **if_not_activated** that jumps to an error-handling sequence. Or you could simply insert a **set_end_of_list**. • **if_not_activated** resets the error bit from **activate_fly_xy**/**activate_fly_xy_encoder** or **activate_fly_2d**/**activate_fly_2d_encoder**. If you still need it for subsequent querying by `get_marking_info( Bit #9 )`, then you can include a renewed call of **activate_fly_2d**/**activate_fly_2d_encoder** or **activate_fly_xy** in your error-handling sequence to set the error bit again (provided that the error situation still exists). • Successful activation by **activate_fly_2d**/**activate_fly_2d_encoder** or **activate_fly_xy**/**activate_fly_xy_encoder** *does not* reset any already-set error bit. It remains set for **get_marking_info** until **get_marking_info** has been called.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**activate_fly_2d**, **activate_fly_2d_encoder**, **activate_fly_xy**, **activate_fly_xy_encoder**, **get_marking_info**

Undelayed Short List Command	**if_not_cond**
Function	*Conditional command execution*: **if_not_cond** immediately executes the directly following list command, if the current `IOvalue` at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector *does not meet* the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are *not* `1` *or* the bits specified in `Mask0` are *not* `0`). Otherwise, this list command is skipped.
Call	`if_not_cond( Mask1, Mask0 )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.
	`Mask0` See `Mask1`.
Multi-board Com'd Name	**n_if_not_cond**
Comments	• See also Chapter 9.3.2 "Execution of Conditional Commands", page 317.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**if_cond**, **if_pin_cond**, **if_not_pin_cond**

Undelayed Short List Command	**if_not_fly_x_overflow**
Function	*Conditional command execution for Processing-on-the-fly applications*: **if_not_fly_x_overflow** immediately executes the directly subsequent list command, if the condition in accordance with `Mode` for the x axis is *not* fulfilled. Otherwise, this list command is skipped.
Call	`if_not_fly_x_overflow( Mode )`
Parameters	`Mode` To-be-evaluated condition. As a signed 32-bit value. = 0: Some kind of boundary exceedance occurred (error bit Bit #4 = 1 or error bit Bit #5 = 1). > 0: An overflow occurred (error bit Bit #5 = 1). < 0: An underflow occurred (error bit Bit #4 = 1).
Multi-board Com'd Name	**n_if_not_fly_x_overflow**
Comments	• For usage of **if_not_fly_x_overflow**, see Section "Customer-Defined Monitoring Area", page 265. • The error bits queried in accordance with `Mode` (error bit Bit #4 and/or error bit Bit #5) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_marking_info**, **set_fly_limits**, **clear_fly_overflow**, **clear_fly_overflow_ctrl**, **if_fly_x_overflow**

Undelayed Short List Command	**if_not_fly_y_overflow**
Function	*Conditional command execution for Processing-on-the-fly applications*: **if_not_fly_y_overflow** immediately executes the directly subsequent list command, if the condition in accordance with `Mode` for the y axis is *not* fulfilled. Otherwise, this list command is skipped.
Call	`if_not_fly_y_overflow( Mode )`
Parameters	`Mode` To-be-evaluated condition. As a signed 32-bit value. = 0: Some kind of boundary exceedance occurred (error bit Bit #6 = 1 or error bit Bit #7 = 1). > 0: An overflow occurred (error bit Bit #7 = 1). < 0: An underflow occurred (error bit Bit #6 = 1).
Multi-board Com'd Name	**n_if_not_fly_y_overflow**
Comments	• For usage of **if_not_fly_y_overflow**, see Section "Customer-Defined Monitoring Area", page 265. • The error bits queried in accordance with `Mode` (error bit Bit #6 and/or error bit Bit #7) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_marking_info**, **set_fly_limits**, **clear_fly_overflow**, **clear_fly_overflow_ctrl**, **if_fly_y_overflow**

Undelayed Short List Command	**if_not_fly_z_overflow**
Function	*Conditional command execution for Processing-on-the-fly applications*: **if_not_fly_z_overflow** immediately executes the directly subsequent list command, if the condition in accordance with `Mode` for the z axis has not been fulfilled. Otherwise, this list command is skipped.
Call	`if_not_fly_z_overflow( Mode )`
Parameters	`Mode` To-be-evaluated condition. As a signed 32-bit value. = 0: Some kind of boundary exceedance occurred (error bit Bit #24 = 1 or error bit Bit #25 = 1). > 0: An overflow occurred (error bit Bit #25 = 1). < 0: An underflow occurred (error bit Bit #24 = 1).
Multi-board Com'd Name	**n_if_not_fly_z_overflow**
Comments	• For usage of **if_not_fly_z_overflow**, see also Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 264. • The error bits queried in accordance with `Mode` (error bit Bit #24 and/or error bit Bit #25) are reset.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_marking_info**, **set_fly_limits_z**, **clear_fly_overflow**, **clear_fly_overflow_ctrl**, **if_fly_z_overflow**

Undelayed Short List Command	**if_not_pin_cond**
Function	*Conditional command execution*: **if_not_pin_cond** immediately executes the directly following list command, if the current `IOvalue` at the LASER Connector's 2-bit digital input port *does not meet* the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are *not* `1` *or* the bits specified in `Mask0` are *not* `0`). Otherwise, this list command is skipped.
Call	`if_not_pin_cond( Mask1, Mask0 )`
Parameters	`Mask1` 2-bit mask. As an unsigned 32-bit value. Only the lower 2 bits are evaluated.
	`Mask0` See `Mask1`.
Multi-board Com'd Name	**n_if_not_pin_cond**
Comments	• See also Chapter 9.3.2 "Execution of Conditional Commands", page 317.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**if_pin_cond**, **if_cond**, **if_not_cond**

Undelayed Short List Command	**if_pin_cond**
Function	*Conditional command execution*: **if_pin_cond** immediately executes the directly following list command, if the current `IOvalue` at the LASER Connector's 2-bit digital input port meets the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are `1` and the bits specified in `Mask0` are `0`). Otherwise, this list command is skipped.
Call	`if_pin_cond( Mask1, Mask0 )`
Parameters	`Mask1` 2-bit mask. As an unsigned 32-bit value. Only the lower 2 bits are evaluated.
	`Mask0` See `Mask1`.
Multi-board Com'd Name	**n_if_pin_cond**
Comments	• See also Chapter 9.3.2 "Execution of Conditional Commands", page 317.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**if_not_pin_cond**, **if_cond**, **if_not_cond**

Ctrl Command	init_fly_2d
Function	Initializes the encoder reference values for 2D encoder compensation of a subsequent **set_fly_2d** Processing-on-the-fly application and resets both encoder values.
Call	`init_fly_2d( OffsetX, OffsetY, No )`
Parameters	`OffsetX` Reference value of the x axis encoder. As a signed 32-bit value. Allowed value range: depends on the compensation table loaded with **load_fly_2d_table**. Default values (after **load_program_file**): = 0.
	`OffsetY` Like `OffsetX` (analogously).
	`No` Number of the 2D compensation table to be used. See also page 259.
Multi-board Com'd Name	**n_init_fly_2d**
Comments	• The final encoder value for the 2D correction value + reference value must not exceed the allowed range. Otherwise, clipping occurs (see also **load_fly_2d_table**). • `No` = 1: use table 1. • `No` = 2: use table 2. • `No` = 0: use the table that has been used up to now • `No` > 2: do not use a table. • The tables to be used must have been validly loaded by **load_fly_2d_table** beforehand. • With `No` > 2, the table remains validly loaded. • By `load_fly_2d_table( Name` = NULL), the table becomes invalid. • **init_fly_2d** is allowed in Boot Phase 2, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	Changed functionality. • Parameter `No`.
Version Info	Available as of DLL 614, OUT 614, RBF 619.
References	**init_fly_2d_list**, **set_fly_2d**, **load_fly_2d_table**, **get_fly_2d_offset**

Normal List Command	init_fly_2d_list	
Function	Like init_fly_2d, but a list command.	
Call	`init_fly_2d_list( OffsetX, OffsetY, No )`	
Parameters	`OffsetX`	Like init_fly_2d.
	`OffsetY`	Like init_fly_2d.
	`No`	Like init_fly_2d.
Multi-board Com'd Name	n_init_fly_2d_list	
Comments	• See init_fly_2d.	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 643, OUT 645.	
References	init_fly_2d	

Ctrl Command	init_rtc6_dll
Function	Initializes control of the installed RTC6 boards for a user program.
Call	`InitErrorNo = init_rtc6_dll()`
Result	Error code. As an unsigned 32-bit value. If multiple errors occurred simultaneously, then multiple bits are set. The assignment between bit numbers, error types and error constants is identical to those for **get_error**.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• **init_rtc6_dll** *must be called by each user program at the beginning*, so that an RTC6 board can be addressed by the user program at all. • **init_rtc6_dll** searches for all present RTC6 PCIe Boards (not for RTC6 Ethernet Boards! – see Chapter 16.5.2 "About Searching for RTC6 Ethernet Boards", page 984 on this topic) and establishes the RTC6 board management. If no RTC6 board is found (for example, when only RTC6 Ethernet Boards are operated), then error code `RTC6_NO_PCIE_CARD_FOUND` is returned (see **get_error**). • **init_rtc6_dll** automatically assigns the user program access rights to the found boards (as by **acquire_rtc**), if access rights are not already assigned to another user program (any number of boards and applications may be used, but each particular board cannot be used simultaneously by multiple applications). The first initialization acquires all found boards for itself. Subsequent initializations started by other applications result in return of an `RTC6_ACCESS_DENIED` error code by **init_rtc6_dll**. The boards are then only accessible by these applications through RTC6 DLL-internal functions that require no access rights – for example, **get_error**, **get_last_error** or **select_rtc** (most multi-board commands, too, are only callable for boards with existing access rights). If a user program has access rights for a board, then the user program must first explicitly release its access rights with **release_rtc** or **free_rtc6_dll** before the board can be used by another user program by **init_rtc6_dll** or **acquire_rtc**. An `RTC6_ACCESS_DENIED` error code is returned if access has been denied by at least one of the found boards. Which board(s) denied access can be determined by `n_get_error( CardNo )` (`CardNo` from 1 to the number of found boards) or directly after **init_rtc6_dll** (before the next command) by `n_get_last_error( CardNo )`. • If a board is acquired by **init_rtc6_dll** (as by **acquire_rtc**), then a version compatibility check is performed. If a version error is detected, then access to the board is denied (return code `RTC6_ACCESS_DENIED`|`RTC6_VERSION_MISMATCH`). • Only one user program can perform initialization at any one time. Subsequent initializations started by other applications wait until the current initialization is complete.

Ctrl Command	init_rtc6_dll
Comments (cont'd)	• If a user program calls **init_rtc6_dll** multiple times, then the RTC6 board management created by the prior call is deleted for this user program and the originally-assigned access rights canceled. The RTC6 board management is then newly created and access rights are newly granted again. • Each initialization of a user program by **init_rtc6_dll** causes the RTC6 DLL-internal numbers for all found RTC6 boards to be newly reassigned. The relationship between the RTC6 DLL-internal numbers and the installed boards can be determined by **get_serial_number**. When the accessible board with the smallest RTC6 DLL-internal number is initialized, it then simultaneously becomes the active board and the target of non-multi-board commands. **select_rtc** can be called at anytime to change the active board. If no access rights exist for any board, then the board with the highest RTC6 DLL-internal number (see **rtc6_count_cards**) is the active board, in which case only RTC6 DLL-internal commands that require no access rights can be used. • Also observe Chapter 6.7.1 "Notes on Board Acquisition by a User Program", page 131. • **init_rtc6_dll** is available even without explicit access rights to any particular RTC6 board. • After initialization of the RTC6 DLL with **init_rtc6_dll**, the RTC6 Standard Mode is set by default. After that, a different RTC6 DLL operational mode can be set, see **set_rtc4_mode** and **set_rtc5_mode**. • **init_rtc6_dll** does not trigger an initialization of RTC6 boards. Only **load_program_file** performs an initialization of RTC6 boards.
RTC4→RTC6	New command.
RTC5→RTC6	New command. The functionalities of **init_rtc6_dll** and the RTC5 command **init_rtc5_dll** are identical.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_rtc4_mode**, **set_rtc5_mode**

Normal List Command	**jump_abs**	
Function	Moves the output point (for the laser focus) along a 2D vector at jump speed from the current position to the specified position (absolute coordinate values) within a 2D Image Field.	
Call	`jump_abs( X, Y )`	
Parameters	`X`	Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`Y`	Like `X` (analogously).
Multi-board Com'd Name	**n_jump_abs**	
Comments	• If the jump speed has not been previously explicitly set by **set_jump_speed** or **set_jump_speed_ctrl**, then the jump is executed at a predefined jump speed of 10000 *bits/ms*. • The Signals for "Laser Active" Operation are switched off before the jump and remain off during the jump. • After a Jump Command, a (variable) Jump Delay is inserted. Exception: a zero-length jump vector's subsequent (variable) Jump Delay is not executed. However, **jump_abs** itself still requires a 10 μs clock cycle for execution. • Previously accumulated coordinate transformations become effective by **jump_abs**, see list item "With `at_once` = 2 …", page 235.	
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the values specified for `X` and `Y` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_jump_speed**, **set_scanner_delays**, **jump_rel**, **timed_jump_abs**, **goto_xy**	

Normal List Command	**jump_abs_3d**	
Function	Moves the output point (of the laser focus) along a 3D vector at jump speed from the current position to the specified position (absolute coordinate values) within the 3D Image Field.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **jump_abs_3d** has the same effect as **jump_abs**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.	
Call	`jump_abs_3d( X, Y, Z )`	
Parameters	`X`	Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`Y`	Like `X` (analogously).
	`Z`	Like `X` (analogously), except Allowed value range: [–524,288…+524,287].
Multi-board Com'd Name	**n_jump_abs_3d**	
Comments	• Except for the additional motion in the third dimension, **jump_abs_3d** functions similarly to **jump_abs** (see comments there). • The `DirectMove3D` parameter of **set_delay_mode** determines the type of z axis motion (linear or with stepwise correction).	
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the values specified for `X`, `Y` and `Z` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range. In RTC5 Compatibility Mode, the RTC6 multiplies the value specified for `Z` by 16. The allowed value range decreases accordingly.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**jump_abs**, **jump_rel_3d**, **timed_jump_abs_3d**	

Normal List Command	**jump_abs_drill**
Call	`jump_abs_drill( X, Y, DrillTime )`
Multi-board Com'd Name	**n_jump_abs_drill**
Comments	• This command is described, for example, in the manual for the intelli*DRILL* $_{de}$ II 20 Module.

Normal List Command	**jump_abs_drill_2**
Call	`jump_abs_drill_2( X, Y, DrillTime, XOff, YOff )`
Multi-board Com'd Name	**n_jump_abs_drill_2**
Comments	• This command is described, for example, in the manual for the intelli*DRILL* $_{de}$ II 20 Module.

Normal List Command	**jump_rel**	
Function	Moves the output point (of the laser focus) along a 2D vector at jump speed from the current position to the specified position (relative coordinate values) within a 2D Image Field.	
Call	`jump_rel( dX, dY )`	
Parameters	`dX`	Relative x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field, for example, for (enabled) Processing-on-the-fly applications.
	`dY`	Like `dX` (analogously).
Multi-board Com'd Name	**n_jump_rel**	
Comments	• The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **jump_rel** is identical to **jump_abs** (see comments there).	
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `dX` and `dY` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**jump_abs**, **jump_rel_3d**, **timed_jump_rel**	

Normal List Command	jump_rel_3d
Function	Moves the output point (of the laser focus) along a 3D vector at jump speed from the current position to the specified position (relative coordinate values) within the 3D Image Field.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **jump_rel_3d** has the same effect as **jump_rel**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.
Call	`jump_rel_3d( dX, dY, dZ )`
Parameters	`dX` Relative x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`dY` Like `dX` (analogously).
	`dZ` Like `dX` (analogously), except Allowed value range: [–524,288…+524,287].
Multi-board Com'd Name	**n_jump_rel_3d**
Comments	• The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **jump_rel_3d** is identical to **jump_abs_3d** (see comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `dX`, `dY` and `dZ` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**jump_abs_3d**, **jump_abs**, **jump_rel**, **timed_jump_rel_3d**

Normal List Command	jump_rel_drill
Call	`jump_rel_drill( dX, dY, DrillTime )`
Multi-board Com'd Name	**n_jump_rel_drill**
Comments	• This command is described, for example, in the manual for the intelli*DRILL* $_{de}$ II 20 Module.

Normal List Command	jump_rel_drill_2
Call	`jump_rel_drill_2( dX, dY, DrillTime, XOff, YOff )`
Multi-board Com'd Name	**n_jump_rel_drill_2**
Comments	• This command is described, for example, in the manual for the intelli*DRILL* $_{de}$ II 20 Module.

Variable List Command	**laser_on_list**
Function	Turns on the Signals for "Laser Active" Operation for a specified time interval.
Call	`laser_on_list( Period )`
Parameters	`Period` Time interval. In bits. As an unsigned 32-bit value. 1 bit equals 10 μs. Allowed value range: 0 ≤ `Period` ≤ (2^{31}–1).
Multi-board Com'd Name	**n_laser_on_list**
Comments	• Bit #31 of `Period` is ignored (see also Comments at **laser_on_pulses_list**). • The Signals for "Laser Active" Operation must first be selected with **set_laser_mode**, defined by further commands, and enabled by **set_laser_control** or **enable_laser** before they can be switched on with **laser_on_list**, see Chapter 7.4 "Laser Control", page 189. • While the Laser Control Signals are turned on, the set position of the scanners is not changed. The next list command is executed when the programmed time interval has passed. • The currently set LaserOn Delay is applied at the beginning of the programmed time interval: The Laser Control Signals turn on after a LaserOn Delay. – As with other [*]mark[*] Commands (for example, **mark_abs**), the Laser Control Signals do *not* explicitly turn off at the end of the time interval, but instead only turn off with a subsequent list command (for example, **jump_abs** or **list_nop**). The currently set LaserOff Delay is applied. • **laser_on_list** is useful for marking single dots, see Chapter 7.1.3 "Marking Single Dots", page 144. • Wobbel Mode, see Chapter 8.4 "Wobbel Mode", page 241, is retained but ignored. • For `Period` = 0 the Laser Control Signals do not turn on. **laser_on_list** is a Short List Command then. • **laser_on_list** does not trigger a Scanner Delay. To wait until the laser (after a LaserOff Delay) is actually off before a jump, then explicitly a **long_delay** should be inserted.
RTC4→RTC6	Unchanged functionality. In addition: increased value range.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**laser_on_pulses_list**, **long_delay**, **para_laser_on_pulses_list**

Variable List Command	**laser_on_pulses_list**	
Function	Turns on the LASERON laser control signal for the specified number of external signal pulses, but for no longer than the specified time interval. For `Pulses` > 65,535 the function of **laser_on_pulses_list** is identical to **laser_on_list**.	
Call	`laser_on_pulses_list( Period, Pulses )`	
Parameters	`Period`	Time interval. In bits. As an unsigned 32-bit value. 1 bit equals 10 μs. Allowed value range: 0 ≤ `Period` ≤ (2^{32}–1).
	`Pulses`	Number of external signal pulses. As an unsigned 32-bit value. Allowed value range: 0 ≤ `Pulses` ≤ 65,535 or larger (see comments below).
Multi-board Com'd Name	**n_laser_on_pulses_list**	
Comments	• **laser_on_pulses_list** is useful for marking single dots in Laser Mode 6, but also effective in the other laser modes. • The external pulses must be supplied as TTL pulses at DIGITAL IN1 of the LASER Connector (see Section "2-Bit Digital Input Port", page 77). By **set_laser_control**(Bit #5), you can specify whether signal pulses should be counted at rising or falling edges. • If `Period` = 0, then **laser_on_pulses_list** has no effect. **laser_on_pulses_list** is a Short List Command then. • If 0 < `Period` ≤ (2^{31}–1), then **laser_on_pulses_list** duration is always `Period` clocks (that is, `Period` × 10 μs), even if the specified number of external signal pulses expire in a shorter time interval. • If 2^{31} ≤ `Period` ≤ (2^{32}–1), **laser_on_pulses_list** maximum duration is (`Period` – 2^{31}) clocks ((`Period` – 2^{31}) × 10 μs). Here, however, **laser_on_pulses_list** terminates as soon as the specified number of external signal pulses has been detected. • If `Pulses` > 65,535, then **laser_on_pulses_list**'s function is identical to **laser_on_list** (external signal pulses are not taken into account, see comments there). Otherwise (for 0 ≤ `Pulses` ≤ 65,535), **laser_on_pulses_list** (in contrast to **laser_on_list**) *do not* toggle the Laser Control Signals between "laser active" and "laser standby" operation, but instead only switches the LASERON signal, whereby Laser Delays are not taken into account. Likewise during this command, unexpired Laser Delays activated by prior commands have no effect (though their effect resume when the LASERON signal switches off again after the final pulse or after `Period`).	

Variable List Command	laser_on_pulses_list
Comments (cont'd)	• If 1 ≤ `Pulses` ≤ 65,535, then the LASERON signal does switch on upon the edge (according the polarity set by **set_laser_control**(Bit #5)) of the first external pulse (unless it is already on due to an unexpired LaserOff Delay) and remains on for the specified number of pulses, but no longer than until the end of the number of 10 µs periods specified by `Period`. Users should ensure to define `Period` large enough for processing `Pulses` number of signal pulses in this time interval. If the DIGITAL IN1 input port does not receive any pulses, then **laser_on_pulses_list** does not alter the LASERON signal. If more signal pulses than specified by `Pulses` are received during the time interval defined by `Period`, then the surplus pulses are ignored. • If `Pulses` = 0, then **laser_on_pulses_list** has no effect (the LASERON signal is not switched on). **laser_on_pulses_list** is a Short List Command then. • The LASERON signal must first be defined and enabled by **set_laser_control** or **enable_laser** before it can be switched on with **laser_on_pulses_list**, see Chapter 7.4 "Laser Control", page 189. • While t**laser_on_pulses_list** is executed, the set position of the scanners is not changed. The next list command is executed when the programmed time interval `Period` has passed. • The Wobbel Mode, see Chapter 8.4 "Wobbel Mode", page 241, is retained but ignored. • Any softstarts that were defined are not executed.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**laser_on_list**, **para_laser_on_pulses_list**

Ctrl Command	laser_signal_off
Function	Turns off the Signals for "Laser Active" Operation immediately.
Call	`laser_signal_off()`
Multi-board Com'd Name	**n_laser_signal_off**
Comments	• **laser_signal_off** is intended for direct laser control in combination with **laser_signal_on**. • **laser_signal_off** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set • **laser_signal_off** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) – the INTERNAL-BUSY list execution status is set
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**laser_signal_on**

Normal List Command	laser_signal_off_list
Function	Like **laser_signal_off**, but a list command.
Call	`laser_signal_off_list()`
Multi-board Com'd Name	**n_laser_signal_off_list**
Comments	• –
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**laser_signal_off**

Ctrl Command	laser_signal_on
Function	Turns on the Signals for "Laser Active" Operation immediately.
Call	`laser_signal_on()`
Multi-board Com'd Name	**n_laser_signal_on**
Comments	• *Caution! Check the beam path before turning on the laser!* • The Signals for "Laser Active" Operation must first be selected by **set_laser_mode**, defined by further commands, and enabled by **set_laser_control** or **enable_laser** before they can be switched on with **laser_signal_on**, see Chapter 7.4 "Laser Control", page 189. • **laser_signal_on** is intended for turning on the laser directly, for example, for alignment purposes. • The Signals for "Laser Active" Operation must be turned off by **laser_signal_off**. • **laser_signal_on** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **laser_signal_on** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set)
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**laser_signal_off**

Normal List Command	laser_signal_on_list
Function	Like **laser_signal_on**, but a list command and never ignored.
Call	`laser_signal_on_list()`
Multi-board Com'd Name	**n_laser_signal_on_list**
Comments	• –
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**laser_signal_on**

Undelayed Short List Command	**list_call**
Function	Causes an unconditional jump to a subroutine that starts at the specified absolute address (in any desired location within RTC6 List Memory).
Call	`list_call( Pos )`
Parameters	`Pos` Absolute jump address [0…(2^{23}–1)]. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_list_call**
Comments	• The first command of a subroutine called by **list_call** is executed (possibly after a **list_continue**) immediately and without delay. Nested or recursive calls are also possible, up to a depth of 63, see also Chapter 6.5.1 "Subroutines", page 115. • Each subroutine must be terminated by **list_return** so that after the subroutine (including the terminating **list_return**) has been processed, execution continues with the command that follows the subroutine-call command. This, too, executes (after a possible **list_continue**) immediately and without delay. If **set_end_of_list** is encountered instead of the expected **list_return**, then list execution terminates or – if previously activated – an automatic list change takes place (for the latter, the current list status is a decisive factor as described below). Under no circumstances does program flow then return again to the calling location, even in the case of nested subroutine calls. Any not-yet-completed **mark_text** does not execute to completion. If the end of a RTC6 List Memory area ("List 1" or "List 2") is reached without having encountered a **list_return** or **set_end_of_list**, then execution continues at the start of the current list. If such a situation occurs in the protected RTC6 List Memory area "List 3", then a compulsory **list_return** command is inserted and executed. • The **list_call** command is replaced by a **list_nop** if `Pos` > (2^{23}–1) or if `Pos` is also the current address (**get_last_error** return code `RTC6_PARAM_ERROR`). • If a called subroutine executes a further **list_call** to the address of the calling **list_call** command (recursive call), then the resulting endless loop is terminated as soon as the 63-nested-call upper limit is reached. Further **list_call** commands are then ignored and the next command is instead executed. • If the subroutine starts directly at the address which follows **list_call**, then the subroutine is executed once again after **list_return** (see also comments on a missing corresponding function call in the **list_return** command description). The next processed command is the one which follows after **list_return** (see also **list_repeat**…**list_until**). This bypasses the possibly unwanted list processing.

Undelayed Short List Command	list_call
Comments (cont'd)	• The BUSY list status readable by **read_status** is altered by **list_call** if the called address is in the RTC6 List Memory area ("List 1" or "List 2"). If this address is instead in protected RTC6 List Memory area "List 3", then the calling location's list status is retained because the protected area does not have its own list status. During execution of a subroutine in protected memory, **get_status** too returns the calling location's List Execution Status, and **get_out_pointer** returns the position of the calling **list_call** as the output pointer's position. • Absolute Vector Commands and "Arc" Commands execute absolutely after **list_call** is called. If the subroutine needs to execute at various locations within the Image Field, then either the subroutine can only contain relative [*]mark[*] Commands, "Arc" Commands or Jump Commands or **list_call_abs** must be used instead. • If **list_call** is at the last possible memory position in a list ("List 1" or "List 2"), then – even if automatic list changing has been previously enabled – execution continues at the start of the same list after processing of the called subroutine. • `list_call( Pos )` is synonymous with `list_call_repeat( Pos, 1 )`.
RTC4→RTC6	Basically unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_continue**, **list_repeat**, **list_return**, **list_until**, **list_call_abs**, **list_call_cond**, **list_call_repeat**, **sub_call**

Undelayed Short List Command	**list_call_abs**
Function	Causes an unconditional jump to a subroutine that starts at the specified absolute address (in any desired area of RTC6 List Memory). In the called subroutine, any absolute Vector Commands and "Arc" Commands receive an offset (based on the current coordinates at the time of the call).
Call	`list_call_abs( Pos )`
Parameters	`Pos` Absolute jump address [0…(2^{23}–1)]. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_list_call_abs**
Comments	• **list_call_abs** is basically identical to **list_call** (see comments there). However, **list_call_abs** results in a different execution of absolute Vector Commands and "Arc" Commands within the called subroutine. When **list_call_abs** executes, an offset is established for the called subroutine and set according to the current position. Thereby, the current position is automatically considered when absolute Vector Commands and "Arc" Commandes of the called subroutine are subsequently executed. Nested calls are taken into account when the offset is determined (with **list_return**, the previous offset values are re-established). Subroutines can thus contain absolute Vector Commands and "Arc" Commands even if they are intended to be repeated in different parts of the Image Field. • If the called subroutine contains no absolute commands, then there is no difference between **list_call_abs** and **list_call**. • `list_call_abs( Pos )` is synonymous with `list_call_abs_repeat( Pos, 1 )`.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_call**, **list_call_abs_cond**, **list_call_abs_repeat**

Undelayed Short List Command	**list_call_abs_cond**
Function	*Conditional subroutine call (AbsCall)*: **list_call_abs_cond** executes **list_call_abs**(`Pos`), if the current `IOvalue` at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector meets the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are `1` and the bits specified in `Mask0` are `0`). Otherwise, the directly following list command is immediately executed.
Call	`list_call_abs_cond( Mask1, Mask0, Pos )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the least significant 16 bits are evaluated.
	`Mask0` See `Mask1`.
	`Pos` Absolute jump address [0…(2^{23}–1)]. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_list_call_abs_cond**
Comments	• See **list_call_abs**. • See also Chapter 9.3.2 "Execution of Conditional Commands", page 317.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_call_abs**

Undelayed Short List Command	**list_call_abs_repeat**
Function	Causes an unconditional jump to a subroutine that starts at the specified absolute address (in any desired area of RTC6 List Memory) and executes its body several times.
Call	`list_call_abs_repeat( Pos, Number )`
Parameters	`Pos` Absolute jump address [0…(2^{23}–1)]. As with **list_call_abs**: As an unsigned 32-bit value. `Number` Number of repetitions. As an unsigned 32-bit value. `Number` = 0 is treated as `Number` = 1.
Multi-board Com'd Name	**n_list_call_abs_repeat**
Comments	• `list_call_abs( Pos )` is synonymous with `list_call_abs_repeat( Pos, 1 )`. • **list_call_abs_repeat** avoids an empty cycle at the repetition, which otherwise inevitably occurs with `list_call_abs`…`list_call_abs` or `list_repeat`…`list_call_abs`…`list_until` constructions. • See **list_call_repeat** and **list_call_abs**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_call**, **list_call_abs**, **list_call_abs_cond**, **list_call_repeat**, **list_repeat**, **list_until**

Undelayed Short List Command	**list_call_cond**
Function	*Conditional subroutine call*: **list_call_abs_cond** executes **list_call**(`Pos`), if the current `IOvalue` at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector meets the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are `1` and the bits specified in `Mask0` are `0`). Otherwise, the directly following list command is immediately executed.
Call	`list_call_cond( Mask1, Mask0, Pos )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.
	`Mask0` See `Mask1`.
	`Pos` Absolute jump address [0…(2^{23}–1)]. As an unsigned 32-bit value.
Multi-board Com’d Name	**n_list_call_cond**
Comments	• See **list_call**. • See also Chapter 9.3.2 ”Execution of Conditional Commands”, page 317.
RTC4→RTC6	Basically unchanged functionality (see **list_call**).
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_call**

Undelayed Short List Command	**list_call_repeat**
Function	Causes an unconditional jump to a subroutine that starts at the specified absolute address (in any desired area of RTC6 List Memory) and executes its body several times.
Call	`list_call_repeat( Pos, Number )`
Parameters	`Pos` Absolute jump address [0…(2^{23}–1)]. As with **list_call**: As an unsigned 32-bit value. `Number` Number of repetitions. As an unsigned 32-bit value. `Number` = 0 is treated as `Number` = 1.
Multi-board Com'd Name	**n_list_call_repeat**
Comments	• `list_call( Pos )` is synonymous with `list_call_repeat( Pos, 1 )`. • **list_call_repeat** avoids an empty cycle at the repetition, which otherwise inevitably occurs with `list_call`…`list_call` or `list_repeat`…`list_call`…`list_until` constructions. • By **list_call_repeat**, for example, trajectories (see Glossary entry on page 31) from micro_vector[*] Commands for runup curves and coast down curves can be seamlessly joined together with shapes from subroutines. • An empty cycle must be inserted if at **list_call_repeat** another subroutine call follows immediately (nested calls). This can be avoided, if there is for example, at least one micro_vector[*] Command between two subroutine calls. • The subroutine start `Pos` can be located in any part of the RTC6 List Memory area. This applies in particular directly after **list_call_repeat**. Thus, the complete RTC6 List Memory can continuously be used for list commands without reserving a special area for protected subroutines. • After a **list_return** the next processed command is the one which follows directly after **list_call_repeat**. However, if the subroutine start follows directly after **list_call_repeat**, then the next processed command is the one which follows directly after **list_return**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_call**, **list_call_abs**, **list_call_abs_repeat**, **list_repeat**, **list_return**, **list_until**, **micro_vector_abs**, **micro_vector_abs_3d**, **micro_vector_rel**, **micro_vector_rel_3d**

Normal List Command	**list_continue**
Function	Inserts a null operation (no operation) into the RTC6 List Memory.
Call	`list_continue()`
Multi-board Com'd Name	**n_list_continue**
Comments	• Execution of **list_continue** (as does **list_nop**) requires 10 μs. • When **list_continue** immediately follows a short list command, it ensures (as does **list_nop**) that the subsequent list command only executes in the next 10 μs clock cycle (the short list command's "effective" execution time is then 10 μs). Unlike **list_nop**, however, **list_continue** neither modifies Laser Control Signals nor pauses for Scanner Delays. In contrast to **list_nop**, therefore, **list_continue** allows postponing short list commands to the next 10 μs clock cycle without interrupting Polylines by switching off the laser. • With exceedance of the maximum allowed number of directly consecutive short list commands, an empty cycle (which exactly corresponds **list_continue**) is automatically inserted. • You can also use **list_continue** to separate from each other several outputs to the same output port. Without **list_continue**, for example, multiple directly consecutive **write_da_x_list** calls (short list commands) would occur in a single 10 μs clock cycle, whereby some values to the analog output port would never get outputted. • The RTC6 never uses **list_continue** as a placeholder for other (rejected) list commands, but instead always uses **list_nop**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_nop**

Undelayed Short List Command	**list_jump_cond**
Function	*Conditional absolute list jump*: **list_jump_cond** executes **list_jump_pos**(`Pos`), if the current `IOvalue` at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector meets the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are `1` and the bits specified in `Mask0` are `0`). Otherwise, the directly following list command is immediately executed.
Call	`list_jump_cond( Mask1, Mask0, Pos )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.
	`Mask0` See `Mask1`.
	`Pos` Absolute jump address [0…(2^{23}–1)]. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_list_jump_cond**
Comments	• **list_jump_cond** is synonymous with **list_jump_pos_cond** (see comments there).
RTC4→RTC6	Basically unchanged functionality. However: • Jumps into or out from the protected RTC6 List Memory area “List 3” are not allowed (illegal Jump Commands are ignored during processing, see also **list_jump_pos**).
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_jump_pos**, **list_jump_pos_cond**

Undelayed Short List Command	**list_jump_pos**
Function	Execution produces an unconditional jump to the specified RTC6 List Memory address. The next command there is executed immediately without delay.
Call	`list_jump_pos( Pos )`
Parameters	`Pos` Absolute jump address [0…(2^{23}–1)]. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_list_jump_pos**
Comments	• For `Pos`, an absolute address can be specified within the configured RTC6 List Memory ("List 1" and "List 2"), but not within the protected "List 3" area or completely outside of the RTC6 List Memory area. • Jumps into or out from the protected RTC6 List Memory area "List 3" are not allowed. • Illegal Jump Commands are transmitted unaltered to the RTC6, but are ignored during processing. Instead, the next command is executed. Hence, the user program does probably no longer perform as expected. Therefore, Jump Commands must be carefully programmed (see also Chapter 6.5.3 "Jumps", page 123). • Jump Commands initiating a jump to themselves (`Pos` = list position of the Jump Command) are also ignored at runtime to prevent an infinite loop that excludes further activities (and that could only be halted by **stop_execution** or an External Stop). • Decisive are the runtime conditions. When reconfiguring RTC6 List Memory or converting a subroutine, an originally valid jump address might become invalid due to new list boundaries or a relocated subroutine storage position. • After a jump to another list area, the status information might under some circumstances not be correct (see also Chapter 6.4.2 "List Status", page 110). • The BUSY list status of the two lists is alternatingly set by **list_jump_pos**, the USED list status of the two lists remains unchanged (see **read_status**). • **list_jump_pos** is synonymous with **set_list_jump**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_list_jump**, **list_jump_rel**, **list_jump_pos_cond**

Undelayed Short List Command	**list_jump_pos_cond**
Function	*Conditional absolute list jump*: **list_jump_pos_cond** executes **list_jump_pos**(`Pos`), if the current `IOvalue` at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector meets the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are `1` and the bits specified in `Mask0` are `0`). Otherwise, the directly following list command is immediately executed.
Call	`list_jump_pos_cond( Mask1, Mask0, Pos )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.
	`Mask0` See `Mask1`.
	`Pos` Absolute jump address [0…(2^{23}–1)]. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_list_jump_pos_cond**
Comments	• See **list_jump_pos**. • Unlike the rules for preventing endless loops (see **list_jump_pos**), jumps by **list_jump_pos_cond** are allowed even if they are to their own address (`Pos` = list position of the Jump Command), for example, to wait for confirmation of a signal. • **list_jump_pos_cond** is synonymous with **list_jump_cond**. • See also Chapter 9.3.2 "Execution of Conditional Commands", page 317.

Examples (Pascal)	• wait until Bit #3 of the input port turns HIGH (= loop while the bit is *LOW*):
	`list_jump_pos_cond(0, $0008, get_input_pointer);`
	• skip the next two list commands if the state of the input port is `xxxx xxxx xxxx 0110`:
	`list_jump_pos_cond(6, 9, get_input_pointer + 3);`
	• See also Section "Example Code (Pascal)", page 318.

RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_jump_pos**

Undelayed Short List Command	**list_jump_rel**
Function	Execution produces an unconditional jump to the specified address within the current list. The next command there is executed immediately without delay.
Call	`list_jump_rel( Pos )`
Parameters	`Pos` Jump distance $[(-2^{23}+1)...(2^{23}-1)]$. As a signed 32-bit value.
Multi-board Com'd Name	**n_list_jump_rel**
Comments	• **list_jump_rel** enables implementation of branching (for example, "if-then-else") independently of the command's list position, in particular also coded independently of the list number because of relative addressing. • The current list input pointer can be queried by **get_list_pointer**. • **list_jump_rel** is usable in all RTC6 List Memory areas even the protected RTC6 List Memory "List 3". • When specifying a jump distance within "List 1" or "List 2" or for non-indexed subroutines within "List 3" be sure that the jump does not exceed the boundaries of the corresponding RTC6 List Memory area. • If **list_jump_rel** is used in an indexed subroutine or character set, then also be sure that the jump does not exceed the boundaries of the subroutine or character set. • Illegal Jump Commands are transmitted unaltered to the RTC6, but are ignored during processing. Instead, the next command is executed. Hence, the user program does probably no longer perform as expected. Therefore, Jump Commands must be carefully programmed (see also Chapter 6.5.3 "Jumps", page 123). • Jump Commands initiating a jump to themselves (`Pos` = 0) is also ignored at runtime to prevent an infinite loop that excludes further activities (and that could only be halted by **stop_execution** or an External Stop). • Decisive are the runtime conditions. When reconfiguring RTC6 List Memory or converting a subroutine, an originally valid jump address might become invalid due to new list boundaries or a relocated subroutine storage position.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_jump_pos**, **list_jump_rel_cond**

Undelayed Short List Command	**list_jump_rel_cond**
Function	*Conditional relative list jump*: **list_jump_rel_cond** executes **list_jump_rel**(`Pos`), if the current `IOvalue` at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector meets the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are `1` and the bits specified in `Mask0` are `0`). Otherwise, the directly following list command is immediately executed.
Call	`list_jump_rel_cond( Mask1, Mask0, Pos )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.
	`Mask0` See `Mask1`.
	`Pos` Jump distance $[(-2^{23}+1)\ldots(2^{23}-1)]$. As a signed 32-bit value.
Multi-board Com'd Name	**n_list_jump_rel_cond**
Comments	• See **list_jump_rel**. • Unlike the rules for preventing endless loops (see **list_jump_rel**), jumps by **list_jump_rel_cond** are allowed even if they are to their own address (`Pos` = 0), for example, to wait for confirmation of a signal. • See also Chapter 9.3.2 "Execution of Conditional Commands", page 317.

Examples (Pascal)	• Wait until Bit #3 of the input port turns HIGH (= loop while the bit is *LOW*):
	`list_jump_rel_cond(0, $0008, 0);`
	• Skip the next two list commands if the state of the input port is `xxxx xxxx xxxx 0110`:
	`list_jump_rel_cond(6, 9, 3);`
	• See also Section "Example Code (Pascal)", page 318.

RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_jump_rel**

Undelayed Short List Command	**list_next**
Function	Executes the next list command.
Call	`list_next()`
Multi-board Com'd Name	**n_list_next**
Comments	• **list_next** reads the next list command and executes it immediately, as long as the maximum allowed number of short list commands has not been exceeded. Otherwise, it is executed within the next 10 μs clock cycle. • **list_next** is a proper place holder for another command in the list, because it prevents an extra 10 μs clock cycle, which is unavoidable with other place holders such as **list_nop** or **list_continue**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_nop**, **list_continue**

Normal List Command	**list_nop**
Function	Inserts a null operation (no operation) into the RTC6 List Memory.
Call	`list_nop()`
Multi-board Com'd Name	**n_list_nop**
Comments	• **list_nop** has the same effect as **long_delay**(1). It switches off the Signals for "Laser Active" Operation after a LaserOff Delay and waits for a possible Scanner Delay. Even if no delays need to be waited for, execution of **list_nop** requires 10 μs. • **list_nop** serves as a placeholder for rejected list commands (**get_last_error** return code `RTC6_IGNORED`). • When **list_nop** immediately follows a short list command, it ensures that the subsequent list command only executes in the next 10 μs clock cycle (the short list command's "effective" execution time is then 10 μs).
RTC4→RTC6	The RTC4 command is a pure placeholder and is more like **list_continue**. The RTC6 command switches the Signals for "Laser Active" Operation off and waits for a Scanner Delay.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**long_delay**, **list_continue**

Undelayed Short List Command	**list_repeat**
Function	Initiates repetition of a group of list commands.
Call	`list_repeat()`
Multi-board Com'd Name	**n_list_repeat**
Comments	• See Chapter 6.5.5 "Loops", page 125. • Any still-pending Delayed Short List Command is executed beforehand. • All list commands between **list_repeat** and the next **list_until** call is possibly repeated multiple times. • Each executed **list_repeat**…**list_until** loop requires a full 10 μs clock cycle. Accordingly, the first list command after a repetition is executed with a 10 μs delay. • Nesting up to 8 **list_repeat**…**list_until** loops deep is allowed. Each further **list_repeat** call beyond that limit is ignored as long as no **list_until** call has terminated a loop. • If a list terminates (by **set_end_of_list** or **stop_execution** or /STOP), then all not-yet-fully-processed **list_repeat**…**list_until** loops are deleted. However, this does not occur if an automatic list change (by **auto_change**, **auto_change_pos** or **start_loop**) is active. • Complete **list_repeat**…**list_until** loops can reside within a list or subroutine. Changing between lists and subroutines or between different subroutines is not permitted. Changing between lists is permitted as long as the list change occurs without explicit list termination (by an automatic list change or by an explicit list jump to another list with **list_jump_pos**). • List jumps into a **list_repeat**…**list_until** loop body or from inside a loop to a loop-external location should be avoided. Careless use could compromise loop management integrity so severely that loops do not execute as expected. But subroutine calls from inside a loop are always reliably possible.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_until**

Undelayed Short List Command	**list_return**
Function	Terminates a previously called subroutine, jumps to the calling location and executes the immediately following list command (possibly after a **list_continue**) immediately and without delay.
Call	`list_return()`
Multi-board Com'd Name	**n_list_return**
Comments	• While loading an indexed subroutine, character or text string in the protected RTC6 List Memory area "List 3", the **list_return** command additionally triggers entries into the corresponding internal management table of data such as the starting address. • In contrast, if **list_return** directly follows a **load_sub**, **load_char** or **load_text_table** command, then the affected subroutine, character or text string are deleted from the corresponding internal management table. • If, during loading into the protected RTC6 List Memory area "List 3", indexed subroutines, characters or text strings are *not* terminated with **list_return** before the input pointer is explicitly set to another position, then they are not stored and are thereafter unavailable. • During loading of **list_return**, a flush of the buffered list input is triggered, see Chapter 6.4.1 "Loading Lists", page 108. • If **list_return** follows **load_sub**, **load_char** or **load_text_table**, then the input pointer after **list_return** is invalid. That is, further commands can not be loaded without a prior request for a new input pointer positioning. • If, during processing, a **list_return** is encountered without the prior explicit calling of a subroutine, character or text string, then processing continues at the absolute address 0 (starting address of "List 1"). With nested calls the integrity of the subroutine structure is destroyed.
RTC4→RTC6	No changes to the previous functionality (termination of a subroutine). New: impact during loading into the protected RTC6 List Memory area "List 3".
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_call**, **sub_call**, **load_char**, **load_text_table**, **load_sub**, **list_continue**

Undelayed Short List Command	**list_until**
Function	Terminates repetition of a group of list commands.
Call	`list_until( Number )`
Parameters	`Number` Number of repetitions. As an unsigned 32-bit value. `Number` = 0 is treated like `Number` = 1.
Multi-board Com'd Name	**n_list_until**
Comments	• See Chapter 6.5.5 "Loops", page 125. • Any still-pending Delayed Short List Command is executed beforehand. • All list commands between **list_until** and the most recent preceding **list_repeat** call is executed `Number` number of times. • Nesting up to 8 **list_repeat**...**list_until** loops deep is allowed. Each further **list_until** call for which there has been no preceding **list_repeat** call is ignored. • An empty loop consisting of a **list_repeat** call directly followed by a **list_until** call is terminated immediately and is not repeated. • If a list terminates (by **set_end_of_list** or **stop_execution** or /STOP), then all not-yet-fully-processed **list_repeat**...**list_until** loops are deleted. However, this does not occur if an automatic list change (by **auto_change**, **auto_change_pos** or **start_loop**) is active. • Complete **list_repeat**...**list_until** loops can reside within a list or subroutine. Changing between lists and subroutines or between different subroutines is not permitted. Changing between lists is permitted as long as the list change occurs without explicit list termination (by an automatic list change or by an explicit list jump to another list with **list_jump_pos**). • List jumps into a **list_repeat**...**list_until** loop body or from inside a loop to a loop-external location should be avoided. Careless use could compromise loop management integrity so severely that loops do not execute as expected. But subroutine calls from inside a loop are always reliably possible.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_repeat**

Ctrl Command	load_auto_laser_control
Function	Loads a table with data points from an ASCII text file. Determines – by linear interpolation – the nonlinearity curve (see Section "Loading and Determining the Nonlinearity Curve", page 212) for "Automatic Laser Control" (except Vector-Defined Laser Control), see Chapter 7.4.9 ""Automatic Laser Control"", page 205.
Call	`NoOfDataPoints = load_auto_laser_control( Name, No )`
Parameters	`Name` Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.
	`No` Determines which table in the text file is to be loaded. The parameter corresponds to the extension `<No>` of `[AutoLaserCtrlTable<No>]` at the beginning of the desired table. As an unsigned 32-bit value.
Result	A positive error code in case of an error. The negative number of found data points in case of success. As a signed 32-bit value. Value – Description –1…– 50 – Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored, see also Section "Loading and Determining the Nonlinearity Curve", page 212. –1024 – For `Name` = NULL (see also comments). 1 – No valid data points found (though Table `No` found). 3 – File not found. 4 – DSP memory error. 5 – BUSY list execution status error, board has been BUSY list execution status or INTERNAL-BUSY list execution status, no download (**get_last_error** return code `RTC6_BUSY`). 8 – Board is locked by another user program (**get_last_error** return code `RTC6_ACCESS_DENIED`). 11 – PCI error (**get_last_error** return code `RTC6_SEND_ERROR`), verify error (**get_last_error** return code `RTC6_VERIFY_ERROR`). 13 – The specified table number could not been found in the file.
Multi-board Com'd Name	**n_load_auto_laser_control**
Comments	• The format requirements for the text file's table entries with data points for the nonlinearity curve are described in Section "Loading and Determining the Nonlinearity Curve", page 212. When loading the table, the RTC6 determines suitable values for the entire range of percent values. • **load_auto_laser_control** overwrites any previously loaded nonlinearity curve. • For `Name` = NULL (as during initialization by **load_program_file**), the function *Scale*(*Percent*)=`1.0` is loaded for the complete percent range (no nonlinearity).

Ctrl Command	load_auto_laser_control
Comments (cont'd)	• **load_auto_laser_control** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **load_auto_laser_control** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • During the runtime of **load_auto_laser_control**, External Starts are suppressed. • Before loading a table, **load_auto_laser_control** performs a DSP memory check that produces error code 4 in case of error. • The table can be saved by **create_dat_file**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_auto_laser_control**, **create_dat_file**

Ctrl Command	load_char
Function	Assigns a desired index to a character defined by subsequent list commands, and loads the character into the protected RTC6 List Memory area “List 3”.
Call	`load_char( Char )`
Parameters	`Char` Index of the indexed character. As an unsigned 32-bit value. Allowed value range: [0…1023].
Multi-board Com’d Name	**n_load_char**
Comments	• Up to 1024 indexed characters, hence 4 character sets with 256 indexed characters per set, can be stored. For defining character sets, the following applies: `Char` = character set number × 256 + ASCII number of the character (character sets are numbered 0 to 3). If `Char` > 1023 then **load_char** is ignored (**get_last_error** return code `RTC6_PARAM_ERROR`). • The addresses in the protected RTC6 List Memory area “List 3” where the character definitions are to be stored are automatically determined and internally managed. This management is independent of that for indexed subroutines (see **load_sub**) and text string definitions (see **load_text_table**). • Indexed character definitions must be terminated with a **list_return** call. This is a prerequisite for actual storage of the commands, entry of the start address into the internal management table, and initiating a flush of the buffered list input, see Chapter 6.4.1 ”Loading Lists”, page 108. Otherwise (the input pointer is altered without a preceding **list_return**), the character definition with this index is not available. • An indexed character definition is not stored if the protected RTC6 List Memory area “List 3” has not been previously configured for a sufficient size beyond “List 1” and “List 2”. • If **list_return** is the next command after **load_char**, then the corresponding character definition is deleted from the internal management table. • Observe all notes in Chapter 6.5.2 ”Character Sets and Text Strings”, page 121.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_return**, **load_sub**, **load_text_table**

Ctrl Command	load_correction_file
Function	Loads the specified correction file into RTC6 memory. Then calls automatically **select_cor_table** with the most recently used parameter values (or the default parameter values).
Call	`ErrorNo = load_correction_file( Name, No, Dim )`
Parameters	`Name` Name of the correction file. As a pointer to a \0-terminated ANSI string.
	`No` Number of the correction table. Allowed values: [1…8]. As an unsigned 32-bit value. See also **number_of_correction_tables**.
	`Dim` Determines whether the correction file content is stored as a 2D correction table or (if possible) as a 3D correction table (see also comments). As an unsigned 32-bit value. = 2: 2D and 3D correction files are stored as a 2D correction table (downgrade, if necessary). = 3: If the Option "3D" *is* enabled, 2D and 3D correction files are stored as a 3D correction table (upgrade, if necessary). If the Option "3D" is *not* enabled, like `Dim` = 2. Others: `Dim` has no effect. A 2D correction file is stored as a 2D correction table. A 3D correction file is stored as a 3D correction table, if the Option "3D" *is* enabled, otherwise as a 2D correction table (downgrade).
Result	Error code. As an unsigned 32-bit value. 0 Success. 1 File error (file corrupt or incomplete). 2 Memory error (RTC6 DLL-internal, Windows system memory). 3 File-open error (empty string submitted for `Name` parameter, file not found, etc.). 4 DSP memory error. 5 PCI download error (RTC6 board driver error), Ethernet download error. 8 RTC6 board driver not found (**get_last_error** return code `RTC6_ACCESS_DENIED`). 10 Parameter error (incorrect `No`).

Ctrl Command	load_correction_file	
Result (cont'd)	11	Access error: board reserved for another user program (**get_last_error** return code RTC6_ACCESS_DENIED) or version check has detected an error (OUT or RTC version not compatible to the current RTC6 DLL version, **get_last_error** return code RTC6_ACCESS_DENIED | RTC6_VERSION_MISMATCH).
	12	Warning: 3D correction table or Dim = 3 selected, but the Option "3D" is not enabled. The system subsequently operates as an ordinary 2D system (this warning is only returned, if no other error has occurred).
	13	Busy error: no download, board is BUSY list execution status or INTERNAL-BUSY list execution status (**get_last_error** return code RTC6_BUSY).
	14	PCI upload error (RTC6 board driver error, only with download verification)
	15	Verify error (only with download verification).
Multi-board Com'd Name	**n_load_correction_file**	
Comments	On loading correction tables: • The RTC6 can store 8 different correction tables at the same time, for example, for use in a multiple scan head configuration. • Before storing a correction file, **load_correction_file** performs a DSP memory check that produces error code 4 in case of error. • If the parameter No is out of range, then no correction table is loaded (return value 10). Users can further restrict the permitted range by **number_of_correction_tables**. • The name of the to-be-loaded correction file must be passed to **load_correction_file** as a pointer to a \0-terminated ANSI string. If Name is passed as a NULL pointer, the corresponding table is replaced by a 1-to-1 table (for Dim = 3 and enabled Option "3D" with a 1-to-1 3D correction table, otherwise with a 1-to-1 2D table). An empty string ("") for Name result in an error return code of 3. • If the Option "3D" is *not* enabled (default), then 2D and 3D correction files are stored as 2D correction tables – regardless of the value specified for Dim. The 3D data sections of 3D correction files are then ignored. • If, on the other hand, the Option "3D" is *enabled*, then both 2D and 3D correction tables can be loaded. – For Dim = 2, a 2D table is always stored (the 3D data section of 3D correction files are ignored) and, accordingly, only 2D corrections are calculated (the z axis thereby remains unchanged, as if no Option "3D" has been enabled). – For Dim = 3, if the Option "3D" is enabled then both 2D and 3D correction files are stored as 3D correction tables. 2D correction tables are thereby automatically expanded to incorporate a linear Z correction. The actually suitable Z correction can subsequently be loaded by **load_z_table** or **load_z_table_no**, see Chapter 7.3.4 "3D Image Field", page 175. – All other values for Dim do not change the type of the correction file.	

Ctrl Command	load_correction_file
Comments (cont'd)	On assigning correction tables by **select_cor_table**: • Use the **select_cor_table** or **select_cor_table_list** command to assign one (or two) correction table(s) stored on the RTC6 to the scan head (or to both scan head connectors). • **load_correction_file** automatically calls **select_cor_table** after loading of a correction table. However, if you call **load_correction_file** *before* loading the program file (by **load_program_file**), then the automatic call of **select_cor_table** has no effect. If you call **load_correction_file** *after* **load_program_file**, then the **select_cor_table** call uses the parameter values (`HeadA` = 1, `HeadB` = 0) or the values most recently used (after **load_program_file**) when having called **select_cor_table**. • **load_correction_file** does not return to the user program until the **select_cor_table**-induced jump to the corrected galvanometer scanner position has been completed. See also Notes, page 181. Others: • RTC5/RTC6 correction tables contain parameters that with an RTC4 must be looked-up in the supplied `*_ReadMe.txt` file first, and then must be entered manually into the user program. With the RTC6 these parameters can be read from the currently loaded correction tables (by **get_table_para**) or from the assigned correction tables (by **get_head_para**). Thus, they are directly integrated into a user program. • During the runtime of **load_correction_file**: – No other control commands are executed – External Starts are suppressed • **load_correction_file** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **load_correction_file** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • If an `RTC6_VERSION_MISMATCH` error occurs (return value 11), a RTC6 DLL version appropriate for the program file must be chosen and the board must be made currentless (to unload the program software) or alternatively program files appropriate for the RTC6 DLL version must be reloaded by **load_program_file** (after `RTC6_ACCESS_DENIED`, the single-board command **load_program_file** does not get access for the board). Only afterward (and after the board has been reacquired by **acquire_rtc** or **select_rtc**) **load_correction_file** can be normally executed again.

Ctrl Command	load_correction_file
RTC4→RTC6	Except its basic function and the parameters `Name` and `No`, the functionality of **load_correction_file** has substantially changed: • **load_correction_file** now uses the new parameter `Dim`. This allows, for instance, storage of 3D correction files as 2D correction tables and storage of 2D correction files as 3D correction tables. • Now, *up to 8* 3D correction tables can be stored in RTC6 memory (in the 3D-enabled version). However, two 3D correction tables can *not* be simultaneously assigned to the scan head connector (see **select_cor_table**). • The parameters `k`, `Phi` and `Offset` no longer exist. Therefore, table transformations (scaling, rotation, translation) are no longer possible during loading of the correction file. Such coordinate transformations can now be specified by **set_scale**, **set_angle** and **set_offset** in addition to **set_matrix**. • **select_cor_table** is automatically called, see comments above and Section "Notes", page 181. • **load_correction_file** does not return to the user program until the correction motion has been completed (see comments above).
RTC5→RTC6	Changed functionality. • **load_correction_file** lets you load 8 correction tables simultaneously into the RTC6 memory. • Overlaps no longer exist between memory areas used elsewhere.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	—

Ctrl Command	load_disk
Function	Loads into the protected RTC6 List Memory area "List 3" the indexed characters, text strings and subroutines previously stored in a binary file by **save_disk** and returns the number of actually loaded list commands.
Call	`NoOfLoadedCommands = load_disk( Name, Mode )`
Parameters	`Name` File name or NULL. As a pointer to a \0-terminated ANSI string.
	`Mode` Determines how the loading procedure is executed. As an unsigned 32-bit value. = 0: The internal management tables for indexed characters, text strings and subroutines are initialized (all old references are thereby lost) and the input pointer is set to the beginning of "List 3" (the resulting loading process overwrites list commands stored there). > 0: Internal management tables are not initialized (all old references are initially retained, but then replaced or supplemented by other references during the loading process, depending on the file's content) and the input pointer is set to the position after the last stored (by **load_char**, **load_text_table** or **load_sub**) indexed character, text string or subroutine (the resulting loading process does *not* overwrite the list commands of old indexed characters, text strings and subroutines).
Result	The number of commands loaded by **load_disk**. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_load_disk**
Comments	• For `Name` = NULL, only the internal management tables are initialized as with `Mode` = 0 (no loading occurs, no "empty" binary file must be provided). Otherwise, **load_disk** can only be used for reading files that were stored with **save_disk**. • For all indexed characters, text strings and subroutines in the specified file, **load_disk** executes a corresponding **load_char**, **load_text_table** or **load_sub** command. The pertaining list commands are thereby written to the protected RTC6 List Memory area "List 3" and the entries are made in the internal management tables in accordance with the index assignment stored in the file. If there is no character, text string or subroutine for a particular index in the specified file, then no list commands are loaded for this index, and if `Mode` > 0 then any already existing entries in the internal management table remains unaltered. Thus, supplementary loading of indexed characters, text strings and subroutines from various files is possible.

Ctrl Command	load_disk
Comments (cont'd)	• Together with **save_disk**, **load_disk** can be used, for example, to defragment the protected RTC6 List Memory area "List 3" and to subsequently protect subroutines, see Section "Subsequent Protection and Conversion of Non-Indexed Subroutines", page 119 and Section "Index Management and Defragmentation", page 118. If the characters, text strings and subroutines come from various files, then defragmentation can be achieved if the first file is loaded in `Mode = 0` and subsequent files are loaded with `Mode` > 0, provided that no indices are used simultaneously in different files. Memory gaps in the protected area caused by dereferencing of indexed characters, text strings or subroutines are thereby closed. • If, during loading, the end of the protected RTC6 List Memory area "List 3" is reached before the end of the file (EOF), then all further list commands in the file is ignored. Likewise, incomplete characters, text strings and subroutines (as with individual **load_char**, **load_text_table** or **load_sub** calls) are not stored. Prior to a **load_disk** call, be sure that the memory configuration provides a sufficiently large protected RTC6 List Memory area "List 3" above the RTC6 List Memory areas ("List 1" and "List 2"). **save_disk** returns the number of list commands stored in the file. If a board changes ownership, it is the user's responsibility to ensure that the memory configuration data is consistent (`Mem1` and `Mem2` are queried from the RTC6 DLL, not from the board – see also **get_config_list** and Chapter 6.7.1 "Notes on Board Acquisition by a User Program", page 131). • If the specified file is corrupt and cannot be read to the end, then only the characters, text strings and subroutines fully readable to that point are stored. • **load_disk** is not executed, if: – "List 3" has no RTC6 List Memory area assigned, that is, `Mem1` + `Mem2` = 2^{23} (**get_last_error** return code `RTC6_PARAM_ERROR`) – the BUSY list execution status is set (**get_last_error** return code `RTC6_BUSY`) – the INTERNAL-BUSY list execution status is set (**get_last_error**-Returncode `RTC6_BUSY`) • **load_disk** is even executed, if: – a list has only been paused by **set_wait** (PAUSED list execution status set). But users are responsible for ensuring that no still-needed commands are overwritten, for example, if **set_wait** has been called from an indexed subroutine. • During the runtime of **load_disk**: – No other commands can be executed – External Starts are suppressed • **load_disk** checks the version info saved by **save_disk** and compares it with the current runtime version. Both must match. Otherwise, **load_disk** is not executed (**get_last_error** return code `RTC6_VERSION_MISMATCH`).

Ctrl Command	load_disk
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 615: version check.
References	**save_disk**

Ctrl Command	load_fly_2d_table	
Function	Loads a 2D table from an ASCII text file for a **set_fly_2d**-Processing-on-the-fly application with 2D encoder compensation for xy positioning stages, see Section "2D Encoder Compensation for xy Positioning Stages", page 258.	
Call	`NoOfDataPoints = load_fly_2d_table( Name, No )`	
Parameters	`Name`	Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.
	`No`	Determines which table in the text file is to be loaded. `No` corresponds to the extension `<No>` of `[Fly2DTable<No>]` at the beginning of the desired table. As an unsigned 32-bit value.
Result	A positive error code in case of an error. The negative number of found data points in case of success. As a signed 32-bit value.	
	< 0	Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored, see Section "For the 2D compensation tables, the following rules apply:", page 259.
	0	For `Name` = NULL (see comments).
	1	No valid data points found (though Table `No` found).
	2	Out of Memory (not enough Windows system memory).
	3	File not found.
	4	DSP memory error.
	5	BUSY list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (**get_last_error** return code `RTC6_BUSY`).
	8	Board is locked by another user program (**get_last_error** return code `RTC6_ACCESS_DENIED`).
	11	PCI error (**get_last_error** return code `RTC6_SEND_ERROR`), verify error (**get_last_error** return code `RTC6_VERIFY_ERROR`).
	13	The specified table number could not be found in the file.
Multi-board Com'd Name	**n_load_fly_2d_table**	
Comments	• `No` = `<No>` loads table number 1; `No` = `<No>` + 65.536 loads table number 2. • The text file's data format requirements for reference points of the 2D encoder compensation table are described in Section "For the 2D compensation tables, the following rules apply:", page 259. The largest of these reference points should not exceed the range –524,288…+524,287 (otherwise precision may be lost). At runtime, the current encoder values (including reference values) must not exceed the largest values specified in the table. Otherwise, clipping occurs. • **load_fly_2d_table** overwrites a previously loaded table for 2D encoder compensation. • If `Name` = NULL, then a 0-correction table for 2D encoder compensation is loaded and the table is marked as invalid (see **init_fly_2d**).	

Ctrl Command	load_fly_2d_table
Comments (cont'd)	• **load_fly_2d_table** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **load_fly_2d_table** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • During the runtime of **load_fly_2d_table**, External Starts are suppressed. • Before loading a table, **load_fly_2d_table** performs a DSP memory check that produces error code 4 in case of error.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. Additional: 2 tables can now be loaded simultaneously.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_fly_2d**, **init_fly_2d**, **get_fly_2d_offset**

Ctrl Command	load_jump_table	
Function	Loads a value table with Jump Delay data points from an ASCII text file (or alternatively performs automatic determination on the attached scan system) and uses linear interpolation to create the internal Jump Delay table for 2D jumps executable in Jump Mode.	
Call	`NoOfDataPoints = load_jump_table( Name, No, PosAck, MinDelay, MaxDelay, ListPos )`	
Parameters	`Name`	See **load_jump_table_offset**.
	`No`	See **load_jump_table_offset**.
	`PosAck`	See **load_jump_table_offset**.
	`MinDelay`	See **load_jump_table_offset**.
	`MaxDelay`	See **load_jump_table_offset**.
	`ListPos`	See **load_jump_table_offset**.
Result	See **load_jump_table_offset**.	
Multi-board Com'd Name	**n_load_jump_table**	
Comments	• **load_jump_table** is identical to **load_jump_table_offset** with `Offset` = 0.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
Reference	**load_jump_table_offset**	

Ctrl Command	load_jump_table_offset	
Function	Loads a value table with Jump Delay data points from an ASCII text file (or alternatively performs automatic determination on the attached scan system) and uses linear interpolation to create the internal Jump Delay table for 2D jumps executable in Jump Mode.	
Call	`NoOfDataPoints = load_jump_table_offset( Name, No, PosAck, Offset, MinDelay, MaxDelay, ListPos )`	
Parameters	`Name`	Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.
	`No`	Specifies: • For `Name` = Filename, which table of the text file should be loaded. The parameter corresponds to the extension `<No>` of `[JumpTable<No>]` at the beginning of the desired table. • For `Name` = NULL, which scan system (or which scan head connector) should be used for automatic determination. Allowed values: [1, 2]. As an unsigned 32-bit value.
	`PosAck`	Position tolerance value in [bits] (only relevant for automatic determination). As an unsigned 32-bit value.
	`Offset`	Offset in [10 µs] which is added to all automatically determined delay values (only relevant for automatic determination). As a signed 32-bit value.
	`MinDelay`	Minimum Jump Delay in [10 µs] (only relevant for automatic determination). As an unsigned 32-bit value.
	`MaxDelay`	Maximum Jump Delay in [10 µs] (only relevant for automatic determination). As an unsigned 32-bit value.
	`ListPos`	List position (in the area of list 1 or 2) for six list commands that can be overwritten (only relevant for automatic determination). As an unsigned 32-bit value.
Result	Error code. As a signed 32-bit value.	
	–1…– 50	Success for `Name` = filename. The absolute value of the return value equals the number of valid data points found in the table. Invalid entry values are ignored, see also Section "Notes on Loading Determined Jump Delay Values", page 228.
	–1024	Success for `Name` = NULL: Table has been automatically determined.
	1	No valid data points found (but Table `No` found).
	3	File not found.
	4	Verify error: DSP memory error.
	5	Busy error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (**get_last_error** return code `RTC6_BUSY`).
	8	Access error: board reserved for another user program (**get_last_error** return code `RTC6_ACCESS_DENIED`).

Ctrl Command	load_jump_table_offset
Result (cont'd)	10 Only if Name = NULL: Param error: HeadNo or ListPos invalid. 11 PCI error during download (**get_last_error** return code RTC6_SEND_ERROR). 13 The specified table number could not be found in the file.
Multi-board Com'd Name	**n_load_jump_table_offset**
Comments	• For information on command usage, see Section "Jump-Length-Dependent Jump Delays", page 227. • See also Chapter 8.1.5 "Jump Mode", page 226. • Format requirements for placing the table with Jump Delay data points into the text file are described in Section "Notes on Loading Determined Jump Delay Values", page 228. When loading the table, the RTC6 uses linear interpolation to establish appropriate values for the complete jump length range. • **load_jump_table_offset** overwrites any previously loaded Jump Delay tables for Jump Mode. • If Name = filename, then the table number No is loaded. The other parameters are not used. • If Name = NULL, then the Jump Delay table for head No is automatically determined and loaded (if Jump Mode has been previously enabled and activated successfully by **set_jump_mode** (Flag = 1)). Here, the parameters PosAck, Offset, MinDelay, MaxDelay and ListPos are used (see Section "Automatic Determination of the Jump Delay Table", page 230). • Jump Mode takes effect upon the next 2D jump only if it has been activated and enabled by **set_jump_mode** or activated by **set_jump_mode_list**. • **load_jump_table_offset** is not executed (**get_last_error** return code RTC6_BUSY), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **load_jump_table_offset** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • During the runtime of **load_jump_table_offset**, External Starts are suppressed. • Before loading a table, **load_jump_table_offset** performs a DSP memory check that produces error code 4 in case of error. • The table can be saved by **create_dat_file**.
RTC4→RTC6	New command. There is no RTC4 Compatibility Mode for this command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
Reference	**load_jump_table**, **set_jump_mode**, **set_jump_mode_list**, **get_jump_table**, **set_jump_table**, **create_dat_file**

Ctrl Command	load_list
Function	Opens the RTC6 List Memory for writing with list commands and sets the input pointer to the specified *relative* position within the desired list (“List 1” or “List 2”), but only if the list is not currently being processed (BUSY list status is not set) or has already been processed (USED list status is set).
Call	`NoOfOpenedList = load_list( ListNo, Pos )`
Parameters	`ListNo` Number of the list in which the input pointer should be set. As an unsigned 32-bit value. Allowed values: [0…3]. Only the two least significant bits are evaluated. = 0: The list is opened for which the BUSY list status is not currently set. “List 1” is opened, if the BUSY list status is not set for either list. = 1: “List 1” is opened, if the BUSY list status for it is not set (BUSY1 not set). = 2: “List 2” is opened if the BUSY list status for it is not set (BUSY2 not set). = 3: The list is opened for which the BUSY list status is not currently set but the USED list status. If for both lists the BUSY list status is not set but the USED list status: that list is opened, which would be executed by a following automatic list change. For `Mem2` = 0 (see **config_list**) “List 1” is opened, if: • the BUSY list status for it is not set (`ListNo` = 0…2) • the BUSY list status for it is not set but the USED list status (`ListNo` = 3)
	`Pos` Position of the input pointer (offset relative to the start of the respective list). As an unsigned 32-bit value. Allowed value range: [0…(2^{23}–1)].
Result	Number of the opened list [1 or 2] if successful, otherwise 0. As an unsigned 32-bit value.
Multi-board Com’d Name	**n_load_list**
Comments	• If **load_list** has been successful, the next list command is stored at the specified address and all subsequent list commands at the following addresses in the selected list. • If **load_list** has not been successful (return code 0), then no list is opened and the input pointer is set to an invalid position. Then no further list commands can be input until the input pointer is correctly set (for example, by repeating **load_list** with a positive result or by the **set_start_list_pos** command etc.). It is the responsibility of the user to react to a return code of 0. **load_list** produces *no* wait loop and *does not* block execution of subsequent control commands.

Ctrl Command	load_list
Comments (cont'd)	• **load_list** (ListNo = 3) is useful in scenarios such as alternating list changes, where you want to wait specifically for a list to be processed (see Section "Alternating List Changes", page 114) without needing to separately query the list status. Unintentional overwriting of not-yet-executed commands is thereby automatically avoided. To instead perform *unconditional* loading, you can use commands such as **set_start_list_pos**. • After a successful check of the list number (BUSY list status not set and, if applicable, USED list status set), **load_list** behaves like **set_start_list_pos** (see comments there). • If ListNo = 0…2 when using **load_list**, then note that the BUSY list status of a list can change between opening of the list with **load_list** and the actual loading of list commands, for example, if in the meantime an automatic list change has occurred that has been previously defined by **auto_change** or **start_loop**. In cases such as this, where loading of a list might occur simultaneously with processing of the same list, users must always ensure that the output pointer does not overtake the input pointer. • In contrast, for ListNo = 3 **load_list** ensures that a list is actually opened for loading only directly after processing (and after any successful automatic list changes). Particularly, if for no list the BUSY list status is set but for both the USED list status (for example, after initialization, after **stop_execution** or after an External Stop), the opened list number is synchronized with the following automatic list change.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_start_list_pos**

Ctrl Command	load_position_control
Function	Loads a table with data points from an ASCII text file and determines – by linear interpolation – the scaling function for Position-Dependent Laser Control (radial correction).
Call	NoOfDataPoints = load_position_control(Name, No)
Parameters	Name: Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.
	No: Determines which table in the text file is to be loaded. The parameter corresponds to the extension <No> of [PositionCtrlTable<No>] at the beginning of the desired table. As an unsigned 32-bit value.
Result	A positive error code in case of an error. The negative number of found data points in case of success. As a signed 32-bit value. Value – Description –1…– 50 – Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored, see also Section ”Notes on Loading a Scaling Function”, page 207. –256 – For Name = NULL (see also comments). 1 – No valid data points found (though Table No found). 3 – File not found. 4 – DSP memory error. 5 – BUSY list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (**get_last_error** return code RTC6_BUSY). 8 – Board is locked by another user program (**get_last_error** return code RTC6_ACCESS_DENIED). 11 – PCI error (**get_last_error** return code RTC6_SEND_ERROR), verify error (**get_last_error** return code RTC6_VERIFY_ERROR). 13 – The specified table number could not been found in the file.
Multi-board Com’d Name	**n_load_position_control**
Comments	• The format requirements for the text file’s table entries with data points for Position-Dependent Laser Control are described in Section ”Notes on Loading a Scaling Function”, page 207. When loading the table, the RTC6 determines suitable values for the entire range of control values. • **load_position_control** overwrites any previously loaded scaling function for Position-Dependent Laser Control. • For Name = NULL (as during initialization by **load_program_file**), the scaling function *Scale*(*Position*)=1.0 is loaded for the complete position range so that no position-dependent correction takes place.

Ctrl Command	load_position_control
Comments (cont'd)	• Position-Dependent Laser Control only takes effect during subsequent [*]mark[*] Commands or Arc Commands if it has been initialized by **set_auto_laser_control**. Position-Dependent Laser Control is deactivated by **set_auto_laser_control** (`Ctrl` = 0) or by loading *Scale*(*Position*)=1.0. See also Section "Position-Dependent Laser Control", page 207. • **load_position_control** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **load_position_control** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • During the runtime of **load_position_control**, External Starts are suppressed. • Before loading a table, **load_position_control** performs a DSP memory check that produces error code 4 in case of error. • The table can be saved by **create_dat_file**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_auto_laser_control**, **create_dat_file**

Ctrl Command	load_position_control_2d_ctrl
Function	Loads a laser correction table from an ASCII text file for 2D Position-Dependent Laser Control.
Call	`ErrorNo = load_position_control_2d_ctrl( Name )`
Parameters	`Name` Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.
Result	Error code. As an unsigned 32-bit value. 0 Success. 1 Error during parsing or not enough data points found. 2 Out of Memory (not enough Windows system memory). 3 File not found (**get_last_error** return code `RTC6_PARAM_ERROR`). 4 DSP memory error. 5 BUSY list execution status error, board has BUSY list execution status or INTERNAL-BUSY list execution status, no download (**get_last_error** return code `RTC6_BUSY`). 6 Invalid calibration factor (K does not match the correction table assigned to the first scan head, `get_head_para( HeadNo = 1, ParaNo = 1 )`. 8 Board is locked by another user program (**get_last_error** return code `RTC6_ACCESS_DENIED`). 11 PCI error (**get_last_error** return code `RTC6_SEND_ERROR`), verify error (**get_last_error** return code `RTC6_VERIFY_ERROR`).
Multi-board Com'd Name	**n_load_position_control_2d_ctrl**
Comments	• 2D Position-Dependent Laser Control only becomes effective for subsequent [*]mark[*] Commands or Arc Commands, if it has been initialized by **set_auto_laser_control**. • 2D Position-Dependent Laser Control is deactivated by: – `set_auto_laser_control( Ctrl = 0 )` – `Name` = NULL (furthermore, Position-Dependent Laser Control, page 207 is activated, if it has been active before) – Activating Position-Dependent Laser Control, page 207 • See Section "2D Position-Dependent Laser Control", page 215. • See also **load_position_control**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 644, OUT 647.
References	**load_position_control**

Ctrl Command	load_program_file	
Function	• Resets the RTC6 (Software Reset) • Initializes the RTC6 List Memory • Performs a DSP memory check • Loads `RTC6RBF.rbf` • Loads `RTC6OUT.out` (with RTC6 Ethernet Boards, `RTC6ETH.out` is loaded instead • Loads `RTC6DAT.dat` • Starts the DSP	
Call	`ErrorNo = load_program_file( Path )`	
Parameters	`Path`	Path name of the directory, where `RTC6OUT.out`, `RTC6RBF.rbf` and `RTC6DAT.dat` are. As a pointer to a \0-terminated string.
Result	Error code. As an unsigned 32-bit value.	
	0	Success.
	1	Not used.
	2	The board is not running. If a renewed call does not bring success, then a Hardware Reset is necessary.
	3	`RTC6DAT.dat` or `RTC6RBF.rbf` not found.
	4	Not used.
	5	Not enough Windows memory.
	6	Another user program has already acquired the board.
	7	Version error: RTC6 DLL version, `RTC6RBF.rbf` version and `RTC6OUT.out`/`RTC6ETH.out` version are not compatible with each other.
	8	RTC6 board driver not found.
	9	Loading of `RTC6OUT.out` or `RTC6ETH.out` failed or has incorrect format or other error.
	10	Not used.
	11	FPGA firmware error: loading of `RTC6RBF.rbf` failed.
	12	Error opening/reading `RTC6DAT.dat`.
	13	Not used.
	14	DSP memory error (external).
	15	Verify memory error.
	16	DSP memory error (internal).
	17	Ethernet error. See **eth_get_last_error** and **eth_get_error**.
	18	Only RTC6 Ethernet Board: NAND Memory error.
Multi-board Com'd Name	**n_load_program_file**	

Ctrl Command	load_program_file
Comments	• If `Path` = NULL, then the path of the user program's current working directory is used. Caution: It is not always the folder from which the user program has been launched. The current working directory can change, for example, when a file from another folder is selected by the Windows Explorer (unless the "NoChangeDir" flag has been set when incorporating the Explorer window into the user program). • After each hardware reset (powerup), the first user program must begin by issuing a **load_program_file** command during initialization of the RTC6 board, see Initializing the Board, page 101. **load_program_file** should also be executed (for example, if another user program acquires the board – see **acquire_rtc**) when the board needs to be returned to the default state. • If multiple RTC6 boards are connected as master and slave, then **load_program_file** must have been called on all boards prior to initializing and operating the individual boards with further commands, see Chapter 6.6.3 "Master/Slave Operation", page 127. • After execution of **load_program_file**: – The laser focus is in the center of the Image Field (0|0) – The laser control is *deactivated* – On the state of the various output ports, see Chapter 7.4.1 "Enabling, Activating and Switching Laser Control Signals", page 189 (LASERON, LASER1, LASER2), Chapter 9.1.1 "16-Bit Digital Output Port", page 303 (EXTENSION 1 Socket Connector), Chapter 9.1.2 "8-Bit Digital Output Port", page 304 (EXTENSION 2 Socket Connector), Chapter 9.1.3 "2-Bit Digital Output Port", page 304 (LASER Connector), Chapter 9.1.4 "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 305 (LASER Connector). • *Caution! In general, sporadic* **load_program_file** *calls in your user program:* – *Contradict the safe switch-on sequence prescribed in* Chapter 5.6 "Safe Start-up and Shutdown Sequences", page 94 – *Pose the risk of personal injury and/or property damage* *(cases have been reported where lasers with poor electric have emitted)* *If you absolutely cannot refrain from sporadic* ***load_program_file*** *calls in your user program, you must implement appropriate messages that warn users accordingly and prompt them to take actions that prevent these hazards.* • **load_program_file** does *not* load correction tables. Even 1-to-1 tables therefore need to be explicitly requested, see **load_correction_file**. Already-loaded correction tables remain loaded after **load_program_file**. • **load_correction_file** assigns a correction table by `select_cor_table( 1, 0 )` but does not execute a galvanometer scanner motion to the corrected output position. • **load_program_file** only returns to the calling user program, when DSP initialization has been completed.

Ctrl Command	load_program_file
Comments (cont'd)	• **load_program_file** automatically executes **stop_execution**, if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • The files `RTC6OUT.out`, `RTC6ETH.out`, `RTC6RBF.rbf` and `RTC6DAT.dat` are included in the RTC6 software package. For easy identifying and archiving of different software versions, the files are also delivered zipped (the `zip` file names `RTC6<…>_<Version>.zip` include the version numbers). Copy or unzip the three files (of desired version) to the hard drive of your PC. • Assorted versions of the RTC6 DLL and the files `RTC6OUT.out`, `RTC6ETH.out`, `RTC6RBF.rbf` and `RTC6DAT.dat` cannot be arbitrarily combined with another (each `zip` file in the RTC6 software includes a text file with version information). **load_program_file** performs a version compatibility check. If there is a version error, then the loaded programs remain in RTC6 memory, but the board is released by **release_rtc** directly after the version check and therefore is not available for further commands other than those not requiring access rights (**get_last_error** return code `RTC6_ACCESS_DENIED | RTC6_VERSION_MISMATCH`). To then load a correct program version, **load_program_file** can be called. Hereby, temporary access rights are requested and released after the download (if the board has not been acquired by another user program; **load_program_file** does not perform an **acquire_rtc**).
RTC4→RTC6	• The command parameter specifies a directory name with RTC6 (in contrast a file name with the RTC4). • **load_program_file** loads three files, with fixed formats and names (`RTC6OUT.out`, `RTC6RBF.rbf`, `RTC6DAT.dat`) (see above). • After execution of the command, the laser control is *deactivated*.
RTC5→RTC6	Changed functionality. • Notes on migrating the source code of RTC5 user programs, etc.: **load_program_file** is downward compatible with the RTC5 and can continue to be used without modification. Though you do not need to change your user program, keep in mind the following: – There is no checking of the BUSY list execution status (hence error code 13 is never be outputted). – Data transfer by the McBSP interface is deactivated without warning. – A running list is terminated without warning, similarly to the control command **stop_execution**. Observe notes there about mirror positions etc. – Other used error codes are provided as listed in the table above.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Ctrl Command	load_stretch_table
Function	Loads a table with data pairs from an ASCII text file for enhanced 3D correction, see Section "Enhanced 3D Correction", page 249.
Call	`NoOfDataPairs = load_stretch_table( Name, No, TableNo )`
Parameters	`Name` — Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.
	`No` — As a signed 32-bit value. • For `No` ≥ 0, this parameter specifies which table in the text file is to be loaded. The parameter corresponds to the extension `<No>` of `[StretchTable<No>]` at the beginning of the desired table. • `No` < 0: Reserved.
	`TableNo` — The already loaded 3D correction table to which the extended correction is assigned. As a signed 32-bit value. Allowed value range: 1…8. See also number_of_correction_tables.
Result	A positive error code in case of an error. The negative number of found data pairs in case of success. As a signed 32-bit value. Value — Description < 0 — Success. The absolute value of the return value is equal to the number of valid data pairs found in the table. 0 — Reserved. 2 — Out of Memory (not enough Windows memory). 3 — File not found. 4 — DSP memory error. 5 — BUSY list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (**get_last_error** return code `RTC6_BUSY`). 6 — Data error: data pairs missing. 11 — PCI download error (**get_last_error** return code `RTC6_SEND_ERROR`). 13 — The specified table number could not been found in the file. 15 — Verify error (**get_last_error** return code `RTC6_VERIFY_ERROR`, only possible with active download verification, see **set_verify**).
Multi-board Com'd Name	**n_load_stretch_table**
Comments	• For details about enhanced 3D correction, see Section "Enhanced 3D Correction", page 249. • The enhanced 3D correction is also switched when **select_cor_table** is called. • A successfully loaded table activates the new enhanced 3D correction. Here, **load_stretch_table** overwrites a previously loaded table of the same `TableNo`. • If `Name` is *not* NULL, but no table has been successfully read, then **load_stretch_table** returns an error code, but otherwise has no effect (that is, any previous successfully downloaded table remains valid).

Ctrl Command	load_stretch_table
Comments (cont'd)	• If `Name` is NULL, then **load_stretch_table** disables any enhanced 3D correction enabled by a previous **load_stretch_table**. • **load_stretch_table** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **load_stretch_table** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • During execution of **load_stretch_table**, External Starts are suppressed. • Before loading a table, **load_stretch_table** performs a DSP memory check that produces error code 4 in case of error.
RTC4→RTC6	New command.
RTC5→RTC6	Changed functionality. • Additional `TableNo` parameter. This allows that each 3D correction table can be assigned its own extended 3D correction.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**select_cor_table**

Ctrl Command	load_sub
Function	Assigns the specified index to a subroutine defined by subsequent list commands and loads the subroutine into the protected RTC6 List Memory area "List 3".
Call	`load_sub( Index )`
Parameters	`Index` Index of the indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0…1023].
Multi-board Com'd Name	**n_load_sub**
Comments	• Up to 1024 indexed subroutines can be stored. If `Index` > 1023 then **load_sub** is ignored (**get_last_error** return code `RTC6_PARAM_ERROR`). • The address in the protected RTC6 List Memory area "List 3" where the subroutine should be stored is automatically determined and internally managed. • Indexed subroutines must be terminated by a **list_return** call. This is a prerequisite for actual storage of the commands, entry of the start address into the internal management table, and initiating a flush of the buffered list input, see Chapter 6.4.1 "Loading Lists", page 108. Otherwise (the input pointer is altered without a preceding **list_return**), the subroutine with this index is not available. • An indexed subroutine is not stored if the protected RTC6 List Memory area "List 3" has not been previously configured for a sufficient size beyond "List 1" and "List 2". • If **list_return** is the next command after **load_sub**, then the corresponding subroutine is deleted from the internal management table. • Indexed subroutines can be called by the **sub_call** command along with the corresponding index (see Section "General Information on Calling Subroutines", page 117). • Observe all notes in Section "Indexed Subroutines", page 116.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_return**, **sub_call**, **load_char**, **load_text_table**

Ctrl Command	load_text_table
Function	Assigns the specified index to a text string defined by subsequent list commands and loads the text string into the protected RTC6 List Memory area "List 3".
Call	`load_text_table( Index )`
Parameters	`Index` Index of the indexed text string. As an unsigned 32-bit value. Allowed value range: [0…41].
Multi-board Com'd Name	**n_load_text_table**
Comments	• Up to 42 indexed text strings can be stored (for marking times, dates and serial numbers by other commands). The following ordering applies: – `Index` = 0…9: digits for marking the time and date [0…9] – `Index` = 10…21: months [January…December] – `Index` = 22…28: days-of-the-week [Sunday…Saturday] – `Index` = 29: blank character for marking serial numbers – `Index` = 30…39: digits for marking serial numbers [0…9] – `Index` = 40: text for "a.m." – `Index` = 41: text for "p.m." If `Index` > 41 then **load_text_table** is ignored (**get_last_error** return code `RTC6_PARAM_ERROR`). • Even if digits are defined by **mark_text** instead of as individual characters with **mark_char**, the character set can still be subsequently switched for this purpose (see **select_char_set**). • The addresses in the protected RTC6 List Memory area "List 3" where the text string definitions are stored are automatically determined and internally managed. Management is independent of that for indexed subroutines (see **load_sub**) and character definitions (see **load_char**). • Indexed text string definitions must be terminated with a **list_return** call. This is a prerequisite for actual storage of the commands, entry of the start address into the internal management table, and initiating a flush of the buffered list input, see Chapter 6.4.1 "Loading Lists", page 108. Otherwise (the input pointer is altered without a preceding **list_return**), the text string with this index is not available. • An indexed text string definition is not stored if the protected RTC6 List Memory area "List 3" has not been previously configured for a sufficient size beyond "List 1" and "List 2". • If **list_return** is the next command after **load_text_table**, then the corresponding text string definition is deleted from the internal management table. • Also observe all notes in the Chapter 6.5.2 "Character Sets and Text Strings", page 121.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_return**, **load_sub**, **load_char**

Ctrl Command	load_varpolydelay
Function	Loads a table with data points from an ASCII text file for the scaling function of the user-defined “Variable Mark Delay”, see Section “User-defined “Variable Polygon Delays””, page 156.
Call	`NoOfDataPoints = load_varpolydelay( Name, No )`
Parameters	`Name` — Name of the text file or NULL. The text file may contain one or more tables. As a pointer to a \0-terminated ANSI string.
	`No` — Table in the text file which is to be loaded. The parameter corresponds to the extension `<No>` of `[VarPolyTable<No>]` at the beginning of the desired table. As an unsigned 32-bit value.
Result	A positive error code in case of an error. The negative number of found data points in case of success. As a signed 32-bit value. **Value** — **Description** –1…– 50 — Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored, see Section ”User-defined “Variable Polygon Delays””, page 156. –1024 — For `Name` = NULL: the table initialized according to 1 – cos(φ) has been internally loaded (as with program start), see Figure 40. 1 — No valid data points found (though Table `No` found). 3 — File not found. 4 — DSP memory error. 5 — BUSY list execution status error, board is BUSY list execution status or INTERNAL-BUSY list execution status, no download (**get_last_error** return code `RTC6_BUSY`). 8 — The board is locked by another user program (**get_last_error** return code `RTC6_ACCESS_DENIED`). 11 — PCI error (**get_last_error** return code `RTC6_SEND_ERROR`), verify error (**get_last_error** return code `RTC6_VERIFY_ERROR`). 13 — The specified table number could not been found in the file.
Multi-board Com’d Name	**n_load_varpolydelay**
Comments	• The format requirements for text file’s table entries with data points for the user-defined “Variable Mark Delay” are described in Section ”User-defined “Variable Polygon Delays””, page 156. When loading the table, the RTC6 determines suitable values for the entire range of angles by linear interpolation. • **load_varpolydelay** overwrites any previously loaded table for the “Variable Mark Delay”. • For `Name` = NULL (as during initialization by **load_program_file**), the internal (default) table for the “Variable Mark Delay” (1 – cos(φ), see Figure 40) is loaded.

Ctrl Command	load_varpolydelay
Comments (cont'd)	• **load_varpolydelay** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set (a list is being processed or has been halted by **pause_list**) – the INTERNAL-BUSY list execution status is set • **load_varpolydelay** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • During the runtime of **load_varpolydelay**, External Starts are suppressed. • Before loading a table, **load_varpolydelay** performs a DSP memory check that produces error code 4 in case of error. • The table can be saved by **create_dat_file**.
RTC4→RTC6	Basically unchanged functionality. However: • To return to the internal standard Mark Delay table, a reset or renewed program loading by **load_program_file** is *no longer* necessary (see `Name` = NULL above). • The ASCII text file can have any filename extension.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**load_program_file**, **set_delay_mode**, **create_dat_file**

Ctrl Command	load_z_table
Function	Like **load_z_table_20b**. However, for a focus length value *l* in RTC4 compatibility range [–32,768…+32,767].
Restriction	Like **load_z_table_no_20b**.
Call	`ErrorNo = load_z_table( A, B, C )`
Parameters	`A` Like **load_z_table_no**.
	`B` Like **load_z_table_no**.
	`C` Like **load_z_table_no**.
Result	Like **load_z_table_no_20b**.
Multi-board Com'd Name	**n_load_z_table**
Comments	• Like **load_z_table_no_20b**. • `load_z_table( A, B, C )` is synonymous with `load_z_table_no( A, B, C, 0 )`.
RTC4→RTC6	Unchanged functionality. In addition: changed value ranges and error codes. If the correct calibration factors are used (see Chapter "3D Commands", page 247), then the same ABC coefficients can be used on the same 3-axis scan system with the RTC4 and RTC6 boards.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_z_distance**, **read_abc_from_file**, **write_abc_to_file**, **load_z_table_no**, **select_cor_table**, **load_z_table_20b**

Ctrl Command	load_z_table_20b	
Function	Loads coefficients *A*, *B* and *C* into the currently assigned 3D correction table. For a focus length value *l* in the RTC6 20-bit range [–524,288…+524,287].	
Restriction	Like **load_z_table_no_20b**.	
Call	`ErrorNo = load_z_table_20b( A, B, C )`	
Parameters	`A`	Coefficient *A* of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value *l* in the RTC6 20-bit range [–524,288…+524,287]. As a 64-bit IEEE floating point value. Allowed value range: [–1,073,741,824.0…+1,073,741,824.0]. Out-of-range values are clipped to the boundary values.
	`B`	Like `A` (analogously). Allowed value range: [–2,048.0…+2,048.0].
	`C`	Like `A` (analogously). Allowed value range: [–1.0…+1.0].
Result	Like **load_z_table_no_20b**.	
Multi-board Com'd Name	**n_load_z_table_20b**	
Comments	• **load_z_table_20b** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **load_z_table_20b** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • **load_z_table_20b** should always be used *after* **load_correction_file**, since **load_correction_file** sets the three coefficients to the default values of the loaded correction table. • The `ABC` values are lost by **select_cor_table**. • See also other comments at **load_z_table_no**.	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 631, OUT 632.	
References	**get_z_distance**, **read_abc_from_file_20b**, **write_abc_to_file_20b**, **load_z_table_no_20b**, **select_cor_table**, **load_z_table_no_20b**	

Ctrl Command	load_z_table_no	
Function	Like **load_z_table_no_20b**. However, for a focus length value *l* in RTC4 compatibility range [–32,768…+32,767].	
Restriction	Like **load_z_table_no_20b**.	
Call	`ErrorNo = load_z_table_no( A, B, C, No )`	
Parameters	`A`	Coefficient A of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value *l* in the RTC4 compatibility range [–32,768…+32,767]). As a 64-bit IEEE floating point value. Allowed value range: [–67.108.864.0…+67.108.864.0]. Out-of-range values are clipped to the boundary values.
	`B`	Like `A` (analogously). Allowed value range: [–2,048.0…+2,048.0].
	`C`	Like `A` (analogously). Allowed value range: [–16.0…+16.0].
	`No`	Number of the 3D correction table to which the three coefficients `A, B, C` are to be assigned. See also **number_of_correction_tables**.
Result	Like **load_z_table_no_20b**.	
Multi-board Com'd Name	**n_load_z_table_no**	
Comments	• Like **load_z_table_no_20b**. • `load_z_table_no( A, B, C, 0 )` is synonymous with `load_z_table( A, B, C )`. `load_z_table_no( A, B, C, No )` is synonymous with `load_z_table_no_20b( A × 16, B , C × 1/16, No )`. The values are lost after a **select_cor_table** or **select_cor_table_list**.	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 611, OUT 611, RBF 616.	
References	**load_z_table**, **get_z_distance**, **read_abc_from_file**, **write_abc_to_file**, **select_cor_table**	

Ctrl Command	load_z_table_no_20b
Function	Loads coefficients *A*, *B* and *C* and then assigns them to the 3D correction table `No`. For a focus length value *l* in the RTC6 20-bit range [–524,288…+524,287].
Restriction	If the Option “3D” has not been enabled or a 3D correction table has not been assigned (see **select_cor_table**), then **load_z_table_no_20b** returns the error code 12 or 13 and otherwise has no effect.
Call	`ErrorNo = load_z_table_no_20b( A, B, C, No )`
Parameters	`A` — Coefficient *A* of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value *l* in the RTC6 20-bit range [–524,288…+524,287]). As a 64-bit IEEE floating point value. Allowed value range: [–1,073,741,824.0…+1,073,741,824.0]. Out-of-range values are clipped to the boundary values.
	`B` — Like `A` (analogously). Allowed value range: [–2,048.0…+2,048.0].
	`C` — Like `A` (analogously). Allowed value range: [–1.0…+1.0].
	`No` — Number of the 3D correction table to which the three coefficients `A, B, C` are to be assigned. See also **number_of_correction_tables**.
Result	Error code. As an unsigned 32-bit value. Error bits with values 1…64 can also occur combined, but not in conjunction with error code 11…15, which only occur separately. Warnings 12 and 13 are only returned if no other errors exist. 0 — No error. 1 — `A` exceeded the maximum allowed value. 2 — `A` undercut the minimum allowed value. 4 — `B` exceeded the maximum allowed value. 8 — `B` undercut the minimum allowed value. 16 — `C` exceeded the maximum allowed value. 32 — `C` undercut the minimum allowed value. 64 — Execution denied (possibly a BUSY list execution status or INTERNAL-BUSY list execution status error; for exact reason see **get_last_error**). 11 — Access denied. 12 — Option “3D” is not enabled. 13 — No 3D correction table is currently assigned. 14 — RTC6 board driver not found. 15 — Invalid table number (> **number_of_correction_tables**).
Multi-board Com’d Name	**n_load_z_table_no_20b**

Ctrl Command	load_z_table_no_20b
Comments	• **load_z_table_no_20b** is only needed for re-calibrating the z axis in a 3-axis scan system. For adjusting corresponding coefficients see Section "Checking the z axis Calibration", page 175. Both positive and negative coefficients can be specified. The coefficients should preferably be chosen so that all z output values $z_{out} = A + Bl + Cl^2$ lie within RTC6 20-bit range [–524,288…+524,287]. • Prior to the next list command that directly follows **load_z_table_no_20b**, a smooth transition from the last Z position to the changed position is performed at jump speed. You can also immediately force this by **select_cor_table**. This way, time delays can be avoided during an External Start. • Coefficients *A*, *B* and *C* can be queried from the loaded 3D correction table by **get_table_para** (and from the currently assigned 3D correction table by **get_head_para**). • By **load_z_table_no_20b**, the three coefficients `A, B, C` can be assigned to the 3D correction table `No`. When **select_cor_table** is executed, they are switched as well. • If `No`= currently assigned table number, the current values are overwritten as well and thus immediately makes them available, given no list is being processed. Otherwise, they are only saved and are only available after a **select_cor_table** or **select_cor_table_list**. • If no list is being processed, `No` = 0 merely overwrites the current values. • `load_z_table_no_20b( A, B, C, No )` is synonymous with `load_z_table_no( A × 1/16, B × 16 , C × 16, No )`. The values are lost after a **select_cor_table** or **select_cor_table_list**. The following applies to `B` and `C`: The allowed value range decreases accordingly.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 631, OUT 632.
References	**load_z_table_no_20b**, **load_z_table_20b**, **get_z_distance**, **read_abc_from_file_20b**, **write_abc_to_file_20b**, **select_cor_table**

Ctrl Command	load_zoom_correction_file
Multi-board Com'd Name	**n_load_zoom_correction_file**
Comments	• **load_zoom_correction_file** is described, for example, in the manual for the intelli*WELD* II FT.

Normal List Command	long_delay
Function	Pauses further processing of the list for the specified time.
Call	`long_delay( Delay )`
Parameters	`Delay` Delay time. In bits. As an unsigned 32-bit value. 1 bit equals 10 μs. Allowed value range: 0 ≤ delay ≤ (2^{32}–1).
Multi-board Com'd Name	**n_long_delay**
Comments	• **long_delay** switches off the Signals for "Laser Active" Operation after a LaserOff Delay, waits for a possible Scanner Delay and pauses further processing of the list for the specified time. • **long_delay** should always be called after changing the lamp current of a YAG laser to obtain a constant laser power. • **list_nop** corresponds to `long_delay( 1 )`.
RTC4→RTC6	Unchanged functionality. In addition: increased value range.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_defocus_list**

Normal List Command	mark_abs	
Function	Moves the laser focus at mark speed along a 2D vector from the current position to the specified position (absolute coordinate values) within a 2D Image Field.	
Call	`mark_abs( X, Y )`	
Parameters	`X`	Absolute x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`Y`	Like `X` (analogously).
Multi-board Com’d Name	**n_mark_abs**	
Comments	• If the mark speed has not been previously explicitly set by **set_mark_speed** or **set_mark_speed_ctrl**, then the marking is executed at a predefined mark speed of 1,000 *bits/ms*. • The Signals for “Laser Active” Operation are automatically turned on at the beginning of the marking (or remain on after a directly preceding [*]mark[*] Command or “Arc” Command). The defined Scanner Delays and Laser Delays are thereby taken into account, see Chapter 7.2 ”Delay Settings – Coordinating Scan Head Control and Laser Control”, page 148. Note that other delays are executed in Sky Writing mode. Exception: zero-length [*]mark[*] Commands, see Section ”Notes”, page 152.	
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the values specified for `X` and `Y` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_mark_speed**, **set_scanner_delays**, **mark_rel**, **arc_abs**, **timed_mark_abs**	

Normal List Command	**mark_abs_3d**
Function	Moves the laser focus at mark speed along a 3D vector from the current position to the specified position (absolute coordinate values) within the 3D Image Field.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **mark_abs_3d** has the same effect as **mark_abs**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.
Call	`mark_abs_3d( X, Y, Z )`
Parameters	`X` — Absolute x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`Y` — Like `X` (analogously).
	`Z` — Like `X` (analogously), except Allowed value range: [–524,288…+524,287].
Multi-board Com'd Name	**n_mark_abs_3d**
Comments	• Except for the additional motion in the third dimension, **mark_abs_3d** functions similarly to **mark_abs** (see comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the values specified for `X`, `Y` and `Z` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range. In RTC5 Compatibility Mode, the RTC6 multiplies the value specified for `Z` by 16. The allowed value range decreases accordingly.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_abs**, **mark_rel_3d**, **timed_mark_abs_3d**

Undelayed Short List Command	mark_char
Function	Marks an indexed character.
Call	`mark_char( Char )`
Parameters	`Char` Index of the indexed character to be marked. As an unsigned 32-bit value. Allowed value range: [0…1023]). The following applies: `Char` = character set number × 256 + ASCII number of the character (character sets are numbered 0…3).
Multi-board Com'd Name	**n_mark_char**
Comments	• **mark_char** reads the indexed character's starting address from the internal management table based on the supplied index and then calls **list_call** (see also the comments there), which then starts the corresponding command list. • **mark_char** starts indexed characters in protected memory (that were loaded and/or referenced by **load_char**, **load_disk** or **copy_dst_src**) as well as indexed subroutines in the unprotected RTC6 List Memory area (that were referenced as characters by **set_char_pointer** or **copy_dst_src**). • If no character is referenced for the supplied index, then the jump is suppressed and execution continues at the command located after the calling position. In some circumstances, a **list_continue** might be executed, see Section "Normal List Command, Short List Command, Variable List Command and Multiple List Command", page 335. • **get_char_pointer** (`Char`) can be used to determine whether a character has been referenced for a particular index. If *no* character has been referenced, **get_char_pointer** returns the value "–1" (that is, 2^{32}–1). • If `Char` > 1023, then **mark_char** is, already during loading, replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • Absolute Vector Commands and "Arc" Commands execute absolutely after being called with **mark_char**. If the character needs to execute at various locations within the Image Field, then either the command list can only contain relative [*]mark[*] Commands, "Arc" Commands or Jump Commands or **mark_char_abs** must be used instead. • The called character should not contain **mark_text** commands that also contain this character. Such text is *not* marked. The called character itself then might not be complete. • See also Chapter 6.5.2 "Character Sets and Text Strings", page 121.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_char_abs**, **mark_text**, **set_char_pointer**, **get_char_pointer**

Undelayed Short List Command	**mark_char_abs**
Function	Marks an indexed character. In the called command list (see below), any absolute Vector Commands and "Arc" Commands receive an offset (based on the current coordinates at the time of the call).
Call	`mark_char_abs( Char )`
Parameters	`Char` Index of the indexed character to be marked. As an unsigned 32-bit value. Allowed value range: [0…1023]). The following applies: `Char` = character set number × 256 + ASCII number of the character (character sets are numbered 0…3).
Multi-board Com'd Name	**n_mark_char_abs**
Comments	• **mark_char_abs** reads the indexed character's starting address from the internal management table based on the supplied index and then calls **list_call_abs** (see also the comments there), which then starts the corresponding command list. • If the command list of the called character contains no absolute commands, then there is no difference between **mark_char_abs** and **mark_char**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_char**

Normal List Command	mark_date
Function	Marks a part of the date previously stored by **time_fix**, **time_fix_f** or **time_fix_f_off** in the selected format at the current position.
Call	`mark_date( Part, Mode )`
Parameters	`Part` Specifies which part of the date should be marked. = 0: Year (only the last two digits). = 1: Month (in customer specific notation). = 2: Day (corresponding to the normal Gregorian date). = 3: Day-of-the-week (in customer specific notation). = 4: Julian day (see also **time_fix_f_off**). = 5: Year (4 digits). = 6: Month as numerals (January = 1,…). = 7: Day-of-the-week as numerals (Sunday = 1,…). As an unsigned 32-bit value. Allowed value range: [0…7].
	`Mode` Selection of the format. As an unsigned 32-bit value. Allowed value range: [0…3]. a) Only affects `Part = 2`, 4, 6 or 7: The number of leading zeros in the day number. Bit #0 = 0: Leading zeros are suppressed. Bit #0 = 1: Day of the month (`Part = 2`), always two digits. Day of the year (`Part = 4`), always three digits. Month (`Part = 6`), always two digits. Day-of-the-week (`Part = 7`), always two digits. b) Only affects `Part = 0`, 2 or 4…7: Desired character set for marking digits. Bit #1 = 0: The indexed text string for digits [0…9], as defined by **load_text_table** or **set_text_table_pointer**, is marked. Bit #1 = 1: The indexed characters for digits [0…9], as defined by **load_char** or **set_char_pointer**, are marked. The desired character set can be specified with **select_char_set**. For `Part` = 1 or 3, days of the month or days of the week are marked by indexed text strings (Index = 10…28) defined with **load_text_table** or **set_text_table_pointer** (here, the parameter `Mode` is ignored). For marking as numerals, also `Part` = 6 or 7 can be used (then `Mode` is considered).
Multi-board Com'd Name	**n_mark_date**

Normal List Command	mark_date
Comments	• Before marking dates (after every boot-up), the RTC6 and PC times should be synchronized (see **time_update**) and the current (to be marked) date value should be stored with **time_fix**, **time_fix_f** or **time_fix_f_off** (see Chapter 7.5 "Marking Dates, Times and Serial Numbers", page 219). • The complete date can be marked by multiple calls of the **mark_date** command. • The **mark_date** command reads (according to the stored date and according to the selected date part) the starting address of the corresponding indexed text string or character from the internal management table and then calls **list_call** (see also the comments there) an appropriate number of times, which then starts the corresponding command list. The command lists must contain marking instructions for digits 0…9 and for the month and day-of-the-week designations (see Section "Defining Indexed Text Strings for Time, Date and Serial Number", page 122). Non-defined text strings or characters are ignored (that is, not marked). The called indexed text strings can also contain calls to indexed characters (**mark_char** or **mark_char_abs**) and complete texts (**mark_text** or **mark_text_abs**). In the latter case, the character set can be switched when needed (before marking by **mark_date**) with **select_char_set** (see also Chapter 6.5.2 "Character Sets and Text Strings", page 121). • If `Part` > 7 and/or `Mode` > 3, then **mark_date** is, already during loading, replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • Absolute Vector Commands and "Arc" Commands execute absolutely after being called with **mark_date**. If date markings need to execute at various locations within the Image Field, then the corresponding indexed text strings (or characters) can only contain relative [*]mark[*] Commands, "Arc" Commands or Jump Commands or **mark_date_abs** must be used instead. • When marking Gregorian dates or Julian days, the transition to the next day occurs at 00:00 o'clock. Leap years are represented in both date styles.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**time_fix**, **time_fix_f**, **time_fix_f_off**, **load_text_table**, **set_text_table_pointer**, **mark_date_abs**

Normal List Command	**mark_date_abs**
Function	Marks a part of the date previously stored by **time_fix**, **time_fix_f** or **time_fix_f_off** in the selected format at the current position. In the called indexed text strings or characters (see below), any absolute Vector Commands and "Arc" Commands receive an offset (based on the current coordinates at the time of the call).
Call	`mark_date_abs( Part, Mode )`
Parameters	`Part` See **mark_date**.
	`Mode` See **mark_date**.
Multi-board Com'd Name	**n_mark_date_abs**
Comments	• **mark_date_abs** has the same effect as **mark_date**. However, internal calling of the indexed text strings (or characters) is by **list_call_abs** instead of **list_call**. • If the command lists of the called indexed text strings (or characters) contain no absolute commands, then there is no difference between **mark_date_abs** and **mark_date**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_date**

Normal List Command	**mark_ellipse_abs**	
Function	Moves the laser focus at mark speed along an elliptical arc around the specified midpoint (absolute coordinate values) within a 2D Image Field.	
Call	`mark_ellipse_abs( X, Y, Alpha )`	
Parameters	`X`	Absolute x coordinate of the ellipse midpoint. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values.
	`Y`	Like `X` (analogously).
	`Alpha`	Angle between elliptical half-axis a (defined by **set_ellipse**) and the x axis (the angle is referenced to the positive x direction, positive angle values correspond to counterclockwise angles). In degrees. As a 64-bit IEEE floating point value. `Alpha` gets normalized to the value range [0…<360°].
Multi-board Com'd Name	**n_mark_ellipse_abs**	
Comments	• The parameters for **mark_ellipse_abs** only determine the position and orientation of the to-be-executed arc. Before execution of **mark_ellipse_abs**, its shape must have been specified by **set_ellipse**. For descriptions of the individual parameters, see also Section "Ellipse Commands", page 141. • If the arc starting point defined by **mark_ellipse_abs** and **set_ellipse** does not equal the current position, then a "Hard Jump" to the starting point is executed prior to marking, see also notes in Section "Ellipse Commands", page 141. • See also all comments for **arc_abs**.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `X` and `Y` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_ellipse**, **mark_ellipse_rel**, **set_mark_speed**, **set_scanner_delays**, **arc_abs**	

Normal List Command	mark_ellipse_rel	
Function	Moves the laser focus at mark speed along an elliptical arc around the specified midpoint (relative coordinate values) within a 2D Image Field.	
Call	`mark_ellipse_rel( dX, dY, Alpha )`	
Parameters	`dX`	Relative x coordinate of the ellipse midpoint. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values.
	`dY`	Like `dX` (analogously).
	`Alpha`	Angle between elliptical half-axis a (defined by **set_ellipse**) and the x axis (see **mark_ellipse_abs**). In degrees. As a 64-bit IEEE floating point value.
Multi-board Com'd Name	**n_mark_ellipse_rel**	
Comments	• The coordinates for the ellipse midpoint are to be specified as relative coordinates with respect to the current position. Otherwise, **mark_ellipse_rel** is identical to **mark_ellipse_abs** (see comments there).	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `dX` and `dY` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**mark_ellipse_abs**, **set_ellipse**	

Normal List Command	mark_rel
Function	Moves the laser focus at mark speed along a 2D vector from the current position to the specified position (relative coordinate values) within a 2D Image Field.
Call	`mark_rel( dX, dY )`
Parameters	`dX` Relative x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`dY` Like `dX` (analogously).
Multi-board Com'd Name	**n_mark_rel**
Comments	• The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **mark_rel** is identical to **mark_abs** (see comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the values specified for `dX` and `dY` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_abs**, **mark_rel_3d**, **timed_mark_rel**

Normal List Command	**mark_rel_3d**	
Function	Moves the laser focus at mark speed along a 3D vector from the current position to the specified position (relative coordinate values) in the 3D Image Field.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **mark_rel_3d** has the same effect as **mark_rel**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.	
Call	`mark_rel_3d( dX, dY, dZ )`	
Parameters	`dX`	Relative x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`dY`	Like `dX` (analogously).
	`dZ`	Like `dX` (analogously), except Allowed value range: [–524,288…+524,287].
Multi-board Com'd Name	**n_mark_rel_3d**	
Comments	• The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **mark_rel_3d** is identical to **mark_abs_3d** (see comments there).	
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the values specified for `dX`, `dY` and `dZ` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range. In RTC5 Compatibility Mode, the RTC6 multiplies the value specified for `dZ` by 16. The allowed value range decreases accordingly.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**mark_abs_3d**, **mark_abs**, **mark_rel**, **timed_mark_rel_3d**	

Normal List Command	**mark_serial**
Function	Marks the current serial number of the serial-number-set most recently selected by **select_serial_set_list** (or of serial-number-set 0 after **load_program_file**) in the selected format at the current position. Afterward the serial number is (optionally) automatically incremented.
Call	`mark_serial( Mode, Digits )`
Parameters	`Mode` Selection of the serial number format and (de)activation of automatic serial number incrementing. As an unsigned 32-bit value. `Mode` = $20 \times M_1 + 10 \times M_2 + M_3$ Allowed value range: for M_1 [0, 1], for M_2 [0, 1], for M_3 [0…2]. a) Selection of the character set for digit marking. • $M_1 = 0$: The indexed text string for digits [30…39], as defined by **load_text_table** or **set_text_table_pointer**, is marked. • $M_1 = 1$: The indexed characters for digits [0…9], as defined by **load_char** or **set_char_pointer**, are marked. The desired character set can be selected (previously) with **select_char_set**. b) Incrementing of the serial number after marking. • $M_2 = 0$: The serial number is incremented after marking. • $M_2 = 1$: The serial number is *not* incremented after marking. c) Marking of leading zeros. • $M_3 = 0$: Leading zeros are marked as zeros. Dependent on M_1 the corresponding indexed text string definition (Index = 30) or the indexed character definition '0' is used. • $M_3 = 1$: Leading zeros are suppressed (left-justified marking). • $M_3 = 2$: Leading zeros are marked as blank characters (right-justified marking). Dependent on M_1 the corresponding indexed text string definition (Index = 29) or the indexed character definition ' ' is used.
	`Digits` Number [0-12] of to-be-marked digits. As an unsigned 32-bit value. Allowed value range: [0-12]. Larger values are clipped.
Multi-board Com'd Name	**n_mark_serial**
Comments	• The first serial number to be marked must have been previously specified by **set_serial**, **set_serial_step** or **set_serial_step_list**; otherwise, the starting serial number is 0. The starting serial number can have a maximum length of 10 digits. • With every call of **mark_serial**, the serial number is formatted in accordance with M_3 and when $M_2 = 0$ it is automatically incremented before the actual marking. Here, the increment size is 1 unless otherwise specified by **set_serial_step** or **set_serial_step_list**. • The current serial number can be queried with **get_list_serial**, for example, after an aborted list to determine if the current number has been incremented or not.

Normal List Command	mark_serial
Comments (cont'd)	• If the incremented serial number exceeds 10^{Digits}, then marking begins again at 0. The control command **set_max_counts** allows specification of the maximum number of External Starts and thus the maximum number of markings. Here, all markings of all serial-number-sets contribute jointly to the count. • If `Digits` = 0, then a "markless" marking is executed. If M_2 = 0, then the serial number is incremented by 1 (any increment size defined by **set_serial_step** or **set_serial_step_list** are not used in this case!). This can be useful, if a single serial number is to be omitted (repeat n times if necessary; n = increment), but can also be used to indirectly increase the serial number by an *additional* increment. • For each to-be-marked serial number digit, the **mark_serial** command reads the starting address of the corresponding indexed text string (or – for M_1 = 1 – of the corresponding indexed character) from the internal management table and then calls **list_call** (see also the comments there) an appropriate number of times, which starts the corresponding command lists. The command lists must contain marking instructions for digits 0…9 (see Section "Defining Indexed Text Strings for Time, Date and Serial Number", page 122). Non-defined text strings or characters are ignored (that is, not marked). The called indexed text strings can also contain calls to indexed characters (**mark_char** or **mark_char_abs**) and complete texts (**mark_text** or **mark_text_abs**). In the latter case, the character set can be switched if needed (before marking by **mark_serial**) with **select_char_set** (see also Chapter 6.5.2 "Character Sets and Text Strings", page 121). • For invalid `Mode` values, the **mark_serial** is, already during loading, replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • Absolute Vector Commands and "Arc" Commands execute absolutely after being called with **mark_serial**. If serial number markings need to execute at various locations within the Image Field, then the corresponding indexed text strings (or characters) can only contain relative [*]mark[*] Commands, "Arc" Commands or Jump Commands or **mark_serial_abs** must be used instead.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_serial**, **set_serial_step**, **set_serial_step_list**, **get_list_serial**, **set_max_counts**, **get_counts**, **load_text_table**, **set_text_table_pointer**, **mark_serial_abs**

Normal List Command	mark_serial_abs
Function	Marks the current serial number of the serial-number-set most recently selected by **select_serial_set_list** (or of serial-number-set 0 after **load_program_file**) in the selected format at the current position. In the called indexed text strings or characters (see below), any absolute Vector Commands and "Arc" Commands receive an offset (based on the current coordinates at the time of the call). Afterward the serial number is (optionally) automatically incremented.
Call	`mark_serial_abs( Mode, Digits )`
Parameters	`Mode` See **mark_serial**.
	`Digits` See **mark_serial**.
Multi-board Com'd Name	**n_mark_serial_abs**
Comments	• **mark_serial_abs** has the same effect as **mark_serial**; however, internal calling of the indexed text strings (or characters) is by **list_call_abs** instead of **list_call**. • If the command lists of the called indexed text strings (or characters) contain no absolute commands, then there is no difference between **mark_serial_abs** and **mark_serial**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_serial**

Variable List Command	**mark_text**
Function	Marks a \0-terminated string.
Call	`mark_text( Text )`
Parameters	`Text` PC memory address of the first character (byte) of the to-be-marked text string. As a pointer to a \0-terminated string.
Multi-board Com'd Name	**n_mark_text**
Comments	• The to-be-marked text (character sequence, byte array, \0-terminated string) must be terminated with a \0 character (0 byte, NULL). The 0\ character itself is not marked. • When a **mark_text** is loaded, the to-be-marked text (if more than 12 characters in length, \0 not included) is split into blocks of 12 characters, with each block receiving its own **mark_text** command in the RTC6 List Memory (make sure that no undesired memory overflow of the respective RTC6 List Memory area occurs). During processing of the individual **mark_text** commands, the corresponding **mark_char** commands (indexed characters) are executed in accordance with the selected character set. • The desired character set can be selected prior to **mark_text** by **select_char_set**. For the default setting, character set 0 is used. If the **select_char_set** is used within a called indexed character, then all subsequently called indexed characters are marked using this character set. • If the end of a list ("List 1" or "List 2") is reached during loading of a **mark_text**, then loading continues at the start of the corresponding list. In contrast, loading in the protected area (as part of an indexed subroutine) is aborted (**get_last_error** return code `RTC6_REJECTED`) and the indexed subroutine is not stored. • Absolute Vector Commands and "Arc" Commands execute absolutely after being called by **mark_text**. If the text string needs to execute at various locations within the Image Field, then either the indexed character definitions can only contain relative [*]mark[*] Commands, "Arc" Commands or Jump Commands or **mark_text_abs** must be used instead. • **mark_text** should not be used within an indexed character definition. The corresponding text is *not* marked and the indexed character is therefore not fully processed.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_text_abs**

Variable List Command	**mark_text_abs**
Function	Marks a \0-terminated string. In the called indexed characters (see below), any absolute Vector Commands and "Arc" Commands receive an offset (based on the current coordinates at the time of the call).
Call	`mark_text_abs( Text )`
Parameters	`Text` PC memory address of the first character (byte) of the to-be-marked text string. As a pointer to a \0-terminated string.
Multi-board Com'd Name	**n_mark_text_abs**
Comments	• During processing of the individual **mark_text_abs**, the corresponding **mark_char_abs** (indexed characters) are executed in accordance with the selected character set. • If the command list of the called indexed character contains no absolute commands, then there is no difference between **mark_text_abs** and **mark_text**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_text**

Normal List Command	**mark_time**
Function	Marks a part of the time previously stored by **time_fix**, **time_fix_f** or **time_fix_f_off** in the specified format at the current position.
Call	`mark_time( Part, Mode )`
Parameters	`Part` Specifies which part of the time to mark. = 0: Hours (24-h time, no a.m./p.m.) = 1: Minutes = 2: Seconds = 3: Hours (12-h time, no a.m./p.m.) = 4: a.m./p.m. (automatically switched in accordance with 24-h time) As an unsigned 32-bit value. Allowed value range: [0…4].
	`Mode` Format. As an unsigned 32-bit value. Allowed value range: [0…3], only affects `Part = 0…3`). a) Number of leading zeros: Bit #0 = 0: Leading zeros are suppressed. Bit #0 = 1: Always two digits. b) Character set choice for marking of digits: Bit #1 = 0: The indexed text strings for digits [0…9], as defined by **load_text_table** or **set_text_table_pointer**, are marked. Bit #1 = 1: The indexed characters for digits [0…9], as defined by **load_char** or **set_char_pointer**, are marked. The desired character set can be specified with **select_char_set**. a.m./p.m. (Part = 4) can only be marked in accordance with the indexed text strings (Index = 40, 41) defined by **load_text_table** or **set_text_table_pointer**. Here, the parameter Mode is ignored.
Multi-board Com'd Name	**n_mark_time**
Comments	• Before marking times (after every boot-up), the RTC6 and PC times should be synchronized (see **time_update**) and the current (to be marked) time should be stored with **time_fix**, **time_fix_f** or **time_fix_f_off**, see Chapter 7.5 "Marking Dates, Times and Serial Numbers", page 219. • The complete time can be marked by multiple calls of the **mark_time** command.

Normal List Command	mark_time
Comments (cont'd)	• **mark_time** reads (according to the stored time and according to the selected time part) the starting address of the corresponding indexed text string (or – for `Part` = 0…3 and `Mode` = 2 or 3 – of the corresponding indexed character) from the internal management table and then calls **list_call** (see also the comments there) an appropriate number of times, which starts the corresponding command list. The command lists must contain marking instructions for digits 0…9 and for a.m./p.m, see Section "Defining Indexed Text Strings for Time, Date and Serial Number", page 122. Non-defined text strings or characters are ignored (that is, not marked). The called indexed text strings can also contain calls to indexed characters (**mark_char** or **mark_char_abs**) and complete texts (**mark_text** or **mark_text_abs**). In the latter case, the character set can be switched, if needed (before marking with **mark_time**), by **select_char_set**, see also Chapter 6.5.2 "Character Sets and Text Strings", page 121. • If `Part` > 4 and/or `Mode` > 3, then **mark_time** is, already during loading, replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • Absolute Vector Commands and "Arc" Commands execute absolutely after being called with **mark_time**. If time markings need to execute at various locations within the Image Field, then the corresponding indexed text strings (or characters) can only contain relative [*]mark[*] Commands, "Arc" Commands or Jump Commands or **mark_time_abs** must be used instead.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**time_fix**, **time_fix_f**, **time_fix_f_off**, **load_text_table**, **set_text_table_pointer**, **mark_time_abs**

Normal List Command	**mark_time_abs**
Function	Marks a part of the time previously stored by **time_fix**, **time_fix_f** or **time_fix_f_off** in the specified format at the current position. In the called indexed text strings or characters (see below), any absolute Vector Commands and "Arc" Commands receive an offset (based on the current coordinates at the time of the call).
Call	`mark_time_abs( Part, Mode )`
Parameters	`Part` See **mark_time**.
	`Mode` See **mark_time**.
Multi-board Com'd Name	**n_mark_time_abs**
Comments	• **mark_time_abs** has the same effect as **mark_time**; however, internal calling of the indexed text strings (or characters) is by **list_call_abs** instead of **list_call**. • If the command lists of the called indexed text strings (or characters) contain no absolute commands, then there is no difference between **mark_time_abs** and **mark_time**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_time**

Ctrl Command	master_slave_config
Function	Sets settings for the master-slave interface of an RTC6 board.
Call	`master_slave_config( Config )`
Parameters	`Config` As an unsigned 32-bit value. Bit #0 Suppression of /Slave-START signals and /Slave-STOP signals. = 0: /Slave-START and /Slave-STOP signals via the master-slave interface are received and processed. = 1: /Slave-START and /Slave-STOP signals received by this board via the master-slave interface are ignored on this board, but are forwarded to other boards of the master-slave interface. Bit #1 /STOP in case of master-slave faults. = 0: The board does not react explicitly to a fault in the master-slave connection. Any effects are undefined. = 1: Triggers a /STOP, if the connection to the master card is interrupted. This occurs, for example, if **load_program_file** is executed on a card of the already synchronized master-slave chain, the master-slave cable is removed or the signal of the master-slave interface is electromagnetically disturbed. Bit #2 Forwarding /STOP in case of master-slave faults. = 0: A /STOP is not forwarded to other boards. = 1: If an error of the master-slave connection is detected and a /STOP is triggered (see Bit #1), this /STOP is forwarded to all still connected cards of the master-slave chain (also upwards to a master card).
Multi-board Com'd Name	**n_master_slave_config**
Comments	• For usage of **master_slave_config**, see Chapter 6.6.3 "Master/Slave Operation", page 127. • The master-slave interface of a board should normally only be configured after successful synchronization. • If Bit #0 is already set before **sync_slaves**, this board cannot be synchronized to a master board with **sync_slaves**. • In certain other malfunction scenarios, fault-free operation may no longer be possible. In this case, a /STOP is always performed on the affected board, even if Bit #1 = 0. • If the connection to a master-slave synchronized board is disturbed, this can also result in further boards in the chain being disturbed, and thus execute a /STOP, even if Bit #2 is not set.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 609, OUT 609, RBF 613.
References	**sync_slaves**

Ctrl Command	mcbsp_init
Function	Defines the data delays for transmitting and receiving data by the McBSP interface, see also Section "McBSP Interface", page 87.
Call	`mcbsp_init( XDelay, RDelay )`
Parameters	`XDelay` Transmission delay. As an unsigned 32-bit value. Allowed value range: [0…2].
	`RDelay` Receiving delay. As an unsigned 32-bit value. Allowed value range: [0…2].
Multi-board Com'd Name	**n_mcbsp_init**
Comments	• For invalid parameter values, **mcbsp_init** is not executed (**get_last_error** return code `RTC6_PARAM_ERROR`). • The initialized values (after program start) are `XDelay` = `RDelay` = 1. • The signals and operating conditions of the McBSP interface are presented in Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87. • The McBSP interface ignores the first FrameSync signal after a **mcbsp_init**. That is, data provided is not transmitted, see page 89. • **mcbsp_init** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**read_mcbsp**, **set_mcbsp_out**, **set_mcbsp_out_ptr**, **set_mcbsp_freq**, **mcbsp_init_spi**

Ctrl Command	mcbsp_init_spi
Function	No function.
Call	`mcbsp_init_spi( ClockLevel, ClockDelay )`
Parameters	`ClockLevel` = 0: inactive low. > 0: inactive high. As an unsigned 32-bit value. `ClockDelay` = 0: Clock signal and data bits at the same time. > 0: Clock signal is delayed a half period. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_mcbsp_init_spi**
Comments	• See Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87.
RTC4→RTC6	New command.
RTC5→RTC6	Changed functionality. • The RTC6 command set contains **mcbsp_init_spi**, but it has no effect. Execution of it is refused and the **get_last_error** return code is `RTC6_REJECTED`. • Notes on migrating the source code of RTC5 user programs, etc.: For hardware reasons, RTC6 boards are no longer configurable for SPI functionality. You need to appropriately modify any such source code sections.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mcbsp_init**, **set_mcbsp_freq**

Ctrl Command	measurement_status
Function	Returns: • Status of a measurement started by set_trigger[*] • Current position of the measurement counter
Call	`measurement_status( &Busy, &Pos )`
Returned Parameter Values	`Busy` Measurement status. As a pointer to an unsigned 32-bit value. > 0: A measurement session is currently in progress. = 0: No measurement session is currently in progress.
	`Pos` Current position of the measurement counter (within the RTC6 measurement data memory) [0…max. channel size, see set_trigger[*]]. As a pointer to an unsigned 32-bit value.
Multi-board Com'd Name	**n_measurement_status**
Comments	• If a measurement session started by set_trigger[*] is no longer active, then `Pos`+1 indicates the number of recorded data pairs up to termination ([0…max. channel size, see set_trigger[*]]). • `Pos` = 2^{32}–1 indicates that data recording still has not occurred after load_program_file. • Stored data can be queried with get_waveform. • The status of the measurement recording is reset by (see set_trigger comments): – set_trigger(`Period` = 0) – set_trigger4(`Period` = 0) – set_trigger8(`Period` = 0) – stop_trigger – stop_execution
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	set_trigger, set_trigger4, set_trigger8

Normal List Command	**micro_vector_abs**
Function	Moves the output point (of the laser focus) by a "Hard Jump" (without split-up into Microsteps) directly from the current position to the specified position (absolute coordinate values) within the 2D Image Field.
Call	`micro_vector_abs( X, Y, LasOn, LasOff )`
Parameters	`X` Absolute x coordinate of the micro vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`Y` Like `X` (analogously).
	`LasOn` LaserOn Delay. 1 bit equals 1/64 µs. As a signed 32-bit value. Allowed value range: [–1…+(2^{15}–1)]. ≥ 0: Delay is newly set. Values over (2^{15}–1) are clipped. < 0: The previously set delay continues unaffected.
	`LasOff` LaserOff Delay. 1 bit equals 1/64 µs. See `LasOn`.
Multi-board Com'd Name	**n_micro_vector_abs**
Comments	• See also Chapter 8.8 "micro_vector[*] Commands", page 282. • Wobbel is not taken into account, see 2 in Chapter 7.3.6 "Output Values to the Scan System", page 186. • The Microvector is always executed as a "Hard Jump". • By `LasOn` ≥ 0 and `LasOff` ≥ 0, you can set a new LaserOn Delay or LaserOff Delay for each individual Microvector. Each delay thereby gets set at the end of the clock cycle in which the new position actually gets outputted (this output clock cycle is delayed by a preceding Scanner Delay). • Negative values (`LasOn` < 0 and `LasOff` < 0) do not affect Laser Delays. Hereby, the laser can remain on or off across multiple clock cycles (mark and jump simulation). • Delays set with `LasOn` and `LasOff` only apply to the execution of Microvectors. For execution of normal [*]mark[*] Commands and "Arc" Commands (such as **mark_abs**), only the Laser Delays defined by **set_laser_delays** apply. `LasOn` and `LasOff` do not overwrite the laser delay parameter from **set_laser_delays**.

Normal List Command	micro_vector_abs
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `X` and `Y` by 16, those for `LasOn` and `LasOff` by 64. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the specified values for `LasOn` and `LasOff` by 32. The allowed value ranges decrease accordingly.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**micro_vector_rel**, **micro_vector_abs_3d**

Normal List Command	**micro_vector_abs_3d**
Function	Moves the output point (of the laser focus) by a "Hard Jump" (without split-up into Microsteps) directly from the current position to the specified position (absolute coordinate values) within the 3D Image Field.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **micro_vector_abs_3d** has the same effect as **micro_vector_abs**.
Call	`micro_vector_abs_3d( X, Y, Z, LasOn, LasOff )`
Parameters	`X` — Absolute x coordinate of the micro vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`Y` — Like `X` (analogously).
	`Z` — Like `X` (analogously), except Allowed value range: [–524,288…+524,287].
	`LasOn` — LaserOn Delay. 1 bit equals 1/64 μs. As a signed 32-bit value. Allowed value range: [–1…+(2^{15}–1)]. ≥ 0: Delay is newly set. Values over (2^{15}–1) are clipped. < 0: The previously set delay continues unaffected.
	`LasOff` — LaserOff Delay. Like `LasOn`.
Multi-board Com'd Name	**n_micro_vector_abs_3d**
Comments	• Except for the additional motion in the third dimension, **micro_vector_abs_3d** functions similarly to **micro_vector_abs** (see comments there).
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `X`, `Y` and `Z` by 16, those for `LasOn` and `LasOff` by 64. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the specified value for `Z` by 16, those for `LasOn` and `LasOff` by 32. The allowed value ranges decrease accordingly.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**micro_vector_abs**, **micro_vector_rel_3d**

Normal List Command	**micro_vector_rel**	
Function	Moves the output point (of the laser focus) by a “Hard Jump” (without split-up into Microsteps) directly from the current position to the specified position (relative coordinate values) within the 2D Image Field.	
Call	`micro_vector_rel( dX, dY, LasOn, LasOff )`	
Parameters	`dX`	Relative x coordinate of the micro vector end point. In bits. Otherwise, like `X` from **micro_vector_abs**.
	`dY`	Relative y coordinate of the micro vector end point. In bits. Otherwise, like `Y` from **micro_vector_abs**.
	`LasOn`	Like `LasOn` from **micro_vector_abs**.
	`LasOff`	Like `LasOff` from **micro_vector_abs**.
Multi-board Com’d Name	**n_micro_vector_rel**	
Comments	• The coordinates for the micro vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **micro_vector_rel** is identical to **micro_vector_abs** (see comments there). • The Microvector is always executed as a “Hard Jump”.	
RTC4→RTC6	New command. RTC4 Compatibility Mode: see **micro_vector_abs**.	
RTC5→RTC6	Unchanged functionality. RTC5 Compatibility Mode: see **micro_vector_abs**.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**micro_vector_abs**, **micro_vector_rel_3d**	

Normal List Command	micro_vector_rel_3d	
Function	Moves the output point (of the laser focus) by a "Hard Jump" (without split-up into Microsteps) directly from the current position to the specified position (relative coordinate values) within the 3D Image Field.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **micro_vector_rel_3d** has the same effect as **micro_vector_rel**.	
Call	`micro_vector_rel_3d( dX, dY, dZ, LasOn, LasOff )`	
Parameters	`dX`	Relative x coordinate of the micro vector end point. In bits. Otherwise, like wie `X` von **micro_vector_abs_3d**.
	`dY`	Relative y coordinate of the micro vector end point. In bits. Otherwise, like wie `Y` von **micro_vector_abs_3d**.
	`dZ`	Relative z coordinate of the micro vector end point. In bits. Otherwise, like wie `Z` von **micro_vector_abs_3d**.
	`LasOn`	Like `LasOn` from **micro_vector_abs_3d**.
	`LasOff`	Like `LasOff` from **micro_vector_abs_3d**.
Multi-board Com'd Name	**n_micro_vector_rel_3d**	
Comments	• The coordinates for the micro vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **micro_vector_rel_3d** is identical to **micro_vector_abs_3d** (see comments there). • The Microvector is always executed as a "Hard Jump".	
RTC4→RTC6	New command. RTC4 Compatibility Mode: see **micro_vector_abs_3d**.	
RTC5→RTC6	Unchanged functionality. RTC5 Compatibility Mode: see **micro_vector_abs_3d**.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**micro_vector_abs_3d**, **micro_vector_rel**	

Ctrl Command	move_to
Multi-board Com'd Name	**n_move_to**
Comments	• **move_to** is described in the manual "Installation and Operation RTC Step Motor Extension for the RTC4 and RTC5 PC interface boards" (available in English only). • **move_to** has been introduced for the "RTC STEP MOTOR EXTENSION" extension board (#0112097). • **move_to** is *not* supported by "RTC5/6 varioSCAN 40 FLEX Extension" Extension Board (#0128683). See also Chapter 2.8.7 "Extension Board "RTC5/6 varioSCAN FLEX Extension"", page 47.

Ctrl Command	number_of_correction_tables
Function	Defines the maximum number of allowed correction tables.
Call	`number_of_correction_tables( Number )`
Parameters	`Number` Maximum number of allowed correction tables. As an unsigned 32-bit value. Allowed value range: [1…8]. Default after **load_program_file**: 8. See also Chapter 8.5.3 ”Using Several Correction Tables”, page 250.
Multi-board Com’d Name	**n_number_of_correction_tables**
Comments	• For outside the allowed value range, **number_of_correction_tables** is ignored (**get_last_error** return code `RTC6_PARAM_ERROR`). • **number_of_correction_tables** serves to protect other commands (for example, such as **load_correction_file** and **select_cor_table**) from unwanted table numbers. • Existing user programs do not have to be changed. The exception is, if user input is to be rejected (using explicit RTC6 error messages) in the future. • **number_of_correction_tables** refers only to subsequent command executions. Existing assignments of correction tables cannot be corrected automatically. This is particularly important, if RTC6 boards that have been initialized by other user programs are subsequently acquired.
RTC4→RTC6	New command.
RTC5→RTC6	Changed impacts (the RTC5-command has not been documented due to the restricted user group).
Version Info	Available as of DLL 609, OUT 609, RBF 613.
References	**load_correction_file**, **select_cor_table**, **select_cor_table_list**

Normal List Command	para_jump_abs
Function	Moves the output point for the laser focus along a 2D vector at jump speed from the current position to the specified position (absolute coordinate values) within a 2D Image Field. Simultaneously varies the signal parameter selected by **set_vector_control** linearly to the specified value.
Call	`para_jump_abs( X, Y, P )`
Parameters	`X` — Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`Y` — Like `X` (analogously).
	`P` — End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by **set_vector_control** (`Ctrl` parameter), identical with **set_vector_control** (`Value` parameter).
Multi-board Com'd Name	**n_para_jump_abs**
Comments	• If Vector-Defined Laser Control has not been previously activated by **set_vector_control**, then **para_jump_abs** behaves like **jump_abs** (see comments there). The parameter `P` is then ignored. • If Vector-Defined Laser Control is activated, then simultaneously with the motion of the output point of the laser focus the signal parameter selected by **set_vector_control** is linearly varied from the last valid value to `P` (see Section "Vector-Defined Laser Control", page 214). • There is no abs mechanism for `P`. `P` is clipped to the maximum allowed value. • If **para_jump_abs** is used along with Position-Dependent Laser Control or Speed-Dependent Laser Control for the same control parameter, then the current value of `P` is used as the basis of the 100% value for laser control, see Section "Vector-Defined Laser Control", page 214.
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `X` and `Y` by 16. The allowed value ranges decrease accordingly. There is no RTC4 Compatibility Mode for the parameter `P`. The original RTC6 units must be used. Exception is `Ctrl` = 7 (Defocus): In RTC4 Compatibility Mode, with `Ctrl` = 7, the RTC6 multiplies the specified values for `Value` by 16. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality. Exception is `Ctrl` = 7 (Defocus): In RTC5 Compatibility Mode, with `Ctrl` = 7, the RTC6 multiplies the specified values for `Value` by 16. The allowed value ranges decrease accordingly.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**jump_abs**, **set_vector_control**, **para_jump_abs_3d**, **para_jump_rel**

Normal List Command	**para_jump_abs_3d**	
Function	Moves the output point (of the laser focus) along a 3D vector at jump speed from the current position to the specified position (absolute coordinate values) within the 3D Image Field. Simultaneously varies the signal parameter selected by **set_vector_control** linearly to the specified value.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **para_jump_abs_3d** has the same effect as **para_jump_abs**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.	
Call	`para_jump_abs_3d( X, Y, Z, P )`	
Parameters	`X`	Absolute x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`Y`	Like `X` (analogously).
	`Z`	Like `X` (analogously), except Allowed value range: [–524,288…+524,287].
	`P`	End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by **set_vector_control** (`Ctrl` parameter), identical with **set_vector_control** (`Value` parameter).
Multi-board Com'd Name	**n_para_jump_abs_3d**	
Comments	• Except for the additional motion in the third dimension, **para_jump_abs_3d** functions similarly to the **para_jump_abs** command (see comments there). • Further comments see **jump_abs_3d**.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `X`, `Y` and `Z` coordinates by 16. The allowed value ranges decrease accordingly. There is no RTC4 Compatibility Mode for the parameter `P`. The original RTC6 units must be used. Exception is `Ctrl` = 7 (Defocus): In RTC4 Compatibility Mode, with `Ctrl` = 7, the RTC6 multiplies the specified values for `Value` by 16. The allowed value ranges decrease accordingly.	
RTC5→RTC6	Unchanged functionality. Exception is `Ctrl` = 7 (Defocus): In RTC5 Compatibility Mode, with `Ctrl` = 7, the RTC6 multiplies the specified values for `Value` by 16. The allowed value ranges decrease accordingly.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**jump_abs_3d**, **set_vector_control**, **para_jump_abs**, **para_jump_rel_3d**	

Normal List Command	para_jump_rel	
Function	Moves the output point (of the laser focus) along a 2D vector at jump speed from the current position to the specified position (relative coordinate values) within a 2D Image Field. Simultaneously varies the signal parameter selected by **set_vector_control** linearly to the specified value.	
Call	`para_jump_rel( dX, dY, P )`	
Parameters	`dX`	Relative x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`dY`	Like `dX` (analogously).
	`P`	End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by **set_vector_control** (`Ctrl` parameter), identical with **set_vector_control** (`Value` parameter).
Multi-board Com'd Name	**n_para_jump_rel**	
Comments	• The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **para_jump_rel** is identical to **para_jump_abs** (see comments there). • `P` is not treated on a relative basis.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `dX` and `dY` by 16. The allowed value range decreases accordingly. For the parameter `P`, see **para_jump_abs**.	
RTC5→RTC6	Unchanged functionality. For the parameter `P`, see **para_jump_abs**.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**jump_rel**, **set_vector_control**, **para_jump_abs**, **para_jump_rel_3d**	

Normal List Command	**para_jump_rel_3d**
Function	Moves the output point (of the laser focus) along a 3D vector at jump speed from the current position to the specified position (relative coordinate values) within the 3D Image Field. Simultaneously varies the signal parameter selected by **set_vector_control** linearly to the specified value.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **para_jump_rel_3d** has the same effect as **para_jump_rel**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.
Call	`para_jump_rel_3d( dX, dY, dZ, P )`
Parameters	`dX` Relative x coordinate of the jump vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`dY` Like `dX` (analogously).
	`dZ` Like `dX` (analogously), except Allowed value range: [–524,288…+524,287].
	`P` End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by **set_vector_control** (`Ctrl` parameter), identical with **set_vector_control** (`Value` parameter).
Multi-board Com'd Name	**n_para_jump_rel_3d**
Comments	• The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **para_jump_rel_3d** is identical to **para_jump_abs_3d** (see comments there). • `P` is not treated on a relative basis.
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `dX` and `dY` by 16. The allowed value range decreases accordingly. For the parameter `P`, see **para_jump_abs**.
RTC5→RTC6	Unchanged functionality. For the parameter `P`, see **para_jump_abs**.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**jump_rel_3d**, **set_vector_control**, **para_jump_abs_3d**, **para_jump_rel**

Variable List Command	**para_laser_on_pulses_list**	
Function	Turns on the LASERON signal for the specified number of external signal pulses (but for no longer than the specified time interval). Simultaneously varies the signal parameter selected by **set_vector_control** linearly to the specified value.	
Call	`para_laser_on_pulses_list( Period, Pulses, P )`	
Parameters	`Period`	Time interval. In bits. As an unsigned 32-bit value. 1 bit equals 10 μs. Allowed value range: 0 ≤ `Period` ≤ (2^{32}–1).
	`Pulses`	Number of external signal pulses. As an unsigned 32-bit value. Allowed value range: 0 ≤ `Pulses` ≤ 65,535 or larger (see comments below).
	`P`	End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by **set_vector_control** (`Ctrl` parameter), identical with **set_vector_control** (`Value` parameter).
Multi-board Com'd Name	**n_para_laser_on_pulses_list**	
Comments	• **para_laser_on_pulses_list** is useful for marking single dots, see Chapter 7.1.3 "Marking Single Dots", page 144. • If Vector-Defined Laser Control has not been previously activated by **set_vector_control**, then **para_laser_on_pulses_list** behaves like **laser_on_pulses_list** and – if `Pulses` > 65,535 – like **laser_on_list** (see comments there). The parameter `P` is then ignored. • If Vector-Defined Laser Control is activated, then the signal parameter selected by **set_vector_control** is linearly varied from the last valid value to `P` within **para_laser_on_pulses_list** duration (`Period` × 10 μs), see Section "Vector-Defined Laser Control", page 214. • There is no abs mechanism for `P`. `P` is clipped to the maximum allowed value. This maximum value is (2^{31}–1) or a lower value depending on what has been selected with **set_vector_control** (`Ctrl` parameter). • If **para_laser_on_pulses_list** is used along with Position-Dependent Laser Control or Speed-Dependent Laser Control for the same control parameter, then the current value of `P` is used as the basis of the 100% value for laser control, see Section "Vector-Defined Laser Control", page 214. • If `Period` = 0, **para_laser_on_pulses_list** has no effect. **para_laser_on_pulses_list** is a Short List Command then. • If 0 < `Period` ≤ (2^{31}–1), then the **para_laser_on_pulses_list** duration is always `Period` clocks (that is, `Period` × 10 μs), even if the specified number of external signal pulses expires in a shorter time interval. • If 2^{31} ≤ `Period` ≤ (2^{32}–1), the **para_laser_on_pulses_list** maximum duration is (`Period` – 2^{31}) clocks (that is, (`Period` – 2^{31}) × 10 μs). Here, however, **para_laser_on_pulses_list** terminates as soon as the specified number of external signal pulses has been detected.	

Variable List Command	para_laser_on_pulses_list
RTC4→RTC6	New command. For the parameter P, see **para_jump_abs**.
RTC5→RTC6	Unchanged functionality. For the parameter P, see **para_jump_abs**.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**laser_on_pulses_list**, **laser_on_list**

Normal List Command	**para_mark_abs**
Function	Moves the laser focus at mark speed along a 2D vector from the current position to the specified position (absolute coordinate values) within a 2D Image Field. Simultaneously varies the signal parameter selected by **set_vector_control** linearly to the specified value.
Call	`para_mark_abs( X, Y, P )`
Parameters	`X` Absolute x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`Y` Like `X` (analogously).
	`P` End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by **set_vector_control** (`Ctrl` parameter), identical with **set_vector_control** (`Value` parameter).
Multi-board Com'd Name	**n_para_mark_abs**
Comments	• If Vector-Defined Laser Control has not been previously activated by **set_vector_control**, then **para_mark_abs** behaves like **mark_abs** (see comments there). The parameter `P` is then ignored. • If Vector-Defined Laser Control is activated, then simultaneously with the laser focus' motion the signal parameter selected by **set_vector_control** is linearly varied from the last valid value to `P` see Section "Vector-Defined Laser Control", page 214. • There is no abs mechanism for `P`. `P` is clipped to the maximum allowed value. • ≤ OUT 618: For mark vector lengths = 0, no change of signal parameter `P` must be programmed: – This change is not outputted – The following [*]para[*] Command (only this one) produces incorrect signal parameter outputs • ≥ OUT 619: For mark vector lengths = 0 where a change of signal parameter `P` is programmed, the following applies: – These are handled as timed vectors with `T` = 10 µs – The end value of the signal parameter `P` is outputted • If **para_mark_abs** is used along with Position-Dependent Laser Control or Speed-Dependent Laser Control for the same control parameter, then the current value of `P` is used as the basis of the 100% value for laser control (see Section "Vector-Defined Laser Control", page 214). • **[*]para_mark[*]** commands generally do not take Sky Writing into account.

Normal List Command	para_mark_abs
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for X and Y by 16. The allowed value range decreases accordingly. For the parameter P, see **para_jump_abs**.
RTC5→RTC6	Unchanged functionality. For the parameter P, see **para_jump_abs**.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change OUT 619: handling of mark vector lengths = 0.
References	**mark_abs**, **set_vector_control**, **para_mark_abs_3d**, **para_mark_rel**

Normal List Command	**para_mark_abs_3d**	
Function	Moves the laser focus at mark speed along a 3D vector from the current position to the specified position (absolute coordinate values) within the 3D Image Field. Simultaneously varies the signal parameter selected by **set_vector_control** linearly to the specified value.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **para_mark_abs_3d** has the same effect as **para_mark_abs**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.	
Call	`para_mark_abs_3d( X, Y, Z, P )`	
Parameters	`X`	Absolute x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`Y`	Like `X` (analogously).
	`Z`	Like `X` (analogously), except Allowed value range: [–524,288…+524,287].
	`P`	End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by **set_vector_control** (`Ctrl` parameter), identical with **set_vector_control** (`Value` parameter).
Multi-board Com'd Name	**n_para_mark_abs_3d**	
Comments	• Except for the additional motion in the third dimension, this command functions similarly to the **para_mark_abs** command (see comments there). • Further comments see **mark_abs_3d**. • **[*]para_mark[*]** commands generally do not take Sky Writing into account.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for the x and y coordinates by 16. The allowed value range decreases accordingly. For the parameter `P`, see **para_jump_abs**.	
RTC5→RTC6	Unchanged functionality. For the parameter `P`, see **para_jump_abs**.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**mark_abs_3d**, **set_vector_control**, **para_mark_abs**, **para_mark_rel_3d**	

Normal List Command	**para_mark_rel**	
Function	Moves the laser focus at mark speed along a 2D vector from the current position to the specified position (relative coordinate values) within a 2D Image Field. Simultaneously varies the signal parameter selected by **set_vector_control** linearly to the specified value.	
Call	`para_mark_rel( dX, dY, P )`	
Parameters	`dX`	Relative x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`dY`	Like `dX` (analogously).
	`P`	End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by **set_vector_control** (`Ctrl` parameter), identical with **set_vector_control** (`Value` parameter).
Multi-board Com’d Name	**n_para_mark_rel**	
Comments	• The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **para_mark_rel** is identical to **para_mark_abs** (see comments there). • `P` is not treated on a relative basis. • **[*]para_mark[*]** commands generally do not take Sky Writing into account.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `dX` and `dY` by 16. The allowed value range decreases accordingly. For the parameter `P` see **para_jump_abs**.	
RTC5→RTC6	Unchanged functionality. For the parameter `P`, see **para_jump_abs**.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**mark_rel**, **set_vector_control**, **para_mark_abs**, **para_mark_rel_3d**	

Normal List Command	**para_mark_rel_3d**	
Function	Moves the laser focus at mark speed along a 3D vector from the current position to the specified position (relative coordinate values) within the 3D Image Field. Simultaneously varies the signal parameter selected by **set_vector_control** linearly to the specified value.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **para_mark_rel_3d** has the same effect as **para_mark_rel**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.	
Call	`para_mark_rel_3d( dX, dY, dZ, P )`	
Parameters	`dX`	Relative x coordinate of the mark vector end point. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values. The complete value range is only usable as a virtual Image Field for example, for (enabled) Processing-on-the-fly applications.
	`dY`	Like `X` (analogously).
	`dZ`	Like `X` (analogously), except Allowed value range: [–524,288…+524,287].
	`P`	End value of the signal parameter. As an unsigned 32-bit value. Allowed values: dependent on the selection made by **set_vector_control** (`Ctrl` parameter), identical with **set_vector_control** (`Value` parameter).
Multi-board Com'd Name	**n_para_mark_rel_3d**	
Comments	• The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **para_mark_rel_3d** is identical to **para_mark_abs_3d** (see comments there). • `P` is not treated on a relative basis. • **[*]para_mark[*]** commands generally do not take Sky Writing into account.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `dX` and `dY` by 16. The allowed value range decreases accordingly. For the parameter `P`, see **para_jump_abs**.	
RTC5→RTC6	Unchanged functionality. For the parameter `P`, see **para_jump_abs**.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**mark_rel_3d**, **set_vector_control**, **para_mark_abs_3d**, **para_mark_rel**	

<table>
<tr><td>Variable List Command</td><td colspan="2">park_position</td></tr>
<tr><td>Function</td><td colspan="2">For temporary parking, this command moves the output point (of the laser focus) along a 2D vector at jump speed from the current position to the specified position (absolute coordinate values) within the 2D Image Field.</td></tr>
<tr><td>Restriction</td><td colspan="2">If the Option Processing-on-the-fly is not enabled, then park_position functions like a normal jump_abs.</td></tr>
<tr><td>Call</td><td colspan="2"><code>park_position(Mode, X, Y)</code></td></tr>
<tr><td rowspan="3">Parameters</td><td><code>Mode</code></td><td>As an unsigned 32-bit value.
= 0: <code>X</code> and <code>Y</code> are interpreted as park position coordinates in the real Image Field. Allowed value range: [–524,288…+524,287].
The current Processing-on-the-fly mode is switched off and the laser focus moved at the currently set jump speed to the specified park position in the real Image Field. During a subsequent list interruption (by wait_for_encoder etc.), the galvanometer scanners remains stationary (even for a Processing-on-the-fly application initiated by set_fly_2d).
> 0: <code>X</code> and <code>Y</code> are interpreted as park position coordinates in the virtual Image Field. Allowed value range: [–268,435,456…+268,435,455].
The current Processing-on-the-fly mode remains switched on and the laser focus moved at the currently set jump speed to the specified park position in the virtual Image Field (subject to Processing-on-the-fly correction and clipped to the real Image Field’s boundaries). During a subsequent list interruption (by wait_for_encoder etc.), the galvanometer scanners’ positions are continuously Processing-on-the-fly-corrected in accordance with the current encoder values (even for a Processing-on-the-fly application initiated by set_fly_x/set_fly_y) and clipped to the real Image Field’s boundaries.</td></tr>
<tr><td><code>X</code></td><td>Absolute x coordinate of the jump vector end point. In bits.
As a signed 32-bit value.
Allowed value range: see above.
Out-of-range values are clipped to the boundary values.</td></tr>
<tr><td><code>Y</code></td><td>Like <code>X</code> (analogously).</td></tr>
<tr><td>Multi-board Com’d Name</td><td colspan="2">n_park_position</td></tr>
</table>

Variable List Command	park_position
Comments	• **park_position** is intended for encoder-based **set_fly_2d** or **set_fly_x**/**set_fly_y** Processing-on-the-fly applications where the laser focus needs to be moved to a safe parking area during an forwarding motion (prior to list interruption by **wait_for_encoder** etc.), see Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 261. For the return jump (away from the park position), use the **park_return** command (refer to all comments there). • If another (or no) Processing-on-the-fly application is active, then **park_position** functions like a normal Jump Command, as does the return jump by **park_return** (but Processing-on-the-fly remains switched off). • If the laser focus has been already previously moved to a safe park position, then **park_position** is a Short List Command with no effect. • **park_position** switches off the Signals for "Laser Active" Operation after a LaserOff Delay, but does not activate a Scanner Delay afterward.
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for X and Y by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**park_return**

Variable List Command	park_position_1_axis
Function	"Fly Extension" Command: Moves the output position for 1 Axis for intermediate parking with jump speed from the current to the specified absolute position in the 2D Image Field.
Restriction	If the Option Processing-on-the-fly is not enabled, then **park_position_1_axis** only carries-out the jump to the specified new output position.
Call	`park_position_1_axis( Mode, Axis, ParkPos )`
Parameters	`Mode` = 0: Intermediate parking position. Absolute coordinate in the real Image Field. Processing-on-the-fly is switched off at the specified Axis. > 0: Intermediate parking position. Absolute coordinate in the virtual Image Field. Processing-on-the-fly remains switched on at the specified Axis. As an unsigned 32-bit value. `Axis` Axis from Table 3, page 268. As an unsigned 32-bit value. Allowed values: `1`…`2`. `ParkPos` Absolute axis coordinate of the new output position. In bits. As a signed 32-bit value. Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_park_position_1_axis**
Comments	• Being an "Fly Extension" Command, **park_position_1_axis** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • See also comments on **park_position**. • To jump back, **park_return_1_axis** can be used. • With an unallowed parameter value, **park_position_1_axis** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**park_position_2_axes**

Variable List Command	park_position_2_axes
Function	"Fly Extension" Command: Moves the output positions for 2 Axes for intermediate parking with jump speed from the current to the specified absolute position in the 2D Image Field.
Restriction	If the Option Processing-on-the-fly is not enabled, then **park_position_2_axes** only carries-out the jump to the specified new output position.
Call	`park_position_2_axes( Mode, ParkPosX, ParkPosY )`
Parameters	`Mode` = 0: Intermediate parking position. Absolute coordinate in the real Image Field. Processing-on-the-fly is switched off at the specified Axes. > 0: Intermediate parking position. Absolute coordinate in the virtual Image Field. Processing-on-the-fly remains switched on at the specified Axes. As an unsigned 32-bit value. `ParkPosX` Absolute axis coordinate of the new output position. In bits. As a signed 32-bit value. Out-of-range values are clipped to the boundary values. `ParkPosY` Like `ParkPosX`.
Multi-board Com'd Name	**n_park_position_2_axes**

Comments	• Being an "Fly Extension" Command, **park_position_2_axes** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251).
	• See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268.
	• See also comments on **park_position**.
	• To jump back, **park_return_2_axes** can be used.
	• With an unallowed parameter value, **park_position_2_axes** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).
	• The following command calls are executed in the same way:
	– `park_position_2_axes( Mode, ParkPosX, ParkPosY ) =`
	`{`
	`park_position_1_axis( Mode, 1, ParkPosX );`
	`park_position_1_axis( Mode, 2, ParkPosY );`
	`}`

RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**park_position_1_axis**

Variable List Command	**park_return**
Function	Moves the output point (of the laser focus) away from a park position along a 2D vector at jump speed to the specified position (absolute coordinate values) within the 2D Image Field.
Restriction	If the Option Processing-on-the-fly is not enabled, then **park_return** functions as a normal Jump Command. If the laser focus is not currently in a park position, then **park_return** is a Short List Command with no effect (see below).
Call	`park_return( Mode, X, Y )`
Parameters	`Mode` As an unsigned 32-bit value. = 0: `X` and `Y` are ignored (see comments). > 0: `X` and `Y` are interpreted as return jump coordinates (see comments).
	`X` Absolute x coordinate of the jump vector end point in the virtual Image Field. In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values.
	`Y` Like `X` (analogously).
Multi-board Com'd Name	**n_park_return**
Comments	• **park_return** is intended for encoder-based **set_fly_2d** or **set_fly_x**/**set_fly_y** Processing-on-the-fly applications where the laser focus should leave a safe parking area previously reached by **park_position** after an intermediate forwarding motion (following list interruption by **wait_for_encoder** etc.), see Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 261. See also comments at **park_position**. • If a **set_fly_2d** or **set_fly_x**/**set_fly_y** Processing-on-the-fly mode has been switched off by a prior **park_position**(`Mode = 0`) call, then **park_return** switches the same Processing-on-the-fly mode back on. Other Processing-on-the-fly modes are not switched back on. • If the park position has been attained by **park_position**, then **park_return**(`Mode = 0`) returns the galvanometer scanners to the most recent valid position before **park_position** has been called, whereas **park_return**(`Mode = 1`) moves them to the position specified with the command. When calculating the new position, the RTC6 takes the current Processing-on-the-fly correction into account. But if clipping to the boundaries of the virtual Image Field occurs (for example, due to a too long forwarding motion during a list interruption by **wait_for_encoder**), then the Processing-on-the-fly mode *is not* reactivated (see also **activate_fly_2d** and **activate_fly_xy**) and the jump executes without Processing-on-the-fly correction.

Variable List Command	park_return
Comments (cont'd)	• If the laser focus *has not been* previously moved to a safe area by **park_position**, then **park_return** is a Short List Command with no further effect. • If no Processing-on-the-fly mode has been previously active or if any Processing-on-the-fly mode is currently active, then **park_return** performs a normal jump to the specified location. • If a coordinate transformation in the virtual Image Field is active, then the specified or most recent valid position is subsequently appropriately transformed (see Section "Coordinate Transformations in the Virtual Image Field", page 174). • After **park_return**, a Jump Delay is activated. • **park_return** switches off the Signals for "Laser Active" Operation after a LaserOff Delay.
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified coordinate values by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**park_position**

Variable List Command	**park_return_1_axis**
Function	"Fly Extension" Command: Moves the output position for 1 Axis with jump speed from the intermediate parking position to the specified absolute position in the 2D Image Field.
Restriction	If the Option Processing-on-the-fly is not enabled, then **park_return_1_axis** only carries-out the jump to the specified new output position.
Call	`park_return_1_axis( Mode, Axis, RetPos )`
Parameters	`Mode` = 0: Return position is ignored. > 0: Return position. Absolute coordinate in the virtual Image Field. As an unsigned 32-bit value. `Axis` Axis from Table 3, page 268. As an unsigned 32-bit value. Allowed values: `1`…`2`. `RetPos` Absolute axis coordinate of the new output position. In bits. As a signed 32-bit value. Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_park_return_1_axis**
Comments	• Being an "Fly Extension" Command, **park_return_1_axis** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • See also comments on **park_return**. • **park_return_1_axis** can be used to jump back to **park_position_1_axis**. • If a Processing-on-the-fly correction has been previously deactivated at the specified axis by `park_position_1_axis`(`Mode` = 0) or `park_position_2_axes`(`Mode` = 0) then **park_return_1_axis** switches it back on again. • With an unallowed parameter value, **park_return_1_axis** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**park_return_2_axes**

Variable List Command	**park_return_2_axes**
Function	"Fly Extension" Command: Moves the output position for 2 Axes with jump speed from the intermediate parking position to the specified absolute position in the 2D Image Field.
Restriction	If the Option Processing-on-the-fly is not enabled, then **park_return_2_axes** only carries-out the jump to the specified new output positions.
Call	`park_return_2_axes( Mode, RetPosX, RetPosY )`
Parameters	`Mode` = 0: Return positions are ignored. > 0: Return positions. Absolute coordinates in the virtual Image Field. As an unsigned 32-bit value. `RetPosX` Absolute axes coordinates of the new output positions. In bits. As a signed 32-bit value. Out-of-range values are clipped to the boundary values. `RetPosY` Like `RetPosX`.
Multi-board Com'd Name	**n_park_return_2_axes**

Comments	• Being an "Fly Extension" Command, **park_return_2_axes** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251).
	• See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268.
	• See also comments on **park_return**.
	• **park_return_2_axes** can be used to jump back to **park_position_2_axes**.
	• If a Processing-on-the-fly correction has been previously deactivated at the specified axis by `park_position_2_axes( Mode = 0 )` then `park_position_2_axes` switches it back on again.
	• With an unallowed parameter value, **park_return_2_axes** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).
	• The following command calls are executed in the same way:
	– `park_return_2_axes( Mode, RetPosX, RetPosY ) =`
	`{`
	`park_return_1_axis( Mode, 1, RetPosX );`
	`park_return_1_axis( Mode, 2, RetPosY );`
	`}`

RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**park_return_1_axis**

Ctrl Command	**pause_list**
Function	Pauses execution of the list and disables the Signals for “Laser Active” Operation.
Call	`pause_list()`
Parameters	–
Multi-board Com’d Name	**n_pause_list**
Comments	• **pause_list** is synonymous with **stop_list**. **stop_list** is often confused with **stop_execution**. Therefore, it is preferable to use **pause_list**. • If **pause_list** is called during execution of a list, then the Signals for “Laser Active” Operation are suppressed (the signals are set to their respective “Off” level) and keeps the scan system in the most recently defined state – even if in the middle of a split-up into Microsteps. The PAUSED list execution status (queryable by **get_status**) is set, but the BUSY list execution status is left unchanged. Continuation by **execute_list_pos** or **release_wait** or by an External Start is not possible. However, **stop_execution** or an External Stop is possible. **restart_list**, **stop_execution** or an External Stop ends suppression of the start. • If processing of a list should be continued, then **restart_list** *must* be used. After a subsequent **restart_list**, the scan system resumes the planned motions (of the current command) and the Laser Control Signals are released again (in general, an interrupted marking cannot be continued without a disruption in the marking result). The PAUSED list execution status is then reset (here too, BUSY list execution status remains unchanged). • If, during calling of **pause_list**, no list is currently executing (BUSY list execution status not set) or a list has already been halted by **pause_list** or **set_wait** (PAUSED list execution status set), then **pause_list** is ignored (**get_last_error** return code `RTC6_BUSY`; the laser is then already off).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**restart_list**, **set_wait**

Ctrl Command	periodic_toggle
Function	Like **periodic_toggle_list**, but a control command.
Call	`periodic_toggle( Port, Mask, P1, P2, Count, Start )`
Parameters	`Port` Like **periodic_toggle_list**. `Mask` Like **periodic_toggle_list**. `P1` Like **periodic_toggle_list**. `P2` Like **periodic_toggle_list**. `Count` Like **periodic_toggle_list**. `Start` Like **periodic_toggle_list**.
Multi-board Com'd Name	**n_periodic_toggle**
Comments	• See **periodic_toggle_list**. • If an unallowed parameter value is supplied for `Port` or 0 for `P1` or `P2`, then **periodic_toggle** is not executed (**get_last_error** return code `RTC6_PARAM_ERROR`). • See also Chapter 9.4 "Periodical I/O Signals", page 323.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 605, OUT 605: endless toggling with `Count` = 2^{32}–1.
References	**periodic_toggle_list**

Undelayed Short List Command	**periodic_toggle_list**
Function	Generates and periodically toggles a signal at an adjustable output port.
Call	`periodic_toggle_list( Port, Mask, P1, P2, Count, Start )`
Parameters	`Port` Output port (0…4, as with **set_port_default**). As an unsigned 32-bit value. `Mask` Defines the bits to be toggled, (the maximum value depends on the port). 1 = bit is toggled. 0 = bit remains unchanged. As an unsigned 32-bit value. `P1` Duration of the toggled signal in [10 μs] (> 0, 16 bit max.). As an unsigned 32-bit value. `P2` Duration of the toggled signal in [10 μs] (> 0, 16 bit max.). As an unsigned 32-bit value. `Count` Number of P1–P2 period repetitions. As an unsigned 32-bit value. `Start` Duration until the first toggling starts in [10 μs]. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_periodic_toggle_list**
Comments	• If an unallowed parameter value is supplied for `Port` or 0 for `P1` or `P2`, then **periodic_toggle_list** is replaced by a **list_nop** and a **get_last_error** return code `RTC6_PARAM_ERROR` is generated. • `Mask` defines the bits to be toggled. Bits which are set to 1 in `Mask` are toggled and bits which are set to 0 remain unchanged. `Mask` is limited to the maximum allowed value of the selected port (see also **set_port_default**). Extra bits are ignored. • The selected bits are toggled. This state is maintained for `P1` $\times$ 10 μs clock cycles. Then they are toggled once again and kept stable for `P2` $\times$ 10 μs clock cycles. Users define the toggle bit start values ("off" state; "on" signal is active-HIGH or active-LOW) by other standard commands (**write_da_x**, **write_8bit_port**, **write_io_port**, etc.). These – and (should the situation arise) other queued Delayed Short List Commands – are executed before **periodic_toggle_list**. • The first toggling occurs after `Start` $\times$ 10 μs clock cycles. Toggling is repeated `Count`-times. `Count` = 0 immediately stops an output in progress. The remaining parameters are then not relevant. If the stop happens in the `P1` period ("On"), a toggle occurs to restore the initial state ("Off"). • `P1` and `P2` are clipped to the value range [0…65,535]. `P1` and `P2` must not be 0 at the same time.

Undelayed Short List Command	periodic_toggle_list
Comments (cont'd)	• The selected `Port` should not concurrently be used for other outputs such as "Automatic Laser Control". • At a **set_end_of_list**, **stop_execution** or external /STOP the periodical signals continue. • **periodic_toggle_list** toggles endless with `Count` = 2^{32}–1. • See also Chapter 9.4 "Periodical I/O Signals", page 323.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 605, OUT 605: endless toggling with `Count` = 2^{32}–1.
References	**periodic_toggle**, **set_port_default**

Ctrl Command	quit_loop
Function	Stops the repeating automatic list change started with **start_loop**.
Call	`quit_loop()`
Multi-board Com'd Name	**n_quit_loop**
Comments	• Before list execution is stopped, the current list is fully executed until the next **set_end_of_list** is encountered. • **start_loop** must be called prior to **quit_loop**. Otherwise, **quit_loop** has no effect.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**start_loop**

Undelayed Short List Command	**range_checking**	
Function	Defines an emergency action, if a galvanometer scanner exceeds its position limit.	
Call	`range_checking( HeadNo, Mode, Data )`	
Parameters	`HeadNo`	The scan system to be monitored. As an unsigned 32-bit value. 0: No monitoring. Default after **load_program_file**. 1: Monitored is Connector for First Scan Head. 2: Monitored is Connector for Second Scan Head. 3: Monitored are both connectors. Only the 2 least significant bits are evaluated.
	`Mode`	Action to take. As an unsigned 32-bit value. 0: The Signals for "Laser Active" Operation are suppressed immediately. List execution continues. 1: The Signals for "Laser Active" Operation are switched-off immediately. List execution is stopped immediately (as with **stop_execution** or /STOP). 2: A **simulate_ext_stop** is passed on to all slave boards.
	`Data`	Data type chosen to be monitored. As an unsigned 32-bit value. 0: Sample values. Like **set_trigger** signal types 7…9. 1: Transformed control values. Like **set_trigger** signal types 25…30. 2: Corrected control values. Like **set_trigger** signal types 10…15. 3: Actual output values. Like **set_trigger** signal types 20…23. 4: Actual galvanometer scanner position (only with iDRIVE Scan Systems, vario*SCAN* is internally connected to an x axis). Like **set_trigger** signal types [1, 2 and 4] or [4, 5 and 1]. 5: Actual galvanometer scanner position (only with iDRIVE Scan Systems, vario*SCAN* is internally connected to a y axis). Like **set_trigger** signal types [1, 2 and 5] or [4, 5 and 2].
Multi-board Com'd Name	**n_range_checking**	

Undelayed Short List Command	range_checking
Comments	• No monitoring takes place if the specified scan head does not have a correction table assigned (see **select_cor_table**). • The used position limits (that is, if exceeded the emergency action is executed) are the parameters of a customer-specific Processing-on-the-fly application monitoring (see **set_fly_limits**, **set_fly_limits_z**). Exceeding these limits only cause error messages in case of Processing-on-the-fly applications. The error messages can be queried with **get_marking_info** or the respective **if_fly_x_overflow**/**if_fly_y_overflow**/**if_fly_z_overflow** and **if_not_fly_x_overflow**/**if_not_fly_y_overflow**/**if_not_fly_z_overflow**. They do not trigger any activities. • Processing-on-the-fly applications use the data type `Data` = 0 (sample values) for customer-specific range limits. Inconsistent error messages and switch-offs may occur if used simultaneously with **range_checking** data types `Data` > 0. • If the laser has been switched-off with `Mode = 0` and the fault condition is gone then the laser is *not* switched on until the next Mark Command. The laser is *not* switched-on right in the middle of a currently executing Mark Command, not even in the middle of a Polyline. • The **range_checking** monitoring is active without Processing-on-the-fly application. • `Data` > 5 causes a **get_last_error** return code `RTC6_PARAM_ERROR`. In this case **range_checking** is sent to the RTC6 board as **list_nop**. • `Data`=2 and `Data`=3 have the identical effect for the z axis. • For `Data` = 4 and `Data` = 5 users must define the set position as the to-be-returned data type by the scan system. A simultaneous use of a Speed-Dependent Laser Control with `Mode = 2` (see **set_auto_laser_control**) is not possible. • `Data` = 4 and `Data` = 5 only differ in regards to the axis onto an vario*SCAN* is connected. For 2D systems without vario*SCAN* both selections have the identical effect. • `Data` = 4 and `Data` = 5 produce nonsensical switch-offs if an intelli*SCAN* is connected but is either not switched-on or a actual position has not been set as feedback.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 615: `Mode` = 2.
References	**set_fly_limits**, **set_fly_limits_z**

Ctrl Command	read_abc_from_file
Function	Like read_abc_from_file_20b. However, for a focus length value *l* in RTC4 compatibility range [–32,768…+32,767].
Call	`ErrorNo = read_abc_from_file( Name, &A, &B, &C )`
Result	`ErrorNo` Like read_abc_from_file_20b.
Parameters	`Name` Like read_abc_from_file_20b.
Returned Parameter Values	`A` `B` `C` Coefficients of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value *l* in the RTC4 compatibility range [–32,768…+32,767]). As 64-bit IEEE floating point values.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• Like read_abc_from_file_20b. • `read_abc_from_file( &A, &B, &C )` is synonymous with `read_abc_from_file_20b( &A, &B, &C × 1/16 )`.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	write_abc_to_file

Ctrl Command	read_abc_from_file_20b
Function	Reads the ABC values directly from a specified correction file on the PC. For a focus length value *l* in the RTC6 20-bit range [–524,288…+524,287].
Call	`ErrorNo = read_abc_from_file_20b( Name, &A, &B, &C )`
Result	`ErrorNo` Error code. As an unsigned 32-bit value. `0` No error. `1` File error (corrupt or incomplete). `2` Memory error (RTC6 DLL-internal, Windows working memory). `3` File-open error (empty string, file not found etc.).
Parameters	`Name` Name of the correction file. As a pointer to a \0-terminated ANSI string.
Returned Parameter Values	`A` `B` `C` Coefficients of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value *l* in the RTC6 20-bit range [–524,288…+524,287]). As 64-bit IEEE floating point values.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• **read_abc_from_file_20b** is available even without explicit access rights to a specific RTC6 board. • The board-specific error variables `LastError` and `AccError`, see Chapter 6.8 "Error Handling", page 133, are neither generated nor altered by **read_abc_from_file_20b**. • `read_abc_from_file_20b( &A, &B, &C )` is synonymous with `read_abc_from_file( &A × 16, &B, &C × 1/16 )`.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 631, OUT 632.
References	**write_abc_to_file_20b**

Ctrl Command	read_analog_in
Function	Reads in the analog input values: • ANALOG IN0 and ANALOG IN1 at the McBSP/ANALOG Socket Connector of the RTC6 PCIe Board • ANALOG IN0, and ANALOG IN1 at the SPI/ANA/UART Socket Connector of the RTC6 Ethernet Board Returns them as 12-bit digital values.
Call	`AnalogValue = read_analog_in()`
Result	As an unsigned 32-bit value. Bit #0 ANALOG IN0 input value as 12-bit digital value. … Bit #11 Bit #12 = 0. Channel number. Bit #13 = 0. Bit #14 = 0. Bit #15 Old bit for ANALOG IN0. Bit #16 ANALOG IN1 input value as 12-bit digital value. … Bit #27 Bit #28 = 1. Channel number. Bit #29 = 0. Bit #30 = 0. Bit #31 Old bit for ANALOG IN1.
Multi-board Com'd Name	**n_read_analog_in**
Comments	• See also Section "Analog Input Ports", page 89. • **read_analog_in** sets the old bits (Bit #15 and Bit #31) to 1 after reading. As soon new data are available at the corresponding analog input they are automatically set to 0. • The analog input values (ANALOG IN0 and ANALOG IN1) can be recorded by **set_trigger[*]** (signal 54). However, the old bits (Bit #15 and Bit #31) are not recorded. The format of the data is ((ANALOG IN1 << 16) + ANALOG IN0).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_trigger**, **set_trigger4**, **set_trigger8**

Ctrl Command	read_encoder
Function	Returns the counts of the two RTC6 encoder counters that were stored by **store_encoder**.
Call	`read_encoder( &Encoder0_0, &Encoder1_0, &Encoder0_1, &Encoder1_1 )`
Returned Parameter Values	`Encoder0_0` Count. As a pointer to a signed 32-bit value. For parameter "Encoder*n*_*m*": • *n* is the number of the encoder counter ("Encoder0", "Encoder1"). • *m* is the number of the storage position (parameter `Pos` of **store_encoder**).
	`Encoder1_0` Like `Encoder0_0` (analogously).
	`Encoder0_1` Like `Encoder0_0` (analogously).
	`Encoder1_1` Like `Encoder0_0` (analogously).
Multi-board Com'd Name	**n_read_encoder**
Comments	• See also Chapter 8.6 "Processing-on-the-fly", page 251 and Chapter 9.3.3 "Synchronization by Encoder Signals", page 320. • If the workpiece motion is registered with an incremental encoder: – Encoder counter "Encoder0" is triggered by the signals at encoder input port ENCODER X – Encoder counter "Encoder1" is triggered by the signals at encoder input port ENCODER Y • If an encoder simulation has been started by **simulate_encoder**, the encoder counters are triggered by an internal periodic 1 MHz clock signal. • For storage positions in which counts were not previously stored by **store_encoder**, the value 0 is returned (initialized value).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**store_encoder**, **get_encoder**, **set_fly_x**, **set_fly_y**, **set_fly_rot**, **wait_for_encoder**

Ctrl Command	read_image_eth
Function	Standalone Functionality: Reads out data for automatic booting from the NAND Memory and saves it as a binary file on the PC.
Prerequisite	RTC6 Software Package ≥ V1.7.0 and BIOS-ETH ≥ 26.
Call	`Result = read_image_eth( Name )`
Parameters	`Name` Name of the binary file. As a pointer to a \0-terminated ANSI string.
Result	`Result` Error code. As an unsigned 32-bit value. Value Description 0 No error. 1 No RTC6 Ethernet Board. 2 Empty string has been specified as file name. 3 Access denied. 4 RTC6 Ethernet Board is BUSY list execution status. 5 Timeout error (RTC6 Ethernet Board does not respond). 6 File cannot be opened. 7 Ethernet error, see **eth_get_last_error**. 8 NAND Memory error. 9 Aborted, see **eth_get_last_error**.
Multi-board Com'd Name	**n_read_image_eth**
Comments	• **read_image_eth** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • If the `Name` cannot be opened, a **get_last_error** return code `RTC6_PARAM_ERROR` is generated. • The content of an already existing binary file is deleted. • During the execution of **read_image_eth** the 10 μs clock cycle of the DSP is interrupted for up to 2 minutes. • **read_image_eth** is only allowed with RTC6 Ethernet Boards. Otherwise, a **get_last_error** return code `RTC6_TYPE_REJECTED` is generated. • See Chapter 16.7 "Standalone Functionality", page 988.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 618, OUT 618, RBF 623.
References	**write_image_eth**, **store_program**

Ctrl Command	**read_io_port**
Function	Returns the current state of the 16-Bit Digital Input Port on the EXTENSION 1 Socket Connector.
Call	`IOPort = read_io_port()`
Result	16-bit value (DIGITAL IN 0…DIGITAL IN 15). As an unsigned 32-bit value.
Multi-board Com'd Name	**n_read_io_port**
Comments	• See Chapter 9.2.1 "16-Bit Digital Input Port", page 310.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_io_port**, **write_io_port_mask**, **set_io_cond_list**, **clear_io_cond_list**, **read_io_port_buffer**, **read_io_port_list**

Ctrl Command	read_io_port_buffer	
Function	Returns the data previously stored in the IOPort buffer by **read_io_port_list** (input value read at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector; the galvanometer scanner set position and time value that has been current at the time of reading).	
Call	`CurrentIndex = read_io_port_buffer( Index, &Value, &XPos, &YPos, &Time )`	
Parameters	`Index`	Number of the to-be-returned entry in the IOPort buffer. As an unsigned 32-bit value. Only the 12 least significant bits are evaluated (the IOPort buffer only has 4,096 entries). Therefore, the user program can use a continuous counter for the index.
Returned Parameter Values	`Value`	16-bit value. As a pointer to an unsigned 32-bit value.
	`XPos`	Set position of the galvanometer scanner (x value). As a pointer to a signed 32-bit value.
	`YPos`	Set position of the galvanometer scanner (y value). As a pointer to a signed 32-bit value.
	`Time`	Time. In seconds. As a pointer to an unsigned 32-bit value.
Result	The index to which data is written upon the next **read_io_port_list**. As an unsigned 32-bit value.	
Multi-board Com'd Name	**n_read_io_port_buffer**	
Comments	• See also comments for **read_io_port_list**. • If no **read_io_port_list** has been called before **read_io_port_buffer**, then the initialization values (Default: 0) are returned.	

Example (C/C++)	Wait until the data under `Index` gets newly written:
	`while (Index == read_io_port_buffer( Index, &Value, &XPos, &YPos, &Time ));`
	Read all data from `Index` to the current (not yet newly written) read position:
	`while (Index != read_io_port_buffer( Index, &Value, &XPos, &YPos, &Time ))`
	`{ Index = (Index+1) & 0xf; …}`	

RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality. Larger IOPort buffer.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**read_io_port_list**	

Undelayed Short List Command	read_io_port_list
Function	Writes into the internal IOPort buffer the current state (DIGITAL IN 0…DIGITAL IN 15) of the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector. At the same time, the current xy set positions and *relative* time in seconds are stored.
Call	`read_io_port_list()`
Multi-board Com'd Name	**n_read_io_port_list**
Comments	• **read_io_port_list** does not return values. However, the values can be subsequently queried from the IOPort buffer by the **read_io_port_buffer** command. • The IOPort buffer holds 4,096 entries and is circularly organized. It always stores the 4,096 most recent entries and overwrites older entries without warning. The incremental Index starts after a reset (program start) at Index 0, and after passing 4,095 does rollover to 0. • The stored time is the current value of an internal seconds counter that is set to 0 at program start (**load_program_file**) or by **time_update**. • See also Chapter 9.2.1 "16-Bit Digital Input Port", page 310.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. Larger IOPort buffer.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**read_io_port_buffer**, **read_io_port**

Ctrl Command	read_mcbsp
Function	Returns the most recent input value that has been fully copied to an internal memory location by the McBSP interface.
Call	`mcbsp_value = read_mcbsp( No )`
Parameters	`No` Number of the internal memory location whose input value should be queried. As an unsigned 32-bit value. 0 Memory location for Processing-on-the-fly applications and for **set_mcbsp_in** input with coding Bit #31 = 0. 1 Memory locations for Online Positioning and for **set_mcbsp_in** input. 2 Wie 1. 3 Memory location for **set_mcbsp_in** input with coding Bit #31 = 1. For **set_multi_mcbsp_in**, the following applies: memory locations 0 through 3 are continuously written to as a ring buffer. Only the 2 least significant bits are evaluated.
Result	Input value. As a signed 32-bit value.
Multi-board Com'd Name	**n_read_mcbsp**
Comments	• For information on the McBSP interface and the related memory locations, see also Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87. • The interpretation as – one signed or unsigned 32-bit data word – two signed 16-bit data words – two signed 15-bit data words is the responsibility of the user: – For Processing-on-the-fly applications (`No` = 0) see Chapter 8.6 "Processing-on-the-fly", page 251 and Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 322. – For "Local Online Positioning" (`No` = 1…2) see Chapter 8.3.1 ""Local Online Positioning"", page 237. – For "Global Online Positioning" (`No` = 1…2) see Chapter 8.3.2 ""Global Online Positioning"", page 240. – For **set_mcbsp_in** input (`No` = 0…3) see Section "Correction via McBSP Interface with Additional McBSP Input", page 255 and Section "Correction via McBSP Interface with Additional McBSP Input", page 257. – For **set_multi_mcbsp_in**, see page 766. • After **load_program_file**, the input values at the McBSP interface are transferred to location 0 (default) even if no Processing-on-the-fly correction is activated. • The McBSP interface ignores the first FrameSync signal after a **load_program_file** or **mcbsp_init**. That is, data provided is not transmitted, see page 89.

Ctrl Command	read_mcbsp
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_mcbsp_out**, **mcbsp_init**, **set_fly_x_pos**, **set_fly_y_pos**, **set_fly_rot_pos**, **set_mcbsp_x**, **set_mcbsp_y**, **set_mcbsp_rot**, **set_mcbsp_matrix**, **apply_mcbsp**, **set_mcbsp_global_x**, **set_mcbsp_global_y**, **set_mcbsp_global_rot**, **set_mcbsp_global_matrix**

Ctrl Command	read_multi_mcbsp
Function	Queries the McBSP multi transmission input value that has been most recently type-sorted and stored.
Call	`mcbsp_value = read_multi_mcbsp( No )`
Parameters	`No` Number of the type-sorted internal memory location. `No` corresponds to coding Bit #0…Bit #2 of the McBSP multi input. As an unsigned 32-bit value. 0 x coordinate of the Processing-on-the-fly application. 1 y coordinate of the Processing-on-the-fly application. 2 z coordinate of the Processing-on-the-fly application. 3 Laser power P. 4 Extra parameter E1. 5 Extra parameter E2. 6 Extra parameter E3. 7 Extra parameter E4. Only the 3 least significant bits are evaluated.
Result	Memory value. As a signed 32-bit value.
Multi-board Com'd Name	**n_read_multi_mcbsp**
Comments	• You must have previously activated and used McBSP multi transmission by **set_multi_mcbsp_in** or **set_multi_mcbsp_in_list**. Otherwise, default or old values are returned.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_multi_mcbsp_in**, **set_multi_mcbsp_in_list**

Ctrl Command	read_status
Function	Returns the RTC6 list status, see Chapter 6.4.2 "List Status", page 110.
Call	`Status = read_status()`
Result	List status. As an unsigned 32-bit value. Bit # / Name / Description Bit #0 (LSB) LOAD1 = 1: Indicates that the input pointer is currently in "List 1", that is, that all following list commands are stored in "List 1". LOAD1 is set if the input pointer is set into "List 1" (for example, by **set_start_list_pos**). LOAD1 is reset if a **set_end_of_list** is written to "List 1" (READY1 set) or if LOAD2 is set (for example, by **set_start_list_pos**). Bit #1 LOAD2 = 1: Indicates that the input pointer is currently in "List 2", that is, that all following list commands are stored in "List 2". LOAD2 is set if the input pointer is set into "List 2" (for example, by **set_start_list_pos**). LOAD2 is reset if a **set_end_of_list** is written to "List 2" (READY2 set) or if LOAD1 is set (for example, by **set_start_list_pos**). Bit #2 READY1 = 1: Indicates that during the loading procedure a **set_end_of_list** has been written to "List 1". READY1 is reset if LOAD1 is newly set (for example, by **set_start_list_pos**). Bit #3 READY2 = 1: Indicates that during the loading procedure a **set_end_of_list** has been written to "List 2". READY2 is reset if LOAD2 is newly set (for example, by **set_start_list_pos**). Bit #4 BUSY1 = 1: Indicates that "List 1" is executing at the moment (or more precisely: that the output pointer currently resides in "List 1" after execution of "List 1" or "List 2" has been started). BUSY1 is set by starting execution of "List 1" (for example, by **execute_list_pos**) or by a list change to "List 1" (automatic list change or jump). BUSY1 is reset if **set_end_of_list** is executed in "List 1", if a jump into "List 2" is executed (for example, **list_jump_pos**), if "List 2" is started (for example, by **execute_list_pos**) or if **stop_execution** is executed.

Ctrl Command	read_status
Result (cont'd)	Bit #5 BUSY2 = 1: Indicates that "List 2" is executing at the moment (or more precisely: that the output pointer currently resides in "List 2" after execution of "List 1" or "List 2" has been started). BUSY2 is set by starting execution of "List 2" (for example, by **execute_list_pos**) or by a list change to "List 2" (automatic list change or jump). BUSY2 is reset if **set_end_of_list** is executed in "List 2", if a jump into "List 1" is executed (for example, **list_jump_pos**), if "List 1" is started (for example, by **execute_list_pos**) or if **stop_execution** is executed. Bit #6 USED1 = 1: Indicates that a **set_end_of_list** has been reached during processing of "List 1". USED1 is reset when LOAD1 is set (for example, by **set_start_list_pos**). Bit #7 USED2 = 1: Indicates that a **set_end_of_list** has been reached during processing of "List 2". USED2 is reset when LOAD2 is set (for example, by **set_start_list_pos**). Bit #8 … Bit #31 0
Multi-board Com'd Name	**n_read_status**
Comments	• When interpreting the status values returned by **read_status**, always take into account the programmed loading or execution processes of the lists. Under some circumstances, the status values can be misleading, as illustrated by the following examples: – Even during a loading process, the LOAD list status can already have been reset and the READY list status set if – after loading of a **set_end_of_list** into a list – further list commands are loaded into the same list. – The status values remain unchanged if a **set_end_of_list** is overwritten with another command (READY list status is not reset). – If – during a loading process– a **set_end_of_list** is processed at the same time in the same list, then the list is regarded as already processed (USED list status set), even though it is still newly loaded (USED list status has been then initially reset). – Even if a completely loaded command list (incl. **set_end_of_list**) has been stored, the READY list status of a list can be reset if the input pointer has been newly set (for example, by **set_input_pointer**) into the list. Then a list's USED list status can be reset too, even though a completely loaded and already processed command list is stored. – For jumps from one list area to another ("List 1" <-> "List 2", for example, by **list_jump_pos**) during execution, the USED list status values of both lists (unlike their BUSY list execution status values) remain unchanged and are therefore not meaningful.

Ctrl Command	read_status
Comments (cont'd)	• If the list status is queried during processing of a subroutine in the protected RTC6 List Memory area "List 3", then the status is returned of the list ("List 1" or "List 2") in which the output pointer most recently resided (typically from where the subroutine has been originally called). • If list execution is interrupted (by **pause_list**, **stop_list** or **set_wait**), then the above-mentioned status values remains unchanged. • The List Execution Status values BUSY list execution status and PAUSED list execution status can be queried by **get_status**. • To read the status signals from the *scan heads*, use **get_head_status**.
RTC4→RTC6	Basically unchanged functionality. However: Additional USED list status.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_status**, **get_head_status**

Ctrl Command	read_user_data
Function	No function.
Multi-board Com'd Name	**n_read_user_data**
Comments	• To date, **read_user_data** has no effect.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**send_user_data**

Normal List Command	regulation3	
Function	Category: List Commands for a Restricted User Group only. Varies laser frequency and pulse length depending on encoder frequency.	
Call	`regulation3( Fmax, Fmin )`	
Parameters	`Fmax`	Max. encoder frequency. In Hz. Allowed value range: 10,000…1,000,000. As an unsigned 32-bit value.
	`Fmin`	Switch-off frequency. In Hz. Allowed value range: 10…[`Fmax` / 10]. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_regulation3**	
Comments	• **regulation3** is executed infinitely until list execution is stopped by **stop_execution** or a /STOP. • For encoder frequency < `Fmin` the laser is switched off. • An encoder frequency > `Fmax` is clipped to `Fmax`. • The value for the laser frequency results from the lookup table of **set_laser_timing_table**. • The value for the pulse length results from the lookup table of **set_duty_cycle_table**. • The following applies for List Commands for a Restricted User Group only: These users have been informed separately in how to handle these RTC6 commands. Therefore, only the command table is available in this manual and further information is omitted.	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 626, OUT 627.	
References	**set_duty_cycle_table**, **set_laser_timing_table**	

Ctrl Command	release_rtc
Function	Releases the specified RTC6 for use by other user programs.
Call	`NoOfReleasedCard = release_rtc( CardNo )`
Parameters	`CardNo` RTC6 DLL-internal number of the RTC6 board. As an unsigned 32-bit value.
Result	The returned value is `CardNo` if the user program has been still in possession of access rights for this board. Otherwise, 0 is returned. As an unsigned 32-bit value.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• After the user program releases a board with **release_rtc**, it no longer has access rights for this board. The board can then be subsequently acquired by the same or another user program by **acquire_rtc** or **init_rtc6_dll**. • If a board released by **release_rtc** has been the active board, then (other than multi-board commands) only those non-multi-board commands not requiring explicit access rights are subsequently available. Another board is not automatically selected as the active board. Activation of another board (for which access rights are assigned to the user program) can be achieved by **select_rtc**. • **release_rtc** is available even without explicit access rights for a particular RTC6 board, but **release_rtc** then has no effect (return value 0). • **release_rtc** also has no effect (return value 0), if: – `CardNo` exceeds the number of RTC6 boards found during initialization (see **rtc6_count_cards**) *and* – No RTC6 Ethernet Board is entered at `CardNo` – `CardNo` = 0 (real boards begin at 1)
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**acquire_rtc**, **init_rtc6_dll**

Ctrl Command	release_wait
Function	Resumes processing of a list that has been interrupted by **set_wait**.
Call	`release_wait()`
Multi-board Com'd Name	**n_release_wait**
Comments	• **release_wait** is only executed if the RTC6 is actually in a wait state (hence if a break point has been previously reached and list processing has been interrupted; the PAUSED list execution status is then set but *not* the BUSY list execution status). Otherwise, **release_wait** is ignored (**get_last_error** return code `RTC6_BUSY`). • **By release_wait,** the PAUSED list execution status (queriable with **get_status**) is reset and the BUSY list execution status is newly set, see also Chapter 6.4.3 "List Execution Status", page 111. • **release_wait** resets the `WaitWord` that receives the break point number during an interrupt to zero. • If a home jump (defined by **home_position** or **home_position_xyz**) has been executed by **set_wait**, then **release_wait** leads to a corresponding home return (the INTERNAL-BUSY list execution status is set while the home return is executed). • The wait state can be queried by **get_wait_status**.
RTC4→RTC6	Basically unchanged functionality. However: Additional PAUSED list execution status. It is set by **set_wait** and reset with **release_wait**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_wait**, **get_wait_status**, **restart_list**

Ctrl Command	reset_error
Function	Resets the cumulative error code.
Call	`reset_error( Code )`
Parameters	`Code` OR connection of the error codes = sum by means of $2^{\text{Bitnumber}}$ of all bits to be reset. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_reset_error**
Comments	• For error handling see Chapter 6.8 "Error Handling", page 133. • The cumulative error code `AccError` can be reset: – Bitwise (individually for each error type, for example, Bit #5 and Bit #6 by `Code` = $2^5 \| 2^6$ or by `Code` = `RTC6_BUSY` | `RTC6_REJECTED`) – Completely (by `Code` = "–1", therefore by `Code` = 2^{32}–1) The meanings of bit numbers, error types and error constants is described at **get_error**. • **reset_error** does not delete the error code of another user program currently assigned access rights to the board. • **reset_error** and **n_reset_error** are available even without explicit access rights to a specific RTC6 board. • The board-specific error variable `LastError` (see Chapter 6.8 "Error Handling", page 133) is neither generated nor altered by **reset_error**. In contrast, `AccError` is altered as specified by `Code`.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_error**, **get_last_error**

Ctrl Command	restart_list
Function	Reenables Signals for "Laser Active" Operation and resumes execution of a list that has been interrupted by **pause_list** or **stop_list**.
Call	`restart_list()`
Multi-board Com'd Name	**n_restart_list**
Comments	• **restart_list** is only executed, if a list has been previously halted by **pause_list** or **stop_list** and if the BUSY list execution status and PAUSED list execution status (queryable with **get_status**) are set. Otherwise (for example, if a list has been halted by **set_wait**), **restart_list** is ignored (**get_last_error** return code `RTC6_BUSY`). • **restart_list** resets the PAUSED list execution status. The BUSY list execution status is left unchanged. • In general, an interrupted marking cannot be continued without a disruption in the marking result.
RTC4→RTC6	Basically unchanged functionality. However: Additional PAUSED list execution status that is set with **pause_list** and **stop_list** and reset with **restart_list**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**pause_list**, **stop_list**, **release_wait**

Ctrl Command	rs232_config
Function	Configures the RS-232 interface for the specified baud rate.
Call	`rs232_config( BaudRate )`
Parameters	`BaudRate` Baud rate. As an unsigned 32-bit value. Allowed value range: [160 Bd…12.8 MBd].
Multi-board Com'd Name	**n_rs232_config**
Comments	• See also Chapter 4.6.5 "RS232 Socket Connector", page 86. • **rs232_config** is synonymous with **uart_config**. However, **rs232_config** has no result. • < DLL 611: value range [300 Bd…+115.200 Bd].
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. As of DLL 611: extended value range.
References	**rs232_write_data**, **rs232_read_data**, **uart_config**

Ctrl Command	rs232_read_data
Function	Reads a value from the input buffer of the RS-232 interface, see Chapter 9.2.3 "RS-232 Interface", page 310.
Call	`RS232Data = rs232_read_data()`
Result	As an unsigned 32-bit value. Bit #0 (LSB) … Bit #7: Next (not yet read by the user program) value of the input buffer. Bit #8: "New" bit: = 1: The value is new (has not been previously read). = 0: The value is old (has been already read once by **rs232_read_data**). Bit #9 … Bit #15: 0. Bit #16 … Bit #23: Number of further (not yet read) characters. Bit #24 … Bit #31: Number of buffer overruns.
Multi-board Com'd Name	**n_rs232_read_data**
Comments	• The RS-232 interface is internally is read in internally asynchronously (one character at a time). If this character is new, then it is stored in a 256-character ring buffer. From there, it can be transferred to the user program by **rs232_read_data** (asynchronously to reading). • Byte #0 (Bit #0…Bit #7) returns only one character from the current reading position. • Byte #1 (Bit #8) indicates if the character has already been read by the user program. • Byte #2 (Bit #16…Bit #23) indicates the number of characters in the input buffer that still have not been read by the user program. If no unread characters are present (byte #2 = 0), then the most recently read character is transferred with "new bit" = 0. • Byte #3 (Bit #24…Bit #31) indicates the number of overruns of the input buffer (a corresponding number of characters were overwritten and therefore irretrievably lost). *Important: To reset the overflow counter, you must call* **rs232_read_data** *correspondingly often!* • Example: return value 459098 = 0x0007015A = (0, 7, 1, 90) means: character 90 ('Z') has been read and is new (1), 7 additional characters remain to be read, the buffer has been never overrun (0).

Ctrl Command	rs232_read_data
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**rs232_write_data**, **uart_config**

Ctrl Command	rs232_write_data
Function	Sends a data word (byte) to the RS-232 interface.
Call	`rs232_write_data( Data )`
Parameters	`Data` Data word. As an unsigned 32-bit value. Only the least significant byte is transferred to the RS-232 interface.
Multi-board Com'd Name	**n_rs232_write_data**
Comments	• The complete transmission of any previous data words is waited for. An overrun at the interface is not possible. • See also Chapter 9.1.6 "RS-232 Interface", page 309.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**rs232_write_text**, **rs232_read_data**, **uart_config**, **rs232_write_text_list**

Ctrl Command	rs232_write_text
Function	Sends a text string (character-by-character) to the RS-232 interface.
Call	`rs232_write_text( pData )`
Parameters	`pData` PC memory address of the first character (byte) of the to-be-sent text string. As a pointer to a \0-terminated string.
Multi-board Com'd Name	**n_rs232_write_text**
Comments	• **rs232_write_text** is split into an appropriate number of **rs232_write_data**. • During execution of **rs232_write_text** no further control commands can be executed, but the list execution on the board is not affected. If an executing list itself contains **rs232_write_text_list**, **rs232_write_text** should preferably not be used so as to avoid conflicts (race conditions). • See also Chapter 9.1.6 "RS-232 Interface", page 309.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**rs232_write_data**, **rs232_write_text_list**, **uart_config**

Variable List Command	**rs232_write_text_list**
Function	Sends a text string (character-by-character) to the RS-232 interface.
Call	`rs232_write_text_list( pData )`
Parameters	`pData` PC memory address of the first character (byte) of the to-be-sent text string. As a pointer to a \0-terminated string.
Multi-board Com'd Name	**n_rs232_write_text_list**
Comments	• When an **rs232_write_text_list** is loaded, the to-be-sent text (if more than 12 characters in length, \0 not included) is split into blocks of 12 characters, with each block receiving its own **rs232_write_text_list** in the RTC6 List Memory (keep this in mind to prevent unintended overflow of the corresponding RTC6 List Memory area). Processing of the individual **rs232_write_text_list** is similar to that of **rs232_write_text** (the 12-character block is split into individual characters and sent sequentially to the RS-232 interface). • **rs232_write_text_list** takes some time for execution, depending on the baud rate and text length. • See also Chapter 9.1.6 "RS-232 Interface", page 309.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**rs232_write_text**, **rs232_read_data**, **uart_config**

Ctrl Command	**rtc6_count_cards**
Function	Returns the number of RTC6 PCIe Boards detected during initialization.
Call	`NoOfCards = rtc6_count_cards()`
Result	Number of RTC6 PCIe Boards. As an unsigned 32-bit value.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• Initialization of the installed RTC6 PCIe Boards is to be made separately for each user program by calling **init_rtc6_dll**. • **rtc6_count_cards** is available even without explicit access rights to a specific RTC6 PCIe Board. • The board-specific error variables `LastError` and `AccError` (see Chapter 6.8 "Error Handling", page 133) are neither generated nor altered by **rtc6_count_cards**.
RTC4→RTC6	New command. Functionality is similar to **rtc4_count_cards** of the RTC4 command set.
RTC5→RTC6	New command. Functionality is similar to **rtc5_count_cards** of the RTC5 command set.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**init_rtc6_dll**

Delayed Short List Command	save_and_restart_timer
Function	Stores the current value of the RTC6 Timer and resets it to zero.
Call	`save_and_restart_timer()`
Multi-board Com'd Name	**n_save_and_restart_timer**
Comments	• The stored RTC6 Timer value can be read by **get_time**. • **save_and_restart_timer** is useful for measuring the execution time of a marking process, see Chapter 8.12 "Time Measurements", page 290. • The RTC6 Timer only counts list-command clock cycles. Counting is paused during interruptions by **set_wait** or **pause_list**. • By **get_lap_time** the number of elapsed list-command clock cycles since the reset to zero can be queried. • To compare RTC6-internal **save_and_restart_timer** time measurements to external time measurements by the BUSY Pin, you should insert a **list_nop** between **save_and_restart_timer** and **set_end_of_list**. This ensures that any Scanner Delay completes before **set_end_of_list**. Without **list_nop**, **save_and_restart_timer** includes the Scanner Delay in its measurement even though it completes only after **set_end_of_list** (and therefore the BUSY Pin is already low). • A 32-bit "Timestamp Counter" and a 64-bit "Timestamp Counter" are available in addition to the RTC6 Timer, see also Chapter 8.12 "Time Measurements", page 290.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_lap_time**, **get_time**

Ctrl Command	save_disk
Function	Stores all indexed characters, text strings and/or subroutines to a binary file on a PC data medium, ordered by index, and returns the number of thereby stored list commands.
Call	`NoOfSavedCommands = save_disk( Name, Mode )`
Parameters	`Name` File name. As a pointer to a \0-terminated ANSI string.
	`Mode` Specifies what is to be stored: Bit #0 = 1: All indexed characters and text strings are stored. Bit #1 = 1: All indexed subroutines are stored. Bit #2 Not evaluated. … Bit #31: Not evaluated. As an unsigned 32-bit value.
Result	The number of list commands saved by **save_disk**. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_save_disk**
Comments	• **save_disk** can be used together with **load_disk**, for example, to perform defragmentation or to apply subsequent protection to subroutines, see Section "Subsequent Protection and Conversion of Non-Indexed Subroutines", page 119 and Section "Index Management and Defragmentation", page 118. • By **save_disk**, always the complete sets (no individual characters, text strings or subroutines!) are stored in the specified file. Indexed characters, text strings or subroutines that are referenced multiple times with **copy_dst_src** are also correspondingly duplicated by **save_disk**, that is, also stored multiple times. The **save_disk** command ignores unreferenced non-indexed (not subsequently referenced by **set_char_pointer**,…) characters, text strings and subroutines. • The number of list commands stored by **save_disk** can differ from the number of the list commands stored in the protected RTC6 List Memory area "List 3" (= 2^{23} – `Mem1` – `Mem2` – **get_list_space**), due to the following: – Indexed characters/text strings/subroutines are referenced several times – Indexed characters/text strings/subroutines reside in the unprotected RTC6 List Memory area "List 1" or "List 2" Prior a subsequent **load_disk**, be sure to compare the returned number with the size of the protected RTC6 List Memory area "List 3" (= 2^{23} – `Mem1` – `Mem2`). • No-longer-needed characters (or text strings or subroutines) should be dereferenced by **load_char** (or **load_text_table** or **load_sub**) directly followed by **list_return** previously to **save_disk** (see Section "Index Management and Defragmentation", page 118).

Ctrl Command	save_disk
Comments (cont'd)	• **save_disk** always stores all characters, text strings and/or subroutines from the referenced address to the next possible **list_return**. Jump Commands (also branches to various **list_return** commands) are thereby neither evaluated nor executed. • **save_disk** automatically replaces unallowed commands (for example, **set_end_of_list**) with **list_nop** commands; missing commands (for example, **list_return** upon reaching the last memory position of "List 3") are added. • **save_disk** is not executed (**get_last_error** return code `RTC6_PARAM_ERROR`), if: – `Mode` = 0 – `Name` = NULL • **save_disk** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **save_disk** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • During the runtime of **save_disk**: – No further commands are executed – External Starts are suppressed • **save_disk** also stores version information which is checked by **load_disk**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 615: version info.
References	**load_disk**

Undelayed Short List Command	**select_char_set**
Function	Selects one of the available character sets (exclusively) for the execution of subsequent **mark_text** and **mark_text_abs**.
Call	`select_char_set( No )`
Parameters	`No` Number of the desired character set. As an unsigned 32-bit value. Allowed value range: [0…3].
Multi-board Com'd Name	**n_select_char_set**
Comments	• The selection remains valid until another is encountered. • The default value (after initialization) is `No` = 0. • If `No` > 3, then **select_char_set** is replaced by a **list_nop**. The current character set then remains valid (**get_last_error** return code `RTC6_PARAM_ERROR`). • A character set change *within* a **mark_text** or **mark_text_abs** is only possible if a **select_char_set** is located within a (called) indexed character. The selection encountered there then applies to all subsequent characters. • If the selected character set is not (fully) defined, then missing characters are not marked (with **mark_text** or **mark_text_abs**).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_text**, **mark_text_abs**

Ctrl Command	select_cor_table
Function	Assigns the previously loaded correction tables to the scan head connector and activates Image Field correction.
Call	`select_cor_table( HeadA, HeadB )`
Parameters	`HeadA` = 0: Turns off the signals for scan head A (Connector for First Scan Head). = 1…8: Assigns correction table number `HeadA` to scan head A. As an unsigned 32-bit value. See also **number_of_correction_tables**.
	`HeadB` = 0: Turns off the signals for scan head B (Connector for Second Scan Head). = 1…8: Assigns correction table number `HeadB` to scan head B. Requires Option "Second Scan Head Control". As an unsigned 32-bit value. See also **number_of_correction_tables**.
Multi-board Com'd Name	**n_select_cor_table**
Comments	• **select_cor_table** or **select_cor_table_list** should be called directly after loading the desired correction table(s) by **load_correction_file** and/or **load_z_table**/**load_z_table_no** (or after a subsequent **load_program_file** – see below) in order to assign the correction table(s) to the corresponding scan head connector (**select_cor_table** is automatically called by **load_correction_file**; see Section "Notes", page 181). **select_cor_table** and **select_cor_table_list** issue a jump with jump speed (no "Hard Jump") to the corrected galvanometer scanner position, so that Image Field correction is immediately applied to the currently set galvanometer scanner position. **select_cor_table** does this automatically and immediately, whereas **select_cor_table_list** inserts the jump in front of the next list command. Depending on the table contents and galvanometer scanner position, this can take a few clock cycles (and at least one clock cycle). • Correction tables should be loaded and assigned prior to first-time starting of a list or issuance of a control command (for example, **goto_xy**) that sets the scan system's galvanometer scanners in motion, and **select_cor_table** or **select_cor_table_list** should only be started *after* loading of the desired correction table(s). Otherwise, the galvanometer scanners can, in some circumstances, be driven to an unexpected position and the laser beam deflected in an unintended direction. • Correction tables, loaded (by **load_correction_file**) *prior* to **load_program_file**, are not fully effective before **select_cor_table** or **select_cor_table_list** is called. • **select_cor_table** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set The list command **select_cor_table_list** can be used without restrictions. • **select_cor_table** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set)

Ctrl Command	select_cor_table
Comments (cont'd)	• If the INTERNAL-BUSY list execution status is set, **select_cor_table** is only executed with a delay (after INTERNAL-BUSY list execution status has been reset again). • The INTERNAL-BUSY list execution status is set while the jump to the corrected galvanometer scanner position is executed. • If the values for parameters `HeadA` or `HeadB` are invalid (values > **number_of_correction_tables**) or 0, then **select_cor_table** turns off the corresponding scan head signals. The galvanometer scanners then remain in their last position. • If the Option "Second Scan Head Control" is not enabled, **select_cor_table** does not assign a correction table to the Connector for Second Scan Head. • The default setting (after **load_program_file**) is `(1,0)`, that is, correction table #1 is used for scan head A, whereas the output signals for scan head B are turned off (this corresponds to parameter values `HeadA` = 1 and `HeadB` = 0). Initially however, after **load_program_file** and until **select_cor_table** is called or the galvanometer scanners are explicitly moved, the galvanometer scanners stay in the position (0|0), even if the correction table content is different. • If the Option "Second Scan Head Control" is not enabled, then signals for the second scan head remain permanently turned off. Even so, multiple correction tables can still be loaded and used at any time for the first scan head. Even with one scan head, you can thereby quickly switch back and forth between several correction tables, for example, one for a pilot laser and one for the processing laser with a different wavelength, see also Chapter 8.5 "Controlling 2D Scan Systems and 3D Scan Systems", page 245. • In a double scan head system, table #1 is typically used for scan head A, and table #2 is used for scan head B: `select_cor_table(1,2)`. But you can make a different assignment at any time. • For 3D systems, the Option "3D" must be enabled. Then several (2D or 3D) correction tables can be loaded. If Option "Second Scan Head Control" is not enabled, then x axis and y axis must be connected to Connector for First Scan Head, the z axis to the x or y channel of the Connector for Second Scan Head. If a 3D correction table is assigned to the Connector for First Scan Head, corrected signals for an xy scan head are then transmitted by the Connector for First Scan Head and corrected signals for the z axis are transmitted by both channels of the Connector for Second Scan Head. If multiple RTC6 boards with enabled Option "3D" are installed in a PC, then that many 3-axis systems can be simultaneously controlled.

Ctrl Command	select_cor_table
Comments (cont'd)	• Provided the Option "3D" and the Option "Second Scan Head Control" are enabled, users specify which signals (xy or z) are to be outputted by which connector when assigning the correction table (see also Chapter 4.5.1 "Scan Head Connectors and Transfer Protocol", page 69 and Section "2D and 3D Correction Files", page 179). 2D correction tables should and 3D correction tables can be thereby assigned only to the xy axes and *not* to the z axis (for example, by `select_cor_table(0,1)` or `select_cor_table(0,2)` for xy at the scan head B connector and z at the scan head A connector). Because unexpected system behavior might otherwise occur and the system would not know where the xy axes and z axis are connected, the signals of both connectors are turned off if both connectors have been assigned a correction table and (at least) one of them is a 3D correction table (for example, for `select_cor_table(1,1)`). • Parameters from currently loaded correction tables can be read by **get_table_para**, or from currently assigned correction tables by **get_head_para** (see also **load_correction_file**). • **select_cor_table** is allowed in Boot Phase 2, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	Unchanged functionality. Exception: up to 8 correction tables in total can be stored in RTC6 memory, see Chapter 8.5.3 "Using Several Correction Tables", page 250. However, *two 3D* correction tables *cannot* be simultaneously assigned to the scan head connectors.
RTC5→RTC6	Changed functionality. • In the RTC6 command set, **select_cor_table** automatically switches on Zoom and/or Stretch, if a table has been previously loaded by **load_stretch_table** or **load_zoom_correction_file** to this table number.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**select_cor_table_list**, **load_correction_file**, **load_program_file**, **load_stretch_table**, **number_of_correction_tables**

Variable List Command	**select_cor_table_list**
Function	Like **select_cor_table**, but a list command.
Call	`select_cor_table_list( HeadA, HeadB )`
Parameters	`HeadA` Like **select_cor_table**.
	`HeadB` Like **select_cor_table**.
Multi-board Com'd Name	**n_select_cor_table_list**
Comments	• See **select_cor_table**. • Even though **select_cor_table_list** is a Short List Command, execution of the directly following list command is delayed by a few clock cycles due to the intermediate jump to the corrected galvanometer scanner position. The extent of this delay depends on the assigned correction table's content for the current galvanometer scanner position (for example, (0|0) after **load_program_file**); but it is at least 10 µs, even if the correction table does not specify a correction.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**select_cor_table**

Ctrl Command	select_rtc
Function	Defines, in a multi-board system, the active RTC6 board for a user program. See Chapter 6.6 "Using Several RTC6 PCIe Boards in One PC", page 126.
Call	`NoOfSelectedCard = select_rtc( CardNo )`
Parameters	`CardNo` RTC6 DLL-internal number of the RTC6 board. As an unsigned 32-bit value.
Result	The returned value is: • `CardNo` if access rights exist for the specified board or if the specified board is not allocated to another user program (in this latter case, access rights are acquired through **select_rtc** – as by **acquire_rtc**, see below). • The number of the active board if `CardNo` exceeds the number of RTC6 boards found during initialization, no RTC6 Ethernet Board is entered there or if `CardNo` = 0. • 0 otherwise (particularly when the specified board is reserved by another user program or if a version compatibility error is detected and activation of the board therefore cannot succeed) (**get_last_error** return code `RTC6_ACCESS_DENIED` and possibly `RTC6_VERSION_MISMATCH`). As an unsigned 32-bit value.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• Activation of a board for a user program already occurs when the RTC6 DLL for this user program is initialized (see **init_rtc6_dll**). The **select_rtc** command offers the possibility of changing the active board at any time. All user program commands subsequent to **select_rtc** (except for multi-board commands) are forwarded to the corresponding active board. • If the specified board is reserved for another user program, **select_rtc** has no effect (return value 0, **get_last_error** return code `RTC6_ACCESS_DENIED`). • If the specified board has no access rights granted and is not acquired by another user program, an attempt is made to acquired it (as by **acquire_rtc**). If a board is acquired (as by **acquire_rtc**), a version compatibility check is performed. If a version error is detected, then access to the board is denied and **select_rtc** has no effect (return code 0, **get_last_error** return code `RTC6_ACCESS_DENIED`|`RTC6_VERSION_MISMATCH`). • If the specified board is already the active board for this user program, **select_rtc** has no effect (return value: `CardNo`). • The **select_rtc** command also has no effect if `CardNo` exceeds the number of RTC6 boards found during initialization (see **rtc6_count_cards**), no RTC6 Ethernet Board is entered there or if `CardNo` = 0 (real boards begin at 1). The return value is then the number of the active board (see above). Therefore, `select_rtc( 0 )` can be used to determine the number of the active board at any time. • **The select_rtc** command is available even without explicit access rights to a particular RTC6 board.

Ctrl Command	select_rtc
RTC4→RTC6	With the RTC4, **select_rtc** only activates the specified board (unconditionally). With the RTC6, **select_rtc** additionally delivers a return value and tries to get access to the specified board granted (as with **acquire_rtc**) or remains ineffective.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Ctrl Command	select_serial_set
Function	Selects a serial-number-set for serial number control commands.
Call	`select_serial_set( No )`
Parameters	`No` Number of the desired serial-number-set. As an unsigned 32-bit value. Allowed value range: [0…3]. Only the two least significant bits are evaluated.
Multi-board Com'd Name	**n_select_serial_set**
Comments	• After initialization with **load_program_file**, serial-number-set 0 is selected. • **select_serial_set** does *not* set the serial-number-set for serial number marking. The list command **select_serial_set_list** is intended for that purpose. • For usage of **select_serial_set**, see Chapter 7.5.2 "Marking Serial Numbers", page 219.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**select_serial_set_list**, **set_serial_step**, **get_serial**

Undelayed Short List Command	select_serial_set_list
Function	Selects a serial-number-set for serial number list commands (as well as for serial number marking).
Call	`select_serial_set_list( No )`
Parameters	`No` Number of the desired serial-number-set. As an unsigned 32-bit value. Allowed value range: [0…3]. Only the two least significant bits are evaluated.
Multi-board Com'd Name	**n_select_serial_set_list**
Comments	• After initialization with **load_program_file**, serial-number-set 0 is selected. • For usage of **select_serial_set_list**, see Chapter 7.5.2 "Marking Serial Numbers", page 219.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**select_serial_set**,**set_serial_step_list**, **get_list_serial**, **mark_serial**, **mark_serial_abs**

Ctrl Command	send_user_data
Function	No function.
Multi-board Com'd Name	**n_send_user_data**
Comments	• To date, **send_user_data** has no effect.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**read_user_data**

Ctrl Command	set_angle
Function	Uses a specified rotation angle to define the rotation matrix M_R for all subsequent coordinate transformations, see Chapter 8.2 "Coordinate Transformations", page 233.
Call	`set_angle( HeadNo, Angle, at_once )`
Parameters	`HeadNo` Number of the scan head connector. As an unsigned 32-bit value. = 1: The definition only affects the Connector for First Scan Head. = 2: The definition only affects the Connector for Second Scan Head. = 0, 3: The definition affects *both* scan head connectors. = 4: The definition affects the virtual Image Field (see also comments). Only the 3 least significant bits are evaluated.
	`Angle` Rotation angle. In degrees. As a 64-bit IEEE floating point value. Positive angles result in a counterclockwise rotation around the centerpoint of the Image Field.
	`at_once` Determines when the defined transformation becomes effective. As an unsigned 32-bit value. Like **set_matrix**.
Multi-board Com'd Name	**n_set_angle**
Comments	• See also Chapter 8.2 "Coordinate Transformations", page 233. • As of DLL 617, OUT 617 the following applies for `HeadNo` = 4: The global coordinate transformations are generally available, even outside a Processing-on-the-fly Session.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Last change DLL 617, OUT 617: `HeadNo` = 4.
References	**set_angle_list**, **set_matrix**, **set_offset**, **set_scale**

<table>
<tr><td>Variable List Command</td><td colspan="2">set_angle_list</td></tr>
<tr><td>Function</td><td colspan="2">Like set_angle, but a list command.</td></tr>
<tr><td>Call</td><td colspan="2"><code>set_angle_list(HeadNo, Angle, at_once)</code></td></tr>
<tr><td rowspan="3">Parameters</td><td><code>HeadNo</code></td><td>Like set_angle. However, <code>HeadNo</code> = 4 is not available.</td></tr>
<tr><td><code>Angle</code></td><td>Like set_angle.</td></tr>
<tr><td><code>at_once</code></td><td>Determines when the defined transformation becomes effective.
As an unsigned 32-bit value.
= 0: The transformation settings are only accumulated and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (for example, by a list command with <code>at_once</code> = 1 or a corresponding control command).
= 1: The transformation is immediately calculated (including all transformation settings that were accumulated until then) and processed prior to the next list command. If necessary, Signals for “Laser Active” Operation are switched off in advance.
= 2: The transformation settings are only accumulated and intermediately stored (as with <code>at_once</code> = 0). However, The transformation is immediately calculated (including all transformation settings that were accumulated and intermediately stored until then) and applied to the current position when the next jump_abs or jump_rel (only if no list is currently being executed: also goto_xy or goto_xyz) is executed.
= 3: Like <code>at_once</code> = 1, but the Laser Control Signals remain unaffected.
> 3: Like <code>at_once</code> = 2.</td></tr>
<tr><td>Multi-board Com’d Name</td><td colspan="2">n_set_angle_list</td></tr>
<tr><td>Comments</td><td colspan="2">• –</td></tr>
<tr><td>RTC4→RTC6</td><td colspan="2">New command.</td></tr>
<tr><td>RTC5→RTC6</td><td colspan="2">Unchanged functionality.</td></tr>
<tr><td>Version Info</td><td colspan="2">Available as of DLL 600, OUT 600, RBF 600.</td></tr>
<tr><td>References</td><td colspan="2">set_angle</td></tr>
</table>

Ctrl Command	set_auto_laser_control	
Function	Initializes or deactivates “Automatic Laser Control“, see Chapter 7.4.9 ““Automatic Laser Control““, page 205.	
Call	`ErrorCode = set_auto_laser_control( Ctrl, Value, Mode, MinValue, MaxValue )`	
Parameters	`Ctrl`	Control parameter. As an unsigned 32-bit value. = 1…7: Defines which signal parameter is corrected by “Automatic Laser Control“. = 1: 12-bit output value at the ANALOG OUT1 output port. = 2: 12-bit output value at the ANALOG OUT2 output port. = 3: Output value at the 8-Bit Digital Output Port. = 4: Pulse length (`PulseLength`) of the Laser Control Signals LASER1 and LASER2. = 5: Output period (`HalfPeriod`) of the Laser Control Signals LASER1 and LASER2. = 6: Output value at the 16-Bit Digital Output Port. = 7: As 5. However, the pulse distance is geometrically constant (“Spot Distance Control“). = 0 or > 7: Deactivates “Automatic Laser Control“ (for `Ctrl` > 7: **get_last_error** return code `RTC6_PARAM_ERROR`).
	`Value`	Defines the 100% value for the parameter selected by `Ctrl`. As an unsigned 32-bit value. Allowed values:
		For `Ctrl` = 1/2: 12-bit values [0…4,095]. Bits with higher significance are ignored. For `Ctrl` = 3: 8-bit values [0…255]. Bits with higher significance are ignored. For `Ctrl` = 4: [0…(2^{32}–1)]. 1 bit equals 1/64 µs. For `Ctrl` = 5: [0…(2^{32}–1)]. 1 bit equals 1/64 µs. For `Ctrl` = 6: 16-bit values [0…65,535]. Bits with higher significance are ignored.
	`Mode`	Speed-Dependent Laser Control (`Mode` =1…5) lets you select which data is to be used for calculating speed-dependent correction. As an unsigned 32-bit value. =1: Set speed. =2: Actual speed (as well as extensions, see below). =3: Reserved. =4: Reserved. =5: Encoder speed (counter pulse rate).

<table>
<tr><th>Ctrl Command</th><th colspan="2">set_auto_laser_control</th></tr>
<tr><td rowspan="3">Parameters
(cont'd)</td><td>Mode
(cont'd)</td><td>For Mode = 2 above, the following extensions are available in any combination:
• M = +4 Encoder speed-dependent correction, see comment, page 662
• M = +16 For SCANahead Systems
• M = +32 Angle-synchronous galvanometer scanner speed is converted into an Image Field synchronous speed with the help of the assigned correction table, see comment, page 663
• M = +64 coordinate transformations for the first scan head are backward transformed, see comment, page 663
• M = +128 "Spot Distance Control" in combination with Sky Writing. The spot distance is even kept constant in a Polyline, which is interrupted by a Sky Writing motion.
Then the following applies: Mode = 2 + sum of extensions M.
For Mode = 0 or none of the above described ones:
Disables the Speed-Dependent Laser Control or Encoder-Speed-Dependent Laser Control
(get_last_error-Returncode: RTC6_PARAM_ERROR). The Position-Dependent Laser Control always remains effective with a valid value for Ctrl.</td></tr>
<tr><td>MinValue</td><td>Defines the lower range limit of the parameter selected by Ctrl for automatic correction and that cannot be undercut.
As an unsigned 32-bit value.
Allowed values: see Value.</td></tr>
<tr><td>MaxValue</td><td>Defines the upper range limit of the parameter selected by Ctrl for automatic correction and that cannot be exceeded.
As an unsigned 32-bit value.
Allowed values: see Value.</td></tr>
<tr><td>Result</td><td colspan="2">ErrorCode.
As an unsigned 32-bit value.
0 No error.
1 No first scan head active.
2 No iDRIVE scan system (see Glossary entry on page 28) active.
3 Invalid Ctrl value.
4 Invalid Mode value.
5 Access denied (board locked, get_last_error return code RTC6_ACCESS_DENIED).
6 Mode = 2 extension M = +16: No SCANahead System connected or not active.</td></tr>
<tr><td>Multi-board Com'd Name</td><td colspan="2">n_set_auto_laser_control</td></tr>
</table>

Ctrl Command	set_auto_laser_control
Comments	• For usage of **set_auto_laser_control**, see Chapter 7.4.9 ""Automatic Laser Control"", page 205. • MinValue and MaxValue are automatically exchanged if MinValue > MaxValue and MinValue is clipped to ≤ Value and MaxValue clipped to ≥ Value. • Control data for Speed-Dependent Laser Control is exclusively derived from the scan system data at the Connector for First Scan Head and then also applied for the Connector for Second Scan Head (if that connector has been activated and a scan system is attached). At the time **set_auto_laser_control**is called, the Connector for First Scan Head must already have been assigned a correction table, otherwise error code 1 is returned and Ctrl is set to 0. If only the Connector for Second Scan Head is being used and/or the scan system at the Connector for First Scan Head is shut off, then Speed-Dependent Laser Control does not function. • For Mode = 2, a functioning i*DRIVE* scan system (see Glossary entry on page 28) must be attached to the Connector for First Scan Head. As a result, **control_command** is unavailable for the Connector for First Scan Head as long as this Mode selection is in effect (because the current speed is utilized – which is transmitted by the scan system). For Mode = 0 or 1, **control_command** is available without any restriction. If no functioning i*DRIVE* scan system is attached, then error code 2 is returned and Ctrl set to 0. • Encoder-Speed-Dependent Laser Control (Mode = 5) functions even if no *scan heads* are attached to the scan-head connectors at runtime. The Option Processing-on-the-fly does not need to be activated here either. The target encoder speed is set by **set_encoder_speed**. • **set_auto_laser_control** with Mode = 2 extension M = +4 converts encoder speeds to galvanometer scanner units (*bits/ms*) by using the scaling factors from **set_fly_x** and **set_fly_y** or **set_fly_2d**. The result is then added to the current galvanometer scanner speeds. This requires an enabled Option Processing-on-the-fly (in contrast to Mode = 5). The current mark speed is used as reference speed. The **set_encoder_speed** command (which is used for Mode = 5) is not taken into account with the +4 extension. If an axis is not activated for Processing-on-the-fly at runtime, its encoder speed is not taken into account. If both axes are not activated for Processing-on-the-fly, +4 extension is not effective. **set_fly_rot**, **set_fly_rot_pos**, **set_fly_x_pos** and **set_fly_y_pos** cannot be combined with the galvanometer scanner speed. • **set_auto_laser_control** with Mode = 2 extension M = +4 even works, if no Processing-on-the-fly correction is active. However, the corresponding scaling factors must have been set beforehand by one of the **[*]fly[*]** commands (for example, **set_fly_x**, **set_fly_2d**, ...). The Processing-on-the-fly correction activated in this way must be deactivated by **fly_disable_list** so that M = +4 continues to work even after the Processing-on-the-fly correction. See Comments, page 418 on **fly_disable_list**.

Ctrl Command	set_auto_laser_control
Comments (cont'd)	• **set_auto_laser_control** with `Mode` = 2 extension M = +64 must be called before setting corresponding coordinate transformations (for example, by **set_scale**, **set_angle**, …). Otherwise, the backward transformation is not executed correctly. • **set_auto_laser_control** with `Mode` = 2 extension M = +128 can be combined with **set_sky_writing_min_speed_ctrl**. The energy input is kept constant though. • For `Ctrl` = 1…6, the selected signal parameter is updated every 10 µs. For `Ctrl` = 5 (output period) it is always only effective after an already started laser control signal period has elapsed. For `Ctrl` = 7, however, the output period is continuously adjusted (from pulse to pulse). • `Ctrl` = 7 ("Spot Distance Control") is only available for SCANahead Systems. For `Ctrl` = 7 to actually take effect the requested geometric pulse distance must be specified. As long as the geometric pulse distance is 0 (default `Value`) this control is ineffective, but not deactivated. The actual controlled variable ("temporal pulse distance") cannot be recorded with **set_trigger[*]** and signal 24. The parameters `MinValue`, `MaxValue` and `Value` are ignored. Therefore, the actual controlled variable ("temporal pulse distance") cannot be limited by `MinValue` and `MaxValue`. It is automatically controlled to the current mark speed. See also Section ""Spot Distance Control"", page 210. • Activating the angle-to-Image Field transformation of the galvanometer scanner speed (+32 extension) is particularly useful, if the correction file has to compensate for strong distortions. • If the values for `Ctrl` or `Mode` are invalid, then **set_auto_laser_control** is not executed (return value 3 or 4, **get_last_error** return code `RTC6_PARAM_ERROR`). • "Automatic Laser Control" should not be combined with the variable laser power of **set_multi_mcbsp_in**. • Only SCANahead Systems: If a SCANahead System is connected and active, make sure that it is no longer busy before calling **set_auto_laser_control** (see **get_status** and HEAD BUSY list execution status). Otherwise, the not yet processed rest of the marking could be compromised. • **set_auto_laser_control** is allowed in Boot Phase 2, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command. In RTC4 Compatibility Mode for `Ctrl` = 1/2, the specified values for `Value`, `MinValue` and `MaxValue` are internally multiplied by 4. For `Ctrl` = 4/5, the parameter values for `Value`, `MinValue` and `MaxValue` are multiplied by 8. The allowed value ranges decrease accordingly.
RTC5→RTC6	Changed functionality. More laser control modes can be set.

Ctrl Command	set_auto_laser_control
Version Info	Available as of DLL 600, OUT 600, RBF 600. Change DLL 609, OUT 609, RBF 613: `Ctrl` = 7. Change DLL 641, OUT 642: `Mode` = 2 extension M = +64. Last change DLL 646, OUT 649, RBF 641: `Mode` = 2 extension M = +128.
References	**load_position_control**, **load_auto_laser_control**, **set_auto_laser_params**, **set_auto_laser_params_list**, **set_fly_x**, **set_fly_y**, **set_fly_2d**, **set_encoder_speed**, **spot_distance**, **spot_distance_ctrl**

Ctrl Command	set_auto_laser_params	
Function	Specifies the to-be-controlled signal parameter of the “Automatic Laser Control“, the 100% value and its corresponding limit values.	
Call	`ErrorCode = set_auto_laser_params( Ctrl, Value, MinValue, MaxValue )`	
Parameters	`Ctrl`	The to-be-controlled signal parameter (see **set_auto_laser_control**). Allowed value range: [1…7].
	`Value`	100% value. See **set_auto_laser_control**.
	`MinValue`	*Lower* range limit. See **set_auto_laser_control**.
	`MaxValue`	*Upper* range limit. See **set_auto_laser_control**.
Result	`ErrorCode`. As an unsigned 32-bit value. 0 No error. 3 Invalid `Ctrl` value.	
Multi-board Com’d Name	**n_set_auto_laser_params**	
Comments	• For information on using **set_auto_laser_params**, see Chapter 7.4.9 ““Automatic Laser Control“”, page 205. • The meaning of the parameters is identical to that of **set_auto_laser_control** (see also comments there). However, **set_auto_laser_params** can neither start nor terminate “Automatic Laser Control“. • If the value for `Ctrl` is invalid (0 or > 7), then **set_auto_laser_params** is not executed (**get_last_error** return code `RTC6_PARAM_ERROR`).	
RTC4→RTC6	New command. RTC4 Compatibility Mode: see **set_auto_laser_control**.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 609, OUT 609, RBF 613: `Ctrl` = 7.	
References	**set_auto_laser_control**, **set_auto_laser_params_list**	

Delayed Short List Command	set_auto_laser_params_list	
Function	Like **set_auto_laser_params**, but a Delayed Short List Command.	
Call	`ErrorCode = set_auto_laser_params_list( Ctrl, Value, MinValue, MaxValue )`	
Parameters	`Ctrl`	Like **set_auto_laser_params**.
	`Value`	Like **set_auto_laser_params**.
	`MinValue`	Like **set_auto_laser_params**.
	`MaxValue`	Like **set_auto_laser_params**.
Multi-board Com'd Name	**n_set_auto_laser_params_list**	
Comments	• If the value for `Ctrl` is invalid (0 or > 7), then **set_auto_laser_params_list** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).	
RTC4→RTC6	New command. RTC4 Compatibility Mode: see **set_auto_laser_control**.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 609, OUT 609, RBF 613: `Ctrl` = 7.	
References	**set_auto_laser_params**, **set_auto_laser_control**	

Ctrl Command	set_char_pointer	
Function	Stores the absolute start address of a command list in the internal management table for indexed characters.	
Call	`set_char_pointer( Char, Pos )`	
Parameters	`Char`	Index of the indexed character whose starting address `Pos` should be entered in the management table. As an unsigned 32-bit value. Allowed value range: [0…1023]. The same applies as for **load_char**: `Char` = character set number × 256 + ASCII number of the character (character sets are numbered 0 to 3).
	`Pos`	Absolute start address. As an unsigned 32-bit value. Allowed value range: [0…(2^{23}–1)].
Multi-board Com'd Name	**n_set_char_pointer**	
Comments	• If `Char` > 1023 and/or `Pos` > (2^{23}–1), then **set_char_pointer** is *not* executed (**get_last_error** return code `RTC6_PARAM_ERROR`). • The **set_char_pointer** command can be used for referencing an indexed character. It thereby becomes an indexed character that is protectable by **save_disk**/**load_disk** and/or callable by the index. • **set_char_pointer** can also be used to reference anew an indexed subroutine, character or text string so that it can also be called by a second index. Here, it is preferable to use the **copy_dst_src** command for index management. • The start addresses of command lists that are to be referenced with **set_char_pointer** can be queried by **get_input_pointer** *before* saving the command lists. • **set_char_pointer** only stores *starting addresses* in the internal management table. An indexed character only gains protection by a subsequent **save_disk**/**load_disk** command. • `Pos` should not be an arbitrary address within a list. Instead, it should be the starting address of an actually existing subroutine that has been finalized by **list_return** and does not contain **set_end_of_list**.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**load_char**	

Ctrl Command	set_char_table
Function	Stores the absolute start address of a command list in the internal management table for indexed text strings.
Call	`set_char_table( Index, Pos )`
Parameters	`Index` Index of the indexed text string whose starting address `Pos` should be entered in the management table. As an unsigned 32-bit value. Allowed value range: [0…41]). The same ordering applies as for the **load_text_table** command: = 0…9: Digits for marking the time and date [0…9]. = 10…21: Months [January…December]. = 22…28: Days-of-the-week [Sunday…Saturday]. = 29: Blank character for marking serial numbers. = 30…39: Digits for marking serial numbers [0…9]. = 40: Text for “a.m.”. = 41: Text for “p.m.”.
	`Pos` Absolute start address. As an unsigned 32-bit value. Allowed value range: [0…(2^{23}–1)].
Multi-board Com’d Name	**n_set_char_table**
Comments	• **set_char_table** is synonymous with **set_text_table_pointer**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_text_table_pointer**

Ctrl Command	set_control_mode
Function	Enables or disables the external control input and resets the counter for External Starts to zero.
Call	`set_control_mode( Mode )`
Parameters	`Mode` As an unsigned 32-bit value.
	Bit # — Value — Description
	Bit #0 (LSB) = 1: The external start input (by /START, /START2 or /Slave-START) is enabled.
	= 0: The external start input is disabled.
	Bit #1 = 1: A /STOP, /STOP2, /Slave-STOP or **simulate_ext_stop** causes explicit cancellation of the external start queue entries (/START, /START2, /Slave-START or **simulate_ext_start**).
	= 0: No effect.
	Bit #2 = 1: The track delay (defined by **simulate_ext_start**, **set_ext_start_delay** or **set_ext_start_delay_list**) that postpones execution of the start relative to the triggering input signal or **simulate_ext_start** or **simulate_ext_start_ctrl** command (see Section "External Start", page 312) is deactivated.
	= 0: No effect. To define and activate the track delay (for example, for Processing-on-the-fly applications), use **simulate_ext_start**, **set_ext_start_delay** or **set_ext_start_delay_list**.
	Bit #3 = 1: The external start input is *not* disabled by an external stop signal.
	= 0: The external start input *is* disabled by an external stop signal.
	Bit #4 Disables **simulate_ext_start_ctrl**.
	Bit #5 Reserved.
	Bit #6 Reserved.
	Bit #7 Reserved.
	Bit #8 Reserved.
	Bit #9 = 1: Encoder resets of the 2 internal encoder counters occur by: • An external start signal • **simulate_ext_start** • **simulate_ext_start_ctrl**, postponed by a track delay set by **simulate_ext_start**, **set_ext_start_delay** or **set_ext_start_delay_list**, see also Bit #2
	= 0: Encoder resets occur immediately with each initiating Processing-on-the-fly command.

Ctrl Command	set_control_mode		
Parameters (cont'd)	Bit #10	= 1:	Track delay configured by **simulate_ext_start**, **set_ext_start_delay** or **set_ext_start_delay_list** is counted beginning with the most recent externally (but not with **execute_list_pos** etc.) triggered or simulated External Start. The interval between subsequent External Starts (in encoder pulses) is thus constant, see also Section "Regular (Periodic) External Starts", page 315. For **stop_execution** or an external stop signal, Bit #10 gets reset to "0".
		= 0:	Track delay configured by **simulate_ext_start**, **set_ext_start_delay** or **set_ext_start_delay_list** is counted beginning with the time point an External Start has been requested (that is, with the corresponding **simulate_ext_start** or **simulate_ext_start_ctrl** command or external start signal). The interval between subsequent External Starts (in encoder pulses) can thus vary.
	Bit #12		Reserved.
	…		…
	Bit #31		Reserved.
Multi-board Com'd Name	**n_set_control_mode**		
Comments	• If execution is aborted by **stop_execution**, then Bit #0 and Bit #10 gets reset to zero, thus deactivating external start input ports and the counting of track delays with respect to a the triggering event. • If Bit #9 = 0, then there is generally a small (random) time offset (10 µs jitter) between the start signal at /START, /START2 and the actual execution start. If Bit #9 = 1, then this 10 µs jitter is not present because the encoder reset then occurs synchronously with the start signal. • Bit #9 = 0 is useful, when several separate Processing-on-the-fly Sessions are to be started within a list. • See also Section "External Stop", page 311 and Section "External Start", page 312. • **set_control_mode** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.		

Ctrl Command	set_control_mode
RTC4→RTC6	Bit #8 cannot not be used to lock or unlock the upper 8 bits of the 16-Bit Digital Output Port for I/O commands. The RTC6 automatically reserves these bits for vario*SCAN* FLEX control by **move_to**. It is not possible to explicitly release these bits (release automatically occurs by **load_program_file**). Otherwise: unchanged functionality for the bits that are also used on the RTC4.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 605, OUT 605: Bit #4.
References	**set_control_mode_list**, **set_extstartpos**, **get_counts**, **set_max_counts**, **get_startstop_info**, **move_to**

Normal List Command	**set_control_mode_list**
Function	Like **set_control_mode**, but a list command.
Call	`set_control_mode_list( Mode )`
Parameters	`Mode` Like **set_control_mode**.
Multi-board Com'd Name	**n_set_control_mode_list**
Comments	• The counter for External Starts is *not* reset by **set_control_mode_list**.
RTC4→RTC6	Unchanged functionality for the bits that are also used on the RTC4.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 605, OUT 605: Bit #4.
References	**set_control_mode**

Ctrl Command	set_controlpreview_compensation_ctrl	
Function	Only for Preprocessing System. Compensates for the `ControlPreview` time by delaying Laser Control Signals output.	
Call	`set_controlpreview_compensation_ctrl( ControlPreview, Mode )`	
Parameters	`ControlPreview`	Delay of the laser control signal output. In 1/64 μs. As an unsigned 32-bit value.
	`Mode`	Reserved.
Multi-board Com'd Name	**n_set_controlpreview_compensation_ctrl**	
Comments	• **set_controlpreview_compensation_ctrl** is exclusively provided to control scan systems that have a `ControlPreview` time specified (for example, *SCAN*cube IV), see Preprocessing System and corresponding scan system manual. • If SCANahead Functionality is active, `ControlPreview` has no effect. • **set_controlpreview_compensation_ctrl** also updates Processing-on-the-fly total prediction values, see **set_fly_tracking_error**. • **set_controlpreview_compensation_ctrl** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 634, OUT 635. Last change DLL 646, OUT 649: updates Processing-on-the-fly total prediction values.	
References	**set_fly_tracking_error**, **fly_prediction**	

Ctrl Command	set_default_pixel
Function	Defines the pulse length default value for the default pixel that terminates Pixel Output Mode.
Call	`set_default_pixel( PulseLength )`
Parameters	`PulseLength` Default pixel pulse length. As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: [0…(2^{32}–1)].
Multi-board Com'd Name	**n_set_default_pixel**
Comments	• In Pixel Output Mode, the pixel pulse length at the end of an image line (at the beginning of the default pixel) gets set to the specified default value, see Chapter 8.7.4 "Synchronization", page 277. • When initializing (by **load_program_file**), the `PulseLength` default value is set to 0.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_port_default**

Undelayed Short List Command	set_default_pixel_list
Function	Like **set_default_pixel**, but a list command.
Call	`set_default_pixel_list( PulseLength )`
Parameters	`PulseLength` Like **set_default_pixel**.
Multi-board Com'd Name	**n_set_default_pixel_list**
Comments	• See **set_default_pixel**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_default_pixel**, **set_port_default**

Ctrl Command	set_defocus
Function	Determines a focus shift for 3D outputs.
Restriction	If the Option "3D" has not been enabled or if no 3D correction table has been assigned (see **select_cor_table**), then **set_defocus** has no effect. Nevertheless, the supplied focus shift value is stored internally and takes effect as soon as a 3D correction table is assigned.
Call	`set_defocus( Shift )`
Parameters	`Shift` Focus shift. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_set_defocus**
Comments	• A focus shift causes a defocusing of the laser focus relative to the working plane. A positive value increases the focal length of the Dynamic Focusing Unit and shifts the focus position, for example, for the control value (0|0|0) by about d = `Shift` / *K* to the plane z = –d [mm]. For the calibration factor *K*, see Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 172; furthermore, #8 in Chapter 7.3.6 "Output Values to the Scan System", page 186. • If the resulting total Z output value is too large, the z axis moves to the limit stop. Make sure that this is avoided as far as possible, see **get_galvo_controls**. • If **set_defocus** is called during output of a vector, then it is only executed directly before the next list command. To avoid "Hard Jumps", a jump to the changed z output is performed at jump speed. Length and duration of the jump depend on the changed z control value. The duration can be calculated by **get_galvo_controls**. If *no* list is currently BUSY list execution status, then the jump executes immediately, whereby no delay occurs at the next start. • If the INTERNAL-BUSY list execution status is set, **set_defocus** is only executed with a delay (after INTERNAL-BUSY list execution status has been reset again). • **set_defocus** sets the INTERNAL-BUSY list execution status while the jump to the changed z output is executed. • After Vector-Defined Laser Control is activated by **set_vector_control**(`Ctrl` = 7), the focus shift changes with parameterized [*]mark[*] Command or Jump Commands, too.
RTC4→RTC6	Basically unchanged functionality. But the RTC6 command avoids "Hard Jumps".
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_defocus_list**, **set_defocus_2_list**, **set_offset_xyz**, **set_defocus_offset**, **get_galvo_controls**

Variable List Command	set_defocus_list
Function	Like **set_defocus**, but a list command.
Restriction	Like **set_defocus**.
Call	`set_defocus_list( Shift )`
Parameters	`Shift` Like **set_defocus**.
Multi-board Com'd Name	**n_set_defocus_list**
Comments	• See **set_defocus**. • Even though **set_defocus_list** is a Short List Command, execution of the directly following list command is delayed by a few 10 µs clock cycles due to the intermediate jump to the changed z output. The extent of this delay depends on the size of the specified focus shift; but it is at least 10 µs, even if `Shift`= 0. • After the jump to the changed z output, a Jump Delay previously configured with **set_scanner_delays** is inserted. Depending on the dynamics of the z axis it may be reasonable to increase the Jump Delay before calling **set_defocus_list**. In case of a SCANahead System with automatic delay calculation, **long_delay** can alternatively be used to compensate for the Tracking Error of the z axis.
RTC4→RTC6	See **set_defocus**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_defocus**, **set_defocus_2_list**, **long_delay**

Undelayed Short List Command	**set_defocus_2_list**	
Function	Like **set_defocus_list**, but with an additional parameter `Mode`.	
Restriction	Like **set_defocus_list**.	
Call	`set_defocus_2_list( Shift, Mode )`	
Parameters	`Shift`	Like **set_defocus_list**.
	`Mode`	Mode. As a signed 32-bit value.
	= 0	`Shift` is immediately applied ("Hard Jump").
	> 0	Reserved.
Multi-board Com'd Name	**n_set_defocus_2_list**	
Comments	• **set_defocus_2_list**(`Mode` = 0) executes – unlike **set_defocus_list** – a "Hard Jump" to the changed z output in the exact same 10 μs clock cycle. • See **set_defocus_list**. • For `Mode` > 0, **set_defocus_2_list** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 646, OUT 649.	
References	**set_defocus_list**	

Ctrl Command	set_defocus_offset
Function	Defines an offset in addition to the focus shift (see **set_defocus**) for all 3D outputs.
Restriction	Like **set_defocus**.
Call	`set_defocus_offset( Shift )`
Parameters	`Shift` Like **set_defocus**.
Multi-board Com'd Name	**n_set_defocus_offset**
Comments	• **set_defocus_offset** has the same effect as **set_defocus**. • **set_defocus_offset** enables setting a global defocus shift that is not overwritten by parameterized commands such as **para_mark_abs**, see **set_vector_control**(`Ctrl` = 7). • **set_defocus_offset** `shift`: – Works additively to **set_defocus** – Cannot be used as a control parameter for Vector-Defined Laser Control – Cannot be recorded via Signal 32 with **set_trigger[*]** – Is taken into account by **get_z_distance**, **get_galvo_controls** and the back transformation (**transform**, **get_transform**/**get_transform_offset**)
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `Shift` by 16.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the specified value for `Shift` by 16.
Version Info	Available as of DLL 609, OUT 609, RBF 614.
References	**set_defocus**, **set_defocus_offset_list**

Variable List Command	set_defocus_offset_list
Function	Like set_defocus_offset, but a list command.
Restriction	Like set_defocus.
Call	`set_defocus_offset_list( Shift )`
Parameters	`Shift` Like set_defocus.
Multi-board Com'd Name	n_set_defocus_offset_list
Comments	• See set_defocus_offset. • See set_defocus_list. • Even though set_defocus_offset_list is a Short List Command, execution of the directly following list command is delayed by a few clock cycles due to the intermediate jump to the changed z position. The extent of this delay depends on the size of the specified focus shift; but it is at least 10 μs, even if `Shift`= 0.
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `Shift` by 16.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the specified value for `Shift` by 16.
Version Info	Available as of DLL 609, OUT 609, RBF 614.
References	set_defocus_offset, set_defocus_list

Ctrl Command	**set_delay_mode**		
Function	Turns the "Variable Polygon Delays" mode and the "Variable Jump Delays" mode on or off and sets some special Scanner Delay-related parameters as well as the 3D Z-Move mode.		
Call	`set_delay_mode( VarPoly, DirectMove3D, EdgeLevel, MinJumpDelay, JumpLengthLimit )`		
Parameters	Name	Allowed Values	Description
	`VarPoly`	> 0	Enables "Variable Polygon Delays" mode.
		= 0	Disables "Variable Polygon Delays" mode. Default setting.
			As an unsigned 32-bit value.
	`DirectMove3D`		This parameter effects only 3D-applications.
		> 0	The z output is changed directly (linearly) to its end value during a jump.
		= 0	The z output is changed to its end-value in such a way that the focus is kept in one plane during the entire jump.
			As an unsigned 32-bit value.
	`EdgeLevel`	0…(2^{32}–1). 1 bit equals 10 µs.	This parameter defines a maximum "laser on" time for the corners of a Polyline. If the Mark Delay is longer than or equal to this value (because the angle φ is close to 180°, for instance), the laser is switched off (after a LaserOff Delay) and a new Polyline is started. This can be useful for preventing burn-in effects. The `EdgeLevel` value must be smaller than twice the set value for the Mark Delay, otherwise it has no effect. See also Figure 40.
			Note: To disable this feature, the `EdgeLevel` value must be set to (2^{32}–1) (default value).
			As an unsigned 32-bit value.
	`MinJumpDelay`	0…(2^{32}–1). 1 bit equals 10 µs.	Minimum Jump Delay that cannot be undercut for jumps shorter than `JumpLengthLimit` (even for jump vectors of zero length, see Figure 36). For jumps longer than `JumpLengthLimit`, the `MinJumpDelay` has no relevance. To avoid anomalies in the range of `MinJumpDelay`, define a value for `MinJumpDelay` that is not larger than the Jump Delay.
			As an unsigned 32-bit value.

Ctrl Command	set_delay_mode		
	`JumpLengthLimit`	0…(2^{32}–1)	Jump length limit. In bits. `JumpLengthLimit` > 0 enables “Variable Jump Delays“ mode. If the jump vector is *longer* than this value, then the fixed Jump Delay (see **set_scanner_delays**) is inserted. For all shorter jump lengths, a linearly interpolated “Variable Jump Delay” (page 151) between `MinJumpDelay` and the Jump Delay is calculated and inserted, see Figure 36. `JumpLengthLimit` = 0 disables “Variable Jump Delays“ mode. As an unsigned 32-bit value.
Multi-board Com’d Name	**n_set_delay_mode**		
Comments	• –		
RTC4→RTC6	Unchanged functionality. In addition: extended value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `JumpLengthLimit` by 16. The allowed value range decreases accordingly.		
RTC5→RTC6	Unchanged functionality.		
Version Info	Available as of DLL 600, OUT 600, RBF 600.		
References	**set_scanner_delays**, **load_varpolydelay**, **set_delay_mode_list**		

Multiple List Command	**set_delay_mode_list**
Function	Like **set_delay_mode**, but a list command.
Call	`set_delay_mode_list( VarPoly, DirectMove3D, EdgeLevel, MinJumpDelay, JumpLengthLimit )`
Parameters	`VarPoly` Like **set_delay_mode**.
	`DirectMove3D` Like **set_delay_mode**.
	`EdgeLevel` Like **set_delay_mode**.
	`MinJumpDelay` Like **set_delay_mode**.
	`JumpLengthLimit` Like **set_delay_mode**.
Multi-board Com'd Name	**n_set_delay_mode_list**
Comments	• See **set_delay_mode**. • **set_delay_mode_list** occupies 2 RTC6 List Memory positions. • **set_delay_mode_list** is executed as 2 undelayed short list commands. • Any still-pending Delayed Short List Command is executed beforehand.
RTC4→RTC6	New command. RTC4 Compatibility Mode: see **set_delay_mode**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_delay_mode**

Ctrl Command	set_dsp_mode
Function	Only for some discontinued systems. Sets a DSP mode for short-command count.
Call	`dsp_mode = set_dsp_mode( Mode )`
Parameters	`Mode` Desired DSP mode. As an unsigned 32-bit value. = 0: Corresponds to 500 MHz RTC5 boards (older type series). = 1: Reserved. = 2: Corresponds to 720 MHz RTC5 boards (newer type series). = 3: Corresponds to RTC6 boards. Default setting.
Result	Set DSP mode prior to **set_dsp_mode** execution. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_dsp_mode**
Comments	• **set_dsp_mode** is used to switch the faster processing of short vectors in standard mode (DSP mode 3) to a RTC5-compatible processing which is slower, see Section "Automatic Delay Adjustments", page 158. • **set_dsp_mode** is required, if: – several RTC6 boards with different DSP modes are to be operated synchronously in a master/slave chain – *AND* if the adaptation of the short command count is not carried out by **sync_slaves** • DSP mode > 3 cannot be set. In this case, `Mode` is clipped to 3. • **set_dsp_mode** is allowed in Boot Phase 2, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	Basically unchanged functionality. However: Standard mode (DSP mode 3) is available.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**sync_slaves**, **get_rtc_version**

Normal List Command	**set_duty_cycle_table**
Function	Category: List Commands for a Restricted User Group only. Sets a value in the lookup table for the pulse length.
Call	`set_duty_cycle_table( Index, DutyCycle )`
Parameters	`Index` — Index in the lookup table for the pulse length. Allowed value range: 0…10. As an unsigned 32-bit value.
	`DutyCycle` — Duty Cycle. In per mille. Allowed value range: 0…1,000. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_duty_cycle_table**
Comments	• The relationship between encoder frequency and laser frequency is defined as follows: – Encoder frequency = `Index` × [`Fmax` / 10] • Within the lookup table, the values are linearly interpolated. • The value for pulse length is set depending on the laser frequency: – Pulse length = laser frequency × `DutyCycle` / 1,000 • The following applies for List Commands for a Restricted User Group only: These users have been informed separately in how to handle these RTC6 commands. Therefore, only the command table is available in this manual and further information is omitted.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 626, OUT 627.
References	**set_laser_timing_table**, **regulation3**

Undelayed Short List Command	**set_ellipse**	
Function	Defines the shape of an elliptical arc that can subsequently be marked by **mark_ellipse_abs** or **mark_ellipse_rel**.	
Call	`set_ellipse( a, b, Phi0, Phi )`	
Parameters	`a`	Length of the elliptical half-axis. In bits. As an unsigned 32-bit value. Allowed value range: [1…+8,388,607]. Out-of-range positive values are clipped to the boundary values.
	`b`	See `a`.
	`Phi0`	Beginning phase angle (the arc starting point position relative to the end point of half-axis a). In degrees. As a 64-bit IEEE floating point value. A positive sign means “clockwise”. `Phi0` gets normalized to the value range [0…<360°].
	`Phi`	Arc angle (to-be-marked ellipse section). In degrees. As a 64-bit IEEE floating point value. A positive sign means “clockwise”. Allowed value range: [–2,880.0°…+2,880.0°] (±8 full ellipses). Out-of-range values are clipped to the boundary values.
Multi-board Com’d Name	**n_set_ellipse**	
Comments	• Specify the to-be-marked elliptical arc’s position and orientation in the 2D Image Field by **mark_ellipse_abs** or **mark_ellipse_rel**. For descriptions of the individual parameters, see Section “Ellipse Commands”, page 141. • For `a` < 1 and/or `b` < 1, **set_ellipse** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `a` and `b` by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 639, OUT 641: Allowed value range for `Phi` has been reduced (previously: [–3,600.0°…+3,600.0°] = ±10 full ellipses).	
References	**mark_ellipse_abs**, **mark_ellipse_rel**	

Ctrl Command	set_encoder_filter_ctrl
Function	Switches on (off) either the Encoder Speed Filter or Encoder Input Filter.
Call	`set_encoder_filter_ctrl( Length, Mode )`
Parameters	`Length` Combined with `Mode` = 0: • Encoder Speed Filter length. As exponent to base 2. As an unsigned 32-bit value. Allowed values: [0, 1, 2, 3, 4, 5, 6, 7]. Combined with `Mode` = 1: • Encoder Input Filter. As an unsigned 32-bit value. = 0 Off. = 1 On. `Mode` Encoder filter type. As an unsigned 32-bit value. = 0 Encoder Speed Filter. = 1 Encoder Input Filter. > 1 Reserved.
Multi-board Com'd Name	**n_set_encoder_filter_ctrl**
Comments	• **set_encoder_filter_ctrl** is not executed for (**get_last_error** return code `RTC6_PARAM_ERROR`): – `Length` > 7, `Mode` = 0 – `Length` > 1, `Mode` = 1 – `Mode` > 1 • Encoder Speed Filter – Smooths encoder speeds on the basis of the moving average. – `Length` = 0 (2^0 = 1) disables the Encoder Input Filter. – A higher `Length` value improves the filter effect at the expense of a certain delay in the encoder speeds used. • Encoder Input Filter – Limits the encoder signal frequency to 4 MHz (16 encoder signal edges / µs). See also Chapter 9.3.3 "Synchronization by Encoder Signals", page 320. – `Length` = 1 switches the Encoder Input Filter on. Default after **load_program_file**. – `Length` = 0 switches the Encoder Input Filter off. This allows encoder signal frequencies of > 4 MHz. However, the RTC6 board then no longer recognizes encoder errors (**get_marking_info**(`Bit #16`…`Bit #21`)). Therefore, you must make sure the following yourself: • Received signals are valid • The Processing-on-the-fly application works as expected • See also Section "Input Ports for External Encoder Signals", page 321.

Ctrl Command	set_encoder_filter_ctrl
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 641, OUT 642, RBF 638. Last change DLL 646, OUT 649, RBF 641: `Mode` = 1.
References	–

Delayed Short List Command	set_encoder_speed
Function	Defines the target encoder speed and further parameters for Encoder-Speed-Dependent Laser Control.
Call	`set_encoder_speed( EncoderNo, Speed, Smooth )`
Parameters	`EncoderNo` — Number of the encoder counter to be used for speed measurement. As an unsigned 32-bit value. Allowed values: = 0: Encoder counter "Encoder0". = 1: Encoder counter "Encoder1". = 2 and 3: Vectorial encoder velocity: a vectorial velocity is calculated from both encoder speeds (by the pulse rates of both encoder counters) for use with Encoder-Speed-Dependent Laser Control in `Mode`= 5. Bits with higher significance are ignored.
	`Speed` — Target encoder speed (counter pulse rate) in [counter pulses/ms]. As a 64-bit IEEE floating point value. Allowed value range: [100.0…16000.0]. Out-of-range values are clipped to the boundary values.
	`Smooth` — Smoothing factor for a 2-stage low-pass filter. As a 64-bit IEEE floating point value. Allowed value range: [0.0…1.0]. Larger values are clipped.
Multi-board Com'd Name	**n_set_encoder_speed**
Comments	• If `Smooth` = 0.0, then only the current encoder speed `CurrentSpeed` (based on counter pulses received in the most recent 10 μs) is used; if `Smooth` = 1.0, then the speed from the previous clock cycle `PreviousSpeed`. In general, the used speed is: Speed = `PreviousSpeed` × `Smooth` + `CurrentSpeed` × (1.0 – `Smooth`). • If `Speed` ≤ 0.0 or `Smooth` < 0.0, then **set_encoder_speed** is, already during loading, replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • One encoder increment results in 4 counter pulses, see Section "Input Ports for External Encoder Signals", page 321. • The maximum value for `Speed` (16000.0) corresponds to a counting rate of 16 MHz. The minimum value for `Speed` (100.0) corresponds to a counting rate of 1/(10 μs), that is, one counter pulse per output period. Beware of the low resolution of Encoder-Speed-Dependent Laser Control for low speed values! Encoder-speed-dependent correction is only recommended if substantially more than one counter pulse per output period (10 μs) are received. • See also Section "Encoder-Speed-Dependent Laser Control", page 212.

Delayed Short List Command	set_encoder_speed
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_encoder_speed_ctrl**, **set_auto_laser_control**

Ctrl Command	set_encoder_speed_ctrl	
Function	Like **set_encoder_speed**, but a control command.	
Call	`set_encoder_speed_ctrl( EncoderNo, Speed, Smooth )`	
Parameters	`EncoderNo`	Like **set_encoder_speed**.
	`Speed`	Like **set_encoder_speed**.
	`Smooth`	Like **set_encoder_speed**.
Multi-board Com'd Name	**n_set_encoder_speed_ctrl**	
Comments	• **set_encoder_speed_ctrl** is not executed (**get_last_error** return code `RTC6_PARAM_ERROR`), if: – `Speed` ≤ 0.0 – `Smooth` < 0.0 • **set_encoder_speed_ctrl** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set • **set_encoder_speed_ctrl** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) – the INTERNAL-BUSY list execution status is set	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_encoder_speed**, **set_auto_laser_control**	

Normal List Command	**set_end_of_list**
Function	Ends execution of a list.
Call	`set_end_of_list()`
Parameters	None.
Result	None.
Multi-board Com'd Name	**n_set_end_of_list**
Comments	• If, during processing of a list, the **set_end_of_list** is encountered and *no* automatic list change has been previously activated, see Chapter 6.4.6 "Changing Lists Automatically", page 113, then list execution ends. The Signals for "Laser Active" Operation are then switched off and a home jump, if defined by **home_position** or **home_position_xyz**, is executed (the INTERNAL-BUSY list execution status is set while the home jump is executed). • In contrast, upon reaching a **set_end_of_list**, execution continues at the other list if an automatic list change has been previously *activated*. The other list can also be "List 1" if "List 2" has not been configured (`Mem2` = 0, see **config_list**). • Upon processing of the **set_end_of_list**, the USED list status of the respective list (USED1 or USED2) is always set and the BUSY list status (BUSY1 or BUSY2) of the list is reset, see also Chapter 6.4.3 "List Execution Status", page 111). The BUSY list execution status, on the other hand, is only reset if *no* automatic list change has been previously activated. • An automatic list change of the input pointer never occurs during loading of **set_end_of_list** (in contrast to an automatic list change of the *output pointer* during execution of **set_end_of_list**, if previously activated by **auto_change_pos** or **start_loop**). • Upon loading **set_end_of_list**, the READY list status (READY1 or READY2) is set and LOAD list status (LOAD1 or LOAD2) is reset. Additionally, flushing of the buffered list input is triggered, see Chapter 6.4.1 "Loading Lists", page 108. • **set_end_of_list** is ignored during loading and execution, that is, replaced with a **list_nop** if an indexed subroutine is currently being loaded or executed (**get_last_error** return code `RTC6_IGNORED`).
RTC4→RTC6	Basically unchanged functionality. However: Additional USED list status.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_start_list**

Ctrl Command	set_eth_boot_control
Function	Standalone Functionality: Activates or deactivates automatic booting.
Prerequisite	RTC6 Software Package ≥ V1.7.0 and BIOS-ETH ≥ 26.
Call	`set_eth_boot_control( Ctrl )`
Parameters	`Ctrl` = 0: Deactivates automatic booting. > 0: Activates automatic booting. As an unsigned 32-bit value.
Result	None.
Multi-board Com'd Name	**n_set_eth_boot_control**
Comments	• **set_eth_boot_control** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – a list has been paused by **set_wait** (PAUSED list execution status set) • During the execution of **set_eth_boot_control** the 10 µs clock cycle of the DSP is interrupted for several ms. • **set_eth_boot_control** is only allowed with RTC6 Ethernet Boards. Otherwise, a **get_last_error** return code `RTC6_TYPE_REJECTED` is generated. • See Chapter 16.7 "Standalone Functionality", page 988. • If **set_eth_boot_control** has not been successful, Bit 27 is set, see **eth_get_last_error** and **eth_get_error**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 618, OUT 618, RBF 623.
References	**eth_boot_dcmd**

Ctrl Command	set_extstartpos
Function	Defines the start address (within the RTC6 List Memory area of "List 1" or "List 2") where execution should continue upon External Starts.
Call	`set_extstartpos( Pos )`
Parameters	`Pos` Absolute address of the first list command to be executed. As an unsigned 32-bit value. Allowed value range: [0…(2^{23}–1)].
Multi-board Com'd Name	**n_set_extstartpos**
Comments	• By default, an External Start results in a continuation or start at the beginning of "List 1" (`Pos` = 0). • `Pos` must be within the range of "List 1" or "List 2". Otherwise, `Pos` = 0 is set. • The specified start address is used for all External Starts until a new address is specified by **set_extstartpos** or **set_extstartpos_list**. An address range validity check is performed on `Pos` before each External Start; `Pos` might therefore become set to 0 at a later point (and remain at this value) if the configuration has been correspondingly changed in the meantime (see **config_list**). • See also Section "External Start", page 312. • **set_extstartpos** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_extstartpos_list**, **set_control_mode**

Undelayed Short List Command	set_extstartpos_list
Function	Like **set_extstartpos**, but a list command.
Call	`set_extstartpos_list( Pos )`
Parameters	`Pos` Like **set_extstartpos**.
Multi-board Com'd Name	**n_set_extstartpos_list**
Comments	• **set_extstartpos_list** can be used within a list, to "link" it to the list that follows. • See **set_extstartpos**.
RTC4→RTC6	Unchanged functionality. In addition: increased value range.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_extstartpos**

Ctrl Command	set_ext_start_delay
Function	Sets a track delay for External Starts, so that the lists are started with a delay relative to the triggering input signal or **simulate_ext_start** or **simulate_ext_start_ctrl** command.
Call	`set_ext_start_delay( Delay, EncoderNo )`
Parameters	`Delay` Track delay (counter steps of the selected encoder counter `EncoderNo`). As a signed 32-bit value. Allowed value range: [-2^{31}…$+(2^{31}-1)$]. `EncoderNo` Number of the to-be-used encoder counter. As an unsigned 32-bit value. Allowed values: = 0: Encoder counter “Encoder0”. = 1: Encoder counter “Encoder1”.
Multi-board Com’d Name	**n_set_ext_start_delay**
Comments	• For External Starts, see Section ”External Start”, page 312. • The track delay is specified in *relative* counting units of the selected encoder counter (the RTC6’s encoder counter is triggered by an external or simulated encoder signal, see Chapter 9.3.3 ”Synchronization by Encoder Signals”, page 320). • Ensure that the sign of the track delay (`Delay` parameter) is appropriate for the selected encoder’s counting direction (for external triggering, this corresponds to the workpiece’s direction of motion and is always positive with simulated encoders). • For `Delay` = 0, the track delay is deactivated. • If a track delay is specified, that causes a start trigger (initiated by **simulate_ext_start** or an external start signal) to occur when the BUSY list execution status or INTERNAL-BUSY list execution status is set (for example, when outputting a list or during **goto_xy**), then no starts are get triggered by this start trigger (in this case, Bit #11 of the **get_startstop_info** return value is set). • Track delays can also be set with **simulate_ext_start**. Track delays are deactivated by initialization (with **load_program_file**), by an External Stop and by **stop_execution**. They can also be deactivated with **set_control_mode**/**set_control_mode_list** (Bit #2). • **set_ext_start_delay** cancels already externally triggered starts that have not yet executed and are still being held in a queue that accommodates up to 8 starts. • If `EncoderNo` > 1, then **set_ext_start_delay** is ignored (**get_last_error** return code `RTC6_PARAM_ERROR`).

Ctrl Command	set_ext_start_delay
RTC4→RTC6	Unchanged functionality. In addition: increased value range. The RTC5 allows this command to be used not only together with an external encoder, but also with an encoder simulation (see Section "Encoder Simulation", page 321) started by **simulate_encoder**. In RTC4 Compatibility Mode, the RTC5 multiplies the specified value for `Delay` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**simulate_ext_start**, **set_ext_start_delay_list**, **set_extstartpos**, **set_extstartpos_list**, **set_control_mode**, **set_control_mode_list**

Normal List Command	**set_ext_start_delay_list**
Function	Like **set_ext_start_delay**, but a list command.
Call	`set_ext_start_delay_list( Delay, EncoderNo )`
Parameters	`Delay` Like **set_ext_start_delay**. `EncoderNo` Like **set_ext_start_delay**.
Multi-board Com'd Name	**n_set_ext_start_delay_list**
Comments	• If `EncoderNo` > 1, then **set_ext_start_delay_list** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).
RTC4→RTC6	See **set_ext_start_delay**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_ext_start_delay**

Ctrl Command	set_firstpulse_killer
Function	Sets the length of the FirstPulseKiller signal.
Call	`set_firstpulse_killer( Length )`
Parameters	`Length` Length of the FirstPulseKiller signal. As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: $[0 \ldots +(2^{26}–1)]$. Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_set_firstpulse_killer**
Comments	• The clock frequency connected to the FirstPulseKiller signal is 64 MHz. 1 bit equals 1/64 μs. • The signal level is set by **set_laser_control**. • For YAG Mode 2, the Q-Switch delay is also correspondingly altered, see Section "YAG Mode Differences", page 197. • In CO_2 Mode, **set_firstpulse_killer** has no effect. • The laser mode is set by **set_laser_mode**, see Chapter 7.4 "Laser Control", page 189. • Q-Switch pulse length and Q-Switch period can be set by **set_laser_pulses_ctrl**, **set_laser_pulses** or **set_laser_timing**.
RTC4→RTC6	Basically unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value by 8. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_laser_mode**, **set_laser_timing**, **config_laser_signals**

Undelayed Short List Command	set_firstpulse_killer_list
Function	Like **set_firstpulse_killer**, but a list command.
Call	`set_firstpulse_killer_list( Length )`
Parameters	`Length` Like **set_firstpulse_killer**.
Multi-board Com'd Name	**n_set_firstpulse_killer_list**
Comments	• –
RTC4→RTC6	As **set_firstpulse_killer**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_firstpulse_killer**

Normal List Command	**set_fly_1_axis**
Function	"Fly Extension" Command: Activates a 1 Axis-Processing-on-the-fly application.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_fly_1_axis** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_fly_1_axis( Axis, Mode, Scale )`
Parameters	`Axis` Axis from Table 3, page 268. As an unsigned 32-bit value. `Mode` Mode from Table 4, page 270. As an unsigned 32-bit value. `Scale` Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: • 1/256 ≤ |`Scale`| ≤ 16.000,0 with linear axis (`1`, `2` or `3`) • |`Scale`| > 100,0 with Rotary axis (`4`) `Scale` can be + or –. Only the absolute value is restricted. `Scale` is ignored for `Mode` = (8/12/16).
Multi-board Com'd Name	**n_set_fly_1_axis**
Comments	• Being an "Fly Extension" Command, **set_fly_1_axis** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • With `Mode`(1…4), **set_fly_1_axis** requires two 10 µs clock cycles for execution. • **set_fly_1_axis** overwrites a previous definition for the same Axis. As a linear axis, **set_fly_1_axis** can be combined with other linear axes, but not with a Rotary axis. Any still-active rotation-fly application (`set_fly_1_axis( Axis =4 )`) is automatically deactivated and vice versa. It is recommended to explicitly switch off incompatible Processing-on-the-fly corrections beforehand, for example, see **fly_return_1_axis** and **fly_return_2_axes**. • With an unallowed parameter value, **set_fly_1_axis** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • The following command calls are executed in the same way: – `set_fly_1_axis( 1, 1, ScaleX )` = `set_fly_x( ScaleX )` – `set_fly_1_axis( 2, 2, ScaleY )` = `set_fly_y( ScaleY )` – `set_fly_1_axis( 3, EncoderNo+3, ScaleZ )` = `set_fly_z( ScaleZ, EncoderNo )` – `set_fly_1_axis( 4, 1, Resolution )` = `set_fly_rot( Resolution )` – `set_fly_1_axis( 1, 6, ScaleX )` = `set_fly_x_pos( ScaleX )` – `set_fly_1_axis( 2, 6, ScaleY )` = `set_fly_y_pos( ScaleY )` – `set_fly_1_axis( 4, 6, Resolution )` = `set_fly_rot_pos( Resolution )` – `set_fly_1_axis( 1, 7, Scale )` corresponds to `set_mcbsp_in( 1, Scale )`
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**set_fly_2_axes**, **set_fly_3_axes**

Normal List Command	**set_fly_2_axes**
Function	"Fly Extension" Command: Activates a 2-Axes-Processing-on-the-fly application.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_fly_2_axes** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_fly_2_axes( Axis1, Mode1, Scale1, Axis2, Mode2, Scale2 )`
Parameters	`Axis1` Axis from Table 3, page 268. Allowed values: `1`…`3` (linear axis only). As an unsigned 32-bit value. `Mode1` Mode from Table 4, page 270. As an unsigned 32-bit value. `Scale1` Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: • 1/256 ≤ |`Scale1`| ≤ 16.000,0 with linear axis (`1`, `2` or `3`) `Scale1` can be + or –. Only the absolute value is restricted. `Scale1` is ignored for `Mode1` = (8/12/16). `Axis2` Like `Axis1`. `Mode2` Like `Mode1`. `Scale2` Like `Scale1`.
Multi-board Com'd Name	**n_set_fly_2_axes**
Comments	• Being an "Fly Extension" Command, **set_fly_2_axes** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • With `Mode`(1…4), **set_fly_2_axes** requires two 10 µs clock cycles for execution. • **set_fly_2_axes** overwrites a previous definition for the same Axes (even individually). As 2 linear axes, **set_fly_2_axes** can be combined with a third linear axis, but not with a Rotary axis. Any still-active rotation-fly application (`set_fly_1_axis( Axis =4 `)) is automatically deactivated and vice versa. It is recommended to explicitly switch off incompatible Processing-on-the-fly corrections beforehand, for example, see **fly_return_2_axes**. • `Axis1` and `Axis2` need to be different linear axes and must not be rotary axes. • `Mode1` and `Mode2` can be the same. If they use the same Encoder, a different scaling factor can be specified by `Mode1`/`Mode2` = 1 / 3 or 2 / 4. • With an unallowed parameter value, **set_fly_2_axes** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).

Normal List Command	**set_fly_2_axes**

Comments (cont'd)	• The following command calls are executed in the same way:
	– `set_fly_2_axes( 1, 1, ScaleX, 2, 2, ScaleY ) = set_fly_2d( ScaleX, ScaleY )`
	– `set_fly_2_axes( 1, 1, ScaleX, 2, 2, ScaleY ) =`
	`{`
	`set_fly_1_axis( 1, 1, ScaleX );`
	`set_fly_1_axis( 2, 2, ScaleY ));`
	`}`
	Encoder resets occur:
	• With a single **set_fly_2_axes** call in the same 10 µs clock cycle
	• With two **set_fly_1_axis** calls in two different 10 µs clock cycles
	– `set_fly_2_axes( 1, 11, Scale, 2, 15, Scale)` corresponds to
	`set_mcbsp_in_list( 3, Scale )`

RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**set_fly_1_axis**, **set_fly_3_axes**

Normal List Command	set_fly_2d	
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece-motion in 2 dimensions (based on the encoder values transferred to the RTC6 by encoder counters "Encoder0" and "Encoder1"). Sets the corresponding scaling factors.	
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_fly_2d** terminates the Processing-on-the-fly process (even though it could not have been activated).	
Call	`set_fly_2d( ScaleX, ScaleY )`	
Parameters	`ScaleX`	Scaling factor for the x direction (encoder counter "Encoder0"). In bits/count. As a 64-bit IEEE floating point value. Allowed value range: 1/256 ≤ |`ScaleX`| ≤ 16000.0.
	`ScaleY`	Scaling factor for the y direction (encoder counter "Encoder1"). In bits/count. As a 64-bit IEEE floating point value. Allowed value range: 1/256 ≤ |`ScaleY`| ≤ 16000.0.
Multi-board Com'd Name	**n_set_fly_2d**	
Comments	• `ScaleX` and `ScaleY` can be negative depending on the motion direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction (for example, determination of the scaling factor or deactivating Processing-on-the-fly correction), see the Chapter 8.6 "Processing-on-the-fly", page 251. For **set_fly_2d** usage, see the Chapter 8.6.4 "Compensating 2D Motions", page 258. • If unallowed parameter values are supplied (for example, for `ScaleX` = 0), then **set_fly_2d** does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by **set_fly_2d** (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed Vector Command or "Arc" Command (without "**set_fly_2d**" Processing-on-the-fly correction). However, Processing-on-the-fly correction successfully activated by **set_fly_2d** switches off any other Processing-on-the-fly correction and does itself get switched off by any other Processing-on-the-fly command, even if that other command contains unallowed parameters, see Section "Overview", page 251. • If an encoder compensation has been set by **load_fly_2d_table** and **init_fly_2d**, the current encoder values are added to the latest reference values of the 2D encoder compensation and then the sums are saved as new reference values. The encoder counters are then reset to 0. • It should *not* be set by **set_control_mode**(Bit #9) that the encoder counters are only reset after the subsequent External Start trigger. Otherwise, the reference values of 2D encoder compensation are lost. **set_fly_2d** always resets the encoder counters, even if they have already been reset by an external start trigger. See also Section "2D Encoder Compensation for xy Positioning Stages", page 258.	

Normal List Command	set_fly_2d
Comments (cont'd)	• Do not intermediately call set_fly_x or set_fly_y to switch on the Processing-on-the-fly application if you intend to use **set_fly_2d** in conjunction with 2D encoder compensation for an xy positioning stage, because here too the reference values are lost. • If no correction table for 2D encoder compensation has yet been loaded onto the board (see load_fly_2d_table), then the encoder values are used without correction. • You can also use activate_fly_2d/activate_fly_2d_encoder to switch on **set_fly_2d** Processing-on-the-fly correction.
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `ScaleX` and `ScaleY` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	init_fly_2d, load_fly_2d_table, get_fly_2d_offset, activate_fly_2d, activate_fly_xy

Multiple List Command	set_fly_3_axes
Function	"Fly Extension" Command: Activates a 3-Axes-Processing-on-the-fly application.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_fly_3_axes** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_fly_3_axes( ModeX, ScaleX, ModeY, ScaleY, ModeZ, ScaleZ )`
Parameters	`ModeX` Mode from Table 4, page 270. As an unsigned 32-bit value. `ScaleX` Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: • 1/256 ≤ |`ScaleX`| ≤ 16.000,0 with linear axis (1, 2 or 3) `ScaleX` can be + or –. Only the absolute value is restricted. `ScaleX` is ignored for `ModeX` = (8/12/16). `ModeY` Like `ModeX`. `ScaleY` Like `ScaleX`. `ModeZ` Like `ModeX`. `ScaleZ` Like `ScaleX`.
Multi-board Com'd Name	**n_set_fly_3_axes**
Comments	• Being an "Fly Extension" Command, **set_fly_2_axes** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • With `Mode`(1…4), **set_fly_3_axes** requires two 10 µs clock cycles for execution. • **set_fly_3_axes** occupies 2 RTC6 List Memory positions. The first part is executed as an Undelayed Short List Command prior to the second part, which executes as a Normal List Command. Any pending Delayed Short List Commands are executed beforehand. • **set_fly_3_axes** overwrites a previous definition for the same Axes (even individually). Any still-active rotation-fly application (`set_fly_1_axis( Axis =4`)) is automatically deactivated and vice versa. It is recommended to explicitly switch off incompatible Processing-on-the-fly corrections beforehand, for example, see **fly_return_3_axes**. • The Axes are automatically set to 1, 2, 3 and therefore, do not need to be specified explicitly. • `ModeX, ModeY, ModeZ` can be same. If they use the same Encoder, a different scaling factor can be specified by `ModeX`/`ModeY`/`ModeZ` = 1 / 3 or 2 / 4. • With an unallowed parameter value, **set_fly_3_axes** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).

Multiple List Command	set_fly_3_axes
Comments (cont'd)	• The following command calls are executed in the same way: – `set_fly_3_axes( 1, ScaleX, 2, ScaleY, 3 or 4, ScaleZ ) = { set_fly_1_axis( 1, 1, Scale X ); set_fly_1_axis( 2, 2, Scale Y ); set_fly_1_axis( 3, 3 or 4, Scale Z ); }` – `set_fly_3_axes( 1, ScaleX, 2, ScaleY, 3 or 4, ScaleZ ) = { set_fly_2_axes( 1, 1, Scale1 X, 2, 2, Scale2 X ); set_fly_1_axis( 3, 3 or 4, Scale Z ); }` Encoder resets occur: • With a single **set_fly_3_axes** call in the same 10 µs clock cycle • With three **set_fly_1_axis** calls in three different 10 µs clock cycles • With one **set_fly_2_axes** call and one **set_fly_1_axis** call in two different 10 µs clock cycles – `set_fly_3_axes( 8, 1, 0, 12, 1, 0, 16, 1, 0 )` corresponds to `set_multi_mcbsp_in_list( Ctrl, P, Mode )`
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**set_fly_1_axis**, **set_fly_2_axes**

Undelayed Short List Command	**set_fly_limits**
Function	Defines the boundaries of the customer-defined monitoring area for Processing-on-the-fly applications.
Call	`set_fly_limits( Xmin, Xmax, Ymin, Ymax )`
Parameters	`Xmin` Range boundary. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
	`Xmax` Like `Xmin` (analogously).
	`Ymin` Like `Xmin` (analogously).
	`Ymax` Like `Xmin` (analogously).
Multi-board Com'd Name	**n_set_fly_limits**
Comments	• For usage of **set_fly_limits**, see Section "Customer-Defined Monitoring Area", page 265. • During initialization (with **load_program_file**), the boundary limits get set to the following default values: – `Xmin` = `Ymin` = –524,288 – `Xmax` = `Ymax` = +524,287 • Range boundaries specified using the parameter value 0 are set to the above-mentioned default values.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_marking_info**

Undelayed Short List Command	**set_fly_limits_z**
Function	Defines the boundaries of the customer-defined monitoring range for z axis Processing-on-the-fly applications.
Call	`set_fly_limits_z( Zmin, Zmax )`
Parameters	`Zmin` Range boundary. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
	`Zmax` Like `Zmin` (analogously).
Multi-board Com'd Name	**n_set_fly_limits_z**
Comments	• For usage of **set_fly_limits_z**, see also Chapter 8.6.9 "Monitoring Processing-on-the-fly Corrections", page 264. • During initialization (with **load_program_file**), the boundary limits get set to the following default values: – `Zmin` = –524,288 – `Zmax` = +524,287 • Boundary limits specified using the parameter value 0 also get set to the above-mentioned default values.
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `Zmin` and `Zmax` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the specified values for `Zmin` and `Zmax` by 16. The allowed value range decreases accordingly.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_fly_limits**, **get_marking_info**

Normal List Command	**set_fly_rot**
Function	Activates Processing-on-the-fly correction for compensation of workpiece rotary motion (based on angular position values transferred to the RTC6 by encoder counter "Encoder0") and sets the corresponding `Resolution` parameter.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_fly_rot** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_fly_rot( Resolution )`
Parameters	`Resolution` Number of steps per revolution. As a 64-bit IEEE floating point value. Allowed value range: |`Resolution`| > 100.0.
Multi-board Com'd Name	**n_set_fly_rot**
Comments	• `Resolution` can be negative depending on the rotation direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determining the `Resolution` parameter, see Chapter 8.6 "Processing-on-the-fly", page 251. • Before executing **set_fly_rot**, you should define the rotation center for Processing-on-the-fly correction by **set_rot_center** or **set_rot_center_list**. Otherwise, the default value (0|0) is applied. • If unallowed parameter values are supplied (for example, for `Resolution` = 0), then **set_fly_rot** does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by **set_fly_rot** (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed Vector Command or "Arc" Command (without "**set_fly_rot**" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section "Overview", page 251. • For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 260. • By **set_control_mode**(Bit #9), it can be set in advance when the encoder counter "Encoder0" is reset by **set_fly_rot**.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_rot_center**, **set_rot_center_list**, **fly_return**, **get_encoder**, **set_fly_rot_pos**

Normal List Command	set_fly_rot_pos
Function	Activates Processing-on-the-fly correction for compensation of workpiece or scan system rotary motion (based on angular position values transferred to the RTC6 by the McBSP interface). It thereby sets the corresponding `Resolution` parameter.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_fly_rot_pos** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_fly_rot_pos( Resolution )`
Parameters	`Resolution` Number of steps per revolution. As a 64-bit IEEE floating point value. Allowed value range: |`Resolution`| > 100.0.
Multi-board Com'd Name	**n_set_fly_rot_pos**
Comments	• `Resolution` can be negative depending on the rotation direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determining the `Resolution` parameter, see Chapter 8.6 "Processing-on-the-fly", page 251. • Before executing **set_fly_rot_pos**, you should define the rotation center for Processing-on-the-fly correction by **set_rot_center** or **set_rot_center_list**. Otherwise, the default value (0|0) is used. • If unallowed parameter values are supplied (for example, for `Resolution` = 0), then **set_fly_rot_pos** does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by **set_fly_rot_pos** (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed Vector Command or "Arc" Command (without "**set_fly_rot_pos**" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section "Overview", page 251. • For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 260. • The McBSP interface cannot be simultaneously used for both Processing-on-the-fly applications and Online Positioning. See also Section "Notes", page 239. • The McBSP interface ignores the first FrameSync signal after a **load_program_file** or **mcbsp_init**. That is, data provided is not transmitted, see page 89.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_rot_center**, **set_rot_center_list**, **fly_return**, **read_mcbsp**, **set_fly_rot**

Ctrl Command	set_fly_tracking_error
Function	Category: Control Commands for a Restricted User Group only. Switches on (off) Tracking Error compensation of encoder values for Processing-on-the-fly Applications.
Call	`set_fly_tracking_error( TrackingErrorX, TrackingErrorY )`
Parameters	`TrackingErrorX` Tracking Error for the x axis. In units of 10 µs. As a signed 32-bit value. Allowed value range: [0…4,095]. Bits with higher significance are ignored.
	`TrackingErrorY` Tracking Error for the y axis. Otherwise, like `TrackingErrorX`.
Multi-board Com'd Name	**n_set_fly_tracking_error**

Comments	• If SCANahead Functionality is active, **set_fly_tracking_error** is not executed (**get_last_error**-return code `RTC6_REJECTED`).
	• `TrackingErrorX` and `TrackingErrorY` are additive to **set_controlpreview_compensation_ctrl**(`ControlPreview`).
	Note their different units:
	– `TrackingErrorX` [10 µs]
	– `TrackingErrorY` [10 µs]
	– `ControlPreview` [1/64 µs]
	• The total prediction values are converted into [10/16 µs]:
	– With **set_fly_tracking_error**
	– With **set_controlpreview_compensation_ctrl**
	As per:
	– `PredictionX` = (`TrackingErrorX` × 16) + (`ControlPreview` / 40)
	– `PredictionY` = (`TrackingErrorY` × 16) + (`ControlPreview` / 40)
	The result must be ≤ 65,535 in *each case*.
	• Alternatively, the total prediction values can also be set explicitly by **fly_prediction**.
	• Always the total prediction values of the last command executed are exclusively effective:
	– For **set_fly_tracking_error**
	`TrackingErrorX`/`TrackingErrorY` + `ControlPreview`
	– For **set_controlpreview_compensation_ctrl**
	`TrackingErrorX`/`TrackingErrorY` + `ControlPreview`
	– For **fly_prediction**
	`PredictionX`/`PredictionY`
	• See Chapter 8.6.10 "Tracking Error Compensation of Encoder Values for Processing-on-the-fly Applications", page 266.
	• **set_fly_tracking_error** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
	• The following applies for Control Commands for a Restricted User Group only: These users have been informed separately in how to handle these RTC6 commands. Therefore, only the command table is available in this manual and further information is omitted.

Ctrl Command	set_fly_tracking_error
RTC4→RTC6	New command.
RTC5→RTC6	Re-implementation: previous functionality is reproduced, but behavior is different in detail. In addition: decreased value range.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 646, OUT 649, RBF 641: functionality implemented (previously: no effect at all).
References	**set_controlpreview_compensation_ctrl**, **fly_prediction**

Normal List Command	**set_fly_x**
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece-motion in the x direction (based on position values transferred to the RTC6 by encoder counter “Encoder0”) and sets the corresponding scaling factor.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_fly_x** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_fly_x( ScaleX )`
Parameters	`ScaleX` Scaling factor for the x direction (encoder counter “Encoder0”). In bits/count. As a 64-bit IEEE floating point value. Allowed value range: 1/256 ≤ |`ScaleX`| ≤ 16000.0.
Multi-board Com’d Name	**n_set_fly_x**
Comments	• `ScaleX` can be negative depending on the motion direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determination of the scaling factor, see Chapter 8.6 “Processing-on-the-fly”, page 251. • If unallowed parameter values are supplied (for example, for `ScaleX` = 0), then **set_fly_x** does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by **set_fly_x** (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed Vector Command or “Arc” Command (without “**set_fly_x**“ Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section “Overview”, page 251. • For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 “Deactivating Processing-on-the-fly Correction”, page 260. • By **set_control_mode**(Bit #9), it can be set in advance when the encoder counter “Encoder0” is reset by **set_fly_x**. • You can also switch on **set_fly_x**/**set_fly_y** Processing-on-the-fly correction by **activate_fly_xy**/**activate_fly_xy_encoder**.
RTC4→RTC6	Unchanged functionality. In addition: changed value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `ScaleX` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**fly_return**, **set_fly_y**, **get_encoder**, **set_fly_x_pos**, **set_fly_y_pos**, **activate_fly_xy**, **set_fly_2d**

Normal List Command	**set_fly_x_pos**
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece or scan system motion in the x direction (based on position values transferred to the RTC6 by the McBSP interface); thereby sets the corresponding scaling factor.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_fly_x_pos** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_fly_x_pos( ScaleX )`
Parameters	`ScaleX` Scaling factor for the x direction. In (RTC6)*bits*/(McBSP)*bit*. As a 64-bit IEEE floating point value. Allowed value range: 1/256 ≤ |`ScaleX`| ≤ 16000.0.
Multi-board Com'd Name	**n_set_fly_x_pos**
Comments	• `ScaleX` can be negative depending on the motion direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determination of the scaling factor, see Chapter 8.6 "Processing-on-the-fly", page 251. • If unallowed parameter values are supplied (for example, for `ScaleX` = 0), then **set_fly_x_pos** does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by **set_fly_x_pos** (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed Vector Command or "Arc" Command (without "**set_fly_x_pos**" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section "Overview", page 251. • For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 260. • For 1D correction (when only **set_fly_x_pos** is used), the McBSP interface provides a (signed) 32-bit value. In contrast, only a (signed) 16-bit value per axis is supplied for 2D correction (**set_fly_x_pos** and **set_fly_x_pos**). Here, **set_fly_x_pos** uses the McBSP interface's lower 16 bits for the x value and **set_fly_x_pos** uses its upper 16-bits for the y value. • The McBSP interface cannot be simultaneously used for both Processing-on-the-fly applications and Online Positioning. See also Section "Notes", page 239. • The McBSP interface ignores the first FrameSync signal after a **load_program_file** or **mcbsp_init**. That is, data provided is not transmitted, see page 89.

Normal List Command	set_fly_x_pos
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `ScaleX` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**fly_return**, **set_fly_y_pos**, **read_mcbsp**, **set_fly_x**, **set_fly_y**

Normal List Command	**set_fly_y**
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece-motion in the Y direction (based on position values transferred to the RTC6 by encoder counter "Encoder1") and sets the corresponding scaling factor.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_fly_y** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_fly_y( ScaleY )`
Parameters	`ScaleY` Scaling factor for the y direction (encoder counter "Encoder1"). In bits/count. As a 64-bit IEEE floating point value. Allowed value range: 1/256 ≤ |`ScaleY`| ≤ 16000.0.
Multi-board Com'd Name	**n_set_fly_y**
Comments	• `ScaleY` can be negative depending on the motion direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determination of the scaling factor, see Chapter 8.6 "Processing-on-the-fly", page 251. • If unallowed parameter values are supplied (for example, for `ScaleY` = 0), then **set_fly_y** does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by **set_fly_y** (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed Vector Command or "Arc" Command (without "**set_fly_y**" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section "Overview", page 251. • For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 260. • By **set_control_mode**(Bit #9), it can be set in advance when the encoder counter "Encoder0" is reset by **set_fly_y**. • You can also switch on **set_fly_x**/**set_fly_y** Processing-on-the-fly correction by **activate_fly_xy**/**activate_fly_xy_encoder**.
RTC4→RTC6	Unchanged functionality. In addition: changed value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `ScaleY` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**fly_return**, **set_fly_x**, **get_encoder**, **set_fly_x_pos**, **set_fly_y_pos**, **activate_fly_xy**, **set_fly_2d**

Normal List Command	set_fly_y_pos
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece or scan system motion in the Y direction (based on position values transferred to the RTC6 by the McBSP interface); thereby sets the corresponding scaling factor.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_fly_y_pos** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_fly_y_pos( ScaleY )`
Parameters	`ScaleY` Scaling factor for the y direction in *(RTC6)bits/(McBSP)bit*. As a 64-bit IEEE floating point value. Allowed value range: 1/256 ≤ |`ScaleY`| ≤ 16000.0.
Multi-board Com'd Name	**n_set_fly_y_pos**
Comments	• `ScaleY` can be negative depending on the motion direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determination of the scaling factor, see Chapter 8.6 "Processing-on-the-fly", page 251. • If unallowed parameter values are supplied (for example, for `ScaleY` = 0), then **set_fly_y_pos** does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by **set_fly_y_pos** (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed Vector Command or "Arc" Command (without "**set_fly_y_pos**" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section "Overview", page 251. • For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 260. • For 1D correction (when only **set_fly_y_pos** is used), the McBSP interface provides a (signed) 32-bit value. In contrast, only a (signed) 16-bit value per axis is supplied for 2D correction (**set_fly_x_pos** and **set_fly_y_pos**). Here, **set_fly_x_pos** uses the McBSP interface's lower 16 bits for the x value and **set_fly_y_pos** uses its upper 16-bits for the y value. • The McBSP interface cannot be simultaneously used for both Processing-on-the-fly applications and Online Positioning. See also Section "Notes", page 239. • The McBSP interface ignores the first FrameSync signal after a **load_program_file** or **mcbsp_init**. That is, data provided is not transmitted, see page 89.
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `ScaleY` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**fly_return**, **set_fly_x_pos**, **read_mcbsp**, **set_fly_x**, **set_fly_y**

Normal List Command	set_fly_z
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece-motion in the z direction (based on position values transferred to the RTC6 by the specified encoder counter) and sets the corresponding scaling factor.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_fly_z** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_fly_z( ScaleZ, EncoderNo )`
Parameters	`ScaleZ` Scaling factor for the z direction. In bits/count. As a 64-bit IEEE floating point value. Allowed value range: 1/256 ≤ |`ScaleZ`| ≤ 16,000.0. `EncoderNo` Number of the to-be-used encoder counter. As an unsigned 32-bit value. Allowed values: = 0: Encoder counter “Encoder0”. = 1: Encoder counter “Encoder1”.
Multi-board Com’d Name	**n_set_fly_z**
Comments	• `ScaleZ` can be negative depending on the motion direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determination of the scaling factor, see Chapter 8.6 ”Processing-on-the-fly”, page 251. • If unallowed `ScaleZ` parameter values are supplied (for example, for `ScaleZ` = 0), then **set_fly_z** does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by **set_fly_z** (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed Vector Command or “Arc” Command (without “**set_fly_z**” Processing-on-the-fly correction). • For deactivating Processing-on-the-fly correction, see Section ”Notes on Usage”, page 267. • By **set_control_mode**(Bit #9), it can be set in advance when the encoder counter “Encoder0” or “Encoder1” is reset by **set_fly_z**. • If `EncoderNo` > 1, **set_fly_z** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `ScaleZ` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**fly_return_z**

Ctrl Command	set_free_variable	
Function	Sets a free variable to the desired value.	
Call	`set_free_variable( No, Value )`	
Parameters	`No`	Number of the free variable to be set. As an unsigned 32-bit value. Allowed value range: [0…7]. Only the 3 least significant bits are evaluated.
	`Value`	Desired variable value. As an unsigned 32-bit value. Allowed value range: [0…(2^{32}–1)].
Multi-board Com'd Name	**n_set_free_variable**	
Comments	• See Chapter 6.9.1 "Free Variables", page 138. • Standalone Functionality: **set_free_variable** is a control command allowed for automatic booting (although the corresponding list command **set_free_variable_list** exists), see Table 8, page 993 in Chapter 16.7 "Standalone Functionality", page 988. By **set_free_variable**, any version code can be assigned to the Boot Image, which in *PC operation* can be read and checked before a /START. • **set_free_variable** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 618: control command allowed for automatic booting, see page 992.	
References	**set_free_variable_list**, **get_free_variable**, **set_trigger**, **eth_boot_dcmd**	

Undelayed Short List Command	set_free_variable_list	
Function	Like **set_free_variable**, but a list command.	
Call	`set_free_variable_list( No, Value )`	
Parameters	`No`	Like **set_free_variable**.
	`Value`	Like **set_free_variable**.
Multi-board Com'd Name	**n_set_free_variable_list**	
Comments	• See **set_free_variable**.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_free_variable**	

Ctrl Command	set_hi
Function	Defines gain values and offset values for the galvanometer scanners of the scan system attached to the specified scan head connector.
Call	`set_hi( HeadNo, GalvoGainX, GalvoGainY, GalvoOffsetX, GalvoOffsetY )`
Parameters	`HeadNo` Number of the scan head connector. As an unsigned 32-bit value. Allowed values: = 1: Connector for First Scan Head. = 2: Connector for Second Scan Head.
	`GalvoGainX` x gain value. As a 64-bit IEEE floating point value. Allowed value range: [0.01…100].
	`GalvoGainY` Like `GalvoGainX` (analogously).
	`GalvoOffsetX` x offset value. In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287].
	`GalvoOffsetY` Like `GalvoOffsetX` (analogously).
Multi-board Com'd Name	**n_set_hi**
Comments	• For usage of **set_hi**, see Section "Customer-Specific Calibration", page 287. • The specified gain values and offset values overwrite the values that were set by **auto_cal** and can themselves be overwritten by a subsequent call of **auto_cal**. • With changed gain values and offset values, the transition is automatically performed at the predefined jump speed (see **set_jump_speed**). • **set_hi** is not executed, if: – A parameter value is invalid (**get_last_error** return code `RTC6_PARAM_ERROR`) – the BUSY list execution status is set (**get_last_error** return code `RTC6_BUSY`) – the INTERNAL-BUSY list execution status is set (**get_last_error** return code `RTC6_BUSY`) • **set_hi** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • For `RTC6_PARAM_ERROR`, the BUSY list execution status is not checked. Therefore the return codes `RTC6_BUSY` and `RTC6_PARAM_ERROR` do not occur simultaneously. • If the Option "Second Scan Head Control" has not been enabled, values specified for the Connector for Second Scan Head have no effect. • **set_hi** is allowed in Boot Phase 2, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**auto_cal**, **get_hi_pos**, **write_hi_pos**

Ctrl Command	set_input_pointer
Function	Opens the RTC6 List Memory for writing with list commands and sets the input pointer to the specified *absolute* address in RTC6 List Memory ("List 1" or "List 2").
Call	`set_input_pointer( Pos )`
Parameters	`Pos` Position (absolute memory address) of the input pointer. [0…(2^{23}–1)]. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_input_pointer**
Comments	• The next list command is stored at the specified address and all further list commands at the subsequent addresses in the selected list. • **set_input_pointer** performs basically like **set_start_list_pos** (see comments there). But **set_input_pointer** sets the input pointer based on a specified *absolute* memory address, whereas **set_start_list_pos** uses a specified list number and a *relative* memory address. • For `Pos` ≥ `Mem1` + `Mem2` (see **config_list**), `Pos` is set to 0.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_start_list_pos**, **get_input_pointer**

Undelayed Short List Command	**set_io_cond_list**
Function	Sets the bits of the 16-Bit Digital Output Port of the EXTENSION 1 Socket Connector that are set in the parameter `MaskSet`, if the current `IOvalue` at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector meets the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are `1` and the bits specified in `Mask0` are `0`).
Call	`set_io_cond_list( Mask1, Mask0, MaskSet )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.
	`Mask0` Like `Mask1`.
	`MaskSet` Like `Mask1`.
Multi-board Com'd Name	**n_set_io_cond_list**
Comments	• **set_io_cond_list** sets only those bits of the digital output port that are set in the parameter `MaskSet` and leaves the other bits unchanged. • See Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81 and Chapter 9.3.2 "Execution of Conditional Commands", page 317.
Examples (Pascal)	• Set Bit #4 of the output port (DIGITAL OUT4), if Bit #0 of the input port (DIGITAL IN0) is set and Bit #1…Bit #3 (DIGITAL IN1…3) of the input port are not set: `set_io_cond_list`($0001, $000E, $0010) • Always set Bit #15 of the output port (and leave the other bits unchanged): `set_io_cond_list`(0, 0, $8000)
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**clear_io_cond_list**, **write_io_port**, **write_io_port_mask**, **get_io_status**, **read_io_port**

Ctrl Command	set_jump_mode	
Function	Enables and activates or disables and deactivates Jump Mode for 2D jumps and sets the related parameters.	
Prerequisite	Enabling is only possible if a jump-tuning-equipped intelliSCAN (with scan system firmware version ≥ 2078) is attached to at least one of the two scan head connectors. Otherwise, **set_jump_mode** has no effect.	
Call	`ErrorCode = set_jump_mode( Flag, Length, VA1, VA2, VB1, VB2, JA1, JA2, JB1, JB2 )`	
Parameters	`Flag`	Switching flag. As a signed 32-bit value. Allowed values: = –1: Jump Mode is disabled. Afterward, switching the `Flag` by **set_jump_mode_list** is *no longer* possible. = 0: Jump Mode is enabled but deactivated. Afterwards it is also possible to switch the flag by **set_jump_mode_list**. = 1: Jump Mode is enabled and activated. Afterwards it is also possible to switch the flag by **set_jump_mode_list**.
	`Length`	Limit of the jump length (per axis) under which 2D jumps – even with activated Jump Mode – are performed in vector mode. As an unsigned 32-bit value.

	`VA1`
	`VA2`
	`VB1`
	`VB2`
	`JA1`
	`JA2`
	`JB1`
	`JB2`
		Numbers of the tunings that should be used for jump execution in Jump Mode:
		V: The Vector tuning that is set at the end of a 2D jump.
		J: The Jump tuning that is set at the beginning of a 2D jump.
		A Connector for First Scan Head.
		B Connector for Second Scan Head.
		1: x axis (STATUS channel, Galvanometer scanner 2).
		2: y axis (STATUS1 channel, Galvanometer scanner 1).
		Allowed values:
		= – 1: Tuning is neither checked nor used.
		= 0…3: After passing a check, tuning is used for Jump Mode. The allowed value range also depends on the number of tunings with which the attached scan system is equipped.
		As a signed 32-bit value.

Ctrl Command	set_jump_mode
Result	Error code. As a signed 32-bit value. 0 — No error: Flag successfully switched to 0 (Jump Mode deactivated, vector mode activated). 1 — No error: Flag successfully switched to 1 (Jump Mode activated, vector mode deactivated). –1 — Flag successfully switched to –1 (Jump Mode deactivated and disabled). –2 — Busy error: board BUSY list execution status or INTERNAL-BUSY list execution status (**get_last_error** return code `RTC6_BUSY`). –3 — Board not responding: no program loaded or PCI error (**get_last_error** return code `RTC6_TIMEOUT`). –4 — Access error: board reserved for another user program (**get_last_error** return code `RTC6_ACCESS_DENIED`). > 1 — Did not pass the check (see also notes). `Flag` is set to –1 (Jump Mode deactivated and disabled). The following is returned: Byte #0 = 255 Byte #1 = Error code for Connector for First Scan Head Byte #2 = Error code for Connector for Second Scan Head Byte #3 = 0 Whereby error code: =1: x axis (galvanometer scanner 2) not responding or no intelliSCAN (with scan system firmware version ≥ 2078) attached. =2: y axis (galvanometer scanner 1) not responding or no intelliSCAN (with scan system firmware version ≥ 2078) attached. =4: no correction table assigned. =8: incorrect tuning number(s): incorrect type or unsuitable for rapid switching.
Multi-board Com'd Name	**n_set_jump_mode**

Ctrl Command	**set_jump_mode**
Comments	• For usage of **set_jump_mode**, see Chapter 8.1.5 "Jump Mode", page 226. • A check (see also Section "Requirements and Activation", page 227) is only performed if `Flag` = –1 (the initialization state) prior to the **set_jump_mode** call and/or if the supplied tuning numbers do not match those stored on the board. Otherwise, only the flag is switched. For the check, the board must not be BUSY list execution status or INTERNAL-BUSY list execution status, because meanwhile the data type to-be-returned by the scan system changes and "Automatic Laser Control" is deactivated (both get restored at the end of the command). Depending on results of the check, different error codes are returned (see above). In case of error, `Flag` gets set to –1 (Jump Mode deactivated and disabled). If the check is successful, then you can afterward (even by **set_jump_mode_list** during processing of a list) switch freely between the states `Flag` = 1 (Jump Mode activated, vector mode deactivated) and `Flag` = 0 (Jump Mode deactivated, vector mode activated) without another check having to be performed. • Use –1 as the tuning number if certain tunings should not be checked (for example, because no intelli*SCAN* scan system is attached or specific tunings are not available – Vector tuning, for example, is not needed in pure drilling applications) or if, after switching to Jump tuning, it is not desirable to return to Vector tuning. As a result, such tunings are neither checked nor switched on. If the Option "Second Scan Head Control" is not enabled, then the tuning numbers for the Connector for Second Scan Head (B) is automatically set to –1 (even if others were supplied). If all tuning numbers are –1, then `Flag` is set to –1 (return value –1). • Even after successful activation of Jump Mode (`Flag` = 1), the first servo switching only occurs after the first subsequent 2D jump (see Section "Functional Principle", page 226). If the currently set tuning then does not match the Jump tuning or Vector tuning specified by **set_jump_mode**, then the first switching can take somewhat longer (approx. 250 ms), depending on the currently set tuning. You can determine ahead of time whether this is so by calling **set_jump_mode** using the currently set tuning as a parameter. If true (return value > 1, error code = 2), then you can achieve the desired operational sequence by calling **control_command** before the first 2D jump to set the tuning to one that has been supplied by **set_jump_mode**. • **set_jump_mode** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_jump_mode_list**, **load_jump_table_offset**, **set_jump_table**

Normal List Command	set_jump_mode_list
Function	Activates or deactivates and disables Jump Mode for 2D jumps.
Prerequisite	See **set_jump_mode**.
Call	`set_jump_mode_list( Flag )`
Parameters	`Flag` See **set_jump_mode**.
Multi-board Com'd Name	**n_set_jump_mode_list**
Comments	• For usage of **set_jump_mode_list**, see Chapter 8.1.5 "Jump Mode", page 226. • **set_jump_mode_list** functions like the control command **set_jump_mode** (see notes there) but, as a list command, has the following differences: – No tunings or jump length limit can be specified by **set_jump_mode_list**. – **set_jump_mode_list** does not perform a check. `Flag` must have previously been successfully set to 0 or 1 by **set_jump_mode**. – Though Jump Mode can be deactivated and disabled by setting `Flag` to –1 by **set_jump_mode** or **set_jump_mode_list**, it can only be reactivated again by **set_jump_mode** (if the check is successful). If `Flag` has been –1 prior to calling **set_jump_mode_list**, then **set_jump_mode_list** has no effect.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
Reference	**set_jump_mode**

Undelayed Short List Command	**set_jump_speed**
Function	Defines the jump speed for Vector Commands.
Call	`set_jump_speed( Speed )`
Parameters	`Speed` Jump speed. In Bits/ms. As a 64-bit IEEE floating point value. Allowed value range: [1.6…800000.0]. Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_set_jump_speed**
Comments	• By default a jump speed of 10000 *bits/ms* is preset. • The specified jump speed is used for all Jump Commands until a new value is specified. • The actual jump speed v_{jump} in the image plane in m/s is derived from the specified `Speed` value [*bits/ms*] and the calibration factor *K* [Bits/mm] as follows: v_{jump} = `Speed` / *K* The calibration factor *K* can be queried from the correction table by **get_table_para** or **get_head_para**. • **set_jump_speed** is also available as control command **set_jump_speed_ctrl**.
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode, the RTC6 multiplies the specified `Speed` value by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**jump_abs**, **jump_rel**, **set_mark_speed**, **set_jump_speed_ctrl**

Ctrl Command	set_jump_speed_ctrl
Function	Like **set_jump_speed**, but a control command.
Call	`set_jump_speed_ctrl( Speed )`
Parameters	`Speed` Like **set_jump_speed**.
Multi-board Com'd Name	**n_set_jump_speed_ctrl**
Comments	• **set_jump_speed_ctrl** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set • **set_jump_speed_ctrl** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) – the INTERNAL-BUSY list execution status is set
RTC4→RTC6	New command. In RTC4 Compatibility Mode: like **set_jump_speed**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_jump_speed**, **set_mark_speed**, **set_mark_speed_ctrl**

Ctrl Command	set_jump_table
Function	Reads the Jump Delay table with 1024 unsigned 16-bit values stored at the supplied PC address and loads them onto the board as the Jump Delay table (see notes for **get_jump_table**).
Call	`ErrorCode = set_jump_table( Addr )`
Parameters	`Addr` PC Address of a 2048-byte area of PC main memory.
Result	Error code. As an unsigned 32-bit value. 0 No error. 1 Busy error: board BUSY list execution status or INTERNAL-BUSY list execution status (**get_last_error** return code `RTC6_BUSY`). 4 Verify error: DSP memory error. 11 RTC6 board driver error.
Multi-board Com'd Name	**n_set_jump_table**
Comments	• **set_jump_table** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **set_jump_table** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set)
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
Reference	**get_jump_table**, **load_jump_table_offset**

Ctrl Command	set_laser_control
Function	Defines and enables or disables the Laser Control Signals.
Call	`set_laser_control( Ctrl )`
Parameters	`Ctrl` As an unsigned 32-bit value. Bit #0 (LSB) Pulse Switch Setting (does not concern Laser Mode 4 or Laser Mode 6). The setting only affects those LASER1 or LASER2 “laser active” modulation pulses in CO_2 Mode or LASER1 Q-Switch pulses in a YAG Mode) that are not yet fully processed at completion of the LASERON signal, see Figure 52 and Figure 53. = 0: The signals are cut off at the end of the LASERON signal. = 1: The final pulse fully executes despite completion of the LASERON signal. See Pulse Completion, page 191. Bit #1 Phase shift of the Laser Control Signals (does not concern Laser Mode 4 or Laser Mode 6). = 0: No phase shift. = 1: With CO_2 Mode, the LASER1 signal is exchanged with the LASER2 signal. With YAG Mode 1, YAG Mode 2, YAG Mode 3, YAG Mode 5, the LASER1 is shifted back half a signal period. Bit #2 Enabling or disabling of Signals for “Laser Active” Operation. = 0: The Signals for “Laser Active” Operation are enabled. = 1: The Signals for “Laser Active” Operation are disabled (the signals are set to their respective “Off” level). Bit #3 Level of LASERON signal. = 0: Is set to active-HIGH. = 1: Is set to active-LOW. If there is no change of the pin assignment with **config_laser_signals**, the LASERON signal corresponds to the signal at the LASERON pin, see Figure 17. Bit #4 Levels of LASER1 signal and LASER2 signal. = 0: Are set to active-HIGH. = 1: Are set to active-LOW. If there is no change of the pin assignment with **config_laser_signals**, the LASER1 signal (LASER2 signal) corresponds to the signal at the LASER1 pin (LASER2 pin), see Figure 17. Bit #5 Determines for **laser_on_pulses_list** whether external signal pulses (at the LASER Connector’s DIGITAL IN1 digital input port) are to be counted at rising or falling edges: = 0: At the falling edge. = 1: At the rising edge.

Ctrl Command	set_laser_control		
Parameters (cont'd)	Bit #6	= 0:	Synchronization is switched off (default setting). See Chapter 7.4.10 "Synchronization of the RTC6 Clock Cycle and an External Clock Signal", page 216.
		= 1:	Synchronization is switched on.
	Bit #7	= 0:	The "constant pulse length" mode is switched off (default setting).
		= 1:	The "constant pulse length" mode is switched on. See Chapter 7.4.8 "Pulse Picking Laser Mode", page 203 and **set_pulse_picking_length**.
	Bit #8	Reserved.	
	Bit #9	Reserved.	
	Bit #10	Reserved.	
	Bit #11	Reserved.	
	Bit #12	Reserved.	
	Bit #13	Reserved.	
	Bit #14	Reserved.	
	Bit #15	Reserved.	
	Bit #16	For the automatic suppression of Laser Control Signals is used: PowerOK of the head A, x axis.	
	Bit #17	Like Bit #16, but: TempOK of the head A, x axis.	
	Bit #18	Like Bit #16, but: PosAck of the head A, x axis.	
	Bit #19	Like Bit #16, but: PowerOK of the head A, y axis.	
	Bit #20	Like Bit #16, but: TempOK of the head A, y axis.	
	Bit #21	Like Bit #16, but: PosAck of the head A, y axis.	
	Bit #22	Like Bit #16, but: PowerOK of the head B, x axis.	
	Bit #23	Like Bit #16, but: TempOK of the head B, x axis.	
	Bit #24	Like Bit #16, but: PosAck of the head B, x axis.	
	Bit #25	Like Bit #16, but: PowerOK of the head B, y axis.	
	Bit #26	Like Bit #16, but: TempOK of the head B, y axis.	
	Bit #27	Like Bit #16, but: PosAck of the head B, y axis.	
	Bit #28	= 1:	In case of error, automatic monitoring (automatic suppression of Laser Control Signals) automatically generates a /STOP signal (list stops, Laser Control Signals get permanently switched off).
	Bit #29	= 1:	In case of error according to Bit #28, the **stop_execution** is forwarded as /Master-STOP (see Figure 70) to all Master/Slave-connected RTC6 boards. See **master_slave_config**.
	Bit #30	Reserved.	
	Bit #31	Reserved.	
Multi-board Com'd Name	n_set_laser_control		

Ctrl Command	set_laser_control
Comments	• In the default setting (after **load_program_file**), all bits are set to 0. After a hardware reset, however, the settings become effective following the first-time call of **set_laser_control**. Prior to this, all laser control signal outputs (LASERON, LASER1 and LASER2) are in the high-impedance mode. TTL states (LOW or HIGH) only become available when **set_laser_control** is called to define the desired TTL level, see also Chapter 7.4 "Laser Control", page 189. Even after **load_program_file**, which deactivates the Laser Control Signals, **set_laser_control** must be called for first-time activation. • For the RTC5/RTC6 predecessors, the laser control signal levels are defined by solder jumpers. The RTC5/RTC6 lets users control them completely by software. **get_startstop_info** queries the current status of the Laser Control Signals (Bit #9) and whether the signals are set to active-HIGH or active-LOW (Bit #13). • Enabling and disabling of Laser Control Signals can also be achieved by **enable_laser** or **disable_laser**. By **get_startstop_info** (Bit #14) the current state can be queried. • Even if the Laser Control Signals were enabled with **set_laser_control** or **enable_laser**, they are not outputted without further commands, see Chapter 7.4 "Laser Control", page 189. • The phase shift of the Laser Control Signals (Bit #1 = 1) can be set for better synchronization of an analog output, for example, in Softstart Mode (not yet implemented) or the Pixel Output Mode, see Chapter 8.7 "Pixel Output Mode", page 272. • **set_laser_control** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 617, OUT 617, RBF 623: **get_startstop_info** (Bit #14) support.
References	**set_laser_mode**, **config_laser_signals**

Undelayed Short List Command	**set_laser_delays**
Function	Sets the LaserOn Delay and the LaserOff Delay.
Call	`set_laser_delays( LaserOnDelay, LaserOffDelay )`
Parameters	`LaserOnDelay` LaserOn Delay. As a signed 32-bit value. 1 bit equals 1/64 μs. Allowed value range: $[-2^{31}...+(2^{21}-1)]$. Values over $(2^{21}-1)$ are clipped.
	`LaserOffDelay` LaserOff Delay. As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: $[0...+(2^{21}-1)]$. Values over $(2^{21}-1)$ are clipped.
Multi-board Com'd Name	**n_set_laser_delays**
Comments	• The delays can be freely chosen within the allowed ranges. If `LaserOffDelay` < `LaserOnDelay`, overlaps of LaserOn and LaserOff are automatically prevented during processing of short vectors, see Section "Automatic Delay Adjustments", page 158. • Observe the notes in Chapter 7.2.1 "Laser Delays", page 148. • A negative `LaserOnDelay` value extends the total marking time. • The XY2-100 Converter (Accessory) introduces a 10 μs runtime latency to scan system control. This runtime latency can be compensated by increasing the LaserOn Delay and LaserOff Delay by 10 μs each. • The default setting after **load_program_file** corresponds to `set_laser_delays( 640, 640 )`.
RTC4→RTC6	Essentially similar functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified delay values by 64. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the values specified for `LaserOnDelay` and `LaserOffDelay` by 32. The allowed value ranges decrease accordingly.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_scanner_delays**

Ctrl Command	**set_laser_mode**
Function	Sets the laser mode of the RTC6.
Call	`set_laser_mode( Mode )`
Parameters	`Mode` = 0: CO_2 Mode. = 1: YAG Mode 1. = 2: YAG Mode 2. = 3: YAG Mode 3. = 4: Laser Mode 4. = 5: YAG Mode 5. = 6: Laser Mode 6. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_laser_mode**
Comments	• The available Laser Control Signals depend on the set laser mode, see also Chapter 7.4 "Laser Control", page 189. • **set_laser_mode** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	Additional laser modes, for example, YAG Mode 5, Laser Mode 6 and Pulse Picking. Otherwise, unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_laser_control**, **set_laser_pulses_ctrl**, **set_laser_pulses**, **set_laser_timing**, **set_firstpulse_killer**, **set_firstpulse_killer_list**, **set_standby**, **set_standby_list**, **set_qswitch_delay**, **set_qswitch_delay_list**

Ctrl Command	set_laser_off_default	
Function	Sets the default output value for the ANALOG OUT1 and ANALOG OUT2 output ports, as well as for the 8-Bit Digital Output Port.	
Call	`set_laser_off_default( AnalogOut1, AnalogOut2, DigitalOut )`	
Parameters	`AnalogOut1`	12-bit value for analog output port ANALOG OUT1, see also Section "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 77. As an unsigned 32-bit value. Bits with higher significance are ignored. Exception: `AnalogOut1/AnalogOut2 =` "–1", see comments for **set_port_default**.
	`AnalogOut2`	12-bit value for analog output port ANALOG OUT2. See `AnalogOut1`.
	`DigitalOut`	8-bit value for the 8-Bit Digital Output Port, see also Section "8-Bit Digital Output Port", page 84. As an unsigned 32-bit value. Bits with higher significance are ignored. Exception: `DigitalOut =` "–1", see comments for **set_port_default**.
Multi-board Com'd Name	**n_set_laser_off_default**	
Comments	• The default values can also be defined by **set_port_default**. See also all comments there.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `AnalogOut1` and `AnalogOut2` by 4.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_port_default**	

Ctrl Command	set_laser_pin_out
Function	Sends a value to the two digital outputs of the LASER Connector.
Call	`set_laser_pin_out( Pins )`
Parameters	`Pins` Output value (DIGITAL OUT1 and DIGITAL OUT2). As an unsigned 32-bit value. Bit # 0: DIGITAL OUT1. Bit # 1: DIGITAL OUT2. Bit # 2: Reserved. … Bit #31: Reserved.
Multi-board Com'd Name	**n_set_laser_pin_out**
Comments	• See also Chapter 9.1.3 "2-Bit Digital Output Port", page 304.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_laser_pin_out_list**, **get_laser_pin_in**

Undelayed Short List Command	set_laser_pin_out_list
Function	Like **set_laser_pin_out**, but a list command.
Call	`set_laser_pin_out_list( Pins )`
Parameters	`Pins` Like **set_laser_pin_out**.
Multi-board Com'd Name	**n_set_laser_pin_out_list**
Comments	• See **set_laser_pin_out**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_laser_pin_out**, **write_port_list**

Undelayed Short List Command	**set_laser_power**
Function	Synchronizes laser power outputs at the output ports (ANALOG OUT1, ANALOG OUT2, 8-Bit Digital Output Port, 16-Bit Digital Output Port) and Laser Delays.
Call	`set_laser_power( Port, Power )`
Parameters	`Port` Output port for which a laser power value `Power` is to be defined. As an unsigned 32-bit value. Allowed values: = 0: ANALOG OUT1 output port. = 1: ANALOG OUT2 output port. = 2: 8-Bit Digital Output Port (EXTENSION 2 Socket Connector). = 3: 16-Bit Digital Output Port (EXTENSION 1 Socket Connector). > 3: **set_laser_power** is not executed (**get_last_error** return code `RTC6_PARAM_ERROR`).
	`Power` Desired laser power value. As an unsigned 32-bit value. Allowed values: For `Port` = 0: 12-bit values [0…4,095]. For `Port` = 1: 12-bit values [0…4,095]. For `Port` = 2: 8-bit values [0…255]. For `Port` = 3: 16-bit values [0…65,535]. Out-of-range values are clipped to the boundary values. **get_last_error** return code `RTC6_PARAM_ERROR`.
Multi-board Com’d Name	**n_set_laser_power**
Comments	• In particular, **set_laser_power** is to be used with a SCANahead System, if laser power needs to be changed within a Polyline. • **set_laser_power** is also necessary, for example, if *short vectors* need unequal laser power and Laser Delays extend beyond the next vector (in particular in DSP mode 3).
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified `Power` value for `Port` = 0 and `Port` = 1 by 4. The allowed value range decreases accordingly.
RTC5→RTC6	New command.
Version Info	Available as of DLL 602, OUT 602, RBF 602.
References	**get_last_error**, **set_dsp_mode**, **set_rtc4_mode**, **write_da_x**

Ctrl Command	**set_laser_pulse_sync**
Function	Switches Pulse Synchronization Mode on (off).
Call	`set_laser_pulse_sync( Mode, Delay )`
Parameters	`Mode` = 0: Switches Pulse Synchronization Mode off. > 0: Switches Pulse Synchronization Mode on. As an unsigned 32-bit value.
	`Delay` Delay. In 1/64 μs. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_laser_pulse_sync**
Comments	• See also Chapter 7.4.11 "Pulse Synchronization Mode", page 217. • See also Section ""Spot Distance Control"", page 210. • **set_laser_pulse_sync** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 625, OUT 625, RBF 628.
References	–

Delayed Short List Command	**set_laser_pulses**	
Function	Defines the output period and the pulse lengths of the Laser Control Signals LASER1 and LASER2 for "laser active" operation.	
Call	`set_laser_pulses( HalfPeriod, PulseLength )`	
Parameters	`HalfPeriod`	*Half* of the output period. In bits. As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: [0…(2^{32}–1)].
	`PulseLength`	Pulse length of the Laser Control Signals LASER1 and LASER2. In bits. As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: [0…(2^{32}–1)].
Multi-board Com'd Name	**n_set_laser_pulses**	
Comments	• By the `HalfPeriod` parameter, *half* the period duration is specified, see Figure 52 and Figure 53. • If `HalfPeriod` = 0 and/or `PulseLength` = 0, no Laser Control Signals are outputted. • With `PulseLength` ≥ (2 × `HalfPeriod`), the laser remains on all the time. • The signal level is defined by **set_laser_control**. • **set_laser_pulses** is also available as the control command **set_laser_pulses_ctrl**. • **set_laser_pulses** is largely identical to the RTC4 command **set_laser_timing**, but has less parameters.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `HalfPeriod` and `PulseLength` by 8. The allowed value ranges decrease accordingly.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_laser_pulses_ctrl**, **set_laser_timing**	

Ctrl Command	set_laser_pulses_ctrl	
Function	Like **set_laser_pulses**, but a control command.	
Call	`set_laser_pulses_ctrl( HalfPeriod, PulseLength )`	
Parameters	`HalfPeriod`	Like **set_laser_pulses**.
	`PulseLength`	Like **set_laser_pulses**.
Multi-board Com'd Name	**n_set_laser_pulses_ctrl**	
Comments	• See **set_laser_pulses**.	
RTC4→RTC6	New command. In RTC4 Compatibility Mode: like **set_laser_pulses**.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_laser_pulses**, **set_laser_timing**	

Delayed Short List Command	**set_laser_timing**	
Function	Defines the output period and the pulse lengths for the Laser Control Signals LASER1 and LASER2 for “laser active” operation.	
Call	`set_laser_timing( HalfPeriod, PulseLength1, (PulseLength2), TimeBase )`	
Parameters	`HalfPeriod`	*Half* of the output period. In bits. As an unsigned 32-bit value. • In RTC6 Standard Mode: 1 bit equals 1/64 µs. Allowed value range: [0…(2^{32}–1)]. • In RTC4 Compatibility Mode: *1 bit equals 1/8 µs or 1 µs*, depending on the selected clock frequency. With respect to the specified `TimeBase`, the value is converted to an integer-multiple of 1/64 µs. The allowed range is correspondingly smaller.
	`PulseLength1`	Pulse lengths of the Laser Control Signals LASER1 and LASER2. In bits. As an unsigned 32-bit value. • In RTC6 Standard Mode: 1 bit equals 1/64 µs. Allowed value range: [0…(2^{32}–1)]. • In RTC4 Compatibility Mode: *1 bit equals 1/8 µs or 1 µs*, depending on the selected clock frequency. With respect to the specified `TimeBase`, the value is converted to an integer-multiple of 1/64 µs. The allowed range is correspondingly smaller.
	`(PulseLength2)`	Value is not used. (As an unsigned 32-bit value.) With the RTC6, the pulse lengths of Laser Control Signals LASER1 and LASER2 are always identical and are defined by `PulseLength1`.
	`TimeBase`	As an unsigned 32-bit value. • In RTC6 Standard Mode, the value is ignored and the clock frequency is fixed at 64 MHz. 1 bit equals 1/64 µs. • In RTC4 Compatibility Mode, the `TimeBase` value is handled as follows: = 0: sets the clock frequency to 1 MHz. *1 bit equals 1 µs*. ≠ 0: sets the clock frequency to 8 MHz. *1 bit equals 1/8 µs*.
Multi-board Com’d Name	**n_set_laser_timing**	

Delayed Short List Command	set_laser_timing
Comments	• In RTC6 Standard Mode, **set_laser_timing** is synonymous with **set_laser_pulses**. See comments there (on HalfPeriod, PulseLength). • The clock frequency settings apply *only* for the parameters of **set_laser_timing**. • In RTC4 Compatibility Mode, SCANLAB generally recommends setting the clock frequency to 8 MHz. A clock frequency of 1 MHz should only be set, if absolutely necessary. • Observe also the notes in Chapter 7.4 "Laser Control", page 189. • In RTC4 Compatibility Mode, in Laser Mode 4 and Laser Mode 6, the time base for signals LASER1 and LASER2 is independently of TimeBase always 1/8 µs.
RTC4→RTC6	• With the RTC6, the pulse lengths of Laser Control Signals LASER1 and LASER2 are always identical. • Clock frequencies: – In RTC6 Standard Mode: 64 MHz, fixed – In RTC4 Compatibility Mode: 1 MHz or 8 MHz
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_laser_pulses_ctrl**, **set_laser_pulses**, **set_laser_mode**, **set_firstpulse_killer**, **set_firstpulse_killer_list**, **set_standby**, **set_standby_list**

Normal List Command	set_laser_timing_table	
Function	Category: List Commands for a Restricted User Group only. Sets a value in the lookup table for the laser frequency.	
Call	`set_laser_timing_table( Index, F )`	
Parameters	Index	Index in the lookup table for the laser frequency. Allowed value range: 0…10. As an unsigned 32-bit value.
	F	Laser frequency. In Hz. Allowed value range: 65…100,000. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_laser_timing_table**	
Comments	• All Comments of **set_duty_cycle_table** apply. • The following applies for List Commands for a Restricted User Group only: These users have been informed separately in how to handle these RTC6 commands. Therefore, only the command table is available in this manual and further information is omitted.	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 626, OUT 627.	
References	**set_duty_cycle_table**, **regulation3**	

Undelayed Short List Command	set_list_jump
Function	Produces an unconditional jump to the specified address within the RTC6 List Memory upon execution. The next command there is executed immediately without delay.
Call	`set_list_jump( Pos )`
Parameters	`Pos` Absolute jump address [0…(2^{23}–1)]. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_list_jump**
Comments	• **set_list_jump** is synonymous with **list_jump_pos**. See the comments there.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_jump_pos**

Delayed Short List Command	**set_mark_speed**
Function	Sets the mark speed for the Vector Commands and "Arc" Commands.
Call	`set_mark_speed( Speed )`
Parameters	`Speed` Marking speed. In Bits/ms. As a 64-bit IEEE floating point value. Allowed value range: [1.6…800000.0]. Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_set_mark_speed**
Comments	• By default a mark speed of 1,000 *bits/ms* is preset. • The specified mark speed is used for all [*]mark[*] Commands and "Arc" Commands until a new value is specified. • The actual mark speed v_{mark} in the image plane in m/s is derived from the specified `Speed` value [*bits/ms*] and the calibration factor *K* [Bits/mm] as follows: v_{mark} = `Speed` / *K* The calibration factor *K* can be queried from the correction table by **get_table_para** or **get_head_para**. • **set_mark_speed** is also available as control command **set_mark_speed_ctrl**.
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode, the RTC6 multiplies the specified `Speed` value by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_abs**, **mark_rel**, **set_jump_speed**, **set_mark_speed_ctrl**

Ctrl Command	set_mark_speed_ctrl
Function	Like **set_mark_speed**, but a control command.
Call	`set_mark_speed_ctrl( Speed )`
Parameters	`Speed` Like **set_mark_speed**.
Multi-board Com'd Name	**n_set_mark_speed_ctrl**
Comments	• **set_mark_speed_ctrl** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set • **set_mark_speed_ctrl** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) – the INTERNAL-BUSY list execution status is set • See also **set_sky_writing_min_speed_ctrl**.
RTC4→RTC6	New command. In RTC4 Compatibility Mode: like **set_mark_speed**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_mark_speed**, **set_jump_speed**, **set_jump_speed_ctrl**, **set_sky_writing_min_speed_ctrl**

Ctrl Command	set_matrix	
Function	Defines the coefficients of the • General transformation matrix M_T for coordinate transformations, see Chapter 8.2 "Coordinate Transformations", page 233 • Global transformation matrix in virtual Image Field, see Section "Coordinate Transformations in the Virtual Image Field", page 174	
Call	`set_matrix( HeadNo, M11, M12, M21, M22, at_once )`	
Parameters	`HeadNo`	Number of the scan head connector. As an unsigned 32-bit value. = 1: The definition only affects the Connector for First Scan Head. = 2: The definition only affects the Connector for Second Scan Head. = 0, 3: The definition affects *both* scan head connectors. = 4: The definition affects the virtual Image Field (see also comments). Only the three least significant bits are evaluated.
	`M11`	Matrix coefficient of the general transformation matrix M_T. As a 64-bit IEEE floating point value. Allowed value range: [–50…+50] in the real Image Field and [–2.0…+2.0] in the virtual Image Field. With invalid value, **set_matrix** is ignored.
	`M12`	Like `M11` (analogously).
	`M21`	Like `M11` (analogously).
	`M22`	Like `M11` (analogously).
	`at_once`	Defines when the defined transformation becomes effective. As an unsigned 32-bit value. For `HeadNo` = 0…3, the following applies: = 0: The new total transformation (total matrix and offset) is only calculated when the next list command is executed and applied to the position which is current at that time. = 1: The new total transformation is calculated immediately (or prior the next list command, if currently the BUSY list execution status or INTERNAL-BUSY list execution status is set) and applied to the current position. Signals for "Laser Active" Operation are switched off in advance. = 2: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next **jump_abs**, **jump_rel**, **goto_xy** or **goto_xyz** is executed. = 3: Like `at_once` = 1, but the Laser Control Signals remain unchanged. > 3: Like `at_once` = 2. For `HeadNo` = 4, the following applies: = 0: Only saved. Is applied at the next Processing-on-the-fly Session start to the current position. = 1: Is applied immediately to the current position if currently no list is BUSY list execution status or no board is INTERNAL-BUSY list execution status. Otherwise, like 0. = 2, 3: Like 0.
Multi-board Com'd Name	**n_set_matrix**	

Ctrl Command	set_matrix
Comments	• Up to DLL 613, OUT 613 the following applies for `HeadNo` = 4: – The global coordinate transformations are only available during a Processing-on-the-fly Session that has been started by **set_fly_2d**. – `at_once` is ignored • As of DLL 614, OUT 614 the following applies for `HeadNo` = 4: – The global coordinate transformations are generally available, even outside a Processing-on-the-fly Session. – Coordinate transformations that have been merely saved, are applied at the next Processing-on-the-fly Session start. • **set_matrix**(`HeadNo` = 4) is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	Unchanged first functionality (transformation matrix definition), however: • The parameters `HeadNo` and `at_once` are new. • Reduced value range for coefficients. • See also Section "Notes for RTC4 Users", page 236.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 614, OUT 614: `HeadNo` = 4 generally available.
References	**set_matrix_list**, **set_angle**, **set_offset**, **set_scale**

Variable List Command	**set_matrix_list**	
Function	Sets *one* of the 4 coefficients of the general transformation matrix M_T during execution of a list, see Chapter 8.2 "Coordinate Transformations", page 233.	
Call	`set_matrix_list( HeadNo, Ind1, Ind2, Mij, at_once )`	
Parameters	`HeadNo`	Like **set_matrix**.
	`Ind1` `Ind2`	Row index and column index of the matrix coefficient to be changed. As an unsigned 32-bit value. Allowed values: [Index uneven: 1, Index even: 2].
	`Mij`	Matrix coefficient. As a 64-bit IEEE floating point value. Allowed value range: [–50…+50]. If the parameter is set to an invalid value, **set_matrix_list** is replaced by a **list_nop**.
	`at_once`	Determines when the defined transformation becomes effective. As an unsigned 32-bit value. = 0: The transformation settings are only accumulated and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (for example, by a list command with `at_once` = 1 or a corresponding control command). = 1: The transformation is immediately calculated (including all transformation settings that were accumulated until then) and processed prior to the next list command. Signals for "Laser Active" Operation are switched off in advance. = 2: The transformation settings are only accumulated and intermediately stored (as with `at_once` = 0). However, The transformation is immediately calculated (including all transformation settings that were accumulated and intermediately stored until then) and applied to the current position when the next **jump_abs** or **jump_rel** (only if no list is currently being executed: also **goto_xy** or **goto_xyz**) is executed. = 3: As with `at_once` = 1, but the Laser Control Signals remain unaffected. > 3: See `at_once` = 2.
Multi-board Com'd Name	**n_set_matrix_list**	
Comments	• **set_matrix_list** only allows changing *one* of the 4 coefficients at a time. To change several coefficients during execution of a list, **set_matrix_list** has to be called repeatedly. Here, we recommend making the first calls with `at_once` = 0 and only the last call with `at_once` = 1. • See Chapter 8.2 "Coordinate Transformations", page 233.	
RTC4→RTC6	See **set_matrix**.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_matrix**	

Ctrl Command	set_max_counts
Function	Defines the maximum number of External Starts.
Call	`set_max_counts( Counts )`
Parameters	`Counts` Maximum number of External Starts. As an unsigned 32-bit value. Allowed value range: [0…(2^{32}–1)].
Multi-board Com'd Name	**n_set_max_counts**
Comments	• When the specified number of External Starts has been reached, the external start input is disabled (see **set_control_mode**, Bit #0 = 0). • If `Counts` = 0, the number of External Starts is unlimited. • When the RTC6 is initialized (by **load_program_file**), `Counts` is set to 0. • The current number of External Starts can be read from the corresponding internal counter by **get_counts**. The counter can be reset by **set_control_mode**. • **set_max_counts** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	Unchanged functionality. In addition: increased value range.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_counts**, **set_control_mode**

Ctrl Command	**set_mcbsp_freq**
Function	Sets the transmission frequency of the McBSP interface.
Call	`mcbsp_freq = set_mcbsp_freq( Freq )`
Parameters	`Freq` Desired transmission frequency of the McBSP interface in Hz. As an unsigned 32-bit value. Allowed value range: [4000000…16000000] (4…16 MHz). Out-of-range values are clipped to the boundary values. Out-of-range values cause the **get_last_error** return code `RTC6_PARAM_ERROR` to be generated.
Result	The actually set frequency in Hz. As an unsigned 32-bit value. With overflowing values 0 is returned.
Multi-board Com'd Name	**n_set_mcbsp_freq**
Comments	• The default transmission frequency (after initialization) is 8 MHz (`Freq` = 8,000,000). • Because not every arbitrary frequency can be implemented, **set_mcbsp_freq** returns the actually set frequency. Example: `mcbsp_freq = set_mcbsp_freq(7,000,000);` returns `mcbsp_freq = 7,200,000`. • The new transmission frequency only becomes effective when you re-initialize the McBSP interface by **mcbsp_init**. • The signals and operating conditions of the McBSP interface are presented in the Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87. • Note: The receiving frequency is exclusively determined by the incoming clock pulses and has a maximum limit of 16 MHz. • **set_mcbsp_freq** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mcbsp_init**, **set_mcbsp_out**, **set_mcbsp_out_ptr**

Ctrl Command	set_mcbsp_global_matrix
Function	Activates matrix correction for "Global Online Positioning" by the McBSP interface.
Call	`set_mcbsp_global_matrix()`
Multi-board Com'd Name	**n_set_mcbsp_global_matrix**
Comments	• See also Chapter 8.3.2 ""Global Online Positioning"", page 240. • A matrix correction cannot be used in conjunction with offset and/or rotation corrections. Any such already-activated options gets deactivated by **set_mcbsp_global_matrix**. Subsequent activation of other options (by **set_mcbsp_global_x**, **set_mcbsp_global_y** or **set_mcbsp_global_rot**) deactivates the matrix option. • The following restrictions apply to the matrix coefficients transferred over the McBSP interface (as with **set_matrix**`(HeadNo = 4, …)`: – The allowed value range for matrix coefficients is [–2.0…+2.0]. – Transferred coefficients exceeding this range are ignored. • Users must individually supply as input value M_{in} to the McBSP interface each matrix coefficient Mij of the transformation matrix M_T as a normalized integer with associated indices i and j as follows: M_{in} = (integer(Mij * 2^{28}) << 2) + (i << 1) + j with M_T = { M00, M01, M10, M11 } = { m_{11}, m_{12}, m_{21}, m_{22} }. Conversely, the RTC6 determines a coefficient from the input value as follows: M_T[M_{in} & 0x3] = (M_{in} >> 2) / 2^{28}. • The coefficients are transferred to internal memory location 1 and can be checked there by querying with `read_mcbsp(1)`. • The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section "Notes", page 239.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**set_mcbsp_global_matrix_list**, **set_mcbsp_matrix**

Undelayed Short List Command	**set_mcbsp_global_matrix_list**
Function	Like **set_mcbsp_global_matrix**, but a list command.
Call	`set_mcbsp_global_matrix_list()`
Multi-board Com'd Name	**n_set_mcbsp_global_matrix_list**
Comments	• See **set_mcbsp_global_matrix**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**set_mcbsp_global_matrix**

Ctrl Command	set_mcbsp_global_rot
Function	Activates or deactivates rotation correction for "Global Online Positioning" by the McBSP interface.
Call	`set_mcbsp_global_rot( Resolution )`
Parameters	`Resolution` As a 64-bit IEEE floating point value. 2^{-26} < `Resolution` < 2^{26}: scaling factor (correction is activated), otherwise: correction is deactivated. `Resolution` = McBSP bits per full circle.
Multi-board Com'd Name	**n_set_mcbsp_global_rot**
Comments	• See also Chapter 8.3.2 ""Global Online Positioning"", page 240. • With an McBSP rotation correction input value of Rot_{in}, **set_mcbsp_global_rot** functions like `set_angle(4, Angle × 360°,…)`, whereby `Angle` (in full circles) = Rot_{in} / `Resolution`. Only `Angle` values in the range [0.0…+20.0 full circles] are allowed. • The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section "Notes", page 239.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**set_mcbsp_global_rot_list**, **set_mcbsp_rot**

Undelayed Short List Command	set_mcbsp_global_rot_list
Function	Like **set_mcbsp_global_rot**, but a list command.
Call	`set_mcbsp_global_rot_list( Resolution )`
Parameters	`Resolution` Like **set_mcbsp_global_rot**.
Multi-board Com'd Name	**n_set_mcbsp_global_rot_list**
Comments	• See **set_mcbsp_global_rot**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**set_mcbsp_global_rot**

Ctrl Command	set_mcbsp_global_x
Function	Activates or deactivates x offset correction for "Global Online Positioning" by the McBSP interface.
Call	`set_mcbsp_global_x( Scale )`
Parameters	`Scale` As a 64-bit IEEE floating point value. 2^{-26} < `Scale` < 2^{26}: scaling factor (correction is activated), otherwise: correction is deactivated.
Multi-board Com'd Name	**n_set_mcbsp_global_x**
Comments	• See also Chapter 8.3.2 ""Global Online Positioning"", page 240. • With an McBSP input value of X_{in} for the x offset correction, **set_mcbsp_global_x** functions like `set_offset_xyz(4, XOffset,…)`, whereby `XOffset` (in bits) = `Scale` × X_{in}. `XOffset` values outside of [–524,288…+524,287] are clipped to the boundary value as long as `Scale` × X_{in} does not exceed the value range $\pm 2^{31}$. • The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section "Notes", page 239.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**set_mcbsp_global_x_list**, **set_mcbsp_x**

Undelayed Short List Command	set_mcbsp_global_x_list
Function	Like **set_mcbsp_global_x**, but a list command.
Call	`set_mcbsp_global_x_list( Scale )`
Parameters	`Scale` Like **set_mcbsp_global_x**.
Multi-board Com'd Name	**n_set_mcbsp_global_x_list**
Comments	• See **set_mcbsp_global_x**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**set_mcbsp_global_x**

Ctrl Command	set_mcbsp_global_y
Function	Activates or deactivates y offset correction for "Global Online Positioning" by the McBSP interface.
Call	`set_mcbsp_global_y( Scale )`
Parameters	`Scale` As a 64-bit IEEE floating point value. 2^{-26} < `Scale` < 2^{26}: scaling factor (correction is activated), otherwise: correction is deactivated.
Multi-board Com'd Name	**n_set_mcbsp_global_y**
Comments	• See also Chapter 8.3.2 ""Global Online Positioning"", page 240. • With an McBSP input value of Y_{in} for the y offset correction, **set_mcbsp_global_y** functions like `set_offset_xyz(4, …, YOffset,…)`, whereby `YOffset` (in bits) = `Scale` × Y_{in}. `YOffset` values outside of [–524,288…+524,287] are clipped to the boundary value as long as `Scale` × Y_{in} does not exceed the value range $\pm 2^{31}$. • The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section "Notes", page 239.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**set_mcbsp_global_y_list**, **set_mcbsp_y**

Undelayed Short List Command	**set_mcbsp_global_y_list**
Function	Like **set_mcbsp_global_y**, but a list command.
Call	`set_mcbsp_global_y_list( Scale )`
Parameters	`Scale` Like **set_mcbsp_global_y**.
Multi-board Com'd Name	**n_set_mcbsp_global_y_list**
Comments	• See **set_mcbsp_global_y**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**set_mcbsp_global_y**

SCANLAB
innovators for industry

Ctrl Command	set_mcbsp_in
Function	Activates Processing-on-the-fly correction for compensation of a workpiece or scan system motion (based on position values transferred to the RTC6 via the McBSP interface). The McBSP interface can also be used for inputting other desired signals.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_mcbsp_in** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_mcbsp_in( Mode, Scale )`
Parameters	`Mode` As an unsigned 32-bit value. Allowed values: = 0: Processing-on-the-fly correction is switched off. = 1: Compensation of linear motion in the x direction. = 2: Compensation of linear motion in the Y direction. = 3: Compensation of linear motion in the x direction and y direction. = 4: Compensation of rotary motion. = 5: Processing-on-the-fly correction is switched off. • `Mode = 0…5`: All McBSP input values are alternatingly copied to internal memory locations 1 and 2. • `Mode = 1…5` (but not for `Mode = 0`): McBSP input values coded with "Bit #31 = 0" is additionally copied to internal memory location 0 and those with "Bit #31 = 1" to internal memory location 3. • `Mode = 1…4`: Values copied to internal memory location 0 are applied for Processing-on-the-fly correction.
	`Scale` Scaling factor or rotation resolution. As a 64-bit IEEE floating point value. • `Mode = 1…3`: scaling factor in (RTC6)*bits/(McBSP)bit.* Allowed value range: 1/256 ≤ |`Scale`| ≤ 16000.0 (if `Mode = 3`, `Scale` applies to both axes). • `Mode = 4`: number of steps (counts) per revolution. Allowed value range: |`Scale`| > 100.0.
Multi-board Com'd Name	**n_set_mcbsp_in**
Comments	• You can query the internal memory locations at any time by **read_mcbsp**. • For Processing-on-the-fly correction and determination of the scaling factor, see Chapter 8.6 "Processing-on-the-fly", page 251. • The various Processing-on-the-fly corrections cannot be arbitrarily combined, see Section "Overview", page 251.

Ctrl Command	set_mcbsp_in
Comments (cont'd)	• For deactivating Processing-on-the-fly correction, see Chapter 8.6.5 "Deactivating Processing-on-the-fly Correction", page 260. • 15 bits per axis (with sign) are effectively available for 2D correction (`Mode` = 3), whereby the x value is in the lower 16 bits of the "Bit #31 = 0"-coded McBSP input value and the y value in the upper 16 bits. • The McBSP interface cannot be simultaneously used for both Processing-on-the-fly applications and Online Positioning. See also Section "Notes", page 239. • The McBSP interface ignores the first FrameSync signal after a **load_program_file** or **mcbsp_init**. That is, data provided is not transmitted, see page 89. • If `Mode` > 5, **set_mcbsp_in** is *not* executed (**get_last_error** return code `RTC6_PARAM_ERROR`). • If an unallowed `Scale` parameter value is supplied (for example, `Scale` = 0), **set_mcbsp_in** behaves as with `Mode` = 0. • Processing-on-the-fly corrections are compatible with the settings from Chapter 8.6.12 ""Fly Extension" Commands", page 268 as of RTC6 Software Package V1.6.1. They can also be subsequently overwritten, for example, with encoder-based Processing-on-the-fly corrections. The McBSP values are then not used for Processing-on-the-fly corrections. However, the McBSP memory transfer described above cannot be changed. If McBSP-based Processing-on-the-fly corrections other than those set according to the `Mode` are set, the user is responsible for transferring the McBSP data in the appropriate format. An encoder-based Processing-on-the-fly correction activated for the z axis is retained.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Last change DLL 617, OUT 617: compatibility with "Fly Extension" Commands.
References	**set_mcbsp_in_list**

Normal List Command	**set_mcbsp_in_list**
Function	Like **set_mcbsp_in**, but a list command.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_mcbsp_in_list** terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	`set_mcbsp_in_list( Mode, Scale )`
Parameters	`Mode` Like **set_mcbsp_in**.
	`Scale` Like **set_mcbsp_in**.
Multi-board Com'd Name	**n_set_mcbsp_in_list**
Comments	• If `Mode` > 5, then **set_mcbsp_in_list** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • See **set_mcbsp_in**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Last change DLL 617, OUT 617: compatibility with "Fly Extension" Commands.
References	**set_mcbsp_in**

Ctrl Command	set_mcbsp_matrix
Function	Activates matrix correction for "Local Online Positioning" by the McBSP interface.
Call	`set_mcbsp_matrix()`
Multi-board Com'd Name	**n_set_mcbsp_matrix**
Comments	• For "Local Online Positioning", see Chapter 8.3.1 ""Local Online Positioning"", page 237. • Matrix corrections cannot be used in conjunction with offset and/or rotation corrections. Any such already-activated options gets deactivated by **set_mcbsp_matrix**. Subsequent activation of other options (by **set_mcbsp_x**, **set_mcbsp_y** or **set_mcbsp_rot**) deactivates the matrix correction. • The following restrictions apply to the matrix coefficients transferred over the McBSP interface (as with **set_matrix**): The allowed value range for matrix coefficients is [–50…+50]. Transferred coefficients exceeding this range are ignored. • You must individually supply as input value M_{in} to the McBSP interface each matrix coefficient Mij of the transformation matrix M_T as a normalized integer with associated indices i and j as follows: M_{in} = (integer(Mij * 2^{24}) << 2) + (i << 1) + j with M_T = { M00, M01, M10, M11 } = { m_{11}, m_{12}, m_{21}, m_{22} }. Conversely, the RTC6 determines a coefficient from the input value as follows: M_T[M_{in} & 0x3] = (M_{in} >> 2) / 2^{24}. • You must separately fetch each transferred matrix coefficient by **apply_mcbsp** or **apply_mcbsp_list**. We recommend fetching the first coefficient with `at_once` = 0 and only the last one with `at_once` > 0. • The coefficients get transferred to internal memory location 1 and can be checked there by querying with `read_mcbsp(1)`. • The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section "Notes", page 239.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_mcbsp_matrix_list**

Undelayed Short List Command	**set_mcbsp_matrix_list**
Function	Like **set_mcbsp_matrix**, but a list command.
Call	`set_mcbsp_matrix_list()`
Multi-board Com’d Name	**n_set_mcbsp_matrix_list**
Comments	• See **set_mcbsp_matrix**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_mcbsp_matrix**

Undelayed Short List Command	set_mcbsp_out
Function	Defines two signal types for output at the McBSP interface, see also Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87.
Call	`set_mcbsp_out( Signal1, Signal2 )`
Parameters	`Signal1` To-be-outputted signal type. As an unsigned 32-bit value.
	`Signal2` To-be-outputted signal type. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_mcbsp_out**
Comments	• The selectable signal types are identical to those of **set_trigger** (refer to the comments there for the allowed value range, signal types and other information). If the value for `Signal1`/`Signal2` is unallowed, then **set_mcbsp_out** is replaced by **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • Both selected data signals are continuously transmitted (once per 10 µs cycle). • Only 16-bit portions of the selected data signals are packed into a common 32-bit data word for output. `Signal1` is the lower half and `Signal2` the upper half of this 32-bit data word. – Only RTC4-compatible Bit #4…Bit #19 of the sample values and status values returned by the scan system (`Signal1, Signal2 =` 1…23, 25…30) are outputted. The least significant 4 bits and any exceeding bits (in the virtual Image Field) are ignored. – For the other data types (`Signal1, Signal2 =` 0, 24, 31…57), the least significant 16 bits are outputted and the upper bits are ignored. – McBSP_Output = ((Out2 & 0x0000FFFF) << 16) | (Out1 & 0x0000FFFF)); • Transmitted values are those of the preceding clock cycle: data is processed at the end of a cycle and transmitted at the beginning of the next clock cycle at the set transmission frequency (see **set_mcbsp_freq**). • The signals and operating conditions of the McBSP interface are presented in Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**read_mcbsp**, **mcbsp_init**, **set_mcbsp_freq**, **set_mcbsp_out_ptr**, **set_trigger**

Ctrl Command	set_mcbsp_out_oie_ctrl
Function	Switches on the OIE Output Mode at the McBSP interface.
Call	`set_mcbsp_out_oie_ctrl( Signal1, Signal2 )`
Parameters	`Signal1` To-be-outputted signal type. As an unsigned 32-bit value.
	`Signal2` To-be-outputted signal type. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_mcbsp_out_oie_ctrl**
Comments	• OIE Output Mode At the McBSP interface, a 32-bit data word of the following structure is outputted permanently (= once every 10 μs clock cycle): – Bit #00…Bit #11 Free variable 0, clipped to 12 bit. – Bit #12…Bit #13 0. – Bit #14…Bit #21 `Signal1`, scaled to [–128…+127]. – Bit #22…Bit #29 `Signal2`, scaled to [–128…+127]. – Bit #30 BUSY list execution status, siehe **get_status**. – Bit #31 LASERON signal. • The selected signal types for `Signal1` and `Signal2` are identical to **set_trigger**. A valid position signal (with value range [–524,288…+524,287]) should be selected for both signal types. • **set_mcbsp_out_oie_ctrl** is not executed with an unallowed parameter value (**get_last_error** return code `RTC6_PARAM_ERROR`).
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 633, OUT 634.
References	**set_mcbsp_out_oie_list**

Undelayed Short List Command	set_mcbsp_out_oie_list
Function	Like set_mcbsp_out_oie_ctrl, but a list command.
Call	`set_mcbsp_out_oie_list( Signal1, Signal2 )`
Parameters	`Signal1` Like set_mcbsp_out_oie_ctrl.
	`Signal2` Like set_mcbsp_out_oie_ctrl.
Multi-board Com'd Name	n_set_mcbsp_out_oie_list
Comments	• See set_mcbsp_out_oie_ctrl. • With an unallowed parameter value, set_mcbsp_out_oie_list is replaced by a list_nop (get_last_error return code `RTC6_PARAM_ERROR`).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 633, OUT 634.
References	set_mcbsp_out_oie_ctrl

Ctrl Command	set_mcbsp_out_ptr	
Function	Defines a list of up to 8 signal types for output at the McBSP interface, see Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87.	
Call	`set_mcbsp_out_ptr( Number, SignalPtr )`	
Parameters	`Number`	As an unsigned 32-bit value. Allowed value range: [0…8]. = 1…8: Number of signal types to be outputted. = 0 Output at the McBSP interface occurs in accordance with the predefined settings or as specified by a prior **set_mcbsp_out**. Bit 31 = 0: Signals are outputted as 24-bit values. Bit 31 = 1: Signals are outputted as 23-bit values. The most significant bit always contains the LASERON status.
	`SignalPtr`	Pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 64-bit value) to an array of `Number` unsigned 32-bit values, where the to-be-outputted `Number` signal type numbers are specified.
Multi-board Com'd Name	**n_set_mcbsp_out_ptr**	
Comments	• The memory area for the `SignalPtr` array must be provided by the user program. • If `Number` > 8 and/or `SignalPtr` = NULL, **set_mcbsp_out_ptr** is *not* executed (**get_last_error** return code `RTC6_PARAM_ERROR`). • The selectable signal types are identical to those of **set_trigger** (refer to the comments there for the allowed value range, signal types and other information). • The up to 8 selected data types are outputted sequentially (one data type per 10 µs clock cycle). Each individual signal belongs to a different clock cycle. Transmitted values are always from the previous clock cycle. The list is repeated until the output is switched off or replaced by another list. • When outputting, each signal value is supplemented by the corresponding signal type number: the signal type number gets inserted into the lowest byte of the 32-bit data word. The actual signal value therefore gets shifted 8 bits to the left, whereby all bits above the 24th get truncated (overflow, no clipping, relevant only for signal types 24, 31 and 37…57): 32-bit output value = (signal value << 8) | (signal type number & 0xFF). • The signals and operating conditions of the McBSP interface are presented in Chapter 4.6.6 "McBSP/ANALOG Socket Connector", page 87. • `Number` Bit 31 is needed for the OIE PID control, for example. • The following applies to `Number`-Bit 31 = 1: At the McBSP interface, the LASERON status is outputted 10 µs later than as recorded by **set_trigger**(`Signal1` = 0). • **set_mcbsp_out_ptr** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.	

Ctrl Command	set_mcbsp_out_ptr
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 632, OUT 633. `Number` Bit 31.
References	**set_mcbsp_out_ptr_list**, **set_mcbsp_out**, **set_trigger**

Multiple List Command	set_mcbsp_out_ptr_list	
Function	Like **set_mcbsp_out_ptr**, but a list command.	
Call	`set_mcbsp_out_ptr_list( Number, SignalPtr )`	
Parameters	`Number`	Like **set_mcbsp_out_ptr**.
	`SignalPtr`	Like **set_mcbsp_out_ptr**.
Multi-board Com'd Name	**n_set_mcbsp_out_ptr_list**	
Comments	• **set_mcbsp_out_ptr_list** requires two RTC6 List Memory positions for `Number` ≥ 1. Sequence of execution: 1. Pending Delayed Short List Commands 2. First command part as an Undelayed Short List Command 3. Second command part as a Normal List Command • See **set_mcbsp_out_ptr**.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 632, OUT 633.	
References	**set_mcbsp_out_ptr**	

Ctrl Command	set_mcbsp_rot
Function	Activates or deactivates rotation correction for "Local Online Positioning" by the McBSP interface.
Call	`set_mcbsp_rot( Resolution )`
Parameters	`Resolution` As a 64-bit IEEE floating point value. 2^{-26} < `Resolution` < 2^{26}: scaling factor (correction is activated), otherwise: correction is deactivated. `Resolution` = McBSP bits per full circle.
Multi-board Com'd Name	**n_set_mcbsp_rot**
Comments	• For "Local Online Positioning", see Chapter 8.3.1 ""Local Online Positioning"", page 237. • For an McBSP rotation correction input value of Rot_{in}, **apply_mcbsp** functions like `set_angle( …, Angle × 360°,…)`, whereby Angle in full circles = Rot_{in} / `Resolution` Only `Angle` values in the range [0.0…+20.0 full circles] are allowed. • The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section "Notes", page 239.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_angle**, **set_mcbsp_rot_list**, **set_mcbsp_x**, **set_mcbsp_y**, **apply_mcbsp**

Undelayed Short List Command	set_mcbsp_rot_list
Function	Like **set_mcbsp_rot**, but a list command.
Call	`set_mcbsp_rot_list( Resolution )`
Parameters	`Resolution` Like **set_mcbsp_rot**.
Multi-board Com'd Name	**n_set_mcbsp_rot_list**
Comments	• See **set_mcbsp_rot**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_mcbsp_rot**

Ctrl Command	set_mcbsp_x
Function	Activates or deactivates X offset correction for "Local Online Positioning" by the McBSP interface.
Call	`set_mcbsp_x( Scale )`
Parameters	`Scale` As a 64-bit IEEE floating point value. 2^{-26} < `Scale` < 2^{26}: scaling factor (correction is activated). Otherwise: correction is deactivated.
Multi-board Com'd Name	**n_set_mcbsp_x**
Comments	• For "Local Online Positioning", see Chapter 8.3.1 ""Local Online Positioning"", page 237. • With an McBSP input value of X_{in} for the X offset correction, **apply_mcbsp** functions like `set_offset( …, XOffset,…)`, whereby `XOffset` (in bits) = `Scale` × X_{in} `XOffset` values outside of [–524,288…+524,287] are clipped to the boundaries as long as `Scale` × X_{in} does not exceed the value range $\pm 2^{31}$. • The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section "Notes", page 239.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_mcbsp_x_list**, **set_mcbsp_y**, **set_mcbsp_rot**, **apply_mcbsp**

Undelayed Short List Command	set_mcbsp_x_list
Function	Like **set_mcbsp_x**, but a list command.
Call	`set_mcbsp_x_list( Scale )`
Parameters	`Scale` Like **set_mcbsp_x**.
Multi-board Com'd Name	**n_set_mcbsp_x_list**
Comments	• See **set_mcbsp_x**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_mcbsp_x**

Ctrl Command	set_mcbsp_y
Function	Activates or deactivates y offset correction for "Local Online Positioning" by the McBSP interface.
Call	`set_mcbsp_y( Scale )`
Parameters	`Scale` As a 64-bit IEEE floating point value. 2^{-26} < `Scale` < 2^{26}: scaling factor (correction is activated). Otherwise: correction is deactivated.
Multi-board Com'd Name	**n_set_mcbsp_y**
Comments	• For "Local Online Positioning", see Chapter 8.3.1 ""Local Online Positioning"", page 237. • With an McBSP input value of Y_{in} for the y offset correction, **apply_mcbsp** functions like `set_offset( …, YOffset,…)`, whereby `YOffset` (in bits) = `Scale` × Y_{in} `YOffset` values outside of [–524,288…+524,287] are clipped to the boundaries as long as `Scale` × Y_{in} does not exceed the value range $\pm 2^{31}$. • The McBSP interface cannot be simultaneously used for an Online Positioning and Processing-on-the-fly applications. See also Section "Notes", page 239.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_mcbsp_y_list**, **set_mcbsp_x**, **set_mcbsp_rot**, **apply_mcbsp**

Undelayed Short List Command	set_mcbsp_y_list
Function	Like **set_mcbsp_y**, but a list command.
Call	`set_mcbsp_y_list( Scale )`
Parameters	`Scale` Like **set_mcbsp_y**.
Multi-board Com'd Name	**n_set_mcbsp_y_list**
Comments	• See **set_mcbsp_y**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_mcbsp_y**

Ctrl Command	set_multi_mcbsp_in
Function	Activates a Processing-on-the-fly application and McBSP interface multi transmission with up to 8 different data types.
Restriction	If the Option Processing-on-the-fly is not enabled, then **set_multi_mcbsp_in** switches off the Processing-on-the-fly process (even if it has been never switched on).
Call	`set_multi_mcbsp_in( Ctrl, P, Mode )`
Parameters	`Ctrl` Control parameter for initializing or deactivating laser power variation. As an unsigned 32-bit value. = 1…6: defines which signal parameter to vary: for a description, see **set_auto_laser_control**. = 0 or > 6: deactivates laser power variation (for `Ctrl` > 6: **get_last_error** return code `RTC6_PARAM_ERROR`). `P` Initialization value for laser power. As an unsigned 32-bit value. Allowed value range: see **set_auto_laser_control**. `Mode` As an unsigned 32-bit value. = 0: The transmitted value is used directly. = 1: The transmitted value is multiplied by `P`/16384 and then put out.
	`Ctrl`, `P` and `Mode` are only relevant for Type 3 (= laser power) of the transmitted data word, see **read_multi_mcbsp**.
Multi-board Com'd Name	**n_set_multi_mcbsp_in**
Comments	• Up to V1.6.0, the following applies: Any other already activated McBSP transmission for Online Positioning and any other activated Processing-on-the-fly application with encoder signals or positional values is terminated first. As of V1.6.1, the following applies: Processing-on-the-fly corrections are compatible with the "Fly Extension" Commands. The Processing-on-the-fly corrections of the 3 axes are correspondingly overwritten. They can also be overwritten later by other corrections, for example, encoder-based Processing-on-the-fly corrections. Data types 0…2 are then not used for Processing-on-the-fly corrections. Hence, transmission of extra parameters can be combined even with encoder-based Processing-on-the-fly corrections. However, the McBSP memory transfer cannot be changed. • **set_multi_mcbsp_in** enables inputting by the McBSP interface of up to 8 different types for asynchronous transmission every 10 μs. Each 10 μs, the data is copied in accordance with its type code to separate memory, from where it can also be queried by **read_multi_mcbsp**. • To avoid faulty sorting, no active McBSP transmission should be occurring when you call **set_multi_mcbsp_in**. • The McBSP interface ignores the first FrameSync signal after a **load_program_file** or **mcbsp_init**. That is, data provided is not transmitted, see page 89.

Ctrl Command	set_multi_mcbsp_in
Comments (cont'd)	• For the transmission, the type assignment must be coded into the 3 least significant bits of the data word according to: – McBSPValue = (Value << 3) | Type The RTC6 board stores the data word according to: – Memory[McBSPValue & 0x7] = (long) McBSPValue >> 3 • For more on type assignments, see **read_multi_mcbsp**. • The remaining (most significant) 29 bits are available for the data word itself. There are no further restrictions other than the type-dependent value ranges themselves. The transmitted values are not checked with respect to their ranges. Clipping or data overflow may occur. • The 4 extra parameters are not the same as the free variables, see Chapter 6.9.1 "Free Variables", page 138, although they can be used for similar purposes. Upon program start, they are initialized with 0 and this command does not further modify them. The transmitted values merely get copied into type-sorted memory. • If more than 4 McBSP transfers complete within a 10 µs clock cycle, then prior values might get overwritten. • If the Option Processing-on-the-fly is enabled, then **set_multi_mcbsp_in** activates a Processing-on-the-fly application with positional values for the three coordinate directions x, y and z. The memory values of their respective types are initialized with 0. • Additionally, laser power can be varied by outputting the transmitted type 3 value at the port assigned by the `Ctrl` parameter. The initialization value `P` gets put out immediately. • For `Mode = 0`, subsequent type-3 transfers are outputted directly at the port assigned by `Ctrl`. For `Mode = 1`, the transferred value is handled as a multiplication factor in accordance with Normalization 1.0 = 16384 (14 bits). Thus, the following is put out: (`P` × the transmitted value / 16384). This mode is an alternative to laser power variation by "freely definable wobbel shapes". • These laser power variation method can be combined with the Vector-Defined Laser Control (with parameterized commands). It cannot be combined with other "Automatic Laser Control" methods. It overwrites other variations sharing the same `Ctrl` parameter. Though unidentical `Ctrl` parameters are allowed, they serve no practical purpose. • If `Ctrl` = 0, then laser power variation is switched off. The values transmitted by McBSP continue to be copied into type-sorted memory, but are not put out. • Special case: if a "freely definable wobbel shape" is active, then `P` is always regarded (irrespective of the `Mode` parameter) as relative laser power and multiplicatively coupled with the wobbel-shape's laser power (see **set_wobbel_vector**). Because this wobbel shape defines its own laser power (see **set_wobbel_control**), the command's `Ctrl` parameter has no relevance. It should be set to an invalid value (for example, with `Ctrl` = 0) to avoid doubled or meaningless output.

Ctrl Command	set_multi_mcbsp_in
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 617, OUT 617: compatibility with "Fly Extension" Commands.
Comments	set_multi_mcbsp_in_list, read_multi_mcbsp, set_wobbel_control, set_wobbel_vector

Normal List Command	set_multi_mcbsp_in_list
Function	Like set_multi_mcbsp_in, but a list command.
Restriction	Like set_multi_mcbsp_in.
Call	`set_multi_mcbsp_in_list( Ctrl, P, Mode )`
Parameters	`Ctrl` Like set_multi_mcbsp_in. `P` Like set_multi_mcbsp_in. `Mode` Like set_multi_mcbsp_in.
Multi-board Com'd Name	n_set_multi_mcbsp_in_list
Comments	• See set_multi_mcbsp_in.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 617, OUT 617: compatibility with "Fly Extension" Commands.
References	set_multi_mcbsp_in, read_multi_mcbsp, set_wobbel_control, set_wobbel_vector

Variable List Command	**set_n_pixel**
Function	In Pixel Output Mode, executes `PortOutValue1` and `PortOutValue2` assigned to the **set_pixel** command `Number` times in immediate succession.
Call	`set_n_pixel( PortOutValue1, PortOutValue2, Number )`
Parameters	`PortOutValue1` Like **set_pixel**.
	`PortOutValue2` Like **set_pixel**.
	`Number` Number of pixels. As an unsigned 32-bit value. Allowed value range: [1…(2^{32}–1)]. 0 is automatically set to 1.
Multi-board Com'd Name	**n_set_n_pixel**
Comments	• For usage of **set_n_pixel**, see Chapter 8.7 "Pixel Output Mode", page 272. • Before the first **set_n_pixel** of a line, **set_pixel_line** must have been called. • **set_n_pixel** defines the parameters for the following `Number` (identical) **set_pixel** commands of an image line. For usage, see comments on **set_pixel**. • If only an individual pixel is to be defined, then **set_pixel** can be used as an alternative to **set_n_pixel**. **set_pixel** is synonymous with `set_n_pixel` (`Number` = 1). • Outside Pixel Output Mode (if **set_n_pixel** is not directly preceded by **set_pixel_line**, **set_pixel** or **set_n_pixel**), **set_n_pixel** is a Short List Command and otherwise ignored. Under some circumstances, a **list_continue** might be inserted, see Section "Normal List Command, Short List Command, Variable List Command and Multiple List Command", page 335.
RTC4→RTC6	New command. For RTC4 Compatibility Mode: see **set_pixel**.
RTC5→RTC6	Unchanged functionality. See **set_pixel**.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_pixel**, **set_pixel_line**, **set_pixel_line_3d**

Ctrl Command	set_offset	
Function	Defines: • The offset for coordinate transformations, see Chapter 8.2 "Coordinate Transformations", page 233. • The global offset in virtual Image Field, see Section "Coordinate Transformations in the Virtual Image Field", page 174	
Call	`set_offset( HeadNo, XOffset, YOffset, at_once )`	
Parameters	`HeadNo`	See **set_offset_xyz**.
	`XOffset`	See **set_offset_xyz**.
	`YOffset`	See **set_offset_xyz**.
	`at_once`	See **set_offset_xyz**.
Multi-board Com'd Name	**n_set_offset**	
Comments	• **set_offset** has the same effect as **set_offset_xyz**, but leaves `ZOffset` unchanged.	
RTC4→RTC6	Unchanged first functionality (offset definition). See **set_offset_xyz**.	
RTC5→RTC6	Unchanged functionality. See **set_offset_xyz**.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_offset_list**, **set_offset_xyz**, **set_angle**, **set_matrix**, **set_scale**	

Variable List Command	set_offset_list	
Function	Like **set_offset_xyz**, but a list command.	
Call	`set_offset_list( HeadNo, XOffset, YOffset, at_once )`	
Parameters	`HeadNo`	Like **set_offset_xyz**. However, `HeadNo` = 4 is not available.
	`XOffset`	Like **set_offset_xyz**.
	`YOffset`	Like **set_offset_xyz**.
	`at_once`	Like **set_offset_xyz_list**.
Multi-board Com'd Name	**n_set_offset_list**	
Comments	• –	
RTC4→RTC6	See **set_offset_xyz**.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_offset_xyz**	

Ctrl Command	set_offset_xyz	
Function	Defines: • The offset for coordinate transformations, see Chapter 8.2 "Coordinate Transformations", page 233. • The global offset in virtual Image Field, see Section "Coordinate Transformations in the Virtual Image Field", page 174	
Call	`set_offset_xyz( HeadNo, XOffset, YOffset, ZOffset, at_once )`	
Parameters	`HeadNo`	Number of the scan head connector. As an unsigned 32-bit value. = 1: The definition only affects the Connector for First Scan Head. = 2: The definition only affects the Connector for Second Scan Head. = 0, 3: The definition affects *both* scan head connectors. = 4: The definition affects the virtual Image Field. Only the three least significant bits are evaluated.
	`XOffset`	Offsets for the x direction (coordinate translation relative to the origin of the Cartesian coordinate system). In bits. As a signed 32-bit value. Allowed value range: [–268,435,456…+268,435,455]. Out-of-range values are clipped to the boundary values.
	`YOffset`	Like `XOffset` (analogously).
	`ZOffset`	Offset for the z direction (coordinate translation relative to the origin of the Cartesian coordinate system). In bits. As a signed 32-bit value. Allowed value range: [–524,288…+524,287]. Out-of-range values are clipped to the boundary values.
	`at_once`	Like **set_matrix**.
Multi-board Com'd Name	**n_set_offset_xyz**	
Comments	• `ZOffset` is stored only once, applying jointly for both scan head connectors (`HeadNo` is therefore ignored). • `ZOffset` ≠ 0 causes a shift of the working plane (opposite to the direction of the laser beam). A positive value increases the z coordinate. For a hypothetical control value of(0|0|0), this would be by d = `ZOffset` / *K* to the plane z = +d. On the calibration factor *K*, see Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 172; furthermore, also #3 and #5 in Chapter 7.3.6 "Output Values to the Scan System", page 186. • If `HeadNo` = 4, then `ZOffset` is ignored. • For `at_once`, see **set_matrix**. • **set_offset_xyz**(`HeadNo` = 4) is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.	

Ctrl Command	set_offset_xyz
RTC4→RTC6	New command. In RTC4 Compatibility Mode,the RTC6 multiplies the specified values for `XOffset`, `YOffset`, `ZOffset` by 16. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality. In RTC4 Compatibility Mode,the RTC6 multiplies the specified value for `ZOffset` by 16. The allowed value range decreases accordingly.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_offset_xyz_list**, **set_offset**, **set_angle**, **set_matrix**, **set_scale**, **set_defocus**

Variable List Command	set_offset_xyz_list	
Function	Like **set_offset_xyz**, but a list command.	
Call	`set_offset_xyz_list( HeadNo, XOffset, YOffset, ZOffset, at_once )`	
Parameters	`HeadNo`	Like **set_offset_xyz**. However, `HeadNo` = 4 is not available.
	`XOffset`	Like **set_offset_xyz**.
	`YOffset`	Like **set_offset_xyz**.
	`ZOffset`	Like **set_offset_xyz**.
	`at_once`	Like **set_matrix_list**.
Multi-board Com'd Name	**n_set_offset_xyz_list**	
Comments	• –	
RTC4→RTC6	See **set_offset_xyz**.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_offset_xyz**, **set_offset_list**, **set_defocus_list**	

Ctrl Command	set_pause_list_cond
Function	Defines the condition at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector under which a **pause_list** automatically is executed.
Call	`set_pause_list_cond( Mask1, Mask0 )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the least 16 bits are evaluated. `Mask0` As `Mask1`.
Multi-board Com'd Name	**n_set_pause_list_cond**
Comments	• If a list is currently executed and the following condition is met, see also Chapter 9.3.2 "Execution of Conditional Commands", page 317: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits in IOValue specified in `Mask1` are 1 and the bits specified in `Mask0` are 0), then automatically a **pause_list** is executed. The condition is checked once per 10 μs clock cycle. • The paused list can only be continued by **restart_list**. • `Mask1` = `Mask0` = 0 disables the condition. • If the condition is disabled or no list is currently executed, nothing else happens. • With **set_pause_list_cond** in the case of an error the currently executed list can be paused immediately by a hardware circuit. This avoids using a time critical call of a command via the operating system. • A conditional **pause_list** takes precedence over a simultaneously present /STOP signal.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 609, OUT 609, RBF 614.
References	**pause_list**, **restart_list**, **set_pause_list_not_cond**

Ctrl Command	**set_pause_list_not_cond**
Function	Defines the "NOT" condition at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector under which a **pause_list** automatically is executed.
Call	`set_pause_list_not_cond( Mask1, Mask0 )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the least 16 bits are evaluated. `Mask0` Like `Mask1`.
Multi-board Com'd Name	**n_set_pause_list_not_cond**
Comments	• If a list is currently executed and the following condition is *not* met conditional commands, see also Chapter 9.3.2 "Execution of Conditional Commands", page 317: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits in IOValue specified in `Mask1` are 1 and the bits specified in `Mask0` are 0), then automatically a **pause_list** is executed. The condition is checked once per 10 µs clock cycle. • The paused list can only be continued by **restart_list**. • `Mask1` = `Mask0` = 0 disables the condition. • If the condition is disabled or no list is currently executed, nothing else happens. • With **set_pause_list_not_cond** in the case of an error the currently executed list can be paused immediately by a hardware circuit. This avoids using a time critical call of a command via the operating system. • A conditional **pause_list** takes precedence over a simultaneously present /STOP signal.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 610, OUT 610, RBF 615.
References	**pause_list**, **restart_list**, **set_pause_list_cond**

Variable List Command	**set_pixel**
Function	In the Pixel Output Mode, defines the laser control parameters (in "Classic Mode" pulse length and analog voltage level) for **one** pixel in an image line.
Call	`set_pixel( PortOutValue1, PortOutValue2 )`
Parameters	`PortOutValue1` For the meaning, see Chapter 8.7.3 "Laser Control", page 274. As an unsigned 32-bit value.
	`PortOutValue2` Like `PortOutValue1`.
Multi-board Com'd Name	**n_set_pixel**
Comments	• **set_pixel** is synonymous with **set_n_pixel** with `Number` = 1 (see comments there).
RTC4→RTC6	• The RTC6 does *not* support querying of analog voltage levels (see the RTC4's `ADChannel` parameter and **read_pixel_ad** command). • RTC4-Pixel Mode 0 is no longer supported. • "Classic Mode" (compatible to RTC4 Pixel Output Mode 1): In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `PulseLength` (`PortOutValue1`) by 8 and the one for `AnalogOut` (`PortOutValue2`) by 4. The allowed value ranges decrease accordingly. The other Pixel Output Modes are not RTC4 compatible.
RTC5→RTC6	Unchanged functionality. Only applies to "Classic Mode", since the other Pixel Output Modes are not supported by the RTC5.
Version Info	Last change DLL 604, OUT 604, RBF 609.
References	**set_n_pixel**, **set_pixel_line**, **set_pixel_line_3d**

Normal List Command	**set_pixel_line**	
Function	Activates a Pixel Output Mode and defines various pixel output parameters.	
Call	`set_pixel_line( Channel, HalfPeriod, dX, dY )`	
Parameters	`Channel`	Defines the Pixel Output Mode as well as the output ports, where the pixel contents are to be outputted. The pixel contents by themselves are defined in **set_pixel** or **set_n_pixel** commands. These must directly follow **set_pixel_line**. The following applies: `Channel` = Mode + Port. As an unsigned 32-bit value. Allowed are: Mode = 0: Output of `PortOutValue1` to the pixel duration (formerly parameter `PulseLength`) and output of `PortOutValue2` to the analog output port Port. The galvanometer scanner motion is not continuous, see Section "RTC5→RTC6", page 779. "Classic Mode". Mode = 16: Two outputs at Port. One 32-bit pixel in each output. "Extended Mode". Mode = 32: Two outputs at Port. Two 16-bit pixels in each output. "Fast Mode". Mode = 64: Two outputs at Port. 4 8-bit pixels in each output. "Ultra Fast Mode". Mode = 256: Like Mode = 0. However, the galvanometer scanner motion is continuous, see Section "RTC5→RTC6", page 779. Mode + 512: The galvanometer scanner motion with Sky Writing (position-accurate pixel line). Cannot be combined with Mode = 256. See also Chapter 8.7.4 "Synchronization", page 277 and Figure 65. Port = 1: 12-Bit Analog Output Port 1 (LASER Connector). See comment, page 779. Port = 2: 12-Bit Analog Output Port 2 (LASER Connector. With RTC6 PCIe Board also at the MARKING ON THE FLY Socket Connector). See comment, page 779. Port = 3: 8-Bit Digital Output Port (EXTENSION 2 Socket Connector). Port = 4: 16-Bit Digital Output Port (EXTENSION 1 Socket Connector). Port = 5: Output of `PortOutValue1` and `PortOutValue2` of **set_pixel** and of **set_n_pixel** to the pixel duration (formerly parameter `PulseLength`). Not allowed with Mode = 0 and Mode = 256.

Normal List Command	set_pixel_line	
Parameters (cont'd)	HalfPeriod	*Half* pixel output period. In bits. 1 bit equals 1/64 µs. As an unsigned 32-bit value. Allowed value range: [Min…(2^{32}–1)] with: Min = 80 for Mode = 0 and 256. Maximum frequency: 0.4 MHz. Min = 40 for Mode = 16. Maximum frequency: 0.8 MHz. Min = 20 for Mode = 32. Maximum frequency: 1.2 MHz. For RTC6 Boards without Option "UFPM" the following applies: Min = 40, maximum frequency: 0.8 MHz. Min = 10 for Mode = 64. Maximum frequency: 3.2 MHz. For RTC6 Boards Option "UFPM" the following applies: Min = 40, maximum frequency: 0.8 MHz.
	dX	Distance in the x direction between adjacent pixels. In bits. As a 64-bit IEEE floating point value.
	dY	Distance in the Y direction between adjacent pixels. In bits. As a 64-bit IEEE floating point value.
Multi-board Com'd Name	n_set_pixel_line	
Comments	• Each image line of a pixel image must be started by **set_pixel_line**. **set_pixel_line** should be preceded by a Jump Command or [*]mark[*] Command to the start point of the image line. • Directly after **set_pixel_line**, the required number of **set_pixel** and **set_n_pixel** commands must follow. These transmit the PortOutValue1 and PortOutValue2 parameters. Their meaning depend from Mode and Port (see above, parameter Channel). See also Chapter 8.7.3 "Laser Control", page 274. • The first list command after **set_pixel_line** that is *not* a **set_pixel** or **set_n_pixel** command turns off the Pixel Output Mode. **set_pixel_line**, too, ends the Pixel Output Mode before starting it again. In the process, a default pixel is inserted. • The default pixel for Port 1…4 is defined by **set_port_default**. But beware: Port numbers of **set_port_default** and **set_pixel_line** do not match! The default pixel for Port 5 (PulseLength) is defined by **set_default_pixel**. • With unallowed Channel values (example: 5, for Mode = 0 and Port = 5) **set_pixel_line** is replaced by a **list_nop** (**get_last_error** return code RTC6_PARAM_ERROR). The Pixel Output Mode is *not* activated in these cases. • Note that *half* the period length must be specified for HalfPeriod. 2 × HalfPeriod is thus the chronological distance between individual pixels, see Chapter 8.7 "Pixel Output Mode", page 272. HalfPeriod must not be smaller than Min (is automatically clipped).	

Normal List Command	set_pixel_line
Comments (cont'd)	• For outputs at 12-Bit Analog Output Port 1 (Port = 1) and 12-Bit Analog Output Port 2 (Port = 2) the following must be obeyed: only for pixel output frequencies up to around 100 kHz (that is, for a `HalfPeriod` < approx. 320) digital-to-analog conversion is always fully completed. With such pixel output frequencies users must carefully verify whether the results are as expected. With pixel output frequencies > 100 kHz, usage of the UFPM Extension Board, see Chapter 17 "Appendix B: UFPM Extension Board", page 1044, is recommended. • The Pixel Output Mode can be combined with Processing-on-the-fly. See also Chapter 8.6 "Processing-on-the-fly", page 251. • The Pixel Output Mode *should not* be used in conjunction with "Automatic Laser Control" (see Chapter 7.4.9 ""Automatic Laser Control"", page 205), if there is a readjustment of the port output, pulse length (`PulseLength`) or output period (`HalfPeriod`) of the Laser Control Signals LASER1 and LASER2. • The Pixel Output Mode is incompatible with the Softstart Mode (not yet implemented). • The Pixel Output Mode *cannot* be combined with Sky Writing. However, a Sky Writing Mode 1-like motion can be set by Mode + 512. This requires Sky Writing to be switched on. In case of a SCANahead System, the automatic delay calculation must be switched on in addition. This allows pixel lines to be positioned accurately. This Sky Writing motion cannot be combined with Sky Writing Mode 2 motions. See also: – Chapter 8.7.4 "Synchronization", page 277 and Figure 65 – Chapter 7.2.4 "Sky Writing", page 163 • The Pixel Output Mode *cannot* be combined with Wobbel. See also Chapter 8.4 "Wobbel Mode", page 241. • See also Chapter 8.7 "Pixel Output Mode", page 272.
RTC4→RTC6	• For differences to the Pixel Output Mode of the RTC4, see Section "Special Functions", page 58. • In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `HalfPeriod` by 8 and those for `dX` and `dY` by 16. The allowed value ranges decrease accordingly.
RTC5→RTC6	In "Classic Mode" (RTC5 compatible), frequencies up to 400 kHz are possible. Furthermore, there are additional Pixel Output Modes.
Version Info	Last change DLL 604, OUT 604, RBF 609. Additional Pixel Output Modes: 16, 32, 64 and +512.
References	**set_default_pixel**, **set_default_pixel_list**, **set_pixel**, **set_n_pixel**, **set_pixel_line_3d**, **set_port_default**

Multiple List Command	**set_pixel_line_3d**	
Function	Activates the Pixel Output Mode and defines various pixel output parameters.	
Call	`set_pixel_line_3d( Channel, HalfPeriod, dX, dY, dZ )`	
Parameters	`Channel`	Like **set_pixel_line**.
	`HalfPeriod`	Like **set_pixel_line**.
	`dX`	Like **set_pixel_line**.
	`dY`	Like **set_pixel_line**.
	`dZ`	Distance in the z direction between adjacent pixels. In bits. As a 64-bit IEEE floating point value.
Multi-board Com'd Name	**n_set_pixel_line_3d**	
Comments	• **set_pixel_line_3d** lets you define the pixel spacing (between two adjacent image points on a line) by a 3D vector. • **set_pixel_line_3d** occupies 2 RTC6 List Memory positions for `dZ` ≠ 0. The first part is executed as an Undelayed Short List Command before the principal part (a Normal List Command). Any still-pending Delayed Short List Command is executed beforehand. • In other respects, **set_pixel_line_3d** behaves similarly to **set_pixel_line**, see Comments, page 778.	
RTC4→RTC6	New command. RTC4 Compatibility Mode: see **set_pixel_line**. In RTC4 Compatibility Mode, the RTC6 *does not* multiply the specified value for `dZ` by 16.	
RTC5→RTC6	Basically unchanged functionality. RTC5 Compatibility Mode: see **set_pixel_line**.	
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change: see **set_pixel_line**.	
References	**set_pixel_line**, **set_pixel**, **set_n_pixel**	

Ctrl Command	set_port_default
Function	Defines the default output value for the selected output port.
Call	`set_port_default( Port, Value )`
Parameters	`Port` Output port for which a default `Value` is to be defined. As an unsigned 32-bit value. Allowed values: = 0: ANALOG OUT1 output port. See also Section "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 77. = 1: ANALOG OUT2 output port. See also Section "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 77. = 2: 8-Bit Digital Output Port. See also Section "8-Bit Digital Output Port", page 84. = 3: 16-Bit Digital Output Port. See also Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81. = 4: 2-Bit Digital Output Port. See also Section "2-Bit Digital Output Port", page 77.
	`Value` Default value. As an unsigned 32-bit value. Allowed values: For `Port` = 0/1: 12-bit values [0…4,095]. Bits with higher significance are ignored. For `Port` = 2: 8-bit values [0…255]. Bits with higher significance are ignored. For `Port` = 3: 16-bit values [0…65,535]. Bits with higher significance are ignored. For `Port` = 4: 2-bit values [0…3]. Bits with higher significance are ignored. For `Value` = "–1" (= 2^{32}–1 = 0xFFFFFFFF), the corresponding default output functionality is switched off (see comment below).
Multi-board Com'd Name	**n_set_port_default**
Comments	• During initialization of the RTC6 (by **load_program_file**), the default values of all output ports are set to "–1" (= 2^{32}–1). • If a default value ≠ "–1" has been specified for an output port, it is set to this default value as soon as processing of a list has ended with **stop_execution** or by an External Stop. For a default value of "–1", the corresponding output port is *not* addressed (any existing high-impedance state of the digital output ports is retained). • Default values (≠ "–1") at the output ports 0…3 are also set at the end of Pixel Output Mode, see Chapter 8.7 "Pixel Output Mode", page 272.

Ctrl Command	set_port_default
Comments (cont'd)	• After initialization of Position-Dependent Laser Control and/or Speed-Dependent Laser Control by **set_auto_laser_control**, the corresponding default value (for output port 0, 1, 2 or 3, depending on the selected laser control signal parameter `Ctrl`), is also outputted when the laser is switched off after marking or when Position-Dependent Laser Control and/or Speed-Dependent Laser Control is set to another `Ctrl` parameter by **set_auto_laser_control** or deactivated by **set_auto_laser_control** (`Ctrl` = 0), see Chapter 7.4.1 "Enabling, Activating and Switching Laser Control Signals", page 189 and Section "General Notes", page 207. If the default value is set to "–1", then the maximum allowed value (4,095, 4,095, 255 or 65,535) is outputted. • If the value for `Port` is invalid, then **set_port_default** is not executed (**get_last_error** return code `RTC6_PARAM_ERROR`). • The default values for the output ports 0…2 can also be defined by **set_laser_off_default**.
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies `Value` for `Port` = 0/1 by 4.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_laser_off_default**, **set_default_pixel**

Undelayed Short List Command	**set_port_default_list**
Function	Like **set_port_default**, but a list command.
Call	`set_port_default_list( Port, Value )`
Parameters	`Port` Like **set_port_default**.
	`Value` Like **set_port_default**.
Multi-board Com'd Name	**n_set_port_default_list**
Comments	• See **set_port_default**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 609, OUT 609, RBF 614.
References	**set_port_default**

Ctrl Command	set_pulse_picking
Function	Switches on Pulse Picking Laser Mode.
Call	`set_pulse_picking( No )`
Parameters	No — As an unsigned 32-bit value. Allowed value range: [0…63]. = 0: LASER2 puts out the LASERON signal. = 1…63: LASER2 puts out each Noth LASER1 pulse. No > 63: No is clipped to 63 (**get_last_error** return code RTC6_PARAM_ERROR).
Multi-board Com'd Name	**n_set_pulse_picking**
Comments	• For Pulse Picking Laser Mode, see Chapter 7.4.8 "Pulse Picking Laser Mode", page 203. • The pulse picking signals are outputted until a different laser mode gets set by **set_laser_mode** (the Pulse Picking Laser Mode gets switched off if any other laser mode switches on by **set_laser_mode**). • You can *not* switch on Pulse Picking Laser Mode by **set_laser_mode**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_laser_mode**, **set_pulse_picking_list**

Ctrl Command	set_pulse_picking_length
Function	Defines a constant pulse length for the LASER2 signal in Pulse Picking Laser Mode.
Call	`set_pulse_picking_length( Length )`
Parameters	`Length` Pulse length. As an unsigned 32-bit value. Allowed value range: [0…65,535]. Bits with higher significance are ignored. 1 bit equals 1/64 µs. The default value after **load_program_file** is 0.
Multi-board Com'd Name	**n_set_pulse_picking_length**
Comments	• For the value to take effect, the following must have been activated: – Pulse Picking Laser Mode by **set_pulse_picking** – The "constant pulse length" mode by **set_laser_control**(Bit #7 = 1). The value then takes immediate effect, even if a marking is carried out. • If Pulse Picking Laser Mode has been activated, but not constant pulse length mode (**set_laser_control**(Bit #7 = 0)), then the pulse-picking signal uses the pulse length of the LASER1 signal, see Chapter 7.4.8 "Pulse Picking Laser Mode", page 203. • If neither Pulse Picking Laser Mode nor constant pulse length mode has been activated, then the value `Length` is irrelevant (**set_pulse_picking**, **set_laser_control** and **set_pulse_picking_length** can be activated in any desired order). • After **set_pulse_picking**(0), LASER2 is continuously output the LASERON signal, but no constant pulse length signal. • **set_pulse_picking_length** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Undelayed Short List Command	set_pulse_picking_list
Function	Like **set_pulse_picking**, but a list command.
Call	`set_pulse_picking_list( No )`
Parameters	`No` Like **set_pulse_picking**.
Multi-board Com'd Name	**n_set_pulse_picking_list**
Comments	• –
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_pulse_picking**

Ctrl Command	set_qswitch_delay
Function	In a YAG Mode, defines the delay length of the first Q-Switch pulse with reference to the FirstPulseKiller signal, see also Figure 53.
Call	`set_qswitch_delay( Delay )`
Parameters	`Delay` Q-Switch delay. As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: [0…(2^{26}–1)].
Multi-board Com'd Name	**n_set_qswitch_delay**
Comments	• Values over (2^{26}–1) are clipped. • A YAG Mode is set by **set_laser_mode** ([1, 2, 3 or 5]). • With YAG Mode 1, YAG Mode 2, YAG Mode 3, the length of the Q-Switch delay can be subsequently changed by **set_qswitch_delay**; and for YAG Mode 2 also by **set_firstpulse_killer** and **set_firstpulse_killer_list**.
RTC4→RTC6	New command. In RTC4 Compatibility Mode,the RTC5 multiplies the specified value for `Delay` by 8. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_qswitch_delay_list**, **set_laser_control**, **set_laser_pulses_ctrl**, **set_laser_pulses**, **set_laser_timing**, **set_firstpulse_killer**, **set_firstpulse_killer_list**

Undelayed Short List Command	set_qswitch_delay_list
Function	Like **set_qswitch_delay**, but a list command.
Call	`set_qswitch_delay_list( Delay )`
Parameters	`Delay` Like **set_qswitch_delay**.
Multi-board Com'd Name	**n_set_qswitch_delay_list**
Comments	• –
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_qswitch_delay**

Ctrl Command	set_rot_center	
Function	Sets the rotation center of a Processing-on-the-fly rotation correction (see **set_fly_rot** and **set_fly_rot_pos**).	
Call	`set_rot_center( X, Y )`	
Parameters	X	Position of the rotation center referenced to the zero point (0|0) of the Image Field. As a signed 32-bit value. Allowed value range: [-2^{24}…(2^{24}–1)].
	Y	Like X (analogously).
Multi-board Com'd Name	**n_set_rot_center**	
Comments	• For Processing-on-the-fly correction, see Chapter 8.6.3 "Compensating Rotary Motions", page 256. • The position of the rotation center should be defined by **set_rot_center** or **set_rot_center_list** before the Processing-on-the-fly correction is activated by **set_fly_rot** or **set_fly_rot_pos**. • The rotation center can also lie outside the Image Field. The allowed area is equivalent to 32x the Image Field. • Usage of a second scan head is only practical if it is set up for exactly the same rotational center. • **set_rot_center** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.	
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for X and Y by 16. The allowed value range decreases accordingly.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_fly_rot**, **set_fly_rot_pos**	

Delayed Short List Command	set_rot_center_list	
Function	Like **set_rot_center**, but a list command.	
Call	`set_rot_center_list( X, Y )`	
Parameters	X	Like **set_rot_center**.
	Y	Like **set_rot_center**.
Multi-board Com'd Name	**n_set_rot_center_list**	
Comments	• –	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_rot_center**	

Ctrl Command	set_rtc4_mode
Function	Sets RTC4 Compatibility Mode as the current RTC6 DLL operation mode.
Call	`set_rtc4_mode()`
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• RTC4 Compatibility Mode is an optional operation mode of the RTC6 DLL. It has been made available so that user programs written for the RTC4 can also be processed by the RTC6 (to a large extent) without needing to modify the programming code. However, a prerequisite here is that the program can only contain RTC4 commands that also exist with unchanged functionality as RTC6 commands. In the Command Descriptions, page 342 of this user manual such changes are noted in the "RTC4→RTC6" row. • RTC6 Standard Mode is predefined as the default RTC6 DLL operation mode and can also be specified subsequently by **set_rtc6_mode**. • The current RTC6 DLL operation mode can be queried by **get_rtc_mode**. • **set_rtc4_mode** is available even without explicit access rights to a particular board. • The scope of **set_rtc4_mode** is not board-specific, but rather global to the RTC6 DLL and all RTC6 boards to which the user program has access rights. • The board-specific error variables `LastError` and `AccError` (see Chapter 6.8 "Error Handling", page 133) are neither generated nor altered by **set_rtc4_mode**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_rtc_mode**, **set_rtc5_mode**, **set_rtc6_mode**

Ctrl Command	set_rtc5_mode
Function	Sets RTC5 Compatibility Mode as the current RTC6 DLL operation mode.
Call	`set_rtc5_mode()`
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• RTC5 Compatibility Mode is an optional operation mode of the RTC6 DLL. It has been made available so that user programs written for the RTC5 can also be processed by the RTC6 (to a large extent) without needing to modify the programming code. However, a prerequisite here is that the program can only contain RTC5 commands that also exist with unchanged functionality as RTC6 commands. In the Command Descriptions, page 342 of this user manual such changes are noted in the "RTC5→RTC6" row. See also Chapter 2.12.2 "Adapting RTC5 Source Code for the RTC6 PCIe Board", step 3, page 63. • In RTC5 Compatibility Mode as RTC6 DLL operation mode Z coordinates and defocus values (for example, parameter `Shift` of **set_defocus**) must be specified with a parameter resolution of 16 bits. These values are automatically multiplied by 16. The allowed value ranges decrease accordingly. • In RTC5 Compatibility Mode as RTC6 DLL operation mode the laser delay values (parameter `LaserOnDelay` and `LaserOffDelay` of **set_laser_delays**) must be specified with a parameter resolution of 1/2 µs. These values are automatically multiplied by 32. The allowed value ranges decrease accordingly. • RTC6 Standard Mode is predefined as the default RTC6 DLL operation mode and can also be specified subsequently by **set_rtc6_mode**. • The current RTC6 DLL operation mode can be queried by **get_rtc_mode**. • **set_rtc5_mode** is available even without explicit access rights to a particular RTC6 board. • The scope of **set_rtc5_mode** is not board-specific, but rather global to the RTC6 DLL and all RTC6 boards to which the user program has access rights. • The board-specific error variables `LastError` and `AccError` (see Chapter 6.8 "Error Handling", page 133) are neither generated nor altered by **set_rtc5_mode**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_rtc_mode**, **set_rtc4_mode**, **set_rtc6_mode**

Ctrl Command	set_rtc6_mode
Function	Sets RTC6 Standard Mode as the current RTC6 DLL operation mode (default setting).
Call	`set_rtc6_mode()`
Multi-board Com’d Name	Not available as a multi-board command.
Comments	• In RTC6 Standard Mode as RTC6 DLL operation mode, z coordinate values and defocus values must be specified with a parameter resolution of 20 bits. • In RTC6 Standard Mode as RTC6 DLL operation mode, laser delay values must be specified with a parameter resolution of 1/64 μs. • The current RTC6 DLL operation mode can be queried by **get_rtc_mode**. • **set_rtc6_mode** is available even without explicit access rights to a particular RTC6 board. • The scope of **set_rtc6_mode** is not board-specific, but rather global to the RTC6 DLL and therefore, to all RTC6 boards to which the user program has access rights. • The board-specific error variables `LastError` and `AccError` (see Chapter 6.8 ”Error Handling”, page 133) are neither generated nor altered by **set_rtc6_mode**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_rtc_mode**, **set_rtc4_mode**, **set_rtc5_mode**

Ctrl Command	set_scale
Function	Uses a specified scaling factor common to the x axis and y axis to define the scaling matrix M_S for all subsequent coordinate transformations, see Chapter 8.2 "Coordinate Transformations", page 233.
Call	`set_scale( HeadNo, Scale, at_once )`
Parameters	`HeadNo` Number of the scan head connector. As an unsigned 32-bit value. = 1: The definition only affects the Connector for First Scan Head. = 2: The definition only affects the Connector for Second Scan Head. = 0, 3: The definition affects *both* scan head connectors. Only the two least significant bits are evaluated.
	`Scale` Scaling factor. As a 64-bit IEEE floating point value. Allowed value range: [–16…+16]. If the parameter is set to an invalid value, it is set to 1. Negative values additionally produce a mirroring around both axes (corresponding to a 180° rotation).
	`at_once` Determines when the defined transformation becomes effective. Like `set_matrix( HeadNo` = 0…3). As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_scale**
Comments	• See Chapter 8.2 "Coordinate Transformations", page 233.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_scale_list**, **set_angle**, **set_matrix**, **set_offset**

Variable List Command	set_scale_list
Function	Like **set_scale**, but a list command.
Call	`set_scale_list( HeadNo, Scale, at_once )`
Parameters	`HeadNo` Like **set_scale**.
	`Scale` Like **set_scale**.
	`at_once` Determines when the defined transformation becomes effective. Like **set_matrix_list**. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_scale_list**
Comments	• –
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_scale**

Ctrl Command	**set_scanahead_laser_shifts**
Function	Only for SCANahead Systems. Shifts the LASERON signals and LaserOff temporally forward or back.
Call	`set_scanahead_laser_shifts( dLasOn, dLasOff )`
Parameters	`dLasOn` Time shift of LASERON signal in [1/64 μs]. As a signed 32-bit value. `dLasOff` Time shift of LaserOff in [1/64 μs]. As a signed 32-bit value.
Multi-board Com'd Name	**n_set_scanahead_laser_shifts**
Comments	• **set_scanahead_laser_shifts** lets you *fine-tune* the Laser Delays, for example, to compensate laser signal propagation times or to take the laser system switching behavior into account. • If **set_scanahead_params** has not been already called and automatic calculation has not been activated (**activate_scanahead_autodelays** `Mode = 0`), then the values only get stored, but not applied. But they are applied as soon as required conditions are fulfilled. • There is a transport delay from the RTC6 board to the scan head of 20 μs. As of DLL 605 this is automatically taken into account. `dLasOn` and `dLasOff` parameter values optimized for < DLL 605 must be decreased by 20 μs to be suitable for ≥ DLL 605.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of version DLL 600, OUT 600. Change as of version DLL 603, OUT 603, RBF 607: increased parameter resolution is now 1/64 μs instead of 1/2 μs.
References	**set_scanahead_laser_shifts_list**, **set_scanahead_params**, **activate_scanahead_autodelays**

Undelayed Short List Command	set_scanahead_laser_shifts_list
Function	Only for SCANahead Systems. Like set_scanahead_laser_shifts, but a list command.
Call	`set_scanahead_laser_shifts_list( dLasOn, dLasOff )`
Parameters	`dLasOn` See set_scanahead_laser_shifts. `dLasOff` See set_scanahead_laser_shifts.
Multi-board Com'd Name	n_set_scanahead_laser_shifts_list
Comments	• **set_scanahead_laser_shifts_list** takes effect upon the next to-be-calculated delay.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of version DLL 600, OUT 600.
References	set_scanahead_laser_shifts

Ctrl Command	set_scanahead_line_params
Function	Only for SCANahead Systems. Influences the quality of marking results at runtime. Smaller percent values increase throughput at the expense of quality.
Call	`set_scanahead_line_params( CornerScale, EndScale, AccScale )`
Parameters	`CornerScale` Corner sharpness with mark/mark sequences in percent. 100% = sharp corners. As an unsigned 32-bit value. `EndScale` Marking accuracy at mark/jump and jump/mark transitions. 100% = straight line ends. As an unsigned 32-bit value. `AccScale` Determines the portion of the acceleration *time* (not: distance traversed) in which the laser is active, in percent. 100% = entire acceleration time. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_scanahead_line_params**
Comments	• Only values from 0% to 100% are useful. Higher values do not improve quality (that is, corners cannot be even sharper than sharp), but instead only extend marking times. • If **set_scanahead_params** has not been already called, then the values only get stored, but not applied. • The parameters `CornerScale` and `EndScale` only affect the galvanometer scanner positioning. They *do not* affect points in time when the laser is to be switched on and off. • The parameter `AccScale` only affects points in time when the laser is to be switched on and off. It *does not* affect the galvanometer scanner positioning! • If Sky Writing is activated, only the parameter `EndScale` is effective. • The other parameters `CornerScale` and `AccScale` are not applied – except there is *no* genuine Sky Writing motion: – If Sky Writing Mode 3 or Sky Writing Mode 4 is activated and the angular limit is undercut, `CornerScale` is effective. • The parameter `EndScale` is effective, if Sky Writing is activated (here, the following applies as well: smaller percent values increase throughput at the expense of quality).
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of version DLL 600, OUT 600.
References	**set_scanahead_line_params_list**

Undelayed Short List Command	set_scanahead_line_params_list
Function	Only for SCANahead Systems. Like set_scanahead_line_params, but a list command.
Call	`set_scanahead_line_params_list( CornerScale, EndScale, AccScale )`
Parameters	`CornerScale` Like set_scanahead_line_params. `EndScale` Like set_scanahead_line_params. `AccScale` Like set_scanahead_line_params.
Multi-board Com'd Name	n_set_scanahead_line_params_list
Comments	• set_scanahead_line_params_list takes effect upon the next to-be-calculated delay.
RTC4→RTC6	New command.
RTC5→ RTC6	New command.
Version Info	Available as of version DLL 600, OUT 600.
References	set_scanahead_line_params

Ctrl Command	set_scanahead_params
Function	Only for SCANahead Systems. Activates the SCANahead Functionality suitable to control an SCANahead System on the RTC6 board. The laser control and digital inputs/outputs for the peripherals are prepared for a SCANahead System, that is, delayed by `PreviewTime`. To have also the Scanner Delays and Laser Delays calculated automatically, **activate_scanahead_autodelays** must be called.
Restriction	**set_scanahead_params** can only be executed, if no list is currently active. Otherwise, **get_last_error** return code gets set to `RTC6_BUSY` and error code 5 is returned. **set_scanahead_params** can only be executed, if the Option "SCANa" is enabled. Otherwise, **get_last_error** return code gets set to `RTC6_REJECTED` and error code 1 is returned.
Call	`Error = set_scanahead_params( Mode, HeadNo, TableNo, PreviewTime, Vmax, Amax )`
Parameters	`Mode` = 0: Deactivates the SCANahead Functionality. The additional parameters `TableNo, PreviewTime, Vmax, Amax` are not taken into account. Default after **load_program_file**. Mode, for example, for operating intelli*SCAN* scan heads. = 1: Queries parameters of the scan head with the specified `HeadNo` and applies them. The parameters `PreviewTime, Vmax, Amax` are not taken into account. See also comments that follow. Mode for operating attached SCANahead Systems. = 2: Uses the parameters of **set_scanahead_params**. Mode, for example, for software development. A SCANahead System does not need to be connected. As an unsigned 32-bit value. `HeadNo` Number of the scan head connector. Allowed values: = 1: Connector for First Scan Head. = 2: Connector for Second Scan Head. Option "Second Scan Head Control" required. As an unsigned 32-bit value. `TableNo` Allowed value range: 1…8. A correction file should have been loaded by **load_correction_file**(`No`). This is used for converting the parameters `Amax` and `Vmax` from control bits (galvanometer scanner rotation angle) into programing bits (Image Field coordinates). See **get_scanahead_params**. As an unsigned 32-bit value.

Ctrl Command	set_scanahead_params
Parameters (cont'd)	`PreviewTime` Precalculation time for galvanometer scanner control. In 10 µs. `PreviewTime` is ignored in `Mode = 0` and `Mode = 1`. As an unsigned 32-bit value. `Vmax` Velocity limit as control bits (galvanometer scanner rotation angle). In bits/10 µs. `Vmax` is ignored in `Mode = 0` and `Mode = 1`. As an unsigned 32-bit value. `Amax` Acceleration limit as control bits (galvanometer scanner rotation angle). In bits/(10 µs)². `Amax` is ignored in `Mode = 0` and `Mode = 1`. As a 64-bit IEEE floating point value.
Result	Error code. As an unsigned 32-bit value. 1 Option "SCANa" is not present. (**get_last_error** return code: `RTC6_REJECTED`). 3 No SCANahead System attached or SCANahead System tuning is not active. No **get_last_error** error is set. 5 A list is currently active (**get_last_error** return code: `RTC6_BUSY`). 6 `HeadNo` = `0` and `HeadNo` >2 (**get_last_error** return code: `RTC6_PARAM_ERROR`). 7 The determined scaling factor exceeds 16 or is below 1/16 (possibly no correction file was loaded). The scaling factor is set to 1. 8 The RTC6 board is not responding. Probably a program has not been loaded yet (**get_last_error** return code: `RTC6_TIMEOUT`). 11 A PCI error occurred (**get_last_error** return code: `RTC6_SEND_ERROR`).
Multi-board Com'd Name	**n_set_scanahead_params**
Comments	• `Vmax` and `Amax` are used for automatic calculation of Scanner Delays and Laser Delays, see **activate_scanahead_autodelays**. • The values for `Vmax` and `Amax` are specified as control bits (galvanometer scanner rotation angles). Information is read out from the correction file and used for an internal conversion into programing bits (sample values). Therefore, the correction file needs to be specified in the `TableNo` parameter. At conversion time, the correction file must have been already loaded onto the RTC6 board, but does not yet need to be assigned. The system cannot reliably detect correction files that have not been loaded. The `Vmax` and `Amax` values which have been converted into programing bits can be queried by **get_scanahead_params**(`HeadNo= 256`).

Ctrl Command	set_scanahead_params
Comments (cont'd)	• For error return information, see **get_scanahead_params**. • `Mode = 1` results in aborted **set_scanahead_params** execution (`Error = 3`) if no SCANahead System is attached or it is not active. The SCANahead Functionality is deactivated. Parameters specified for `Mode = 2` are not applied! • **set_scanahead_params**(`Mode = 1`) cannot be executed when an automatic laser control is active that uses data transmitted by an *iDRIVE* scan system (see Glossary entry on page 28), see **set_auto_laser_control**. You have to deactivate it temporarily. • **set_scanahead_params** waits until the last session has finished traversing, that is, HEAD BUSY list execution status (Bit #23) from **get_status** is no longer set. In any case, **set_scanahead_params** waits no longer than 255 × [10 µs]. • **set_scanahead_params** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of version DLL 600, OUT 600. Last change with version DLL 605: data type of `Amax`.
References	**get_scanahead_params**, **activate_scanahead_autodelays**, **get_last_error**, **get_status**, **load_correction_file**

Ctrl Command	set_scanahead_speed_control
Function	Only for SCANahead Systems. Controls the Scanner Delays either depending on the target speed or actual achievable speed.
Call	`set_scanahead_speed_control( Mode )`
Parameters	`Mode` Mode. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_scanahead_speed_control**
Comments	• `Mode` with: – Bit #0 controls mark commands – Bit #1 controls jump commands – Bit #n = 0 controls the Scanner Delays depending on the target speed – Bit #n = 1 controls the Scanner Delays depending on the actual achievable speed • Default after **load_program_file**: `Mode` = 2. • **set_scanahead_speed_control** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of version DLL 610, OUT 610. Last change with version OUT 611: adjustment of Laser Delays.
References	–

Delayed Short List Command	set_scanner_delays	
Function	Sets the Scanner Delays.	
Call	`set_scanner_delays( Jump, Mark, Polygon )`	
Parameters	`Jump`	Scanner Delay of type: Jump Delay. As an unsigned 32-bit value. 1 bit equals 10 µs. Allowed value range: [0…(2^{32}–1)].
	`Mark`	Scanner Delay of type: Mark Delay. As an unsigned 32-bit value. 1 bit equals 10 µs. Allowed value range: [0…(2^{32}–1)].
	`Polygon`	Scanner Delay of type: Mark Delay. As an unsigned 32-bit value. 1 bit equals 10 µs. Allowed value range: [0…(2^{32}–1)].
Multi-board Com'd Name	**n_set_scanner_delays**	
Comments	• See also Chapter 7.2.2 "Scanner Delays", page 150. • Variable Delays for jumps and Polylines can be set by **set_delay_mode**. • The specified delays are automatically adjusted by the RTC6 to avoid laser control errors, see Section "Automatic Delay Adjustments", page 158. • The default setting after **load_program_file** corresponds to `set_scanner_delays( 9, 6, 3 )`.	
RTC4→RTC6	Unchanged functionality. In addition: increased value range.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_delay_mode**, **set_laser_delays**	

Ctrl Command	set_serial	
Function	Sets the starting serial number of the serial-number-set most recently selected by **select_serial_set** (= 0 after **load_program_file**) and sets the increment size for this serial-number-set to 1.	
Call	`set_serial( No )`	
Parameters	`No`	Serial number. As an unsigned 32-bit value. Allowed value range: [0…(2^{32}–1)].
Multi-board Com'd Name	**n_set_serial**	
Comments	• **set_serial** is synonymous with **set_serial_step** with `Step` = 1 (see comments there).	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_serial_step**, **select_serial_set**	

Ctrl Command	set_serial_step
Function	Sets the starting serial number and the increment size for the serial-number-set most recently selected by **select_serial_set** (= 0 after **load_program_file**).
Call	`set_serial_step( No, Step )`
Parameters	`No` Serial number. As an unsigned 32-bit value. Allowed value range: [0…(2^{32}–1)].
	`Step` Increment size. As an unsigned 32-bit value. Allowed value range: [0…9999]; only the last 4 decimal digits are used.
Multi-board Com'd Name	**n_set_serial_step**
Comments	• **load_program_file** sets the starting serial number to 0 and the increment size to 1. • If **mark_serial** or **mark_serial_abs** has been called with Mode M_2 = 1 (that is, automatic serial-number incrementing has been deactivated), then the increment size setting has no effect. • If `Step` = 0, then incrementing does not occur, except in the case of markless marking (see **mark_serial** or **mark_serial_abs**: `digits` = 0), which always increments serial numbers by 1. • For usage of **set_serial_step**, see Chapter 7.5.2 "Marking Serial Numbers", page 219.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**mark_serial**, **mark_serial_abs**, **set_serial**, **select_serial_set**, **select_serial_set_list**

Normal List Command	set_serial_step_list
Function	Like **set_serial_step**, but a list command.
Call	`set_serial_step_list( No, Step )`
Parameters	`No` Like **set_serial_step**.
	`Step` Like **set_serial_step**.
Multi-board Com'd Name	**n_set_serial_step_list**
Comments	• See **set_serial_step**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_serial_step**, **select_serial_set_list**, **get_list_serial**, **mark_serial**, **mark_serial_abs**

Ctrl Command	set_short_cmd_mode_ctrl
Function	Sets the behavior for Short List Commands.
Call	`set_short_cmd_mode_ctrl( Mode )`
Parameters	`Mode` Counting of Short List Commands. As an unsigned 32-bit value. = 0 Off. = 1 On. Default after **load_program_file**. > 1 Reserved. **get_last_error** return code `RTC6_PARAM_ERROR`.
Multi-board Com'd Name	**n_set_short_cmd_mode_ctrl**
Comments	• For `Mode` = 1, the RTC6 board automatically limits the maximum number of executable Short List Commands / 10 μs clock cycle to avoid clock overflows. See Short List Command, page 335. • For `Mode` = 0, an unlimited number of Short List Command (as well as 1 additional Normal List Command, page 335) / 10 μs clock cycle is possible. Make sure that there are no clock overflows by trial and error and **get_overrun** calls. If necessary, manually initiate a 10 μs clock cycle change by **list_nop** or **list_continue**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 628, OUT 629.
References	**set_short_cmd_mode_list**

Normal List Command	set_short_cmd_mode_list
Function	Like **set_short_cmd_mode_ctrl**, but a list command.
Call	`set_short_cmd_mode_list( Mode )`
Parameters	`Mode` Like **set_short_cmd_mode_ctrl**.
Multi-board Com'd Name	**n_set_short_cmd_mode_list**
Comments	• For `Mode` > 1 **set_short_cmd_mode_list** is replaced by a **list_nop**. • See **set_short_cmd_mode_ctrl**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 628, OUT 629.
References	**set_short_cmd_mode_ctrl**

Ctrl Command	set_sky_writing	
Function	Activates Sky Writing Mode 1 and sets the corresponding parameters or switches off Sky Writing.	
Call	`set_sky_writing( Timelag, LaserOnShift )`	
Parameters	`Timelag`	Like **set_sky_writing_para**.
	`LaserOnShift`	Like **set_sky_writing_para**.
Multi-board Com'd Name	**n_set_sky_writing**	
Comments	• **set_sky_writing** is identical to `set_sky_writing_para( Timelag, LaserOnShift, Nprev, Npost )` with `Nprev` = approx. (0.15 × `Timelag`) and `Npost` = approx. (0.1 × `Timelag`). • See also information about **set_sky_writing_para** and Chapter 7.2.4 "Sky Writing", page 163.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the value specified for `LaserOnShift` by 32. The allowed value range decreases accordingly.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_sky_writing_para**, **set_sky_writing_list**	

Ctrl Command	set_sky_writing_limit
Function	Defines the limit for Sky Writing switching in Sky Writing Mode 3 and Sky Writing Mode 4.
Call	`set_sky_writing_limit( Limit )`
Parameters	`Limit` Limit value. As a 64-bit IEEE floating point value. Allowed value range: [–1.0…+1.0]. Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_set_sky_writing_limit**
Comments	• For usage of **set_sky_writing_limit**, see Chapter 7.2.4 "Sky Writing", page 163. • `Limit` is the cosine of the angular limit (for angular change between consecutive vectors or arcs within a Polyline) for which a Sky Writing motion should be performed. • The initialized value (after program start) is `Limit` = 0 (angular limit = 90°).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_sky_writing_limit_list**

Undelayed Short List Command	set_sky_writing_limit_list
Function	Like **set_sky_writing_limit**, but a list command.
Call	`set_sky_writing_limit_list( Limit )`
Parameters	`Limit` Like **set_sky_writing_limit**.
Multi-board Com'd Name	**n_set_sky_writing_limit_list**
Comments	• See **set_sky_writing_limit**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_sky_writing_limit**

Normal List Command	set_sky_writing_list
Function	Like set_sky_writing, but a list command.
Call	`set_sky_writing_list( Timelag, LaserOnShift )`
Parameters	`Timelag` Like set_sky_writing_para.
	`LaserOnShift` Like set_sky_writing_para.
Multi-board Com'd Name	**n_set_sky_writing_list**
Comments	• See set_sky_writing, set_sky_writing_para and set_sky_writing_para_list.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the value specified for `LaserOnShift` by 32. The allowed value range decreases accordingly.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	set_sky_writing, set_sky_writing_para, set_sky_writing_para_list.

Ctrl Command	set_sky_writing_min_speed_ctrl
Function	Sets the minimum mark speed for Sky Writing.
Call	`set_sky_writing_min_speed_ctrl( Speed )`
Parameters	`Speed` Minimum mark speed for Sky Writing. Like **set_mark_speed_ctrl**.
Multi-board Com'd Name	**n_set_sky_writing_min_speed_ctrl**
Comments	• **set_sky_writing_min_speed_ctrl** only works in: – Sky Writing Mode 2 – Sky Writing Mode 3 – Sky Writing Mode 4 • Sky Writing with Minimum Mark Speed, page 171 is only available for: – SCANahead Systems (**set_scanahead_params**) and automatic calculation of Scanner Delays and Laser Delays switched on (**activate_scanahead_autodelays**) • Sky Writing motions are shortened such that mark speed is no longer reached at the mark vector start/end, but `Speed`. • Regular Sky Writing motions are carried out for: – `Speed` < 100 – `Speed` ≥ mark speed (see **set_mark_speed_ctrl**) • **set_sky_writing_min_speed_ctrl** is not executed, if: – automatic calculation of Scanner Delays and Laser Delays has not been switched on before (**get_last_error**-return code `RTC6_REJECTED`) – the BUSY list execution status is set (**get_last_error** return code `RTC6_BUSY`) • The following applies to "Spot Distance Control" in combination with Sky Writing: – set_auto_laser_control with `Mode` = 2 extension M = +128 can be combined with set_sky_writing_min_speed_ctrl. The energy input is kept constant though.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 646, OUT 649.
References	**set_mark_speed_ctrl**

Ctrl Command	**set_sky_writing_mode**
Function	Toggles the Sky Writing mode.
Call	`set_sky_writing_mode( Mode )`
Parameters	`Mode` Sky Writing mode. As an unsigned 32-bit value. Allowed value range: [0…(2^{32}–1)]. = 0: Sky Writing is deactivated. = 1: Sky Writing Mode 1 is activated. = 2: Sky Writing Mode 2 is activated. = 3: Sky Writing Mode 3 is activated. > 3: Sky Writing Mode 4 is activated.
Multi-board Com'd Name	**n_set_sky_writing_mode**
Comments	• Sky Writing Mode 2, Sky Writing Mode 3 and Sky Writing Mode 4 can only be activated (by `Mode` > 1), if: – Sky Writing Mode 1 has been activated by `set_sky_writing_para` (`Timelag` ≥ 1/8) before – AND is still active • After `set_sky_writing_mode( Mode =0 )`, subsequent Sky Writing reactivation: – Is possible by `set_sky_writing_mode( Mode > 0 )` – Not possible after `set_sky_writing_para` (`Timelag` < 1/8) • Each mode switch by **set_sky_writing_mode** becomes effective as of the next list command. • If you *reactivate* Sky Writing Mode 1 (switching from `Mode =` 0 to 1) during execution of a list, then an already-begun Mark Command still executes to completion *without* Sky Writing. • If you *deactivate* Sky Writing Mode 1 (switching from `Mode =` 1 to 0) during execution of a list, then a Mark Command already begun in Sky Writing Mode 1 still executes to completion in Sky Writing Mode 1 and then is appended with a Mark Delay. • Activation or deactivation of Sky Writing Mode 2, Sky Writing Mode 3 or Sky Writing Mode 4 (switching to or from `Mode =` 2 or 3 or 4) is not possible if the BUSY list execution status is set. Then **set_sky_writing_mode** is ignored (**get_last_error** return code `RTC6_BUSY`). • **set_sky_writing_mode** is executed when a list has been paused by **set_wait**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. No `Mode` > 3.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 646, OUT 649. `Mode` > 3.
References	**set_sky_writing_mode_list**, **set_sky_writing_para**

Normal List Command	**set_sky_writing_mode_list**
Function	Like **set_sky_writing_mode**, but a list command.
Call	`set_sky_writing_mode_list( Mode )`
Parameters	`Mode` Like **set_sky_writing_mode**.
Multi-board Com'd Name	**n_set_sky_writing_mode_list**
Comments	• By **set_sky_writing_mode_list**, Sky Writing Mode 2, Sky Writing Mode 3 and Sky Writing Mode 4 can be activated or deactivated even within a list (switching to or from `Mode = 2` or 3 or 4). Here, an already-begun Mark Command is finished with Sky Writing Mode 1 and the next is started with it. • Deactivation of Sky Writing Mode 1 by **set_sky_writing_mode_list** (switching from `Mode = 1` to 0) results in the addition of a Mark Delay defined prior to activation of Sky Writing – provided that no other delay is in effect (for example, a Jump Delay).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. No `Mode` > 3.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 646, OUT 649. `Mode` > 3.
References	**set_sky_writing_mode**

<table>
<tr><th>Ctrl Command</th><th>set_sky_writing_para</th></tr>
<tr><td>Function</td><td colspan="2">Activates Sky Writing Mode 1 and sets the corresponding parameters or switches Sky Writing off.</td></tr>
<tr><td>Call</td><td colspan="2"><code>set_sky_writing_para(Timelag, LaserOnShift, Nprev, Npost)</code></td></tr>
<tr><td>Parameters</td><td><code>Timelag</code></td><td>Sky Writing parameter.
1.0 equals 1 µs.
The <code>Timelag</code> value is used with an accuracy of 1/64 µs.
As a 64-bit IEEE floating point value.
≥ 1/4: Sky Writing Mode 1 is activated.
From 1/8 to 1/4, Sky Writing is activated,
but <code>Timelag</code> = 0 is internally applied.
< 1/8: Sky Writing is deactivated.</td></tr>
<tr><td></td><td><code>LaserOnShift</code></td><td>Shift (for positive values: delay) of the point of time, where the are switched on (see Figure 46).
1 bit equals 1/64 µs.
As a signed 32-bit value.
Negative values smaller than the complete run-in phase are clipped at runtime, therefore the following applies automatically:
<code>LaserOnShift</code> — Mode
≥ ca. (–40) × <code>Nprev</code> — 1
≥ ca. (–20) × <code>Nprev</code> — 2 or 3</td></tr>
<tr><td></td><td><code>Nprev</code></td><td>Defines the duration of the run-in phase (see Figure 46).
1 bit equals 10 µs.
As an unsigned 32-bit value.
Allowed value range: 0 ≤ <code>Nprev</code> ≤ (2^{32}–1).
Duration of run-in [µs] — Mode
20 × <code>Nprev</code> — 1
10 × <code>Nprev</code> — 2 or 3</td></tr>
<tr><td></td><td><code>Npost</code></td><td>Defines the duration of the run-out phase (see Figure 46).
1 bit equals 10 µs.
As an unsigned 32-bit value.
Allowed value range: 0 ≤ <code>Npost</code> ≤ (2^{32}–1).
Duration of run-out [µs] — Mode
20 × <code>Npost</code> — 1
10 × <code>Npost</code> — 2 or 3</td></tr>
<tr><td>Multi-board Com’d Name</td><td colspan="2">n_set_sky_writing_para</td></tr>
<tr><td>Comments</td><td colspan="2">• For information on Sky Writing mode and a description of the parameters, see Chapter 7.2.4 ”Sky Writing”, page 163.
• If <code>Nprev</code> ≥ 65,535, then set_sky_writing_para behaves similarly to set_sky_writing: <code>Nprev</code> is set to a value of approx. (0.15 × <code>Timelag</code>).
• If <code>Npost</code> ≥ 65,535, then set_sky_writing_para behaves similarly to set_sky_writing: <code>Npost</code> is set to a value of approx. (0.1 × <code>Timelag</code>).</td></tr>
</table>

Ctrl Command	set_sky_writing_para
Comments (cont'd)	• **set_sky_writing_para** cannot be used to switch back and forth between Sky Writing Mode 1, Sky Writing Mode 2, Sky Writing Mode 3 and Sky Writing Mode 4 (**set_sky_writing_mode** is designed for this). • When **set_sky_writing_para** is called and *no* list is running or a list has been halted by **set_wait**, the following applies: – With `Timelag` ≥ 1/8, Sky Writing Mode 1 is activated, if Sky Writing has not been active before. Sky Writing Mode 2, Sky Writing Mode 3 or Sky Writing Mode 4 remain, but the next Mark Command is always started in Sky Writing Mode 1 – With `Timelag` < 1/8, Sky Writing is deactivated. • When **set_sky_writing_para** is called and *a* list is running or a list has been halted by **pause_list**, the following applies: – If Sky Writing Mode 2, Sky Writing Mode 3 or Sky Writing Mode 4 is active, then **set_sky_writing_para** is not executed (**get_last_error** return code `RTC6_BUSY`) – If no Sky Writing or Sky Writing Mode 1 is active, **set_sky_writing_para** parameters are going to be effective upon the next list command. • In Sky Writing Mode 1, an already started Mark Command is executed with the previous parameters and ended with Sky Writing Mode 1. • In Sky Writing Mode 1, the next Mark Command is started in Sky Writing Mode 1, if `Timelag` ≥ 1/8. Otherwise, it is terminated and a Mark Delay is inserted in Sky Writing Mode 1. • For Sky Writing Mode 2, Sky Writing Mode 3 or Sky Writing Mode 4, `Nprev` = 0 and/or `Npost` = 0 automatically get(s) internally corrected to 1 (this is not treated as an error). If you subsequently activate Sky Writing Mode 1 (by **set_sky_writing_mode**), then `Nprev` = 0 and/or `Npost` = 0 is/are reused.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the value specified for `LaserOnShift` by 32. The allowed value range decreases accordingly.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_sky_writing**, **set_sky_writing_para_list**

Normal List Command	set_sky_writing_para_list	
Function	Like **set_sky_writing_para**, but a list command.	
Call	`set_sky_writing_para_list( Timelag, LaserOnShift, Nprev, Npost )`	
Parameters	`Timelag`	Like **set_sky_writing_para**.
	`LaserOnShift`	Like **set_sky_writing_para**.
	`Nprev`	Like **set_sky_writing_para**.
	`Npost`	Like **set_sky_writing_para**.
Multi-board Com'd Name	**n_set_sky_writing_para_list**	
Comments	• **set_sky_writing_para_list** can be executed within a list, even if Sky Writing Mode 2, Sky Writing Mode 3 or Sky Writing Mode 4 is active. Here, an already-begun Mark Command is finished with Sky Writing Mode 1 and the next is started with it. • By **set_sky_writing_para_list**, you cannot switch from Sky Writing Mode 2, Sky Writing Mode 3 oder Sky Writing Mode 4 to Sky Writing Mode 1 permanently. For this, use **set_sky_writing_mode_list**. • Deactivation of Sky Writing Mode 1, Sky Writing Mode 2, Sky Writing Mode 3 or Sky Writing Mode 4 by **set_sky_writing_para_list** results in the addition of a Mark Delay defined prior to activation of Sky Writing – provided that no other delay is in effect (for example, a Jump Delay).	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies the value specified for `LaserOnShift` by 32. The allowed value range decreases accordingly.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_sky_writing_para**	

Ctrl Command	set_softstart_level	
Function	No function.	
Call	`set_softstart_level( Index, Level )`	
Parameters	`Index`	As an unsigned 32-bit value.
	`Level`	As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_softstart_level**	
Comments	• **set_softstart_level** has no effect on the user program.	
RTC4→RTC6	Unchanged functionality.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_softstart_level_list**	

Normal List Command	set_softstart_level_list	
Function	No function.	
Call	`set_softstart_level_list( Index, Level1, Level2, Level3 )`	
Parameters	`Index`	Like **set_softstart_level**.
	`Level1`	Like **set_softstart_level**.
	`Level2`	Like **set_softstart_level**.
	`Level3`	Like **set_softstart_level**.
Multi-board Com'd Name	**n_set_softstart_level_list**	
Comments	• Like **set_softstart_level**.	
RTC4→RTC6	Unchanged functionality.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_softstart_level**	

Ctrl Command	set_softstart_mode	
Function	No function.	
Call	`set_softstart_mode( Mode, Number, Delay )`	
Parameters	`Mode`	As an unsigned 32-bit value.
	`Number`	As an unsigned 32-bit value.
	`Delay`	As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_softstart_mode**	
Comments	• **set_softstart_mode** has no effect on the user program.	
RTC4→RTC6	Unchanged functionality.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	–	

Variable List Command	set_softstart_mode_list	
Function	No function.	
Call	`set_softstart_mode_list( Mode, Number, Delay )`	
Parameters	`Mode`	Like **set_softstart_mode**.
	`Number`	Like **set_softstart_mode**.
	`Delay`	Like **set_softstart_mode**.
Multi-board Com'd Name	**n_set_softstart_mode_list**	
Comments	• See **set_softstart_mode**.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	–	

Ctrl Command	set_standby	
Function	Defines the output period and the pulse length of the standby pulses for "laser standby" operation or – in Laser Mode 4 and Laser Mode 6 – the continuously-running laser signals for Signals for "Laser Active" Operation and "laser standby" operation.	
Call	`set_standby( HalfPeriod, PulseLength )`	
Parameters	`HalfPeriod`	*Half* of the standby output period. As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: [0…+(2^{32}–1)].
	`PulseLength`	Pulse length of the standby pulses. As an unsigned 32-bit value. 1 bit equals 1/64 μs. Allowed value range: [0…(2^{32}–1)].
Multi-board Com'd Name	**n_set_standby**	
Comments	• After a hardware reset, (and after **load_program_file**), the LASER1, LASER2 and LASERON ports must be first-time-activated with **set_laser_control** before standby pulses can be activated by **set_standby** or **set_standby_list**, see Chapter 7.4 "Laser Control", page 189. At the same time, the signal level of the standby pulses are set by **set_laser_control**. • The standby pulses are available in *all* laser modes (CO_2 Mode, YAG Mode 1, YAG Mode 2, YAG Mode 3, Laser Mode 4, YAG Mode 5, Laser Mode 6). • The standby pulses can be deactivated (turned off) by setting the standby pulse length and/or the standby output period to zero (default). • With `HalfPeriod`, *half* the standby output period must be specified, see Figure 52 and Figure 54. • If `PulseLength` is larger than the output period (2 × `HalfPeriod`), the laser is permanently on. • The laser mode is set with **set_laser_mode**, see also Chapter 7.4 "Laser Control", page 189. To set the active output period and pulse length for the "laser active" Laser Control Signals (beside for Laser Mode 4 and Laser Mode 6), are set by **set_laser_pulses_ctrl**, **set_laser_pulses** or **set_laser_timing**.	
RTC4→RTC6	Basically unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `HalfPeriod` and `PulseLength` by 8. The allowed value ranges decrease accordingly.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_standby_list**, **get_standby**	

Delayed Short List Command	set_standby_list	
Function	Like **set_standby**, but a list command.	
Call	`set_standby_list( HalfPeriod, PulseLength )`	
Parameters	`HalfPeriod`	Like **set_standby**.
	`PulseLength`	Like **set_standby**.
Multi-board Com'd Name	**n_set_standby_list**	
Comments	• –	
RTC4→RTC6	See **set_standby**.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_standby**	

Ctrl Command	set_start_list
Function	Opens the RTC6 List Memory for writing of list commands and sets the input pointer to the start of the specified list ("List 1" or "List 2").
Call	`set_start_list( ListNo )`
Parameters	`ListNo` Number of the list in which the input pointer should be set. As an unsigned 32-bit value. Allowed values: [uneven: "List 1", even: "List 2"].
Multi-board Com'd Name	**n_set_start_list**
Comments	• **set_start_list** is synonymous with **set_start_list_pos** for `Pos` = 0. • Alternatively, **set_start_list_1** and **set_start_list_2** (with no parameter) can be used.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**execute_list**, **read_status**, **set_start_list_pos**

Ctrl Command	set_start_list_1
Function	See **set_start_list**.
Call	`set_start_list_1()`
Multi-board Com'd Name	**n_set_start_list_1**
Comments	• –
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_start_list**

Ctrl Command	set_start_list_2
Function	See **set_start_list**.
Call	`set_start_list_2()`
Multi-board Com'd Name	**n_set_start_list_2**
Comments	• –
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_start_list**

Ctrl Command	set_start_list_pos
Function	Opens the RTC6 List Memory for writing of list commands and sets the input pointer to the specified *relative* position in the specified list ("List 1" or "List 2").
Call	`set_start_list_pos( ListNo, Pos )`
Parameters	`ListNo` Number of the list for which the input pointer should be set. As an unsigned 32-bit value. Allowed values: [uneven: "List 1", even: "List 2"].
	`Pos` Position of the input pointer (offset relative to the start of the respective list). As an unsigned 32-bit value. Allowed value range: [0…(2^{23}–1)].
Multi-board Com'd Name	**n_set_start_list_pos**
Comments	• The next list command is stored at the specified address and all further list commands at the subsequent addresses in the selected list. • The specified list is unconditionally opened for loading. There is no checking of whether the list is currently being processed, see also Chapter 6.4.1 "Loading Lists", page 108 and **load_list**. • The input pointer *cannot* be set to the protected RTC6 List Memory area "List 3". If the protected area is to be written to (at the full risk of users) with **set_start_list_pos**, then this area can be temporarily allocated to "List 2" by `config_list( Mem1, -1 )`. After writing, configuration of the area's protection should be restored: `config_list( Mem1, Mem2 )`. • For uneven `ListNo` values, "List 1" is opened, otherwise "List 2". This facilitates continuous automatic list changing though incrementing counts. • If "List 2" has not been assigned memory (`Mem2` = 0, see **config_list**) then "List 1" is opened. • If `Pos` is specified as being larger than the memory area of the respective list (`Pos` > `Mem1` or `Pos` > `Mem2`), then `Pos` is set to 0. • The status values of the selected list (see also **read_status**) are set as follows: LOAD list status set, READY list status reset, USED list status reset. The LOAD list status of the other corresponding list is reset. • **set_start_list_pos** triggers a flush of the buffered list input, see Chapter 6.4.1 "Loading Lists", page 108. • **set_start_list_pos** also covers the specialized variants **set_start_list_1**, **set_start_list_2**, **set_start_list** and **set_input_pointer**.

Ctrl Command	set_start_list_pos
Comments (cont'd)	• *Caution! If the end of the respective RTC6 List Memory area is reached, the list input pointer is automatically reset to the start of the same RTC6 List Memory area. Make sure not to overwrite any commands still needed by your user program.*
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**execute_list_pos**, **read_status**

Ctrl Command	set_sub_pointer
Function	Stores the absolute start address of a command list in the internal management table for indexed subroutines.
Call	`set_sub_pointer( Index, Pos )`
Parameters	`Index` Index of the indexed subroutine whose starting address `Pos` should be entered in the management table. As an unsigned 32-bit value. Allowed value range: [0…1023]).
	`Pos` Absolute start address. As an unsigned 32-bit value. Allowed value range: [0…(2^{23}–1)].
Multi-board Com'd Name	**n_set_sub_pointer**
Comments	• If `Index` > 1023 and/or `Pos` > (2^{23}–1), then **set_sub_pointer** is *not* executed (**get_last_error** return code `RTC6_PARAM_ERROR`). • The **set_sub_pointer** command can be used for referencing a nonindexed subroutine, which thereby becomes an indexed subroutine that is protectable by **save_disk**/**load_disk** and/or callable by the index. • **set_sub_pointer** can also be used to reference anew an indexed subroutine, character or text string so that it can also be called by a second index. Here, it is preferable to use the **copy_dst_src** command for index management. • The start addresses of command lists that are to be referenced with **set_sub_pointer** can be queried by **get_input_pointer** *before* loading the command lists. • **set_sub_pointer** only stores *starting addresses* in the internal management table. An indexed subroutine only gains protection by a subsequent **save_disk**/**load_disk** command. • `Pos` should not be an arbitrary address within a list. Instead, it should be the starting address of an actually existing subroutine that has been finalized by **list_return** and does not contain **set_end_of_list**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**load_sub**

Ctrl Command	set_text_table_pointer
Function	Stores the absolute start address of a command list in the internal management table for indexed text strings.
Call	`set_text_table_pointer( Index, Pos )`
Parameters	`Index` Index of the indexed text string whose starting address `Pos` should be entered in the management table. As an unsigned 32-bit value. Allowed value range: [0…41]. The same assignment applies as for **load_text_table**: = 0…9: Digits for marking the time and date [0…9]. = 10…21: Months [January…December]. = 22…28: Days-of-the-week [Sunday…Saturday]. = 29: Blank character for marking serial numbers. = 30…39: Digits for marking serial numbers [0…9]. = 40: Text for “a.m.”. = 41: Text for “p.m.”.
	`Pos` Absolute start address. As an unsigned 32-bit value. Allowed value range: [0…(2^{23}–1)].
Multi-board Com’d Name	**n_set_text_table_pointer**
Comments	• Indexed text strings can be used for marking time, date or serial numbers, see Section ”Calling Indexed Text Strings”, page 123. • If `Index` > 41 and/or `Pos` > (2^{23}–1), then **set_text_table_pointer** is *not* executed (**get_last_error** return code `RTC6_PARAM_ERROR`). • **set_text_table_pointer** can be used for referencing a nonindexed subroutine, which thereby becomes an indexed text string that is protectable by **save_disk**/**load_disk** and/or callable by the index. • **set_text_table_pointer** can also be used to reference anew an indexed subroutine, character or text string so that it can also be called by a second index. Here, it is preferable to use the **copy_dst_src** command for index management. • The start addresses of command lists that are to be referenced with **set_text_table_pointer** can be queried by **get_input_pointer** *before* loading the command lists. • **set_text_table_pointer** only stores *starting addresses* in the internal management table. An indexed text string only gains protection by a subsequent **save_disk**/**load_disk** command. • `Pos` should not be an arbitrary address within a list. Instead, it should be the starting address of an actually existing subroutine that has been finalized by **list_return** and does not contain **set_end_of_list**.

Ctrl Command	set_text_table_pointer
RTC4→RTC6	New command. **set_text_table_pointer** is synonymous with **set_char_table**, which is available for the RTC4 *SCAN*alone Board (standalone version of the RTC4 board).
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**load_text_table**, **mark_date**, **mark_serial**, **mark_time**

Ctrl Command	set_timelag_compensation
Function	Compensates the various Tracking Errors of the xy axes and z axis.
Call	`set_timelag_compensation( HeadNoXY, TimelagXY, TimelagZ )`
Parameters	`HeadNoXY` Number of the xy scan head connector. As an unsigned 32-bit value. Allowed value range: [1…2]. `TimelagXY` Tracking Error of the xy axis in [10 µs]. As an unsigned 32-bit value. Allowed value range: [0…255]. `TimelagZ` Tracking Error of the z axis in [10 µs]. As an unsigned 32-bit value. Allowed value range: [0…255].
Multi-board Com'd Name	**n_set_timelag_compensation**
Comments	• Unallowed parameter values produce a **get_last_error** return code of `RTC6_PARAM_ERROR` and **set_timelag_compensation** is not executed. • If list execution is currently active, then the **get_last_error** return code gets set to `RTC6_BUSY` (the control command **get_status** returns BUSY list execution status or INTERNAL-BUSY list execution status or PAUSED list execution status) and **set_timelag_compensation** is not executed. • Faster axes are held back towards the slower axes. • **set_timelag_compensation** can also be used with SCANahead Systems. If **set_scanahead_params** configured the RTC6 board for controlling a SCANahead System, then `TimelagXY` is ignored. Instead, the `PreviewTime` value from **set_scanahead_params** gets applied. **set_timelag_compensation** automatically waits until a HEAD BUSY list execution status gets reset (see "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual), hence execution of **set_timelag_compensation** can last up to `PreviewTime`. • With SCANahead Systems, the parameter `PreviewTime` from **set_scanahead_params** is automatically used as "Tracking Error". • Regarding the evaluation of data records (**set_trigger**), see comment, page 830. • **set_timelag_compensation** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC5→ RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_scanahead_params**, **get_last_error**, **get_status**

<table>
<tr><td>Delayed Short List Command</td><td colspan="2">set_trigger</td></tr>
<tr><td>Function</td><td colspan="2">Starts measurement value recording of the specified signals.</td></tr>
<tr><td>Call</td><td colspan="2"><code>set_trigger(Period, Signal1, Signal2)</code></td></tr>
<tr><td>Parameters</td><td><code>Period</code></td><td>Measurement period (time interval between 2 data pair recordings).
1 bit equals 10 μs.
Allowed value range: [0…(2^{31}–1)].
Bit #30 = 0: Automatic recording stop at the end of the list.
Bit #30 = 1: No automatic recording stop at the end of the list.
Bit #31 = 0: Automatic stop at the max. channel size, see set_trigger4/set_trigger8.
Bit #31 = 1: Endless recording (circular buffer).
As an unsigned 32-bit value.</td></tr>
<tr><td></td><td><code>Signal1</code></td><td>Signal type for measurement channel 1.
As an unsigned 32-bit value.
Allowed value range: see below.
0 LASERON signal
1 = laser control signal on. 0 = laser control signal off.
1 StatusAX
Status signal of x axis, Connector for First Scan Head.
2 StatusAY
Status signal of y axis, Connector for First Scan Head.
3 Reserved.
4 StatusBX
Status signal of x axis, Connector for Second Scan Head.
5 StatusBY
Status signal of y axis, Connector for Second Scan Head.
6 Reserved.
7 SampleX
Cartesian control value for the x axis
(= up to #4 in 7.3.6, page 186).
8 SampleY
Cartesian control value for the y axis
(= up to #4 in 7.3.6, page 186).
9 SampleZ
Cartesian control value for the z axis
(= up to #4 in 7.3.6, page 186).</td></tr>
</table>

Delayed Short List Command	set_trigger		
Parameters (cont'd)	`Signal1` (cont'd)	10	SampleAX_Corr Corrected x axis control value for the Connector for First Scan Head (= up to #7 in 7.3.6, page 186).
		11	SampleAY_Corr Corrected y axis control value for the Connector for First Scan Head (= up to #7 in 7.3.6, page 186).
		12	SampleAZ_Corr Corrected z axis control value, if xy are connected to the Connector for First Scan Head (= up to #10 in 7.3.6, page 186). Identical to the effective output value for the z axis.
		13	SampleBX_Corr Corrected control value for the x axis, Connector for Second Scan Head (= up to #7 in 7.3.6, page 186).
		14	SampleBY_Corr Corrected control value for the y axis, Connector for Second Scan Head (= up to #7 in 7.3.6, page 186).
		15	SampleBZ_Corr Corrected z axis control value, if xy are connected to the Connector for Second Scan Head (= up to #10 in 7.3.6, page 186). Identical to the effective output value for the z axis.
		16	StatusAX+LASERON signal StatusAX for laser control signal on. –524,288 for laser control signal off.
		17	StatusAY+LASERON signal StatusAY for laser control signal on. –524,288 for laser control signal off.
		18	StatusBX+LASERON signal StatusBX for laser control signal on. –524,288 for laser control signal off.
		19	StatusBY+LASERON signal StatusBY for laser control signal on. –524,288 for laser control signal off.

Delayed Short List Command	set_trigger		
Parameters (cont'd)	`Signal1` (cont'd)	20	SampleAX_Out Actual output value for the x axis, Connector for First Scan Head (= up to #11 in 7.3.6, page 186). Not usable for z axis output values.
		21	SampleAY_Out Actual output value for the y axis, Connector for First Scan Head (= up to #11 in 7.3.6, page 186). Not usable for z axis output values.
		22	SampleBX_Out Actual output value for the for the x axis, Connector for Second Scan Head (= up to #11 in 7.3.6, page 186). Not usable for z axis output values.
		23	SampleBY_Out Actual output value for the y axis, Connector for Second Scan Head (= up to #11 in 7.3.6, page 186). Not usable for measuring z axis output values.
		24	Laser control parameter of "Automatic Laser Control" See **set_auto_laser_control**.
		25	SampleAX_Trans Transformed control value for the x axis, Connector for First Scan Head (= up to #6 in 7.3.6, page 186).
		26	SampleAY_Trans Transformed control value for the y axis, Connector for First Scan Head (= up to #6 in 7.3.6, page 186).
		27	SampleAZ_Trans Transformed z axis control value, if xy are connected to the Connector for First Scan Head (= up to #6 in 7.3.6, page 186).
		28	SampleBX_Trans Transformed control value for the x axis, Connector for Second Scan Head (= up to #6 in 7.3.6, page 186).
		29	SampleBY_Trans Transformed control value for the y axis, Connector for Second Scan Head (= up to #6 in 7.3.6, page 186).
		30	SampleBZ_Trans Transformed z axis control value, if xy are connected to the Connector for Second Scan Head (= up to #6 in 7.3.6, page 186).

Delayed Short List Command	set_trigger		
Parameters (cont'd)	Signal1 (cont'd)	31	Laser control parameter of Vector-Defined Laser Control See **set_vector_control**.
		32	Focus shift See **set_vector_control**, **set_defocus**, **set_defocus_list**.
		33	12-bit output value at the ANALOG OUT1 output port See **set_vector_control** and Chapter 9.1.4 "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 305.
		34	12-bit output value at the ANALOG OUT2 output port See **set_vector_control** and Chapter 9.1.4 "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 305.
		35	Output value at the 16-Bit Digital Output Port See **set_vector_control** and Chapter 9.1.1 "16-Bit Digital Output Port", page 303.
		36	Output value at the 8-Bit Digital Output Port See **set_vector_control** and Chapter 9.1.2 "8-Bit Digital Output Port", page 304.
		37	Pulse length (PulseLength) of the Laser Control Signals LASER1 and LASER2 See **set_vector_control**.
		38	Output period (HalfPeriod) of the Laser Control Signals LASER1 and LASER2 See **set_vector_control**.
		39	FreeVariable0
		40	FreeVariable1
		41	FreeVariable2
		42	FreeVariable3

Delayed Short List Command	set_trigger		
Parameters (cont'd)	`Signal1` (cont'd)	43	Counter value of encoder counter "Encoder0"
		44	Counter value of encoder counter "Encoder1"
		45	Marking speed From **set_mark_speed**, **set_mark_speed_ctrl**.
		46	16-Bit Digital Input Port (EXTENSION 1 Socket Connector)
		47	Only for intelli*WELD* II: Zoom value
		48	FreeVariable4
		49	FreeVariable5
		50	FreeVariable6
		51	FreeVariable7
		52	32-bit "Timestamp Counter" See Chapter 8.12 "Time Measurements", page 290.

Delayed Short List Command	set_trigger		
Parameters (cont'd) Parameters (cont'd)	Signal1 (cont'd)	53	Wobbel amplitude See **set_wobbel**, **set_wobbel_mode** and Chapter 8.4 "Wobbel Mode", page 241.
		54	ReadAnalogIn See **read_analog_in**.
		55	Scaled encoder value for x
		56	Scaled encoder value for y
		57	Scaled encoder value for z
		58	RS-232
		59	read_mcbsp(0)
		60	read_multi_mcbsp(0)
		61	read_multi_mcbsp(1)
		62	read_multi_mcbsp(2)
	Signal2	Like (analogously) Signal1.	
Multi-board Com'd Name	n_set_trigger		

Delayed Short List Command	set_trigger
Comments	*General Comments* • If Signal1 and Signal2 are not from the above list, then **set_trigger** (except for Period = 0) is replaced by a **list_nop** (**get_last_error** return code RTC6_PARAM_ERROR). • After a **set_trigger** with Period > 0, a data pair (data couple) is recorded immediately and subsequently at intervals defined by the specified measurement period. – Period(Bit #31 = 0): • The measurement value recording ends automatically with 2^{24} data pairs (if Signal2 is invalid, even with 2^{25} data entries, see also **set_trigger4**/**set_trigger8**). • It ends earlier, if one of the following occurs: - **set_trigger** (Period = 0) is called - **set_trigger4**(Period = 0) is called - **set_trigger8**(Period = 0) is called • Period(Bit #31 = 1): • Starts an endless data recording according Period = "measurement period | 0x80000000" with 0 < measurement period < 2^{31}–1. In this case, the data recording does not end automatically. Instead, it starts from the beginning each time and overwrites previously recorded data (ring buffer). In the meantime, the data can be read out with **get_waveform_offset** from any location and in any packets. **measurement_status** also provides information about the current status of the measurement value recording. • A measurement value recording started by **set_trigger** occurs only during the execution of a list. • Given the measurement value recording has not been terminated by **set_trigger** (Period = 0) or **set_trigger4**(Period = 0), it is automatically continued with the start of a new list (the status of the measurement value recording is not reset by **set_end_of_list**). During the execution of a list, the status is reset by **set_trigger**(Period = 0), **set_trigger4**(Period = 0) or **stop_execution**. By **stop_trigger** the status can be reset if no list is executed. • Sample values and status values returned by the scan system and stored on the RTC6 by **set_trigger** (Signal1, Signal2 = 1…23, 25…30) are always in the RTC6 20-bit range, even if the RTC6 DLL is set to RTC4 Compatibility Mode. The measured and stored values can be subsequently transferred to the PC by the **get_waveform** command for evaluation. It is generally recommended to end the measurement explicitly before reading out the data. The measured values are transferred to the PC by **get_waveform** as 32-bit values and must be evaluated accordingly by users (see comments for **get_value**). • The current status of the measurement value recording can be queried by **measurement_status**.

Delayed Short List Command	**set_trigger**
Comments (cont'd)	*General Comments (cont'd)* • An active measured value recording is terminated by (no matter which of the following started it): – **set_trigger**(Period = 0) – **set_trigger4**(Period = 0) – **set_trigger8**(Period = 0) • If you abort a measurement session with **set_trigger** (Period = 0) or **set_trigger4**(Period = 0), then previously recorded measurement values are *not* lost and the measurement counter halts at its most recent value. This allows subsequent querying by **measurement_status** of the number of data entries. In contrast, if a measurement session is newly started with **set_trigger** (Period > 0) or **set_trigger4**(Period > 0), the measurement counter is reset and the measurements obtained thus far are overwritten. It is not possible to resume an explicitly or automatically halted measurement session. *Comments on the Data* • The type of scan system being used determines which status signals are generated and returned by the status channels. Specific information can be found in your scan system's operating manual. **control_command** can be used with iDRIVE Scan Systems to specify which information is returned on the status channels. • Only X measurement signals contain meaningful data with single-status channel scan systems. • With 3D scan systems, the output and status values of the z axis are transmitted over the scan head connector channel to which the z axis is attached (if correspondingly configured by **select_cor_table**). Example: If the z axis is attached to the Connector for Second Scan Head x channel (**select_cor_table**(HeadA, 0)), then its status signal can be queried by StatusBX (Signal1/Signal2 = 4). • The signals 12 and 15 as well as 27 and 30 are identical, each: SampleAZ_Corr = SampleBZ_Corr and SampleAZ_Trans = SampleBZ_Trans. • Status signals Status<…> are a few 10 μs clock cycles later than control signals Sample<…>. See also Section "Reading Out Data", page 222. • Signal1, Signal2 = 0, 24, 31…42 and 47…51 enables logging of values that were outputted during the previous clock cycle. Data recording of different signals is not synchronous because of the internal processing chain. Depending on how the signals are combined, you must later correct the data by a fixed time offset.

Delayed Short List Command	set_trigger
Comments (cont'd)	*Comments on the Data (cont'd)* • Note when evaluating data recorded with active Tracking Error compensation (**set_timelag_compensation**): – A call of `set_timelag_compensation( TimelagXY < TimelagZ )` delays the xy axis as well as the Laser Control Signals by [`TimelagZ` – `TimelagXY`] – For `Signal1, Signal2 = 7`, 8, 10, 11, 13, 14, 20, 21 the undelayed x axis/y axis control value is recorded respectively – In contrast to the recorded values, for `Signal1, Signal2 = 20`, 21, the actual control value is outputted at the scan head connector with a [`TimelagZ` – `TimelagXY`] delay. – For `Signal1, Signal2 = 0`, the LASERON signal delayed by [`TimelagZ` – `TimelagXY`] is recorded. • `Signal1, Signal2 = 24` enables logging of the signal parameter that is outputted by "Automatic Laser Control" (but not with Vector-Defined Laser Control; see **set_auto_laser_control**). Logging only occurs when "Automatic Laser Control" has been activated and the laser is on. When the laser is switched off, 0 is recorded. With "Spot Distance Control", signal 24 has no direct meaning, it does *not* mean "HalfPeriod" in particular. • `Signal1, Signal2` = 31 enables logging of the signal parameter specified for Vector-Defined Laser Control (see **set_vector_control**). Logging is also possible without executing [*]para[*] Commands, but then the signal values remain unchanged. • Pulse length and output period (`Signal1, Signal2 = 37`, 38) are only logged for Signals for "Laser Active" Operation (not for standby signals). In Softstart Mode (not yet implemented) and in Pixel Output Mode, the pulse length and output period might possibly change faster than the 10 µs clock cycle (time resolution of logging by **set_trigger**). These values cannot be recorded. • For `Signal1, Signal2 = 39`…42 and 48…51, see Chapter 6.9.1 "Free Variables", page 138.
RTC4→RTC6	Basically unchanged functionality. However: • The measurement session can be aborted by `Period` = 0. • The signals `StatusAZ` and `StatusBZ` (`Signal1`/`Signal2` = 3, 6) are no longer available. • All coordinate values are referred to (0|0|0) (value range [-2^{19}…($2^{19}-1$)]) and are no longer zero point shifted.
RTC5→RTC6	Basically unchanged functionality. Increased memory area for data recordings by merging channels, see **set_trigger4**.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Change: DLL 610, OUT 610, RBF 615: endless recording.
References	**get_waveform**, **get_waveform_offset**, **measurement_status**, **control_command**, **get_value**, **get_values**, **set_mcbsp_out**, **set_mcbsp_out_ptr**, **set_trigger4**, **set_trigger8**

Delayed Short List Command	**set_trigger4**
Function	Starts the measurement value recording of the specified measurement signals (has the same effect as **set_trigger**, but with up to 4 simultaneous measurement signals).
Call	`set_trigger4( Period, Signal1, Signal2, Signal3, Signal4 )`
Parameters	`Period` Like **set_trigger**.
	`Signal1` Like **set_trigger**.
	`Signal2` Like **set_trigger**.
	`Signal3` Like **set_trigger**.
	`Signal4` Like **set_trigger**.
Multi-board Com'd Name	**n_set_trigger4**
Comments	• See comments for **set_trigger**. • 2^{23} entries per measurement channel are available with **set_trigger4**, if all 4 signals are within the allowed range (see **set_trigger**). The measurement value recording ends automatically with 2^{23} data quadruples. • If Signal 3 and Signal 4 are not allowed (see **set_trigger**), **set_trigger4** is synonymous with **set_trigger**. The memory areas of Signal 3 and Signal 4 are added to the memory areas of Signal 1 and Signal 2. This allows recording of 2^{24} data pairs. The measured value recording ends automatically with 2^{24} data *pairs*. • If only Signal 1 is within the permissible range, the memory areas of all 4 channels are merged into a single one. With this up to 2^{25} data values can be recorded. The measured value recording ends automatically at 2^{25} data *values*. • An active measured value recording is terminated by (no matter which of the 3 commands started it): – **set_trigger**(`Period` = 0) – **set_trigger4**(`Period` = 0) – **set_trigger8**(`Period` = 0) • If **set_trigger**(`Period` > 0) or **set_trigger4**(`Period` > 0) is used to start a new measurement value recording, then the measurement value counter is reset. Existing data is overwritten with new measurement values. See also the corresponding comments for **set_trigger**. • As with **set_trigger**/**set_trigger4**, **get_waveform**/**get_waveform_offset** can be used to transfer the recorded values to the PC.
RTC4→RTC6	New command.
RTC5→RTC6	Basically unchanged functionality. Increased memory area for data recordings by merging unused channels, endless recording.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 610, OUT 610, RBF 615: see **set_trigger**.
References	**set_trigger**, **set_trigger8**, **stop_trigger**, **get_waveform**, **get_waveform_offset**

Delayed Short List Command	**set_trigger8**
Function	Starts the measurement value recording of the specified measurement signals (has the same effect as **set_trigger**, but with up to 8 simultaneous measurement signals).
Call	`set_trigger8( Period, Signal1, Signal2, Signal3, Signal4, Signal5, Signal6, Signal7, Signal8 )`
Parameters	`Period` Like **set_trigger**. `Signal1` Like **set_trigger**. `Signal2` Like **set_trigger**. `Signal3` Like **set_trigger**. `Signal4` Like **set_trigger**. `Signal5` Like **set_trigger**. `Signal6` Like **set_trigger**. `Signal7` Like **set_trigger**. `Signal8` Like **set_trigger**.
Multi-board Com'd Name	**n_set_trigger8**
Comments	• See comments for **set_trigger**. • 2^{22} entries per measurement channel are available with **set_trigger8**. The measurement value recording ends automatically with 2^{22} data octuples. • A currently running measured value recording is terminated by (no matter which of the 3 commands started it): – **set_trigger**(`Period` = 0) – **set_trigger4**(`Period` = 0) – **set_trigger8**(`Period` = 0) • Other than **set_trigger** and **set_trigger4**, **set_trigger8** does not automatically increase the memory area for data recording by merging unused channels. A maximum of 2^{22} entries per measuring channel is always recorded. • The measurement value counter is reset when a new measurement value recording is started again by: – **set_trigger**(`Period` > 0) – **set_trigger4**(`Period` > 0) – **set_trigger8**(`Period` > 0) Already existing data are overwritten with new data. See also the corresponding comments under **set_trigger**. • As with **set_trigger**/**set_trigger4**, **get_waveform**/**get_waveform_offset** can be used to transfer the recorded values to the PC.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 635, OUT 636.
References	**set_trigger**, **set_trigger4**, **stop_trigger**, **get_waveform**, **get_waveform_offset**

<table>
<tr><td>Undelayed Short List Command</td><td colspan="2">set_vector_control</td></tr>
<tr><td>Function</td><td colspan="2">Initializes or deactivates Vector-Defined Laser Control.</td></tr>
<tr><td>Call</td><td colspan="2"><code>set_vector_control(Ctrl, Value)</code></td></tr>
<tr><td rowspan="2">Parameters</td><td><code>Ctrl</code></td><td>Control parameter for initializing or deactivating Vector-Defined Laser Control (for <code>Ctrl</code> = 1…6: identical with set_auto_laser_control).
As an unsigned 32-bit value.
= 1…8: Defines which signal parameter is to be varied by Vector-Defined Laser Control.
= 1: 12-bit output value at the ANALOG OUT1 output port.
= 2: 12-bit output value at the ANALOG OUT2 output port.
= 3: Output value at the 8-Bit Digital Output Port.
= 4: Pulse length (<code>PulseLength</code>) of the Laser Control Signals LASER1 and LASER2.
= 5: Output period (<code>HalfPeriod</code>) of the Laser Control Signals LASER1 and LASER2.
= 6: Output value at the 16-Bit Digital Output Port.
= 7: Focus shift (“Defocus”).
= 8: Reserved for intelliWELD.
= 0 or > 8: Deactivates Vector-Defined Laser Control.</td></tr>
<tr><td><code>Value</code></td><td>Defines the starting value for the parameter selected by <code>Ctrl</code>.
As an unsigned 32-bit value.
Allowed values (out-of-range values are clipped to the boundary values):
For <code>Ctrl</code> = 1/2: 12-bit values [0…4,095].
For <code>Ctrl</code> = 3: 8-bit values [0…255].
For <code>Ctrl</code> = 4: [0…(2^{32}–1)]. 1 bit equals 1/64 µs.
For <code>Ctrl</code> = 5: [0…(2^{32}–1)]. 1 bit equals 1/64 µs.
For <code>Ctrl</code> = 6: 16-bit values [0…65,535].
For <code>Ctrl</code> = 7: [–524,288…+524,287],
<code>Value</code> = focus shift + 524,288.</td></tr>
<tr><td>Multi-board Com’d Name</td><td colspan="2">n_set_vector_control</td></tr>
<tr><td>Comments</td><td colspan="2">• If <code>Ctrl</code> is an invalid value, then Vector-Defined Laser Control is deactivated (<code>Ctrl</code> = 0, initialization state). Otherwise, <code>Value</code> is applied as the starting value of the next [*]para[*] Command. If <code>Value</code> is invalid, then it is clipped to the maximum allowed value.
• set_vector_control only affects [*]para[*] Commands.
• If an “Automatic Laser Control“ has been activated by set_auto_laser_control with the same control parameter (<code>Ctrl</code> = 1…6), then set_vector_control sets the 100% value of the “Automatic Laser Control“ to value <code>Value</code>.
However, this does not apply to <code>Ctrl</code> = 5: set_auto_laser_control(<code>Ctrl</code> = 7) (“Spot Distance Control”) and Vector-Defined Laser Control with set_vector_control (<code>Ctrl</code> = 5) (<code>HalfPeriod</code>) cannot be combined.</td></tr>
</table>

Undelayed Short List Command	set_vector_control
Comments (cont'd)	• For Ctrl = 7, the focus shift is linearly varied as with **set_defocus** and **set_defocus_list**, see comments there) with [*]para[*] Commands (in order for the z outputs to be outputted, the Option "3D" must be enabled and a 3D correction file must be loaded and assigned as well) up to the specified end value (this requires enabling the Option "3D" as well as loading and assigning a 3D correction file). If, for the first [*]para[*] Command, the currently set focus shift does not match the initial value (Value) specified by **set_vector_control**, then the command begins with a "Hard Jump" (in the z output) to Value. The setting defined by **set_defocus** or **set_defocus_list** is lost. Unlike signed values in the range [–524,288…+524,287] for **set_defocus** and **set_defocus_list**, the focus shift (Value) specified for **set_vector_control** must be an unsigned number shifted upward by 524,288: Value = focus shift + 524,288. • "Automatic Laser Control" should not be combined with the variable laser power of **set_multi_mcbsp_in** or the "freely definable wobbel shape".
RTC4→RTC6	New command. This command has no RTC4 Compatibility Mode for Value. In RTC4 Compatibility Mode, the RTC6 multiplies with Ctrl = 7 the specified value for Value by 16 (Value = focus shift + 32,768). The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality. In RTC5 Compatibility Mode, the RTC6 multiplies with Ctrl = 7 the specified value for Value by 16 (Value = focus shift + 32,768). The allowed value range decreases accordingly.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 634: Ctrl = 8.
References	**set_auto_laser_control**

Ctrl Command	set_verify
Function	Activates or deactivates a download verification. See Chapter 6.8.1 "Download Verification", page 134.
Call	`OldVerify = set_verify( Verify )`
Parameters	`Verify` Setting parameter. As an unsigned 32-bit value. = 0: Verification is deactivated. > 0: Verification is activated.
Result	The `Verify` setting parameter that has been active before calling **set_verify**. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_set_verify**
Comments	• If verification is activated, the download times are extended. • Verification of correction file downloads only works if the correction file contains a checksum (otherwise the **get_last_error** return code `RTC6_VERIFY_ERROR` is generated). To ensure that a checksum is present (which is not the case for older `ct5` correction files), you should test your correction file prior to loading by using the control command **verify_checksum**. If the correction file does not contain a check sum, then **verify_checksum** enters it. • **set_verify** is available even without explicit access rights to a specific RTC6 board. **set_verify** only change settings in the RTC6 DLL of the specified (or default) board. It has no effect on the board itself. • The board-specific error variables `LastError` and `AccError` (see Chapter 6.8 "Error Handling", page 133) are neither generated nor altered by **set_verify**. • Activation of the download verification by **set_verify** can generate the **get_last_error** return code `RTC6_OUT_OF_MEMORY`, if a RTC6 DLL-internal Windows memory request fails. In this case, the download verification remains deactivated.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_error**, **get_last_error**, **verify_checksum**

Normal List Command	set_wait
Function	Inserts a numbered break point (“wait marker”) into the list.
Call	`set_wait( WaitWord )`
Parameters	`WaitWord` Number of the break point. As an unsigned 32-bit value. Allowed value range: [1…(2^{32}–1)]. 0 is corrected to 1.
Multi-board Com’d Name	**n_set_wait**
Comments	• The list processing is interrupted at each break point and remains halted until continued by **release_wait**. This provides a way to implement synchronization. • The Signals for “Laser Active” Operation are switched off by **set_wait** and a home jump defined by **home_position** or **home_position_xyz** might be executed (the INTERNAL-BUSY list execution status is set while the home jump is executed). Continuation by **execute_list_pos**, **restart_list** or an External Start is consequently not possible. However, **stop_execution** or an External Stop is possible. **release_wait**, **stop_execution** or an External Stop removes suppression of the start. • **set_wait** sets the PAUSED list execution status and resets the BUSY list execution status (both queryable with **get_status**). The opposite occurs with a subsequent **release_wait**, see also Chapter 6.4.3 ”List Execution Status”, page 111. • If processing has been stopped at a break point, then **get_wait_status** returns the number of this break point.
RTC4→RTC6	Basically unchanged functionality. However: Additional PAUSED list execution status which is set by **set_wait**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_wait_status**, **release_wait**, **pause_list**, **stop_list**

Delayed Short List Command	**set_wobbel**	
Function	Defines the parameters for an ellipse-shaped wobbel motion. "Wobbel" is a motion of the output position which is added to the regular marking motion. See Chapter 8.4 "Wobbel Mode", page 241.	
Call	`set_wobbel( Transversal, Longitudinal, Freq )`	
Parameters	`Transversal`	Amplitude of the elliptical *perpendicular* to the momentary direction of motion or to the one defined by **set_wobbel_direction**. In bits. As an unsigned 32-bit value. Allowed value range: [0…131.071 (=2^{17}-1)]. Larger values are clipped.
	`Longitudinal`	Amplitude of the elliptical *parallel* to the momentary direction of motion or to the one defined by **set_wobbel_direction**. In bits. As an unsigned 32-bit value. Allowed value range: [0…131.071 (=2^{17}-1)]. Larger values are clipped.
	`Freq`	Frequency of the wobbel motion in *Hz* (number of ellipses per second). As a 64-bit IEEE floating point value. Allowed value range: [–6000…6000]. Larger values are clipped.
Multi-board Com'd Name	**n_set_wobbel**	
Comments	• The Wobbel Mode can be used for marking lines with various line widths. An ellipse-shaped motion is added to the linear marking motion, resulting in a spiral motion of the laser focus in the Image Field. Alternatively, a figure-of-8 wobbel shape (horizontal or vertical to the direction of motion) can be activated by **set_wobbel_mode**. The line width can be set by appropriate values for the amplitude and frequency (frequency and mark speed should be coordinated, see Chapter 8.4.1 "Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values", page 243). For arcs, too, the wobbel motion follows the current marking direction (exception: see below). Therefore, independently of the Cartesian angle, the effective line width is always the same. • The Wobbel Mode is terminated by setting both amplitudes and/or the frequency to zero (more accurate for $-n \le$ `Freq` $\le n$ with n = 50000/65536 = 0.7629…). • The frequency is signed. The wobbel vector rotates clockwise for positive values and counterclockwise for negative values. Thus, the inner and outer point densities of arcs can be individually set independently of their actual direction of rotation. • At the beginning of a marking, (after **set_wobbel** or a jump), the wobbel start point is generally set for the same value relative to the vector/arc startpoint; it is repeatedly continued, however, for Polylines (including arcs).	

Delayed Short List Command	set_wobbel
Comments (cont'd)	• For identical amplitudes (`Longitudinal` = `Transversal`), the wobbel startpoint is permanently referenced to the coordinate system, that is, independent of the current direction of motion. By **set_wobbel_direction**, a fixed reference direction can be set which differs from it. • For an angle, ellipse-shaped wobbel motions can result in more or less small jumps after reaching the corner, depending on the current wobbel phase. This does not occur (= "rounded corners") if the wobbel motion is exactly circular (`Longitudinal` = `Transversal`). • `Longitudinal` = 0 produces a sine-shaped wobbel motion across the direction of motion. • When defining the wobbel shape and its frequency take the dynamics of the scan head and laser into account. Otherwise, an overheating and even a permanent damage of the system may occur, see Chapter 8.4.1 "Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values", page 243. • The present wobbel amplitude can be recorded by **set_trigger[*]** (signal 53). The format of the data is ((transversal << 16) + longitudinal).
RTC4→RTC6	Basically unchanged functionality. More wobbel shape: elliptical, direction dependent adjustable, see also **set_wobbel_mode**. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `Longitudinal` and `Transversal` by 16. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_mark_speed**, **set_wobbel_mode**, **set_wobbel_direction**

Undelayed Short List Command	**set_wobbel_control**
Function	Specifies laser control parameters for laser power variation with “freely definable wobbel shapes”.
Call	`set_wobbel_control( Ctrl, Value, MinValue, MaxValue )`
Parameters	`Ctrl` Control parameter for initializing or deactivating laser power variation. As an unsigned 32-bit value. = 0: Deactivates laser power variation. = 1…6: Defines which signal parameter to vary. On the meaning, see **set_auto_laser_control**. = 7: Deactivates laser power variation (**get_last_error** return code `RTC6_PARAM_ERROR`). = 8: Activates laser power variation where ANALOG OUT1 and ANALOG OUT2 alternate. > 8: Same as 7. `Value` Nominal laser power P0 (100%). As an unsigned 32-bit value. Allowed value range: see **set_auto_laser_control**; the maximum is [0…65,535] or 0xFFFFFFFF (with `Ctrl` = 1…6) or 0xFFFF (with `Ctrl` = 8). Excessive values are clipped. `MinValue` Limit that cannot be exceeded, see **set_auto_laser_control**. As an unsigned 32-bit value. The allowed value range depends on the selected `Ctrl` parameter and on `Value`. `MaxValue` See `MinValue`.
Multi-board Com’d Name	**n_set_wobbel_control**
Comments	• Any still-pending Delayed Short List Command is executed beforehand. • For `Ctrl` = 0, `Ctrl` = 7 and `Ctrl` > 8, laser power variation is switched off (initialization state after **load_program_file**). The values for McBSP multi transfers are then not used. • `Ctrl` = 8 activates a laser power variation where ANALOG OUT1 and ANALOG OUT2 alternate. The “Free definable wobbel figure” starts with ANALOG OUT1. Subsequently, alternation occurs with every Microstep. See also **set_wobbel_vector_2**. • With `Ctrl` = 8, `Value`, `MinValue` and `MaxValue` are to be interpreted as 2 independent 16-bit values: – The upper 16 bits apply to ANALOG OUT2 – The lower 16 bits apply to ANALOG OUT1 • **set_wobbel_control**(`Ctrl` = 8) and **set_wobbel_mode**(`Mode` = 3), see page 849, cannot be combined.

Undelayed Short List Command	**set_wobbel_control**
Comments (cont'd)	• The conditions for `Value`, `MinValue`, `MaxValue` are the same as for "Automatic Laser Control" (see **set_auto_laser_control**), but with a maximum of [0…65,535]. Initialized values are `MinValue` = 0 and `MaxValue` = 0xFFFFFFFF, that is, no restrictions. • The laser power can be combined with the Vector-Defined Laser Control, if the corresponding `Ctrl` parameters have been chosen identically. • If you want variable laser power along a wobbel shape, then **set_wobbel_control** must execute before you activate the "freely definable wobbel shape" by **set_wobbel_mode**. Otherwise, laser power is not varied, even if you make a change in the data sets. • Special case: For – `Value` = 0xFFFFFFFF (with `Ctrl` = 1…6) – `Value` = 0xFFFF (with `Ctrl` = 8) the nominal laser power is derived from the port assigned by the signal parameter `Ctrl`, instead of from **set_wobbel_control**. This is then the current content at this timepoint set by other normal commands, for example, **write_da_1_list** for `Ctrl` = 1. These are executed prior to **set_wobbel_control**. But beware: with execution of a "freely definable wobbel shape", the value at the port can change at any time. Subsequent calls of **set_wobbel_control** then return other values for the nominal laser power.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 631, OUT 632: With `Ctrl` = 8, now alternation occurs with every Microstep (no longer with every **set_wobbel_vector** call).
References	**set_wobbel_vector**, **set_multi_mcbsp_in**, **set_multi_mcbsp_in_list**, **set_wobbel_vector_2**

Undelayed Short List Command	**set_wobbel_direction**
Function	Defines a direction vector for wobbel motions.
Call	`set_wobbel_direction( dX, dY )`
Parameters	`dX` x component of the direction vector. In bits. As a signed 32-bit value.
	`dY` y component of the direction vector. In bits. As a signed 32-bit value.
Multi-board Com’d Name	**n_set_wobbel_direction**
Comments	• For non-zero direction vectors (`dX` and/or `dY` non-zero), wobbel motion (longitudinal and transversal) is relative to *this* direction vector instead of to the momentary direction vector. The direction vector’s length is inconsequential. The RTC6 normalizes the direction vector. • If `dX` = `dY` = 0, then the function is deactivated. • The direction vector setting remains in effect even after the Wobbel Mode is switched off by **set_wobbel** or **set_wobbel_mode**, and continues being used if you switch the Wobbel Mode back on.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_wobbel**, **set_wobbel_direction**, **set_wobbel_mode**, **set_wobbel_offset**

Delayed Short List Command	**set_wobbel_mode**	
Function	Switches the Wobbel Mode on and off and defines the parameters for an ellipse-shaped or figure-of-8 wobbel motion, see Chapter 8.4 "Wobbel Mode", page 241.	
Call	`set_wobbel_mode( Transversal, Longitudinal, Freq, Mode )`	
Parameters	`Transversal`	Amplitude of the wobbel motion *perpendicular* to the momentary direction of motion or to the one defined by **set_wobbel_direction**. In bits. As an unsigned 32-bit value. Allowed value range: [0…131.071 (=2^{17}-1)]. Excessive values are clipped.
	`Longitudinal`	Amplitude of the wobbel motion *parallel* to the momentary direction of motion or to the one defined by **set_wobbel_direction**. In bits. As an unsigned 32-bit value. Allowed value range: [0…131.071 (=2^{17}-1)]. Excessive values are clipped.
	`Freq`	Frequency of the wobbel motion in *Hz* (number of ellipses or figure-of-8s per second). As a 64-bit IEEE floating point value. Allowed value range: [–6000…6000]. Larger values are clipped.
	`Mode`	Defines the wobbel shape. As a signed 32-bit value. = 0: Ellipse-shaped wobbel motion. < 0: Figure-of-8 wobbel motion perpendicular to the motion direction (vertical 8). = 1: Figure-of-8 wobbel motion parallel to the motion direction (horizontal 8). > 1: "Freely definable wobbel shape", see **set_wobbel_vector**.
Multi-board Com'd Name	**n_set_wobbel_mode**	
Comments	• If `Mode` = 0, **set_wobbel_mode** functions identically to **set_wobbel** (see comments there). • If `Mode` ≠ 0, then a figure-of-8 wobbel motion is activated. With figure-of-8s, broader mid-line processing can be achieved by appropriate parameter values. If the specified transverse and longitudinal amplitudes are identical, then the wobbel shape remains stationary in space; otherwise the orientation of the wobbel shape follows the current direction of motion. If **set_wobbel_mode** executes while a list contains a "freely definable wobbel shape" (see **set_wobbel_vector** and Chapter 8.4 "Wobbel Mode", page 241), then `Mode` > 1 selects this shape, otherwise the horizontal figure-8 shape is selected similarly to `Mode` = 1. If you *subsequently* define a wobbel shape and the Wobbel Mode has been activated with the parameter `Mode` > 1, then a switch from the horizontal figure-8 to the wobbel shape automatically occurs. But beware: it is then not possible to vary the power along the wobbel figure, see **set_wobbel_control**.	

Delayed Short List Command	set_wobbel_mode
Comments (cont'd)	• The Wobbel Mode is terminated by setting both amplitudes and/or the frequency to zero (more accurate for –n ≤ Freq ≤ n with n = 50000/65536 = 0.7629…). • The frequency is signed. During the figure-of-8's first loop, the wobbel vector rotates clockwise for positive values and counterclockwise for negative values. Thus, (especially for ellipse-shaped wobbel motions), the inner and outer point densities of arcs can be individually set independently of their actual direction of rotation. • At the beginning of a marking, (after **set_wobbel** or a jump), the wobbel start point is generally set for the same value relative to the vector/arc startpoint; it is repeatedly continued, however, for Polyline (including arcs). For identical amplitudes (Longitudinal = Transversal), the wobbel startpoint is permanently referenced to the coordinate system, that is, independent of the current direction of motion. • For an angle, elliptical or figure-8 wobbel motions can result in more or less small jumps after reaching the corner, depending on the current wobbel phase. This does not occur (= "rounded corners") if the wobbel motion is exactly circular (Longitudinal = Transversal). • Longitudinal = 0 produces a sine-shaped wobbel motion across the direction of motion. • For "freely definable wobbel shapes", the parameter Freq has no meaning. It must nevertheless lie within the valid range, because otherwise the Wobbel Mode gets deactivated. This also applies to the parameters Transversal and Longitudinal. • If the Wobbel Mode gets deactivated, then any already-defined wobbel shape remains active and is used upon the next switch-on of the Wobbel Mode with Mode > 1. This also applies, if the wobbel figure has been previously saved by **create_dat_file** and then loaded by **load_program_file**. • When defining the wobbel shape and its frequency take the dynamics of the scan head and laser into account. Otherwise, an overheating and even a permanent damage of the system may occur, see Chapter 8.4.1 "Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values", page 243.

Delayed Short List Command	set_wobbel_mode
Comments (cont'd)	• The present wobbel amplitude can be recorded by **set_trigger[*]** (signal 53). The format of the data is ((transversal << 16) + longitudinal). • The Wobbel Mode cannot be combined with: – Sky Writing – Pixel Output Mode – Jumps – **laser_on_list**
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `Longitudinal` and `Transversal` by 16. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_wobbel**, **set_wobbel_control**, **set_wobbel_offset**, **set_wobbel_vector**

Delayed Short List Command	**set_wobbel_mode_phase**
Function	Switches a classical Wobbel Mode on and off as with **set_wobbel_mode**. In addition, defines a start phase.
Call	`set_wobbel_mode_phase( Transversal, Longitudinal, Freq, Mode, Phase )`
Parameters	`Transversal` As with **set_wobbel_mode**. `Longitudinal` As with **set_wobbel_mode**. `Freq` As with **set_wobbel_mode**. `Mode` As with **set_wobbel_mode**. `Phase` Start phase. As a 64-bit IEEE floating point value. Allowed value range: [0.0°…360.0°]. Negative values are clipped to 0.0°. Too big positive values are reduced by the corresponding multiples of 360.0°.
Multi-board Com'd Name	**n_set_wobbel_mode_phase**
Comments	• `Mode` ≥ 2 ("freely definable wobbel shape") is not allowed with **set_wobbel_mode_phase** (is clipped to 1). • The start phase is converted to an integer value with a 16-bit resolution for a full circle. • By **set_wobbel_mode_phase**, the wobbel shape continues also with: – **jump** commands – **timed_jump** commands – **para_jump** commands – **list_nop** – **long_delay** Here, the wobbel shape follows the last valid marking direction. • **set_wobbel_direction** and **set_wobbel_offset** are effective.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 609, OUT 609, RBF 614.
References	**set_wobbel_mode**

Delayed Short List Command	set_wobbel_offset	
Function	Defines a wobbel shape shift in the direction of motion or perpendicular to the direction of motion.	
Call	`set_wobbel_offset( OffsetTrans, OffsetLong )`	
Parameters	`OffsetTrans`	Transversal offset. As a signed 32-bit value. Allowed value range: ±32,767. Larger values are clipped. Initialization values after **load_program_file**: (0,0).
	`OffsetLong`	Longitudinal offset. Otherwise, like `OffsetTrans`.
Multi-board Com'd Name	**n_set_wobbel_offset**	
Comments	• Offsets can be defined for "classic" wobbel shapes (circle, ellipse, sine, figure-8) as well as for "Freely Definable Wobbel Shapes", see Chapter 8.4 "Wobbel Mode", page 241. • The summed up wobbel amplitude including offsets may never exceed $\pm(2^{17}-1)$. • At the beginning of the wobbel marking offsets are set as "Hard Jumps". This applies also in case of switching-off or non-using the Wobbel Mode, for example, when using a Jump Command (analogously to "classical" wobbel shapes longitudinal amplitudes).	
RTC4→RTC6	New command. In RTC4 Compatibility Mode the RTC6 multiplies the specified values for `OffsetTrans` and `OffsetLong` by 16. The allowed value ranges decrease accordingly.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**set_wobbel_mode**, **set_wobbel_vector**, **set_wobbel_direction**	

Undelayed Short List Command	**set_wobbel_vector**
Function	Defines a linear section of a wobbel shape.
Call	`set_wobbel_vector( dTrans, dLong, Period, dPower )`
Parameters	`dTrans` `dLong` Microstep of a linear wobbel shape section. In bits. As a 64-bit IEEE floating point value. Allowed value range: [–256.0…+255.0]. `dTrans` is the wobbel excursion perpendicular to the direction of motion (which is either the laser trajectory (see Glossary entry on page 31) or the direction of motion defined by **set_wobbel_direction**). I`dLong` is correspondingly longitudinal to it. `Period` As an unsigned 32-bit value. Allowed value range: [0…65,535]. = 1…65,535: number of Microsteps. = 0: The wobbel shape is switched off. `dPower` Microstep of the relative laser power. As a 64-bit IEEE floating point value. Allowed value range: [–1.0…+1.0]. Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_set_wobbel_vector**
Comments	• `Period` = 0 is not allowed as a shape section. `Period` = 0 means explicitly switch off the "freely definable wobbel shape" (not the Wobbel Mode itself, see **set_wobbel_mode**). Subsequent calls of **set_wobbel_vector** begin a new wobbel shape. • Each call of **set_wobbel_vector** adds a new section at the end of the previous section independently of when the call is performed. • Up to 1023 wobbel shape sections can be defined. After 1023 sections, storage automatically wraps around to the first section and overwrites it. Each further call does the same to the next section. • The wobbel shape automatically begins with wobbel vector (0,0), that is, directly on the marking itself (the "Hard Jump" of "classic" wobbel shapes is eliminated if the longitudinal amplitude $\neq 0$) and also ends there without requiring explicit declaration. • Each wobbel shape section consists of a vector defined by Microsteps and their number. The parameters `dTrans`, `dLong` get internally rounded to 7 bit decimal places. At runtime each Microstep is executed within 10 μs. For large `Period` values, you should therefore take rounding error into account. Positive `dTrans` values cause that in respect to the direction of motion the start is to the right (which applies to circular wobbel shapes as well). Positive `dLong` values cause that in respect to the direction of motion the start is forward. The last section's endpoint is also the first section's start point. Independently of its respective position, it always ends at (0,0) and get executed as a "Hard Jump". With the last step the laser power is reset to the initial nominal laser power.

Undelayed Short List Command	set_wobbel_vector
Comments (cont'd)	• The summed up wobbel amplitude may never exceed $\pm(2^{17}-1)$. • If activated by **set_wobbel_control** and set_wobbel_mode(Mode = 2), the following applies: – The RTC6 varies the current laser power P within the wobbel shape section according to $$P(i, n) = P_{nom} \times (1 + (\sum_{j=1}^{i-1} dPower_j \times Period_j) + dPower_i \times n)$$ with $1 \leq i \leq$ wobbel vector number and $1 \leq n \leq$ $Period_i$. n represents the current number of each Microstep. Nominal laser power P_{nom} is set by the list command **set_wobbel_control**. – If activated by **set_multi_mcbsp_in** or **set_multi_mcbsp_in_list**, the nominal laser power P_{nom} is multiplicatively varied by the laser power parameter P across multiple McBSP transfers: $P_{nom\ McBSP} = P_{nom} \times P / 16{,}384$. – Beware here that for each both corrections above a maximum laser power of only $P_{nom} \times 4.0$ is allowed, whereby the maximum range of the laser control parameter likewise must not be exceeded (see **set_wobbel_control**). – For laser power variation with Ctrl = 5 (half period), dPower needs to have the opposite sign. Unlike with "Automatic Laser Control", the multiplication factor cannot be inverted. – By using an "empty" wobbel shape set_wobbel_vector(0.0, 0.0, 1, 0.0), you can also multiplicatively vary the nominal laser power P_{nom} without needing to explicitly wobbel (for another alternative, see **set_multi_mcbsp_in**). – When you switch off the wobbel shape (not by deactivation by **set_wobbel_mode**), the nominal laser power P_{nom} is emitted at the port assigned by Ctrl if **set_wobbel_control** has been last called with Value = 0xFFFFFFFF. Otherwise, the laser power P_{nom} multiplied by the external factor from **set_multi_mcbsp_in** is emitted.

Undelayed Short List Command	**set_wobbel_vector**
	• If activated by **set_wobbel_control** and set_wobbel_mode(Mode = 3), the following applies: – The RTC6 varies the current laser power P within the wobbel shape section according to $$P(i, n) = P_{100} \times (\text{Factor} + (\sum_{j=1}^{i-1} \text{dPower}_j \times \text{Period}_j) + \text{dPower}_i \times n)$$ – P100 (as with Mode = 2) is the nominal laser power as it would be without the wobbel figure. It is set by `set_wobbel_control( Ctrl, Value, MinValue, MaxValue )`: • With Value = 0…65,535, the specified Value value is used • With Value = 0xFFFFFFFF, the value is taken over that has been last outputted to the port Ctrl. – Factor is an unsigned 16+bit value with scaling 16,384 = one (default value). The initial Factor value can be set to nnnn by Value = 0xFF00nnnn and another **set_wobbel_control** call. The parameters Ctrl, MinValue and MaxValue are taken over from the latest **set_wobbel_control** call. – To change the 100% performance, **set_wobbel_control** can be called at any time. However, the initial Factor value only becomes effective when the Wobbel Mode is switched on *newly*. At wobbel shape runtime, Factor cannot be changed by **set_wobbel_control**. – Factor is changed per Microstep additively by the dPower value from `set_wobbel_vector( dTrans, dLong, Period, dPower )`. The maximum value is 65,535 (Factor 4) and cannot be exceeded. – Power modulation via McBSP (as with Mode = 2) is not possible.

Undelayed Short List Command	set_wobbel_vector
Comments (cont'd)	• If the wobbel shape gets switched off while the Wobbel Mode remains active, then the shape automatically switches to `Mode = 1` (horizontal figure-8). • The Wobbel Mode with "freely definable wobbel shapes" also needs to be switched on by **set_wobbel_mode**. • Variable laser power for wobbel shapes cannot be combined with variable laser power for automatic laser control or Vector-Defined Laser Control (parameterized [*]mark[*] Commands). Wobbel variation overwrites other variations if the respective `Ctrl` parameters are identical. Though unidentical `Ctrl` parameters are allowed, they serve no practical purpose. • When defining "freely definable wobbel shapes" make sure that the Microsteps of each individual wobbel vector does not exceed the maximum positioning speed significantly! Otherwise, a galvanometer scanner overheating may occur, see also Chapter 8.4.1 "Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values", page 243. • **create_dat_file** saves a "freely definable wobbel shape", see comment on page 377.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_multi_mcbsp_in**, **set_multi_mcbsp_in_list**, **set_wobbel_control**, **set_wobbel_offset**, **set_wobbel_vector_2**

Undelayed Short List Command	**set_wobbel_vector_2**
Function	Like set_wobbel_vector. However, only for set_wobbel_control(`Ctrl` = 8).
Call	`set_wobbel_vector_2( dTrans, dLong, Period, dPower, dPower2, Ctrl )`
Parameters	`dTrans` Like set_wobbel_vector. `dLong` Like set_wobbel_vector. `Period` Like set_wobbel_vector. `dPower` Microstep of the relative laser power. Refers to ANALOG OUT1. As a 64-bit IEEE floating point value. Allowed value range: [–1.0…+1.0]. `dPower2` Microstep of the relative laser power. Refers to ANALOG OUT2. As a 64-bit IEEE floating point value. Allowed value range: [–1.0…+1.0]. `Ctrl` Reserved. As an unsigned 32-bit value. Out-of-range values are clipped to the boundary values.
Multi-board Com'd Name	**n_set_wobbel_vector_2**
Comments	• **set_wobbel_vector_2** can only be used with set_wobbel_control(`Ctrl` = 8). • Like set_wobbel_vector.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 631, OUT 632.
References	set_wobbel_vector

Ctrl Command	**set_zoom**
Call	`set_zoom( Zoom )`
Multi-board Com'd Name	**n_set_zoom**
Comments	• This command is described, for example, in the manual for the intelli*WELD* II FT.

Undelayed Short List Command	**set_zoom_list**
Call	`set_zoom_list( Zoom )`
Multi-board Com'd Name	**n_set_zoom_list**
Comments	• This command is described, for example, in the manual for the intelli*WELD* II FT.

Ctrl Command	simulate_encoder
Function	Activates or deactivates encoder simulation for the specified encoder.
Call	`simulate_encoder( EncoderNo )`
Parameters	`EncoderNo` Encoder number as an unsigned 32-bit value. Allowed values: = 1: ENCODER X pulses are simulated and encoder counter "Encoder0" thereby incremented. = 2: ENCODER Y pulses are simulated and encoder counter "Encoder1" thereby incremented. = 3: Pulses for ENCODER X *and* ENCODER Y are simulated. = 0: The encoder simulation is deactivated.
Multi-board Com'd Name	**n_simulate_encoder**
Comments	• The encoder simulation is driven by an internal 1 MHz clock (see also Section "Encoder Simulation", page 321). • **simulate_encoder** does not trigger a reset of the encoder counter. • If `EncoderNo` > 3, then **simulate_encoder** is ignored (**get_last_error** return code `RTC6_PARAM_ERROR`). • **simulate_encoder** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_encoder**, **store_encoder**, **read_encoder**, **wait_for_encoder**, **set_fly_x**, **set_fly_y**, **set_fly_rot**

Normal List Command	simulate_ext_start
Function	After the specified track delay, causes a simulated External Start.
Call	`simulate_ext_start( Delay, EncoderNo )`
Parameters	`Delay` Track delay (in counter steps of the selected `EncoderNo` encoder counter). As a signed 32-bit value. Allowed value range: $[-2^{31}\ldots+(2^{31}-1)]$. `EncoderNo` Number of the to-be-used encoder counter. As an unsigned 32-bit value. Allowed values: = 0: Encoder counter "Encoder0". = 1: Encoder counter "Encoder1".
Multi-board Com'd Name	**n_simulate_ext_start**
Comments	• For External Starts, see Section "External Start", page 312. • The track delay is specified in *relative* counting units of the selected encoder counter (the RTC6 encoder counters are triggered by an external or simulated encoder signal, see Chapter 9.3.3 "Synchronization by Encoder Signals", page 320). The start trigger for the start occurs only after the internal encoder counter has reached the specified track delay. External Starts initiated by **simulate_ext_start** or by an external start signal, but whose execution has been postponed according to the specified track delay, are placed in a queue that accommodates up to 8 starts. **simulate_ext_start** cancels a previous queue and starts a new one. • A start trigger initiated by **simulate_ext_start** or an external start signal only triggers a start if it does not occur when the BUSY list execution status is set (for example, when outputting a list), when the INTERNAL-BUSY list execution status is set (for example, during **goto_xy**) and/or when the PAUSED list execution status is set (after **pause_list**, **stop_list** or **set_wait**). Otherwise, Bit #11 of the **get_startstop_info** return value is set. Therefore, if an unsuitable track delay is specified (for example, `Delay = 0`), no start is triggered. If **simulate_ext_start** is the first command in a list, then **get_startstop_info** can be used for checking whether processing of this list can finish within the defined track delay. • Ensure that the sign of the track delay (`Delay` parameter) is appropriate for the selected encoder's counting direction (for external triggering, this corresponds to the workpiece's direction of motion). • Track delays can also be set with **set_ext_start_delay** or **set_ext_start_delay_list**. Track delays are deactivated by initialization (with **load_program_file**), by external stops and by **stop_execution**. They can also be deactivated by **set_control_mode**(Bit #2). • The **simulate_ext_start** command alone *does not* cause an encoder reset. But if accordingly set with **set_control_mode**(Bit #9), a start trigger (initiated by **simulate_ext_start** or an external start signal) causes an encoder reset. • If `EncoderNo` > 1, then **simulate_ext_start** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).

Normal List Command	simulate_ext_start
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified value for `Delay` by 16. The allowed value range decreases accordingly.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**simulate_ext_start_ctrl**, **set_ext_start_delay**, **set_ext_start_delay_list** **set_extstartpos**, **set_extstartpos_list**, **set_control_mode**, **simulate_ext_stop**

Ctrl Command	simulate_ext_start_ctrl
Function	Causes a simulated External Start.
Call	`simulate_ext_start_ctrl()`
Multi-board Com'd Name	**n_simulate_ext_start_ctrl**
Comments	• For External Starts, see Section "External Start", page 312. • The start trigger for the start occurs only after a track delay previously set by **set_ext_start_delay**, **set_ext_start_delay_list** or **simulate_ext_start**. Track delays are deactivated by initialization (with **load_program_file**), by external stops and by **stop_execution**. They can also be deactivated with **set_control_mode** (Bit #2). External Starts initiated by **simulate_ext_start_ctrl** or by an external start signal, but whose execution has been postponed according to the specified track delay, are placed in a queue that accommodates up to 8 starts. In contrast to **simulate_ext_start**, **simulate_ext_start_ctrl** does *not* cancel the previous queue. • A start trigger initiated by **simulate_ext_start_ctrl** or by an external start signal does only trigger a start if it does not coincide with the output of a list (otherwise, Bit #11 of the **get_startstop_info** return value gets set). • The **simulate_ext_start_ctrl** command alone *does not* cause an encoder reset. But if accordingly set with **set_control_mode**(Bit #9), a start trigger initiated by **simulate_ext_start_ctrl**, **simulate_ext_start** or by an external start signal causes an encoder reset if the start trigger is preceded by one of the Processing-on-the-fly commands **set_fly_x**, **set_fly_y**, **set_fly_2d** or **set_fly_rot**. • **simulate_ext_start_ctrl** can be disabled by **set_control_mode**(Bit #4 = 1) or **set_control_mode_list**(Bit #4 = 1). • As of version DLL 609, OUT 609, RBF 614, the following applies: provided that a start is allowed, **simulate_ext_start_ctrl** waits 30 μs before it returns. This closes a potential timing gap (with **get_status**) between command call and actual start.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 609, OUT 609, RBF 614: see above.
References	**simulate_ext_start**

Ctrl Command	simulate_ext_stop
Function	Causes a simulated External Stop.
Call	`simulate_ext_stop()`
Multi-board Com'd Name	**n_simulate_ext_stop**
Comments	• For external stops, see Section "External Stop", page 311. • **simulate_ext_stop** simultaneously halts the master board and all slave boards. • In contrast, **stop_execution** only halts the master board.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**simulate_ext_start**, **simulate_ext_start_ctrl**, **stop_execution**, **sync_slaves**

Undelayed Short List Command	**spot_distance**
Function	As **spot_distance_ctrl**, but an list command.
Call	`spot_distance( Dist )`
Parameters	`Dist` See **spot_distance_ctrl**.
Multi-board Com'd Name	**n_spot_distance**
Comments	• See **spot_distance_ctrl**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 609, OUT 609, RBF 613.
References	**spot_distance_ctrl**, **set_auto_laser_control**, **set_auto_laser_params**, **set_auto_laser_params_list**

Ctrl Command	**spot_distance_ctrl**
Function	Defines the geometric pulse distance for the "Automatic Laser Control" with `Ctrl` = 7.
Call	`spot_distance_ctrl( Dist )`
Parameters	`Dist` Pulse distance. In bits. As a 64-bit IEEE floating point value.
Multi-board Com'd Name	**n_spot_distance_ctrl**
Comments	• The pulse distance is resolved with an accuracy of 1/40 bit (Image Field coordinates). • `Dist` = 0 suppresses the "Automatic Laser Control" with `Ctrl` = 7, but does not switch it off. • `Dist` must not exceed the maximum value 26,214. A practical `Dist` value depends on mark speed, laser frequency and scan system.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 609, OUT 609, RBF 613.
References	**spot_distance**, **set_auto_laser_control**, **set_auto_laser_params**, **set_auto_laser_params_list**

Ctrl Command	start_loop
Function	Starts a repeating automatic list change.
Call	`start_loop()`
Multi-board Com’d Name	**n_start_loop**
Comments	• A list change or execution restart activated with **start_loop** repeats until execution is ended by calling **quit_loop**. • **start_loop** can be called at any time but only has effect upon the next **set_end_of_list**. • If the automatic list change is activated during processing of a list, then upon reaching **set_end_of_list** execution continues without delay at the other list. If there is only *one* list (`Mem2` = 0, see **config_list**), then upon reaching **set_end_of_list** execution continues at Pos = 0 (that is, at the list’s beginning). • During processing of a list, the other list (and also the current list) can be newly loaded, see Chapter 6.4.6 ”Changing Lists Automatically”, page 113. • So that **start_loop** can function at all, the already active list must absolutely be finalized by **set_end_of_list**; the new list should already be loaded and the input pointer should be sufficiently ahead of the output pointer (otherwise, “old” commands are executed). If, during list execution, the end of the list is reached without encountering a **set_end_of_list**, then execution automatically continues at the beginning of the current list, not at the beginning of the other list. • If, during processing of a list, **start_loop** *and* **auto_change_pos**(`Pos` > 0) are called, then upon the next **set_end_of_list** the command **auto_change_pos**(`Pos` > 0) is executed; and at the next one **start_loop** is executed. • The current List Status can be queried by **read_status**. • The current List Execution Status can be queried by **get_status**. • **start_loop** triggers a flush of the buffered list input, see Chapter 6.4.1 ”Loading Lists”, page 108. • **start_loop** is allowed in Boot Phase 2, see also Chapter 16.7.6 ”Automatic Booting – Process in Detail”, page 994.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**quit_loop**

Ctrl Command	stepper_abs
Function	Triggers set-position motions to the specified absolute set positions by both stepper motor output ports.
Call	`stepper_abs( Pos1, Pos2, WaitTime )`
Parameters	`Pos1` Absolute set position in CLOCK pulse units for stepper motor output ports 1. As signed 32-bit values. Allowed value range: [-2^{31}…+(2^{31}-1)].
	`Pos2` Like `Pos1` (analogously).
	`WaitTime` Determines when **stepper_abs** returns at the latest. As an unsigned 32-bit value. *1 bit equals 1 s*. Allowed value range: [0…+(2^{32}–1)].
Multi-board Com'd Name	**n_stepper_abs**
Comments	• For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 306. • **stepper_abs** sets the new set-position values even if a previously started set-position motion is still in progress: – If `Pos1`/`Pos2` in the current direction of motion lies in front of the internal position variable's value, then the motion continues and the Busy status remains set. – If `Pos1`/`Pos2` equals the current value of the internal position variable, then the motion stops. The Busy status gets reset. – If `Pos1`/`Pos2` in the current direction of motion already lies past the internal position variable's value, then the corresponding stepper motor's direction of motion reverses, see Section "Notes", page 306. The Busy status remains set. • If no set-position motion is in progress, then one starts and the Busy status gets set. • During performance of a reference run (Init status set, see **stepper_init**), **stepper_abs** does not execute (**get_last_error** return code `RTC6_PARAM_ERROR`). • If the CLOCK pulse period has been set to 0 by **stepper_init**, **stepper_control** or **stepper_control_list**, then no stepper motor motion occurs at the corresponding stepper motor output. • If `WaitTime` = 0, then **stepper_abs** returns immediately so that system control is restored to the user program.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**stepper_abs_no**, **stepper_rel**, **stepper_rel_no**, **stepper_abs_list**

Undelayed Short List Command	stepper_abs_list	
Function	Like **stepper_abs**, but a list command and without `WaitTime` parameter.	
Call	`stepper_abs_list( Pos1, Pos2 )`	
Parameters	`Pos1`	Like **stepper_abs**.
	`Pos2`	Like **stepper_abs**.
Multi-board Com'd Name	**n_stepper_abs_list**	
Comments	• See **stepper_abs**. • During performance of a reference run (see **stepper_init**), execution of **stepper_abs_list** is delayed until the reference run completes.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**stepper_abs**	

Ctrl Command	stepper_abs_no	
Function	Triggers a set-position motion to the specified absolute set position at *one* stepper motor output port.	
Call	`stepper_abs_no( No, Pos, WaitTime )`	
Parameters	`No`	Number of the stepper motor output port. As an unsigned 32-bit value. Allowed values: = 1: Stepper motor output port 1. = 2: Stepper motor output port 2. If the value is invalid, then **stepper_abs_no** is not executed (**get_last_error** return code `RTC6_PARAM_ERROR`).
	`Pos`	Absolute set position in CLOCK pulse units. As a signed 32-bit value. Allowed value range: $[-2^{31}...+(2^{31}-1)]$.
	`WaitTime`	Determines when **stepper_abs_no** returns at the latest. *1 bit equals 1 s.* As an unsigned 32-bit value. Allowed value range: $[0...+(2^{32}-1)]$.
Multi-board Com'd Name	**n_stepper_abs_no**	
Comments	• A set-position motion is only performed at the stepper motor output port specified by `No`. Otherwise, **stepper_abs_no** is identical to **stepper_abs** (see comments there).	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**stepper_abs**, **stepper_abs_no_list**	

Undelayed Short List Command	**stepper_abs_no_list**
Function	Like **stepper_abs_no**, but a list command and without `WaitTime` parameter.
Call	`stepper_abs_no_list( No, Pos )`
Parameters	`No` Number of the stepper motor output. As an unsigned 32-bit value. Allowed values: = 1: Stepper motor output 1. = 2: Stepper motor output 2. If the value is invalid, then **stepper_abs_no_list** is, already during loading, replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).
	`Pos` Like **stepper_abs_no**.
Multi-board Com'd Name	**n_stepper_abs_no_list**
Comments	• See **stepper_abs_no**. • During performance of a reference run (see **stepper_init**), execution of **stepper_abs_no_list** is delayed until the reference run completes.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**stepper_abs_no**

Ctrl Command	stepper_control
Function	Sets the CLOCK signal pulse periods for stepper motor control.
Call	`stepper_control( Period1, Period2 )`
Parameters	`Period1` Pulse period of the CLOCK signals for stepper motor output ports 1. 1 bit equals 10 μs. As a signed 32-bit value. Allowed values: [0…+(2^{24}-1)] or < 0. Larger values are clipped. > 0: The new period waits for an already-running CLOCK pulse period at the corresponding stepper motor output before starting. = 0: An already-running CLOCK pulse period aborts at each stepper motor output (the corresponding stepper motor motion gets stopped, the Init- and/or Busy statuses for the respective stepper motor outputs get reset). < 0: the corresponding stepper motor control remains unchanged.
	`Period2` Pulse period of the CLOCK signals for stepper motor output ports 2. Otherwise, like `Period1`.
Multi-board Com'd Name	**n_stepper_control**
Comments	• For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 306. • `Period1` or `Period2` = 0 can be used as an emergency stop, see also the Section "Terminating Infinite Motions", page 308.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**stepper_control_list**

Undelayed Short List Command	stepper_control_list
Function	Like **stepper_control**, but a list command.
Call	`stepper_control_list( Period1, Period2 )`
Parameters	`Period1` Like **stepper_control**.
	`Period2` Like **stepper_control**.
Multi-board Com'd Name	**n_stepper_control_list**
Comments	• See **stepper_control**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**stepper_control**

Ctrl Command	**stepper_disable_switch**
Function	Controls the usage of the stepper motor control SWITCH signals.
Call	`stepper_disable_switch( Disable1, Disable2 )`
Parameters	`Disable1` Instruction how the SWITCH signals from stepper motor input port 1 are to be used. As a signed 32-bit value.. Allowed value range: [-2^{32}…$+(2^{32}-1)$]. > 0: The SWITCH signal at the stepper motor input port is not used. = 0: The SWITCH signal at the stepper motor input port is used. < 0: The use of the stepper motor input signal remains unchanged.
	`Disable2` Like `Disable1` (analogously).
Multi-board Com'd Name	**n_stepper_disable_switch**
Comments	• For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 306. • The limit switch can be ignored during normal forwarding motions. For example, this may make sense for continuously rotating axes. • The SWITCH signals are always used with forwarding motions initiated by **stepper_init**.
Version Info	Available as of DLL 542, OUT 542.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**stepper_init**

Ctrl Command	**stepper_enable**
Function	Sets the ENABLE signals of the stepper motor control.
Call	`stepper_enable( Enable1, Enable2 )`
Parameters	`Enable1` ENABLE signal for stepper motor output 1. As a signed 32-bit value. Allowed value range: [-2^{32}…+(2^{32}–1)]. > 0: The ENABLE signal at the stepper motor output gets set. = 0: The ENABLE signal at the stepper motor output gets reset. < 0: The stepper motor output signal remains unchanged.
	`Enable2` Like `Enable1` (analogously).
Multi-board Com'd Name	**n_stepper_enable**
Comments	• For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 306.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**stepper_enable_list**

Undelayed Short List Command	**stepper_enable_list**
Function	Like **stepper_enable**, but a list command.
Call	`stepper_enable_list( Enable1, Enable2 )`
Parameters	`Enable1` Like **stepper_enable**.
	`Enable2` Like **stepper_enable**.
Multi-board Com'd Name	**n_stepper_enable_list**
Comments	• See **stepper_enable**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**stepper_enable**

Ctrl Command	**stepper_init**	
Function	Performs a stepper motor initialization.	
Call	`stepper_init( No, Period, Dir, Pos, Tol, Enable, WaitTime )`	
Parameters	`No`	Number of the stepper motor output. As an unsigned 32-bit value. Allowed values: = 1: Stepper motor output 1. = 2: Stepper motor output 2. If the value is invalid, then **stepper_init** is not executed (**get_last_error** return code `RTC6_PARAM_ERROR`).
	`Period`	Pulse period of the CLOCK signal. As an unsigned 32-bit value. 1 bit equals 10 µs. Allowed value range: [0…+(2^{24}-1)]. Larger values are clipped. If `Period` = 0, then no reference run is performed and the function immediately returns (see comments).
	`Dir`	Direction of stepper motor motion during the reference run. As a signed 32-bit value. Allowed value range: [–2^{31}…+(2^{31}-1)]. > 0: The DIRECTION signal gets set; during the reference run the internal position variable increments. = 0: The DIRECTION signal gets reset; during the reference run the internal position variable decrements. < 0: The DIRECTION signal remains unchanged, no reference run is performed and the function immediately returns.
	`Pos`	New position variable value. In CLOCK pulse units (see comments). As a signed 32-bit value. Allowed value range: [–2^{31}…+(2^{31}-1)].
	`Tol`	Tolerance for the reference run (see comments). As an unsigned 32-bit value. Allowed value range: [0…+(2^{32}–1)]. If `Dir` < 0 and/or `Period` = 0, then the value `Tol` = 0 is irrelevant, otherwise `Tol` = 0 causes **stepper_init** not to be executed (**get_last_error** return code `RTC6_PARAM_ERROR`).
	`Enable`	ENABLE signal. As an unsigned 32-bit value. Allowed value range: [0…+(2^{32}–1)]. = 0: The ENABLE signal is reset. > 0: The ENABLE signal is set.
	`WaitTime`	Defines when **stepper_init**, at the latest, returns (see comments). *1 bit equals 1 s.* As an unsigned 32-bit value. Allowed value range: [0…+(2^{32}–1)].
Multi-board Com'd Name	**n_stepper_init**	

Ctrl Command	stepper_init
Comment	• For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 306. • **stepper_init** immediately stops all previously started motions of the stepper motor specified by the `No` parameter. • The ENABLE signal `Enable` is merely forwarded and always correspondingly set, but has no effect on internal operations. • If `Period` > 0 and `Dir` ≥ 0, then stepper motor `No` starts a reference run with the supplied CLOCK pulse period in the defined direction and the Init status gets set. The first Clock pulse is only generated after a full CLOCK pulse period. – If a limit switch is activated right from the beginning, then the controller attempts to seek a position within the ±`Tol` range of the current position, initially opposite to the defined direction, with the limit switch deactivated. If this does not bring success, then the attempt terminates. In this case, the SWITCH status bit remains set (see **get_stepper_status**). – If a limit switch gets activated during a motion, then the reference run stops there. Afterward, the limit switch position is crossed 4× to arrive at an averaged value for this position. Finally, the stepper motor is driven in the opposite direction by a normal set-position motion (Init status reset, Busy status set) within the tolerance value `Tol`. Here, the DIRECTION status signal changes, whereas it remains constant during seeking motions with multiple direction changes. The internal position variable (for the current position) gets set to the value defined by the `Pos` parameter. Thus, this value represents a positional offset by `Tol` with respect to the defined position. With `Pos` = `Tol`, the middle limit switch position corresponds to position 0. – If *no* limit switch is found (for example, because no limit switch exists in the defined direction), then the stepper motor performs an infinite motion. **stepper_init** then returns after `WaitTime` seconds to restore system control to the user program. But the stepper motor's infinite motion continues until it is aborted by a new **stepper_init** command or **stepper_control**(`Period1`/`Period2` = 0). For more on this, see the Section "Terminating Infinite Motions", page 308. • If `Period` = 0, `Dir` < 0 and/or `WaitTime` = 0, then **stepper_init** returns straight away to immediately restore system control to the user program. You can then call **get_stepper_status** to check whether the reference run has completed. During the seeking motion the status "Init" (see page 466) is set and during the terminating set-position motion to (limit switch + `Tol`) the status "Busy" (see page 466) is set. • `Period` = 0 and/or `Dir` < 0 can be used as an emergency stop. Then a previously started stepper motor motion gets aborted, but no reference run is performed. Here, too, the position variable gets set to the value `Pos`. The DIRECTION signal remains unchanged. • If `Period` = 0, then status "Init" (see page 466) and status "Busy" (see page 466) get reset. No further clock pulses are outputted until `Period` is again set to a positive value.

Ctrl Command	stepper_init
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Ctrl Command	stepper_rel	
Function	Triggers set-position motions to the specified relative positions by both stepper motor output ports.	
Call	`stepper_rel( dPos1, dPos2, WaitTime )`	
Parameters	`dPos1`	Relative set positions in CLOCK pulse units for stepper motor output port 1. As a signed 32-bit value. Allowed value range: $[-2^{31}...+(2^{31}-1)]$.
	`dPos2`	Relative set positions in CLOCK pulse units for stepper motor output port 2. As a signed 32-bit value. Allowed value range: $[-2^{31}...+(2^{31}-1)]$.
	`WaitTime`	Like **stepper_abs**.
Multi-board Com'd Name	**n_stepper_rel**	
Comments	• The set positions should be specified relative to the current position values. Otherwise, **stepper_rel** is identical to **stepper_abs** (see comments there).	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**stepper_abs**, **stepper_rel_list**	

Undelayed Short List Command	stepper_rel_list	
Function	Like **stepper_rel**, but a list command and without `WaitTime` parameter.	
Call	`stepper_rel_list( dPos1, dPos2 )`	
Parameters	`dPos1`	Like **stepper_rel**.
	`dPos2`	Like **stepper_rel**.
Multi-board Com'd Name	**n_stepper_rel_list**	
Comments	• See **stepper_rel**. • During performance of a reference run (see **stepper_init**), execution of **stepper_rel_list** is delayed until the reference run completes.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**stepper_rel**	

Ctrl Command	stepper_rel_no	
Function	Triggers a set-position motion to the specified relative position at *one* stepper motor output port.	
Call	`stepper_rel_no( No, dPos, WaitTime )`	
Parameters	`No`	Like **stepper_abs_no**.
	`dPos`	Relative position. In CLOCK pulse units. As a signed 32-bit value. Allowed value range: $[-2^{31} \ldots +(2^{31}-1)]$.
	`WaitTime`	Like **stepper_abs_no**.
Multi-board Com'd Name	**n_stepper_rel_no**	
Comments	• The set positions should be specified relative to the current position values (the -position value is correspondingly get newly set) and a set-position motion is only performed at the stepper motor output specified by `No`. Otherwise, **stepper_rel_no** is identical to **stepper_abs** (see comments there).	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**stepper_abs_no**, **stepper_abs**, **stepper_rel_no_list**	

Undelayed Short List Command	stepper_rel_no_list	
Function	Like **stepper_rel_no**, but a list command and without `WaitTime` parameter.	
Call	`stepper_rel_no_list( No, dPos )`	
Parameters	`No`	Number of the stepper motor output port. As an unsigned 32-bit value. Allowed values: = 1: Stepper motor output port 1. = 2: Stepper motor output port 2. If the value is invalid, then **stepper_rel_no_list** is, already during loading, replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).
	`dPos`	Like **stepper_rel_no**.
Multi-board Com'd Name	**n_stepper_rel_no_list**	
Comments	• See **stepper_rel_no**. • During performance of a reference run (see **stepper_init**), execution of **stepper_rel_no_list** is delayed until the reference run completes.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**stepper_rel_no**	

Normal List Command	**stepper_wait**
Function	Interrupts further execution of a list until a previously started (at the specified stepper motor output) stepper motor motion completes.
Call	`stepper_wait( No )`
Parameters	`No` Number of the stepper motor output. As an unsigned 32-bit value. Allowed values: = 1: Stepper motor output port 1. = 2: Stepper motor output port 2. = 0, 3: Both stepper motor output ports. Only the two least-significant bits are evaluated.
Multi-board Com'd Name	**n_stepper_wait**
Comments	• For programming the stepper motor signals, see Chapter 9.1.5 "Controlling Stepper Motors", page 306. • If *no* stepper motor motion had been previously started at the specified stepper motor output port, then **stepper_wait** still needs 10 µs to execute, even though it otherwise has no effect. • **stepper_wait** does *not* influence: – the Signals for "Laser Active" Operation – the List Status – the List Execution Status
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	–

Ctrl Command	stop_execution
Function	Stops execution of the list and deactivates the "laser active" Laser Control Signals immediately.
Call	`stop_execution()`
Multi-board Com'd Name	**n_stop_execution**
Comments	• **stop_execution** deactivates the Signals for "Laser Active" Operation even if no list is active (here **stop_execution** has no other effects; here too: **get_last_error** return code `RTC6_BUSY`). • With **stop_execution**, the galvanometer scanners stay in the current position, unless a home jump has been previously defined by **home_position** or **home_position_xyz** (a home jump is executed). Therefore, before a new list is loaded, the galvanometer scanners should be set to a defined position using **goto_xy**. • The external start input ports are disabled, see Section "External Start", page 312. • The Processing-on-the-fly correction is switched off. • The BUSY list status values (see **read_status**) and the BUSY list execution status-List Execution Status value (see **get_status**) are reset. • A list that has been interrupted by **stop_execution** *cannot* be resumed. It must instead be newly started (for example, by **execute_list_pos**). To only temporarily halt a list and later resume it, you can use **pause_list**. • **stop_execution** only affects the addressed RTC6 board. In a master/slave chain, **stop_execution** is not passed on to the slave boards. If all RTC6 boards of a master/slave chain are to be synchronously stopped, then **simulate_ext_stop** or an external stop signal must be called to any card of the master/slave chain, see also Chapter 6.6.3 "Master/Slave Operation", page 127.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality. However: Master/slave change.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_startstop_info**

Ctrl Command	stop_list
Function	Pauses execution of the list and deactivates the Signals for "Laser Active" Operation.
Call	`stop_list()`
Multi-board Com'd Name	**n_stop_list**
Comments	• **stop_list** is synonymous with **pause_list** (see comments there).
RTC4→RTC6	Basically unchanged functionality. However: Additional PAUSED list execution status which is set by **stop_list**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**pause_list**

Ctrl Command	stop_trigger
Function	Resets (to `Busy`= 0) the measurement session status that can be queried by **measurement_status**.
Call	`stop_trigger()`
Multi-board Com'd Name	**n_stop_trigger**
Comments	• **stop_trigger** is only needed if a measurement session has been started by **set_trigger[*]**, but not subsequently terminated by: – **set_trigger**(`Period` = 0) – **set_trigger4**(`Period` = 0) – **set_trigger8**(`Period` = 0) Here, you can reset the measurement session status even when no list is active (see comments at **set_trigger**). • **stop_trigger** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • **stop_trigger** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set)
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**measurement_status**, **set_trigger**, **set_trigger4**, **set_trigger8**

Undelayed Short List Command	**store_encoder**
Function	Stores the current counts of the two RTC6 encoder counters in a cache on the RTC6.
Call	`store_encoder( Pos )`
Parameters	`Pos` Storage position. As an unsigned 32-bit value. • Bit #0 = 0: The counter values are saved to storage position 0. • Bit #0 = 1: The counter values are saved to storage position 1.
Multi-board Com'd Name	**n_store_encoder**
Comments	• **store_encoder** can be used, for instance, to determine the exact number of encoder increments occurring within a specific list-processing period. Here, you could begin the command list by saving the counts in storage position 0 by `store_encoder(0)`. At the end of the command list, you could save the counts to storage position 1 by `store_encoder(1)` and subsequently retrieve by **read_encoder**. • See also Chapter 8.6 "Processing-on-the-fly", page 251 and Chapter 9.3.3 "Synchronization by Encoder Signals", page 320.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**read_encoder**, **get_encoder**, **set_fly_x**, **set_fly_y**, **set_fly_rot**, **wait_for_encoder**

Ctrl Command	store_program
Function	Standalone Functionality: Saves (deletes) data for automatic booting to (from) the NAND Memory.
Prerequisite	RTC6 Software Package ≥ V1.7.0 and BIOS-ETH ≥ 26.
Call	`Error = store_program( Mode )`
Parameters	`Mode` = 0: Saves data for "Standalone Basic State" (see below). = 1: Erases the NAND Memory content. = 2: Like = 0, and in addition some files for "Standalone Full State" (see below). ≥ 3: Like = 2, but no list commands. Designed for Remote Interface Mode only (as list commands can be loaded later there). As an unsigned 32-bit value.
Result	`Error` Error code. As an unsigned 32-bit value. Value Description 0 No error. 1 NAND Memory not addressable. 2 NAND Memory end reached early. 3 Data have not been or only partially stored in NAND Memory. 4 Not an RTC6 Ethernet Board.
Multi-board Com'd Name	**n_store_program**
Comments	• **store_program** is only allowed with RTC6 Ethernet Boards. Otherwise, a **get_last_error** return code `RTC6_TYPE_REJECTED` is generated. • **store_program** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – a list has been paused by **set_wait** (PAUSED list execution status set) • During the execution of **store_program** the 10 μs clock cycle of the DSP is interrupted: – `Mode` = 0 approx. 02 s – `Mode` = 2 approx. 60 s – `Mode` = 3 approx. 10 s • Data for "Standalone Basic State" are: – `RTC6ETH.out`, `RTC6RBF.rbf` and `RTC6DAT.dat` • Data for "Standalone Full State" are: – Data for "Standalone Basic State" (see bullet above) – The required control commands, list commands and correction files • In case of an error, a **get_last_error** return code `RTC6_FLASH_ERROR` is generated. • See Chapter 16.7 "Standalone Functionality", page 988.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 618, OUT 618, RBF 623. Last change DLL 640, OUT 641: `Mode` ≥ 3.
References	**read_image_eth**, **write_image_eth**

Ctrl Command	store_timestamp_counter
Function	Saves the current 32-bit "Timestamp Counter" value.
Call	`store_timestamp_counter()`
Parameters	None.
Multi-board Com'd Name	**n_store_timestamp_counter**
Comments	• See Chapter 8.12 "Time Measurements", page 290. • The current 32-bit "Timestamp Counter" value is stored as time reference `TimeStampStorage` for **wait_for_timestamp_counter**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**store_timestamp_counter_list**, **wait_for_timestamp_counter**

Undelayed Short List Command	**store_timestamp_counter_list**
Function	Like **store_timestamp_counter**, but a list command.
Call	`store_timestamp_counter_list()`
Parameters	None.
Multi-board Com'd Name	**n_store_timestamp_counter_list**
Comments	• See **store_timestamp_counter**.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**store_timestamp_counter**, **wait_for_timestamp_counter**

Undelayed Short List Command	**sub_call**
Function	Causes an unconditional jump to an indexed subroutine.
Call	`sub_call( Index )`
Parameters	`Index` Index of the called indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0…1023].
Multi-board Com'd Name	**n_sub_call**
Comments	• **sub_call** reads the indexed subroutine's starting address from the internal management table based on the supplied index and then calls **list_call** (see also the comments there). **list_call** then triggers the jump to the subroutine. • **sub_call** starts indexed subroutines in protected memory (that were loaded and/or referenced by **load_sub**, **load_disk** or **copy_dst_src**) as well as indexed subroutines in the unprotected list area (that were referenced by **set_sub_pointer** or **copy_dst_src**). • If no subroutine is referenced for the supplied index, then the jump is suppressed and execution continues at the command located after the calling position. If applicable, a **list_continue** is executed. `get_sub_pointer( Index )` can be used to determine whether a subroutine has been referenced for a particular index. If *no* subroutine has been referenced, this command returns the value "–1" (= 2^{32}–1). • If `Index` > 1023, then **sub_call** is, already during loading, replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • Absolute Vector Commands and "Arc" Commands execute absolutely after being called with **sub_call**. If the subroutine needs to execute at various locations within the Image Field, then either the subroutine can only contain relative [*]mark[*] Commands, arc commands and Jump Commands or **sub_call_abs** must be used instead.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**list_call**, **sub_call_abs**, **sub_call_cond**

Undelayed Short List Command	**sub_call_abs**
Function	Causes an unconditional jump to an indexed subroutine. In the called subroutine, any absolute Vector Commands and "Arc" Commands receive an offset (corresponding to the current coordinates at the time of the call).
Call	`sub_call_abs( Index )`
Parameters	`Index` Index of the called indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0…1023].
Multi-board Com'd Name	**n_sub_call_abs**
Comments	• **sub_call_abs** reads the indexed subroutine's starting address from the internal management table based on the supplied index and then calls **list_call_abs** (see also the comments there). **list_call_abs** then triggers the jump to the subroutine. • If the called subroutine contains no absolute commands, then there is no difference between **sub_call_abs** and **sub_call**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**sub_call**, **sub_call_abs_cond**

Undelayed Short List Command	**sub_call_abs_cond**
Function	*Conditional call (AbsCall) of an indexed subroutine*: **sub_call_abs_cond** executes **sub_call_abs**(`Index`), if the current `IOvalue` at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector meets the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are `1` and the bits specified in `Mask0` are `0`). Otherwise, the directly following list command is immediately executed.
Call	`sub_call_abs_cond( Mask1, Mask0, Index )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.
	`Mask0` See `Mask1`.
	`Index` Index of the called indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0…1023].
Multi-board Com'd Name	**n_sub_call_abs_cond**
Comments	• See **sub_call_abs**. • See also Chapter 9.3.2 "Execution of Conditional Commands", page 317.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**sub_call_abs**

Undelayed Short List Command	**sub_call_abs_repeat**
Function	Causes an unconditional jump to an indexed subroutine and executes its body several times.
Call	`sub_call_abs_repeat( Index, Number )`
Parameters	`Index` Index of the to be called indexed subroutine (as with **sub_call_abs**). `Number` Number of repetitions. 0 is treated as 1. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_sub_call_abs_repeat**
Comments	• `sub_call_abs( Index )` is synonymous with `sub_call_abs_repeat( Index, 1 )`. • See **sub_call_repeat** and **sub_call_abs**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**sub_call_repeat**, **sub_call_abs**, **sub_call**

Undelayed Short List Command	**sub_call_cond**
Function	*Conditional call of an indexed subroutine*: **sub_call_cond** executes **sub_call**(`Index`), if the current `IOvalue` at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector meets the following condition: `((IOvalue AND Mask1) = Mask1) AND (((not IOvalue) AND Mask0) = Mask0)` (= if the bits specified in `Mask1` are `1` and the bits specified in `Mask0` are `0`). Otherwise, the directly following list command is immediately executed.
Call	`sub_call_cond( Mask1, Mask0, Index )`
Parameters	`Mask1` 16-bit mask. As an unsigned 32-bit value. Only the lower 16 bits are evaluated.
	`Mask0` See `Mask1`.
	`Index` Index of the to-be-called indexed subroutine. As an unsigned 32-bit value. Allowed value range: [0…1023].
Multi-board Com'd Name	**n_sub_call_cond**
Comments	• See **sub_call**. • See also Chapter 9.3.2 "Execution of Conditional Commands", page 317.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**sub_call**

Undelayed Short List Command	**sub_call_repeat**
Function	Causes an unconditional jump to an indexed subroutine and executes its body several times.
Call	`sub_call_repeat( Index, Number )`
Parameters	`Index` Index of the to be called indexed subroutine (as with **sub_call**). `Number` Number of repetitions. As an unsigned 32-bit value. `Number` = 0 is treated like `Number` = 1.
Multi-board Com'd Name	**n_sub_call_repeat**
Comments	• `sub_call( Index )` is synonymous with `sub_call_repeat( Index, 1 )`. • **sub_call_repeat** avoids an empty cycle at the repetition, which otherwise inevitably occurs with `sub_call`...`sub_call` or `list_repeat`...`sub_call`...`list_until` constructions. • By **sub_call_repeat**, for example, trajectories (see Glossary entry on page 31) from micro vector commands for runup curves and coast down curves can be seamlessly joined together with shapes from subroutines.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**sub_call_abs_repeat**, **sub_call**, **sub_call_abs**, **micro_vector_abs**,**micro_vector_abs_3d**,**micro_vector_rel**,**micro_vector_rel_3d**

Undelayed Short List Command	**switch_ioport**	
Function	Executes a relative list jump **list_jump_rel** (`Pos`) whose jump distance `Pos` (>1) is determined at runtime by the current value (`IOvalue`) at the 16-Bit Digital Input Port of the EXTENSION 1 Socket Connector. You can specify which of the 16-Bit Digital Input Port bits are to be evaluated for this purpose.	
Call	`switch_ioport( MaskBits, ShiftBits )`	
Parameters	`MaskBits`	Number of contiguous bits of the 16-Bit Digital Input Port to be evaluated for determining the jump distance. As an unsigned 32-bit value. Allowed value range: [1…16].
	`ShiftBits`	Position of the least significant to-be-evaluated bit of the 16-Bit Digital Input Port. As an unsigned 32-bit value. Allowed value range: [0…15].
Multi-board Com'd Name	**n_switch_ioport**	
Comments	• With invalid values of `MaskBits` or `ShiftBits` and with (`MaskBits` + `ShiftBits`) > 16, **switch_ioport** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • The following applies: `Mask` = ((1 << `MaskBits`) – 1) << `ShiftBits` and `SwitchNo` = (`Mask` & `IOvalue`) >> `ShiftBits`. Here, a **list_jump_rel**(`Pos`) with `Pos` = (`SwitchNo` + 1) list positions are then executed. • The jump distance is at least 1. This prevents infinite loops when no signal is present. Jumps to the same address (`Pos` = 0) are not possible with **switch_ioport**, but can be simulated by **list_jump_rel** (–1) as the directly subsequent command. • The maximum jump distance is 2^{16} list positions. • See also **list_jump_rel**. • See also Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81 and Chapter 9.3.2 "Execution of Conditional Commands", page 317.	
Example (Pascal)	• It is assumed that the current value at the 16-Bit Digital Input Port is $F152 at runtime: then **switch_ioport**($0008, $0004) executes **list_jump_rel**($0016), that is, a relative list jump of length 22 list positions.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**list_jump_rel**, **list_jump_rel_cond**	

Ctrl Command	**sync_slaves**
Function	No function. **sync_slaves** is no longer necessary as of DLL 614, OUT 614, RBF 619. Synchronizes all slave boards (connected in a master/slave chain with the slave connection of the addressed RTC6 board) stably to the 10 μs clock cycle of the addressed RTC6 board.
Call	`sync_slaves()`
Multi-board Com'd Name	**n_sync_slaves**
Comments	• For usage of **sync_slaves**, see Chapter 6.6.3 "Master/Slave Operation", page 127. • With RTC6 Software Package ≥ V1.5.2, the actions described below for RTC6 Software Packages <V1.5.0 are no longer executed. • RTC6 Software Packages <V1.5.0 (< RBF 619): – SCANLAB recommends executing synchronization immediately after all boards have been initialized by **load_program_file** and **load_correction_file**. Otherwise, all involved boards (that is, the master board and the downstream slave boards allocated to the user program) should already have been halted prior to the call of **sync_slaves**. To avoid irregularities during execution of **sync_slaves**, you should neither apply external stop signals to the boards nor trigger External Starts (this is not automatically prevented). – During the course of **sync_slaves**, a **simulate_ext_stop** is passed to the addressed board. This halts the addressed board and all downstream slave boards in the master/slave chain (including boards not allocated to the user program). Users themselves are responsible to ensure that any running processes are not disrupted by that. – After execution of **sync_slaves**, the scan system axes of all involved boards are in either the coordinate center position (0, 0 [,0]) or the HomeJump position (possibly shifted by an offset set by **set_offset**, **set_defocus** or **set_hi**). – **sync_slaves** (relating to synchronization) only affects RTC6 boards connected in a master/slave chain to the Master connector of the addressed RTC6 board. It does not affect the addressed board itself or any boards connected to the Slave connector of the addressed board. Therefore, if all slave boards of a master/slave chain is to be synchronized with the master board, then **sync_slaves** must address the master board of the master/slave chain. **sync_slaves** has no effect, if no board is connected to the Master connector of the addressed board.

Ctrl Command	sync_slaves
Comments (cont'd)	– Synchronization of downstream slave boards by **sync_slaves** occurs even when the boards' BUSY list execution status or INTERNAL-BUSY list execution status is set (they are automatically halted). Nevertheless, the only slave boards to get synchronized are those allocated to the user program (allocation is not requested automatically). If the user program possesses access rights for the addressed board but no further boards, then **sync_slaves** has no effect. – During the course of **sync_slaves**, all **get_startstop_info** error bits are cleared on all boards (including upstream boards) allocated to the user program. • **sync_slaves** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status of the addressed board is currently set – the INTERNAL-BUSY list execution status of the addressed board is set • **sync_slaves** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status is set)
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality. However: Master/Slave functionality has been changed.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 615: **sync_slaves** has no function anymore.
References	**get_master_slave**, **get_sync_status**, **master_slave_config**

Ctrl Command	time_control_eth
Function	Sets a parameter to fine-tune the accuracy of the real-time clock.
Call	`time_control_eth( PPM )`
Parameters	`PPM` Deviation. As a 64-bit IEEE floating point value. Allowed value range: `PPM` / 4.34 = [–64…+63]
Multi-board Com'd Name	**n_time_control_eth**
Comments	• The execution of **time_control_eth** can take several 100 μs, see **time_update**. • **time_control_eth** is not executed with RTC6 PCIe Boards. • `PPM` means the deviation in parts per million. A positive value slows down the clock.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 612, ETH 612, RBF 617.
References	**time_update**

Normal List Command	time_fix
Function	Stores the current time and date of the RTC clock/calender in a cache for use with **mark_date** and **mark_time**.
Call	`time_fix()`
Multi-board Com'd Name	**n_time_fix**
Comments	• **time_fix** is synonymous with **time_fix_f_off** with `FirstDay` = 0 and `Offset` = 0 (see comments there).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**time_fix_f**, **time_fix_f_off**

Normal List Command	time_fix_f
Function	Stores the current time and date of the RTC clock/calender in a cache for use with **mark_date** and **mark_time**.
Call	`time_fix_f( FirstDay )`
Parameters	`FirstDay` Like **time_fix_f_off**.
Multi-board Com'd Name	**n_time_fix_f**
Comments	• **time_fix_f** is synonymous with **time_fix_f_off** with `Offset` = 0 (see comments there).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**time_fix_f**, **time_fix_f_off**, **time_update**, **mark_time**, **mark_date**

Normal List Command	**time_fix_f_off**	
Function	Stores the current time and date of the RTC clock/calender or a forward-dated date and time for use with **mark_date** and **mark_time** in a cache.	
Call	`time_fix_f_off( FirstDay, Offset )`	
Parameters	`FirstDay`	Defines the starting number for determining the Julian calendar day from the current date of the RTC calendar: Counting proceeds from `FirstDay` to `FirstDay` + 364 (+1 for leap years). As an unsigned 32-bit value.
	`Offset`	Forward dating. In seconds. As an unsigned 32-bit value. Allowed value range: [0…(2^{32}–1)].
Multi-board Com'd Name	**n_time_fix_f_off**	
Comments	• Before calling **time_fix_f_off**, **time_fix_f** or **time_fix**, synchronization of the RTC6 and PC time should be performed (for RTC6 boards, at least once after each **load_program_file**) by **time_update**. • The complete time can be marked through multiple calls of **mark_time** and the complete date through multiple calls of **mark_date**. **time_fix_f_off**, **time_fix_f** or **time_fix** must therefore be called *before* these marking commands so that the to-be-marked time or date do not change during marking. • If **time_fix_f_off**, **time_fix_f** or **time_fix** are not called again before a time or date marking, then the last marked time is marked again. If **time_fix_f_off**, **time_fix_f** or **time_fix** are not called at all after a **load_program_file**, then a time of 00:00 or a date of January 1, 2000 is marked. • If `Offset` = 0, then the current date and current time are fixed. One practical use of forward dating (`Offset` > 0) is for setting a date of expiry based on the current date. • Backdating (`Offset` < 0) is not possible.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**time_fix**, **time_fix_f**, **time_update**, **mark_time**, **mark_date**	

Ctrl Command	time_update
Function	Sets the 24-hour clock and calendar of the RTC6 board to the current PC time.
Call	`time_update()`
Multi-board Com'd Name	**n_time_update**
Comments	• **time_update** must be called after each **load_program_file**, if 24-hour clock and calendar of the board are to be synchronized with the PC time. • The base value for internal time counting is set to January 1, 2000, 00:00 by **load_program_file** whereas to the PC time by **time_update**. An internal seconds counter is set to 0 by **load_program_file** or by **time_update**, but is always driven by the quartz-controlled 10 μs clock. • Before marking with **mark_date** or **mark_time**, you must call **time_fix**, **time_fix_f** or **time_fix_f_off** so that the current time can be captured (as sum of the base value and the current value of the internal seconds counter) and formatted. • The RTC6 Ethernet Board a real-time clock. Therefore, **time_update** must be called only once. **time_update** takes several hundred μs. During this time the 10 μs clock cycle of the RTC6 Ethernet Board is interrupted. Therefore, **time_update** should not be called during list processing. The clock continues to run even if the power supply is switched off (> about 1 week). See also **time_control_eth**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600. Last change DLL 612, ETH 612, RBF 617: real-time clock of the RTC6 Ethernet Board.
References	**time_fix**, **time_fix_f**, **time_fix_f_off**, **mark_date**, **mark_time**, **time_control_eth**

Normal List Command	timed_arc_abs
Function	Moves the laser focus for the specified marking duration from the current position along an arc with the specified angle and center point (absolute coordinate values) within a 2D Image Field.
Call	`timed_arc_abs( X, Y, Angle, T )`
Parameters	`X` Like **arc_abs**.
	`Y` Like **arc_abs**.
	`Angle` Like **arc_abs**.
	`T` Duration of the complete arc marking process. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_arc_abs** behaves like **arc_abs**.
Multi-board Com'd Name	**n_timed_arc_abs**
Comments	• Unlike **arc_abs**, **timed_arc_abs** does not execute the marking process with the specified (by **set_mark_speed** or **set_mark_speed_ctrl**) mark speed. Instead, the speed (that is, the number of Microsteps) is adjusted so that the arc lasts as long as specified (see Chapter 8.9 "Timed Commands", page 283). The total marking time is (for `T` ≥ 5) the sum of the specified (rounded) time and the set delays. • See also comments on **arc_abs**.
RTC4→RTC6	New command. See **arc_abs**.
RTC5→RTC6	Unchanged functionality. See **arc_abs**.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**arc_abs**, **timed_arc_rel**

Normal List Command	timed_arc_rel	
Function	Moves the laser focus for the specified marking duration from the current position along an arc with the specified angle and center point (relative coordinate values) within a 2D Image Field.	
Call	`timed_arc_rel( dX, dY, Angle, T )`	
Parameters	`dX`	Like **arc_rel**.
	`dY`	Like **arc_rel**.
	`Angle`	Like **arc_rel**.
	`T`	Duration of the complete arc marking process. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_arc_rel** behaves like **arc_rel**.
Multi-board Com'd Name	**n_timed_arc_rel**	
Comments	• The coordinates for the arc center are to be supplied as relative coordinates with respect to the current position. Otherwise, **timed_arc_rel** is analogous to **timed_arc_abs** (see the comments there).	
RTC4→RTC6	New command. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `dX` and `dY` by 16. The allowed value ranges decrease accordingly.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**timed_arc_abs**, **arc_rel**	

Normal List Command	timed_jump_abs
Function	Moves the output point (of the laser focus) for the specified jump duration along a 2D vector from the current position to the specified position (absolute coordinate values) within a 2D Image Field.
Call	`timed_jump_abs( X, Y, T )`
Parameters	`X` Like **jump_abs**.
	`Y` Like **jump_abs**.
	`T` Duration of the complete jump vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_jump_abs** behaves like **jump_abs**.
Multi-board Com'd Name	**n_timed_jump_abs**
Comments	• Unlike **jump_abs**, **timed_jump_abs** does not execute the jump with the specified (by **set_jump_speed** or **set_jump_speed_ctrl**) jump speed. Instead, the speed (that is, the number of Microsteps) is adjusted so that the vector lasts as long as specified, see Chapter 8.9 "Timed Commands", page 283. The total jump time is (for `T` ≥ 5) the sum of the specified (rounded) time and the set delays. • See also comments on **jump_abs**.
RTC4→RTC6	Unchanged functionality. In addition: increased value range. In RTC4 Compatibility Mode, the RTC6 multiplies the specified values for `X` and `Y` by 16. The allowed value ranges decrease accordingly.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**jump_abs**, **timed_jump_rel**, **timed_jump_abs_3d**

Normal List Command	**timed_jump_abs_3d**	
Function	Moves the output point (of the laser focus) for the specified jump duration along a 3D vector from the current position to the specified position (absolute coordinate values) within the 3D Image Field.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **timed_jump_abs_3d** has the same effect as **timed_jump_abs**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.	
Call	`timed_jump_abs_3d( X, Y, Z, T )`	
Parameters	`X`	Like **jump_abs_3d**.
	`Y`	Like **jump_abs_3d**.
	`Z`	Like **jump_abs_3d**.
	`T`	Duration of the complete jump vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_jump_abs_3d** behaves like **jump_abs_3d**).
Multi-board Com'd Name	**n_timed_jump_abs_3d**	
Comments	• Except for the additional motion in the third dimension, **timed_jump_abs_3d** functions similarly to the **timed_jump_abs** command (see the comments there). • See also comments on **jump_abs_3d**.	
RTC4→RTC6	New command. See **jump_abs_3d**.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**timed_jump_abs**, **jump_abs_3d**, **timed_jump_rel_3d**	

Normal List Command	**timed_jump_rel**	
Function	Moves the output point (of the laser focus) for the specified jump duration along a 2D vector from the current position to the specified position (relative coordinate values) within a 2D Image Field.	
Call	`timed_jump_rel( dX, dY, T )`	
Parameters	`dX`	Like **jump_rel**.
	`dY`	Like **jump_rel**.
	`T`	Duration of the complete jump vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_jump_rel** behaves like **jump_rel**.
Multi-board Com'd Name	**n_timed_jump_rel**	
Comments	• The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **timed_jump_rel** is identical to **timed_jump_abs** (see the comments there).	
RTC4→RTC6	Unchanged functionality. In addition: increased value range. See **jump_rel**.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**timed_jump_abs**, **jump_rel**, **timed_jump_rel_3d**	

Normal List Command	timed_jump_rel_3d	
Function	Moves the output point (of the laser focus) for the specified jump duration along a 3D vector from the current position to the specified position (relative coordinate values) within the 3D Image Field.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **timed_jump_rel_3d** has the same effect as **timed_jump_rel**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.	
Call	`timed_jump_rel_3d( dX, dY, dZ, T )`	
Parameters	`dX`	Like **jump_rel_3d**.
	`dY`	Like **jump_rel_3d**.
	`dZ`	Like **jump_rel_3d**.
	`T`	Duration of the complete jump vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_jump_rel_3d** behaves like **jump_rel_3d**.
Multi-board Com'd Name	**n_timed_jump_rel_3d**	
Comments	• The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **timed_jump_rel_3d** is identical to **timed_jump_abs_3d** (see the comments there).	
RTC4→RTC6	New command. See **jump_rel_3d**.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**timed_jump_abs_3d**, **jump_rel_3d**, **timed_jump_rel**	

Normal List Command	**timed_mark_abs**	
Function	Moves the laser focus for the specified marking duration along a 2D vector from the current position to the specified position (absolute coordinate values) within a 2D Image Field.	
Call	`timed_mark_abs( X, Y, T )`	
Parameters	`X`	Like **mark_abs**.
	`Y`	Like **mark_abs**.
	`T`	Duration of the complete mark vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_mark_abs** behaves like **mark_abs**.
Multi-board Com'd Name	**n_timed_mark_abs**	
Comments	• Unlike **mark_abs**, the **timed_mark_abs** command does not execute the marking process with the specified (by **set_mark_speed** or **set_mark_speed_ctrl**) mark speed. Instead, the speed (that is, the number of Microsteps) is adjusted so that the vector lasts as long as specified (see Chapter 8.9 "Timed Commands", page 283). The total marking time is (for `T` ≥ 5) the sum of the specified (rounded) time and the set delays. • See also comments on **mark_abs**.	
RTC4→RTC6	Unchanged functionality. In addition: increased value range. See **mark_abs**.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**mark_abs**, **timed_mark_rel**, **timed_mark_abs_3d**	

Normal List Command	**timed_mark_abs_3d**	
Function	Moves the laser focus for the specified marking duration along a 3D vector from the current position to the specified position (absolute coordinate values) within the 3D Image Field.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **timed_mark_abs_3d** has the same effect as **timed_mark_abs**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.	
Call	`timed_mark_abs_3d( X, Y, Z, T )`	
Parameters	`X`	Like **mark_abs_3d**.
	`Y`	Like **mark_abs_3d**.
	`Z`	Like **mark_abs_3d**.
	`T`	Duration of the complete mark vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_mark_abs_3d** behaves like **mark_abs_3d**.
Multi-board Com'd Name	**n_timed_mark_abs_3d**	
Comments	• Except for the additional motion in the third dimension, **timed_mark_abs_3d** functions similarly to **timed_mark_abs** (see the comments there). • See also comments on **mark_abs_3d**.	
RTC4→RTC6	New command. See **mark_abs_3d**.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**timed_mark_abs**, **mark_abs_3d**, **timed_mark_rel_3d**	

Normal List Command	**timed_mark_rel**
Function	Moves the laser focus for the specified marking duration along a 2D vector from the current position to the specified position (relative coordinate values) within a 2D Image Field.
Call	`timed_mark_rel( dX, dY, T )`
Parameters	`dX` Like **mark_rel**.
	`dY` Like **mark_rel**.
	`T` Duration of the complete mark vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_mark_rel** behaves like **mark_rel**).
Multi-board Com'd Name	**n_timed_mark_rel**
Comments	• The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **timed_mark_rel** is analogous to **timed_mark_abs** (see the comments there).
RTC4→RTC6	Unchanged functionality. In addition: increased value range. See **mark_rel**.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**timed_mark_abs**, **mark_rel**, **timed_mark_rel_3d**

Normal List Command	**timed_mark_rel_3d**
Function	Moves the laser focus for the specified marking duration along a 3D vector from the current position to the specified position (relative coordinate values) within the 3D Image Field.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **timed_mark_rel_3d** has the same effect as **timed_mark_rel**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.
Call	`timed_mark_rel_3d( dX, dY, dZ, T )`
Parameters	`dX` Like **mark_rel_3d**.
	`dY` Like **mark_rel_3d**.
	`dZ` Like **mark_rel_3d**.
	`T` Duration of the complete mark vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_mark_rel_3d** behaves like **mark_rel_3d**.
Multi-board Com'd Name	**n_timed_mark_rel_3d**
Comments	• The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **timed_mark_rel_3d** is identical to **timed_mark_abs_3d** (see the comments there).
RTC4→RTC6	New command.. See **mark_rel_3d**.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**timed_mark_abs_3d**, **timed_mark_abs**, **mark_rel_3d**, **timed_mark_rel**

Normal List Command	timed_para_jump_abs
Function	Moves the output point (of the laser focus) for the specified jump duration along a 2D vector from the current position to the specified position (absolute coordinate values) within a 2D Image Field and, simultaneously as well as linearly, changes the signal parameter selected by **set_vector_control** to the specified value.
Call	`timed_para_jump_abs( X, Y, P, T )`
Parameters	`X` Like **para_jump_abs**.
	`Y` Like **para_jump_abs**.
	`P` Like **para_jump_abs**.
	`T` Duration of the complete jump vector. In μs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_para_jump_abs** behaves like **para_jump_abs**.
Multi-board Com'd Name	**n_timed_para_jump_abs**
Comments	• Unlike **para_jump_abs**, the **timed_para_jump_abs** command does not execute the jump with the specified (by **set_jump_speed** or **set_jump_speed_ctrl**) jump speed. Instead, the speed (that is, the number of Microsteps) is adjusted so that the vector lasts as long as specified (see Chapter 8.9 "Timed Commands", page 283). The total jump time is (for `T` ≥ 5) the sum of the specified (rounded) time and the set delays. • **timed_para_jump_abs** requires two RTC6 List Memory positions. At runtime, the command part on the first RTC6 List Memory position is executed as a Short List Command and afterward the second part is executed as a Normal List Command. Thereby, both command parts are executed within the same 10 μs clock, unless further preceding Short List Commands force a **list_continue** between the two parts. • See also comments on **para_jump_abs**.
RTC4→RTC6	New command. See **para_jump_abs**.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**para_jump_abs**, **timed_jump_abs**, **jump_abs**, **timed_para_jump_rel**, **timed_para_jump_abs_3d**

Multiple List Command	timed_para_jump_abs_3d
Function	Moves the output point (of the laser focus) for the specified jump duration along a 3D vector from the current position to the specified position (absolute coordinate values) within the 3D Image Field and, simultaneously as well as linearly, changes the signal parameter selected by **set_vector_control** to the specified value.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **timed_para_jump_abs_3d** has the same effect as **timed_para_jump_abs**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.
Call	`timed_para_jump_abs_3d( X, Y, Z, P, T )`
Parameters	`X` Like **para_jump_abs_3d**.
	`Y` Like **para_jump_abs_3d**.
	`Z` Like **para_jump_abs_3d**.
	`P` Like **para_jump_abs_3d**.
	`T` Duration of the complete jump vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_para_jump_abs_3d** behaves like **para_jump_abs_3d**.
Multi-board Com'd Name	**n_timed_para_jump_abs_3d**
Comments	• Except for the additional motion in the third dimension, **timed_para_jump_abs_3d** functions similarly to **timed_para_jump_abs** (see the comments there). • **timed_para_jump_abs_3d** occupies 2 RTC6 List Memory positions for `T` ≥ 5. The first part is executed as an Undelayed Short List Command before the principal part (a Normal List Command). • See also comments on **para_jump_abs_3d**.
RTC4→RTC6	New command. See **para_jump_abs_3d**.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**timed_para_jump_abs**, **para_jump_abs_3d**, **jump_abs_3d**, **timed_para_jump_rel_3d**

Normal List Command	timed_para_jump_rel	
Function	Moves the output point (of the laser focus) for the specified jump duration along a 2D vector from the current position to the specified position (relative coordinate values) within a 2D Image Field and, simultaneously as well as linearly, changes the signal parameter selected by **set_vector_control** to the specified value.	
Call	`timed_para_jump_rel( dX, dY, P, T )`	
Parameters	`dX`	Like **para_jump_rel**.
	`dY`	Like **para_jump_rel**.
	`P`	Like **para_jump_rel**.
	`T`	Duration of the complete jump vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_para_jump_rel** behaves like **para_jump_rel**).
Multi-board Com'd Name	**n_timed_para_jump_rel**	
Comments	• The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **timed_para_jump_rel** is analogous to **timed_para_jump_abs** (see the comments there).	
RTC4→RTC6	New command. See **para_jump_rel**.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**timed_para_jump_abs**, **para_jump_rel**, **timed_jump_rel**, **timed_para_jump_rel_3d**	

Multiple List Command	**timed_para_jump_rel_3d**
Function	Moves the output point (of the laser focus) for the specified jump duration along a 3D vector from the current position to the specified position (relative coordinate values) within the 3D Image Field and, simultaneously as well as linearly, changes the signal parameter selected by **set_vector_control** to the specified value.
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **timed_para_jump_rel_3d** has the same effect as **timed_para_jump_rel**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective jump speed in the xy plane.
Call	`timed_para_jump_rel_3d( dX, dY, dZ, P, T )`
Parameters	`dX` Like **para_jump_rel_3d**.
	`dY` Like **para_jump_rel_3d**.
	`dZ` Like **para_jump_rel_3d**.
	`P` Like **para_jump_rel_3d**.
	`T` Duration of the complete jump vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_para_jump_rel_3d** behaves like **para_jump_rel_3d**.
Multi-board Com'd Name	**n_timed_para_jump_rel_3d**
Comments	• The coordinates for the jump vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **timed_para_jump_rel_3d** is analogous to **timed_para_jump_abs_3d** (see comments there). • **timed_para_jump_rel_3d** occupies 2 RTC6 List Memory positions for T ≥ 5. The first part is executed as an Undelayed Short List Command before the principal part (a Normal List Command).
RTC4→RTC6	New command. See **para_jump_rel_3d**.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**timed_para_jump_abs_3d**, **para_jump_rel_3d**, **timed_jump_rel_3d**, **timed_para_jump_rel**

Normal List Command	timed_para_mark_abs
Function	Moves the laser focus for the specified marking duration along a 2D vector from the current position to the specified position (absolute coordinate values) within a 2D Image Field and, simultaneously as well as linearly, changes the signal parameter selected by **set_vector_control** to the specified value.
Call	`timed_para_mark_abs( X, Y, P, T )`
Parameters	`X` Like **para_mark_abs**.
	`Y` Like **para_mark_abs**.
	`P` Like **para_mark_abs**.
	`T` Duration of the complete mark vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_para_mark_abs** behaves like **para_mark_abs**.
Multi-board Com'd Name	**n_timed_para_mark_abs**
Comments	• Unlike **para_mark_abs**, **timed_para_mark_abs** does not execute the marking process with the specified (by **set_mark_speed** or **set_mark_speed_ctrl**) mark speed. Instead, the speed (that is, the number of Microsteps) is adjusted so that the vector lasts as long as specified (see Chapter 8.9 "Timed Commands", page 283). The total marking time is (for `T` ≥ 5) the sum of the specified (rounded) time and the set delays. • **timed_para_mark_abs** occupies 2 RTC6 List Memory positions for `T` ≥ 5. At runtime, the command part on the first RTC6 List Memory position is executed as a Short List Command and afterward the second part is executed as a Normal List Command. Thereby, both command parts are executed within the same 10 µs clock, unless further preceding Short List Commands induce a **list_continue** between the two parts. • See also comments on **para_mark_abs**.
RTC4→RTC6	New command. See **para_mark_abs**.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**para_mark_abs**, **timed_mark_abs**, **mark_abs**, **timed_para_mark_rel**, **timed_para_mark_abs_3d**

Multiple List Command	**timed_para_mark_abs_3d**	
Function	Moves the laser focus for the specified marking duration along a 3D vector from the current position to the specified position (absolute coordinate values) within the 3D Image Field and, simultaneously as well as linearly, changes the signal parameter selected by **set_vector_control** to the specified value.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **timed_para_mark_abs_3d** has the same effect as **timed_para_mark_abs**. However, the split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.	
Call	`timed_para_mark_abs_3d( X, Y, Z, P, T )`	
Parameters	`X`	Like **para_mark_abs_3d**.
	`Y`	Like **para_mark_abs_3d**.
	`Z`	Like **para_mark_abs_3d**.
	`P`	Like **para_mark_abs_3d**.
	`T`	Duration of the complete mark vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_para_mark_abs_3d** behaves like **para_mark_abs_3d**.
Multi-board Com'd Name	**n_timed_para_mark_abs_3d**	
Comments	• Except for the additional motion in the third dimension, **timed_para_mark_abs_3d** functions similarly to **timed_para_mark_abs** (see the comments there). • **timed_para_mark_abs_3d** occupies 2 RTC6 List Memory positions for `T` ≥ 5. The first part is executed as an Undelayed Short List Command before the principal part (a Normal List Command). • See also comments on **para_mark_abs_3d**.	
RTC4→RTC6	New command. See **para_mark_abs_3d**.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**timed_para_mark_abs**, **para_mark_abs_3d**, **mark_abs_3d**, **timed_para_mark_rel_3d**	

Normal List Command	**timed_para_mark_rel**
Function	Moves the laser focus for the specified marking duration along a 2D vector from the current position to the specified position (relative coordinate values) within a 2D Image Field and, simultaneously as well as linearly, changes the signal parameter selected by **set_vector_control** to the specified value.
Call	`timed_para_mark_rel( dX, dY, P, T )`
Parameters	`dX` Like **para_mark_rel**.
	`dY` Like **para_mark_rel**.
	`P` Like **para_mark_rel**.
	`T` Duration of the complete mark vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_para_mark_rel** behaves like **para_mark_rel**.
Multi-board Com'd Name	**n_timed_para_mark_rel**
Comments	• The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **timed_para_mark_rel** is analogous to **timed_para_mark_abs** (see the comments there). • See also comments on **para_mark_rel**.
RTC4→RTC6	New command. See **para_mark_rel**.
RTC5→RTC6	Unchanged functionality. In addition: increased value range.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**timed_para_mark_abs**, **para_mark_rel**, **timed_mark_rel**, **timed_para_mark_rel_3d**

Multiple List Command	**timed_para_mark_rel_3d**	
Function	Moves the laser focus for the specified marking duration along a 3D vector from the current position to the specified position (relative coordinate values) within the 3D Image Field. Simultaneously varies the signal parameter selected by **set_vector_control** to the specified value.	
Restriction	If the Option "3D" is not enabled or no 3D correction table has been assigned (see **select_cor_table**), then **timed_para_mark_rel_3d** has the same effect as **timed_para_mark_rel**. However, split-up into Microsteps is calculated like a 3D command and hence influences the effective mark speed in the xy plane.	
Call	`timed_para_mark_rel_3d( dX, dY, dZ, P, T )`	
Parameters	`dX`	Like **para_mark_rel_3d**.
	`dY`	Like **para_mark_rel_3d**.
	`dZ`	Like **para_mark_rel_3d**.
	`P`	Like **para_mark_rel_3d**.
	`T`	Duration of the complete mark vector. In µs. As a 64-bit IEEE floating point value. Allowed value range: [0…167,772,160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If `T` < 5, then **timed_para_mark_rel_3d** behaves like **para_mark_rel_3d**.
Multi-board Com'd Name	**n_timed_para_mark_rel_3d**	
Comments	• The coordinates for the mark vector end point are to be supplied as relative coordinates with respect to the current position. Otherwise, **timed_para_mark_rel_3d** is analogous to **timed_para_mark_abs_3d** (see the comments there). • **timed_para_mark_rel_3d** occupies 2 RTC6 List Memory positions for `T` ≥ 5. The initial component is executed as an Undelayed Short List Command before the principal part (a Normal List Command). • See also comments on **para_mark_rel_3d**.	
RTC4→RTC6	New command. See **para_mark_rel_3d**.	
RTC5→RTC6	Unchanged functionality. In addition: increased value range.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**timed_para_mark_abs_3d**, **para_mark_rel_3d**, **timed_mark_rel_3d**, **timed_para_mark_rel**	

Ctrl Command	transform	
Function	Performs a backward transformation of individual position values.	
Call	`TransformErrorCode = transform( &Sig1, &Sig2, Ptr, Code )`	
Parameters and Returned Parameter Values	`Sig1`	Parameter: to-be-transformed position value. As a pointer to a signed 32-bit value. Returned parameter value: transformed position value. As a signed 32-bit value. The input values are overwritten.
	`Sig2`	Like `Sig1` (analogously).
Parameters	`Ptr`	Pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 64-bit value) to the area of PC main memory to which the correction and transformation settings for backward transformation were previously transferred by **upload_transform**.
	`Code`	Controls aspects of the backward transformation, particularly which partial transformations to perform: If a partial transformation is *not* to be performed, then its corresponding bit (#2…#5) should be set to 1. As an unsigned 32-bit value. `Code` has the same meaning as **get_transform**/**get_transform_offset** (`Sig1` corresponds to `Ptr1`, `Sig2` corresponds to `Ptr2`). If Bit #0 = 0, then both supplied position values (`Sig1` and `Sig2`) are backward transformed as xy coordinates: Bit #1 = 0: The value specified with `Sig1` is backward transformed as x coordinate. The value specified with `Sig2` is backward transformed as y coordinate. = 1: The value specified with `Sig1` is backward transformed as y coordinate. The value specified with `Sig2` is backward transformed as x coordinate. Bit #2 = 0: Gain/offset correction of automatic self-calibration is backward transformed. Bit #3 = 0: Image Field correction is backward transformed. Bit #4 = 0: Offset of the defined coordinate transformation is backward transformed. Bit #5 = 0: Total matrix of the defined coordinate transformation is backward transformed. Bit #6 Reserved. … … Bit #31 Reserved.

Ctrl Command	transform
Parameters (cont'd)	`Code` (cont'd) If Bit #0 = 1, then only one of the two supplied position values is backward transformed as the z coordinate: Bit #1 = 0: The value supplied by `Sig1` is backward transformed as the z coordinate. `Sig2` remains unchanged. = 1: The value supplied by `Sig2` is backward transformed as the z coordinate. `Sig1` remains unchanged. Bit #2 = 0: Offset to the focal length defined by **set_defocus** or **set_defocus_list** is backward transformed. Bit #3 = 0: The ABC correction is backward transformed. Bit #4 = 0: The offset to the z coordinate defined by **set_offset_xyz** or **set_offset_xyz_list** is backward transformed. Bit #5 Reserved. … … Bit #31 Reserved.
Result	Error code. As an unsigned 32-bit value. 0 Success. 1 `Ptr` = NULL (no memory area specified). 2 No valid data at `Ptr` (**upload_transform** did not execute). 3 Erroneous data at `Ptr` (a corresponding error indication has been stored by **upload_transform**). 4 z axis inversion not possible.
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• For backward transformation of position values see Chapter 8.1.3 "Monitoring the Positioning", page 223. • The execution of **transform** must be preceded by a call to **upload_transform**. Additionally, position values should have been requested by **get_values**. • If execution of **transform** results in an error (returned error code > 0), then no transformation occurs (`Sig1` and `Sig2` then remain unchanged). Errors also include `Ptr` = NULL (error code = 1) or errors resulting from prior, erroneous execution of **upload_transform** (error code = 3). • If backward transformation of z values is requested (`Code` Bit #0 = 1), but only a 2D correction table has been assigned at the timepoint of the prior successful call to **upload_transform**, then the offsets to the focal length and z coordinates are initialized with 0 and the values A, B and C are initialized with 0, 1, 0 (1-to-1 backward transformation).

Ctrl Command	transform
Comments (cont'd)	• For backward transformation of xy position values (`Code` Bit #0 = 0), only the z = 0 plane is transformed. xy stretching and Z defocus resulting from z deviations (particularly with non-F-Theta systems) are not taken into account. • Because **transform** does not access any RTC6 board, calling it does not require explicit access rights to a specific board. If both the **upload_transform** data and the queried data recorded by **get_values** or **get_waveform** have been binarily stored on the PC, then offline operation of **transform** is also possible (then **transform** does not require the presence of an RTC6 board on the PCIe bus). • The board-specific error variables `LastError` and `AccError` (see Chapter 6.8 "Error Handling", page 133) are neither generated nor altered by **transform**.
RTC4→RTC6	New command. In the RTC4 Compatibility Mode, all backward transformed values (including z values) are in the RTC6 20-bit range.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**upload_transform**, **get_transform**, **get_transform_offset**, **get_values**

Ctrl Command	uart_config
Function	Configures the internal UART interface for the specified baud rate.
Call	`RealBaudRate = uart_config( BaudRate )`
Parameters	`BaudRate` Baud rate. As an unsigned 32-bit value. Allowed value range: [160 Bd…12.8 MBd].
Result	`RealBaudRate`
Multi-board Com'd Name	**n_uart_config**
Comments	• **uart_config** extends **rs232_config** with a higher value range for the baud rate. • **uart_config** is synonymous with **rs232_config**, but returns the nearest possible actually used baud rate. • The default value is 9,600 baud. • The other RS-232 interface parameters cannot be altered (data bits: 8, start bits: 1, stop bits: 1, parity: none). • See also Chapter 4.6.5 "RS232 Socket Connector", page 86. • **uart_config** is allowed in Boot Phase 1, see also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 611, OUT 611, RBF 616.
References	**rs232_config**, **rs232_write_data**, **rs232_read_data**

Ctrl Command	upload_transform		
Function	Transfers from the RTC6 board to the PC all correction and transformation settings currently assigned to the scan system.		
Call	`UploadErrorCode = upload_transform( HeadNo, Ptr )`		
Parameters	`HeadNo`	Number of the scan head connector whose settings should be queried. As an unsigned 32-bit value. Allowed values: = 1: Connector for First Scan Head. = 2: Connector for Second Scan Head.	
	`Ptr`	Pointer (in C and C++ data type ULONG_PTR, an unsigned 32-bit value or unsigned 64-bit value) to the PC's area of memory that should receive the queried settings.	
Result	Error code. As an unsigned 32-bit value.		
	Bit #0	=1:	X gain = 0 (gain of automatic self-calibration for Galvanometer scanner 2).
	Bit #1	=1:	Y gain = 0 (gain of automatic self-calibration for Galvanometer scanner 1).
	Bit #2	=1:	The total matrix of the defined coordinate transformation is noninvertable.
	Bit #3	=1:	No correction table assigned.
	Bit #4	=1:	The ABC values (z axis) are noninvertable.
	Bit #5	=1:	Error querying correction table.
	Bit #6	=1:	Parameter error: invalid `HeadNo` or `Ptr` = 0.
	Bit #7	=1:	Busy error, board has BUSY list execution status or INTERNAL-BUSY list execution status (**get_last_error** return code `RTC6_BUSY`).
	Bit #8	Reserved.	
	…	…	
	Bit #31	Reserved.	
Multi-board Com'd Name	**n_upload_transform**		

Ctrl Command	upload_transform
Comments	• The queried and transferred data can be used for backward transforming actual position values, see also Chapter 8.1.3 "Monitoring the Positioning", page 223 by: – **transform** – **get_transform**/**get_transform_offset** • To save a read data set, the user program must provide a memory area of size 528,524 bytes on the PC in each case (at an address specified by `Ptr`). • In case of error (except for Bit #6 = 1), an error indication is stored at `Ptr` to indicate that the data are erroneous. **transform** and **get_transform**/**get_transform_offset** recognize this error information and ensure that the backward transformation is not executed (**transform** then generates a corresponding error code, **get_transform**/**get_transform_offset** generates a **get_last_error** return code `RTC6_PARAM_ERROR`). • **upload_transform** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status of the addressed board is set – the INTERNAL-BUSY list execution status is set • **upload_transform** is even executed, if: – a list has been paused by **set_wait** (PAUSED list execution status set) • During the runtime of **upload_transform**, External Starts are suppressed.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**transform**, **get_transform**, **get_transform_offset**

Ctrl Command	**verify_checksum**
Function	Checks or creates the checksum of a correction file.
Call	`verify_checksum( Name )`
Parameters	`Name` Name of the correction file. As a pointer to a \0-terminated ANSI string.
Result	Result of the test. As an unsigned 32-bit value. = 0: No error (the tested checksum is OK.). = 1: A checksum has been newly created (no info about file integrity). = 2: The tested checksum is incorrect. = 3: A checksum could not be determined (file error, etc.).
Multi-board Com'd Name	Not available as a multi-board command.
Comments	• Verification of correction file downloads only works for files that contain a checksum (see Loading of correction files, page 135 and **set_verify**). • The **verify_checksum** command is available even without explicit access rights to a particular RTC6 board. • The programs `CorrectionFileConverter.exe` (version 1.04) and `correXion5.exe` (version 1.01) together with `RTC5Base.dll` (version 1.0.0.4) already automatically create check sums for the output files. • The board-specific error variables `LastError` and `AccError` (see Chapter 6.8 "Error Handling", page 133) are neither generated nor altered by **verify_checksum**.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**set_verify**

Normal List Command	**wait_for_1_axis**
Function	"Fly Extension" Command: Waits until the value for the specified Mode has been exceeded or underrun.
Call	`wait_for_1_axis( Value, Mode, WaitMode, LaserMode )`
Parameters	`Value` Value to be waited for. As a signed 32-bit value. `Mode` Mode from Table 4, page 270. As an unsigned 32-bit value. `WaitMode` < 0: Undercutting. = 0: Equal. > 0: Overrun. The galvanometer scanner follow the object motion. `WaitMode`+16: The galvanometer scanner stop. As a signed 32-bit value. `LaserMode` = 0: The laser remains unchanged. > 0: The laser is switched off after a LaserOff Delay. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_wait_for_1_axis**
Comments	• Being an "Fly Extension" Command, **wait_for_1_axis** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • Depending on the set Mode, it can be waited for an encoder value or (any) McBSP value. In case of McBSP, a corresponding Processing-on-the-fly correction should be enabled. Make sure that the data is transferred in the correct format. • With `wait_for_1_axis` and `wait_for_2_axes`, Mode 1…4 must not be specified. Instead, Mode 17…20 is to be used. Mode 17…18 is to be used with a SCANahead System, if waiting is yet to occur within `PreviewTime`. Outside of this, Mode 19…20 can be used. Mode 17…18 and Mode 19…20 are identical with intelliSCAN systems. • `LaserMode` = 0: like before. • `LaserMode` > 0: Laser is switched off after a LaserOff Delay. • `WaitMode` is like `Mode` of **wait_for_encoder_mode**. • `WaitMode` and `WaitMode`+16 differentiate the galvanometer scanner motion, not **set_fly_2d** and **set_fly_x**/**set_fly_y**. • **wait_for_encoder** with automatic positions-dependent selection of direction is not supported. • With an unallowed parameter value, **wait_for_1_axis** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`).

Normal List Command	wait_for_1_axis
Comments (cont'd)	• The following command calls are executed in the same way: – `wait_for_1_axis( Value, EncoderNo + 19, Mode, 0 )` = `wait_for_encoder_mode( Value, EncoderNo, Mode )` and **set_fly_2d** session – `wait_for_1_axis( Value, EncoderNo + 19, Mode+16, 0 )` = `wait_for_encoder_mode( Value, EncoderNo, Mode )` and **set_fly_x**/**set_fly_y** session – `wait_for_1_axis( Value, 6, Mode, 0 )` = `wait_for_mcbsp( Axis, Value, Mode )` and **set_fly_x_pos** session – `wait_for_1_axis( Value, Axis×4+10, Mode+16, 0 )` = `wait_for_mcbsp( Axis, Value, Mode )` and **set_fly_x_pos**/**set_fly_y_pos** session
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**wait_for_2_axes**

Multiple List Command	**wait_for_2_axes**
Function	"Fly Extension" Command: Waits until the values for both modes (see Mode) are within or outside the specified range.
Call	`wait_for_2_axes( ModeX, MinValueX, MaxValueX, ModeY, MinValueY, MaxValueY, WaitMode, LaserMode )`
Parameters	`ModeX` Mode from Table 4, page 270. As an unsigned 32-bit value. `MinValueX` Lower limit for the x axis Mode value to be waited for. As a signed 32-bit value. `MaxValueX` Upper limit for the x axis Mode value to be waited for. As a signed 32-bit value. `ModeY` Mode from Table 4, page 270. As an unsigned 32-bit value. `MinValueY` Lower limit for the y axis Mode value to be waited for. As a signed 32-bit value. `MaxValueY` Upper limit for the y axis Mode value to be waited for. As a signed 32-bit value. `WaitMode` $\geq$ 0: Within limit values. < 0: Outside limit values. The galvanometer scanner follow the object motion. `WaitMode`+16: The galvanometer scanners stop. As a signed 32-bit value. `LaserMode` = 0: The laser remains unchanged. > 0: The laser is switched off after a LaserOff Delay. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_wait_for_2_axes**
Comments	• Being an "Fly Extension" Command, **wait_for_2_axes** must not be used mixed with "Classic" Processing-on-the-fly commands (see Footnote, page 251). • See Chapter 8.6 "Processing-on-the-fly", page 251 and Section ""Fly Extension" Commands", page 268. • **wait_for_2_axes** occupies 2 RTC6 List Memory positions. The first part is executed as an Undelayed Short List Command prior to the second part, which executes as a Normal List Command. Any pending Delayed Short List Commands are executed beforehand. • Depending on the set Mode, it can be waited for an encoder value or (any) McBSP value to be inside or outside the limits. In case of McBSP, a corresponding Processing-on-the-fly correction should be enabled. Make sure that the data is transferred in the correct format. • With `wait_for_1_axis` and `wait_for_2_axes`, Mode 1…4 must not be specified. Instead, Mode 17…20 is to be used. Mode 17…18 is to be used with a SCANahead System, if waiting is yet to occur within `PreviewTime`. Outside of this, Mode 19…20 can be used. Mode 17…18 and Mode 19…20 are identical with intelliSCAN systems.

Multiple List Command	wait_for_2_axes
Comments (cont'd)	• `LaserMode` = 0: like before. • `LaserMode` > 0: The laser is switched off after a LaserOff Delay. • `WaitMode` is like `Mode` of **wait_for_encoder_mode**. • `WaitMode` and `WaitMode`+16 differentiate the galvanometer scanner motion not **set_fly_2d** and **set_fly_x**/**set_fly_y**. • With an unallowed parameter value, **wait_for_2_axes** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • The following command calls are executed in the same way: – `wait_for_2_axes( 19, EncXmin, EncXmax, 20, EncYmin, EncYmax, 0, 0 ) = wait_for_encoder_in_range_mode(EncXmin, EncXmax, EncYmin, EncYmax, 0 )` and **set_fly_2d** session, intelliSCAN – `wait_for_2_axes( 19, EncXmin, EncXmax, 20, EncYmin, EncYmax, 16, 0 ) = wait_for_encoder_in_range_mode(EncXmin, EncXmax, EncYmin, EncYmax, 0 )` and **set_fly_x**/**set_fly_y** session, intelliSCAN – `wait_for_2_axes( 17, EncXmin, EncXmax, 18, EncYmin, EncYmax, 0, 0 ) = wait_for_encoder_in_range_mode(EncXmin, EncXmax, EncYmin, EncYmax, 2 )` and **set_fly_2d** session, SCANahead System – `wait_for_2_axes( 17, EncXmin, EncXmax, 18, EncYmin, EncYmax, 16, 0 ) = wait_for_encoder_in_range_mode(EncXmin, EncXmax, EncYmin, EncYmax, 2 )` and **set_fly_x**/**set_fly_y** session, SCANahead System
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**wait_for_1_axis**

Normal List Command	**wait_for_encoder**
Function	Waits until the selected encoder counter has overstepped or understepped the specified count for the first time.
Call	`wait_for_encoder( Value, EncoderNo )`
Parameters	`Value` Count. As a signed 32-bit value. Allowed value range: $[-2^{31}...+(2^{31}-1)]$.
	`EncoderNo` Number of the to-be-used encoder counter. As an unsigned 32-bit value. Allowed values: = 0: Encoder counter "Encoder0". = 1: Encoder counter "Encoder1".
Multi-board Com'd Name	**n_wait_for_encoder**
Comments	• **wait_for_encoder** is synonymous with **wait_for_encoder_mode** with parameter `Mode` = 0 (see comments there).
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**wait_for_encoder_mode**

Multiple List Command	**wait_for_encoder_in_range**	
Function	Waits until both encoder counters simultaneously lie within the specified range (including limits).	
Call	`wait_for_encoder_in_range( EncXmin, EncXmax, EncYmin, EncYmax )`	
Parameters	`EncXmin`	Limit value. As a signed 32-bit value. Allowed value range: $[-2^{31}\ldots+(2^{31}-1)]$.
	`EncXmax`	Like `EncXmin` (analogously).
	`EncYmin`	Like `EncXmin` (analogously).
	`EncYmax`	Like `EncXmin` (analogously).
Multi-board Com'd Name	**n_wait_for_encoder_in_range**	
Comments	• For usage of **wait_for_encoder_in_range**, see Chapter 9.3.3 "Synchronization by Encoder Signals", page 320 and Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 261. • **wait_for_encoder_in_range** occupies 2 RTC6 List Memory positions. The first part is executed as an Undelayed Short List Command prior to the second part, which executes as a Normal List Command. Any pending Delayed Short List Commands are executed beforehand. • If `EncXmin` > `EncXmax`, then both values are interchanged. • If `EncYmin` > `EncYmax`, then both values are interchanged. • If no encoder-based Processing-on-the-fly correction is active, then **wait_for_encoder_in_range** merely creates a waiting period (without galvanometer scanner motion). • If `EncXmin` = `EncXmax` (or `EncYmin` = `EncYmax`), then waiting until a specific encoder value is possible. • **wait_for_encoder_in_range** is available even if the Option Processing-on-the-fly is not enabled. • **wait_for_encoder_in_range** does not alter the Laser Control Signals. If you want the laser off during the wait, then this command must be preceded by some other command that switches off the Signals for "Laser Active" Operation (for example, a **list_nop**). • The active Processing-on-the-fly mode determines whether the galvanometer scanners remain stationary during the wait or move in accordance with encoder changes, see Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 261: They move with **set_fly_2d**, but otherwise remain stationary.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**wait_for_encoder_mode**, **park_position**, **park_return**	

Multiple List Command	**wait_for_encoder_in_range_mode**
Function	Waits until both encoder counters simultaneously lie within the specified range (including limits).
Call	`wait_for_encoder_in_range_mode( EncXmin, EncXmax, EncYmin, EncYmax, Mode )`
Parameters	`EncXmin` Limit value. As a signed 32-bit value. Allowed value range: $[-2^{31}...+(2^{31}-1)]$.
	`EncXmax` Like `EncXmin` (analogously).
	`EncYmin` Like `EncXmin` (analogously).
	`EncYmax` Like `EncXmin` (analogously).
	`Mode` Mode. As a signed 32-bit value. = 0 or 1: Waits for direct encoder values ("classical behavior"). = 2: Waits for encoder values that are expected to be present in (**set_scanahead_params** parameter) `PreviewTime` (SCANahead System behavior).
Multi-board Com'd Name	**n_wait_for_encoder_in_range_mode**
Comments	• For usage, see Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 261 and Chapter 9.3.3 "Synchronization by Encoder Signals", page 320. • See also comments on **wait_for_encoder_in_range**. • Use `Mode` like in **wait_for_encoder_mode**: – 0, 1 for intelliSCAN or SCANahead Systems outside an marking process – 2 for SCANahead Systems during an marking process • **wait_for_encoder_in_range** is synonymous with `wait_for_encoder_in_range_mode`(`Mode` = ,0).
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 612, OUT 612, RBF 617.
References	**wait_for_encoder_mode**

Normal List Command	**wait_for_encoder_mode**
Function	Waits until the selected encoder counter has overstepped or understepped the specified count for the first time.
Call	`wait_for_encoder_mode( Value, EncoderNo, Mode )`
Parameters	`Value` Count. As a signed 32-bit value. Allowed value range: [-2^{31}…$+(2^{31}-1)$].
	`EncoderNo` Number of the to-be-used encoder counter. As an unsigned 32-bit value. Allowed values: = 0: Encoder counter "Encoder0". = 1: Encoder counter "Encoder1".
	`Mode` Mode. As a signed 32-bit value. = 0: Waits for understepping/overstepping (position dependent). > 0: Waits for overstepping (position independent). < 0: Waits for understepping (position independent).
Multi-board Com'd Name	**n_wait_for_encoder_mode**
Comments	• For usage of **wait_for_encoder_mode**, see Chapter 9.3.3 "Synchronization by Encoder Signals", page 320. • If `Mode` = 0, ensure that the size and sign of the parameter `Value` is appropriate for the counting direction of the selected encoder (for external triggering, this corresponds to the workpiece's direction of motion). If `Value` > 0, **wait_for_encoder_mode** waits for overstepping, otherwise for understepping. If `Value` is positive and already less than the current encoder count, then **wait_for_encoder_mode** waits for a complete traversal of the counter (likewise if `Value` is negative and larger than the current encoder count). At a 1 MHz counter rate, this can take up to approx. 36 minutes! • If `Mode` ≠ 0, then **wait_for_encoder_mode** waits for overstepping/understepping of the `Value` parameter independently of the current position and direction of motion. • If `EncoderNo` > 1, then **wait_for_encoder_mode** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • If no encoder-based Processing-on-the-fly correction is active, then **wait_for_encoder_mode** merely creates a waiting period (without galvanometer scanner motion) that allows implementation of an externally triggered synchronization (as an alternative to External Starts, **set_wait** or **if_cond**, etc.). • **wait_for_encoder_mode** is available even if the Option Processing-on-the-fly is not enabled. • For `Mode` = 0, **wait_for_encoder_mode** is synonymous with **wait_for_encoder**. • **wait_for_encoder_mode** does not alter the Laser Control Signals. If you want the laser off during the wait, then **wait_for_encoder_mode** must be preceded by some other command that switches off the Signals for "Laser Active" Operation (for example, a **list_nop**).

Normal List Command	wait_for_encoder_mode
Comments (cont'd)	• The active Processing-on-the-fly mode determines whether the galvanometer scanners remain stationary during the wait or move in accordance with encoder changes, see Chapter 8.6.7 "Synchronizing Processing-on-the-fly Applications", page 261: They move with **set_fly_2d**, but otherwise remain stationary. • Mode = 0 or ±1 waits for direct encoder values: "classic" behavior. • Mode = ±2 waits for encoder values that are expected to be present in (**set_scanahead_params** parameter) `PreviewTime`: SCANahead System behavior.
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_encoder**, **store_encoder**, **read_encoder**, **simulate_encoder**, **wait_for_encoder**, **wait_for_encoder_in_range**, **park_position**, **park_return**

Normal List Command	wait_for_mcbsp	
Function	Waits until the input value at the McBSP interface has reached, overstepped or understepped the specified value for the first time.	
Call	`wait_for_mcbsp( Axis, Value, Mode )`	
Parameters	`Axis`	Selects which half-word of the input value is used for evaluation (see below). As an unsigned 32-bit value. Allowed values: = 0: lower half-word (x axis, galvanometer scanner 2) = 1: upper half-word (y axis, galvanometer scanner 1)
	`Value`	Limit value. As a signed 32-bit value.
	`Mode`	Mode. As a signed 32-bit value. = 0: Waits for equality. > 0: Waits for overstepping. < 0: Waits for understepping.
Multi-board Com'd Name	**n_wait_for_mcbsp**	
Comments	• For usage of **wait_for_mcbsp**, see Chapter 9.3.4 "Synchronization and Online Positioning by McBSP Signals", page 322. • **wait_for_mcbsp** is comparable to **wait_for_encoder_mode**, but the McBSP interface is queried. • If **set_fly_x_pos** and **set_fly_y_pos** are *simultaneously* activated, then `Value` consists of two 16-bit half-words, see Section "Correction via McBSP Interface", page 254. Only in this case does `Axis` control which half-word is used for evaluation. In all other cases, `Axis` is irrelevant and `Value` is interpreted as a signed 32-bit value. • `Axis` must be either 0 or 1 (even if it is irrelevant). Otherwise, **wait_for_mcbsp** is replaced by **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • If neither **set_fly_x_pos**, **set_fly_y_pos** nor **set_fly_rot_pos** are activated, then **wait_for_mcbsp** merely creates a waiting period (without galvanometer scanner motion) that allows implementation of an externally triggered synchronization (as an alternative to External Starts, **set_wait** or **if_cond**, etc.). • **wait_for_mcbsp** is available even if the Option Processing-on-the-fly is not enabled. • **wait_for_mcbsp** does not alter the Laser Control Signals. If you want the laser off during the wait, then **wait_for_mcbsp** must be preceded by some other command that switches off the Signals for "Laser Active" Operation (for example, a **list_nop**).	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**wait_for_encoder_mode**	

Normal List Command	**wait_for_timestamp_counter**
Function	Waits for a 32-bit "Timestamp Counter" value.
Call	`wait_for_timestamp_counter( TimeStampCounter )`
Parameters	`TimeStampCounter` 32-bit offset to wait for. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_wait_for_timestamp_counter**
Comments	• **wait_for_timestamp_counter** waits until the current 32-bit "Timestamp Counter" has reached the value `TimeStampStorage` (from **store_timestamp_counter** / **store_timestamp_counter_list**) + `TimeStampCounter`. • Delayed Short List Commands are executed beforehand. • This allows absolute time references from the point in time `TimeStampStorage` to be established when loading a list. Accuracy: 10 µs. • If the specified time has already passed when **wait_for_timestamp_counter** is reached, a full 32-bit "Timestamp Counter" cycle is waited for (duration: approx. 12 hours). This can be canceled by **stop_execution** or /STOP. • **wait_for_timestamp_counter** is synonymous with **wait_for_timestamp_counter_mode**(`Mode` > 2). • Synchronization with other RTC6 commands, such as **wait_for_encoder** or conditional commands, depends on external hardware. • See Chapter 8.12 "Time Measurements", page 290.
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 617, OUT 617, RBF 623.
References	**store_timestamp_counter**, **store_timestamp_counter_list**, **wait_for_timestamp_counter_mode**

Normal List Command	**wait_for_timestamp_counter_long**	
Function	Waits for an absolute 64-bit “Timestamp Counter” value.	
Call	`wait_for_timestamp_counter_long( WaitCounterL, WaitCounterH, MaxWaitTime, Mode )`	
Parameters	`WaitCounterL`	Lower part of the 64-bit “Timestamp Counter” value to wait for. As an unsigned 32-bit value.
	`WaitCounterH`	Upper part of the 64-bit “Timestamp Counter” value to wait for. As an unsigned 32-bit value.
	`MaxWaitTime`	Longest waiting time. As an unsigned 32-bit value.
	`Mode`	Routines after a timeout. As an unsigned 32-bit value. = 0: Ignore the RTC6 command. Continue with next list command. = 1: Abort list by **stop_execution**. = 2: Abort list by **simulate_ext_stop** including forward to all slaves, see Chapter 6.6.3 ”Master/Slave Operation”, page 127 and Chapter 9.3.1 ”Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization”, page 311. > 2: Wait endlessly until an External Stop occurs.
Multi-board Com’d Name	**n_wait_for_timestamp_counter_long**	
Comments	• **wait_for_timestamp_counter_long** waits until the current 64-bit “Timestamp Counter” has reached the following value: `WaitCounterL` + (`WaitCounterH` << 32) • This allows absolute time references to a synchronization point in time defined in the user program to be established when loading a list. Accuracy: 10 µs. • An error flag is set, if: – The specified point in time has already passed when **wait_for_timestamp_counter_long** is reached – The requested waiting time is beyond the running 64-bit “Timestamp Counter” by more than `MaxWaitTime`. The error flag can be read out by **get_startstop_info**(Bit #5). The `Mode` parameter determines how to proceed in case of a timeout. • Synchronization with other RTC6 commands, such as **wait_for_encoder** or conditional commands, depends on external hardware. • See Chapter 8.12 “Time Measurements”, page 290.	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 624, OUT 624.	
References	**get_timestamp_long**	

Normal List Command	**wait_for_timestamp_counter_mode**
Function	Like **wait_for_timestamp_counter**. In addition, it can be defined what should happen after a timeout.
Call	`wait_for_timestamp_counter_mode( TimeStampCounter, Mode )`
Parameters	`TimeStampCounter` 32-Bit offset to wait for. As an unsigned 32-bit value.
	`Mode` Routines after a timeout. As an unsigned 32-bit value. = 0: Ignore the RTC6 command. Continue with next list command. = 1: Abort list by **stop_execution**. = 2: Abort list by **simulate_ext_stop** including forward to all slaves, see Chapter 6.6.3 "Master/Slave Operation", page 127 and Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 311. > 2: Wait for a full 32-bit "Timestamp Counter" cycle (as with **wait_for_timestamp_counter**).
Multi-board Com'd Name	**n_wait_for_timestamp_counter_mode**
Comments	• **wait_for_timestamp_counter_mode** waits until the current 32-bit "Timestamp Counter" has reached the value `TimeStampStorage` (from **store_timestamp_counter** / **store_timestamp_counter_list**) + `TimeStampCounter`. • Delayed Short List Commands are executed beforehand. • This allows absolute time references with an accuracy of 10 µs from the point in time `TimeStampStorage` to be established when loading a list. Accuracy: 10 µs. • An error flag is set, if: – The specified point in time has already passed when **wait_for_timestamp_counter_mode** is reached The error flag can be read out by **get_startstop_info**(Bit #5). The `Mode` parameter determines how to proceed in case of a timeout. • **wait_for_timestamp_counter** is synonymous with **wait_for_timestamp_counter_mode**(`Mode` > 2). • Synchronization with other RTC6 commands, such as **wait_for_encoder** or conditional commands, depends on external hardware. • See Chapter 8.12 "Time Measurements", page 290.

Normal List Command	wait_for_timestamp_counter_mode
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 619, OUT 619, RBF 624.
References	**store_timestamp_counter**, **store_timestamp_counter_list**, **wait_for_timestamp_counter**

Ctrl Command	write_8bit_port
Function	Writes a value to the 8-Bit Digital Output Port on the EXTENSION 2 Socket Connector.
Call	`write_8bit_port( Value )`
Parameters	`Value` 8-bit output value (DATA0…DATA7). As an unsigned 32-bit value. Only the lower 8 bits are evaluated.
Multi-board Com'd Name	**n_write_8bit_port**
Comments	• See also Chapter 9.1.2 "8-Bit Digital Output Port", page 304.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_8bit_port_list**

Delayed Short List Command	write_8bit_port_list
Function	Like **write_8bit_port**, but a list command.
Call	`write_8bit_port_list( Value )`
Parameters	`Value` Like **write_8bit_port**.
Multi-board Com'd Name	**n_write_8bit_port_list**
Comments	• See **write_8bit_port**. • As of version DLL 602, OUT 602: When the 8-Bit Digital Output Port (for example, for laser power) is to be executed synchronous to the laser switch times, use `set_laser_power( 2, Value )`.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_8bit_port**, **set_laser_power**, **write_port_list**

Ctrl Command	write_abc_to_file	
Function	Like **write_abc_to_file_20b**. However, for a focus length value *l* in RTC4 compatibility range [–32,768…+32,767].	
Call	`ErrorNo = write_abc_to_file( Name, A, B, C )`	
Parameters	`Name`	Like **write_abc_to_file_20b**.
	`A`	Coefficient A of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values (focus length value *l* in the RTC4 compatibility range [–32,768…+32,767]). As a 64-bit IEEE floating point value. Allowed value range:: see **load_z_table**.
	`B`	Like `A` (analogously).
	`C`	Like `A` (analogously).
Result	Like **write_abc_to_file_20b**.	
Multi-board Com'd Name	Not available as a multi-board command.	
Comments	• Like **write_abc_to_file_20b**. • `write_abc_to_file_20b( A × 16, B, C × 1/16 )` is synonymous with `write_abc_to_file( A, B, C )`.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**read_abc_from_file**	

<table>
<tr><th>Ctrl Command</th><th colspan="2">write_abc_to_file_20b</th></tr>
<tr><td>Function</td><td colspan="2">Writes the ABC values directly into a specified correction file on the PC.
For a focus length value l in the RTC6 20-bit range [–524,288…+524,287].</td></tr>
<tr><td>Call</td><td colspan="2"><code>ErrorNo = write_abc_to_file_20b(Name, A, B, C)</code></td></tr>
<tr><td>Parameters</td><td><code>Name</code></td><td>Name of the correction file.
As a pointer to a \0-terminated ANSI string.</td></tr>
<tr><td></td><td><code>A</code></td><td>Coefficient A of the parabolic function $z_{out} = A + Bl + Cl^2$ which is used for calculating the Z output values
(focus length value l in the RTC6 20-bit range [–524,288…+524,287]).
As a 64-bit IEEE floating point value.
Allowed value range: see load_z_table_20b.</td></tr>
<tr><td></td><td><code>B</code></td><td>Like <code>A</code> (analogously).</td></tr>
<tr><td></td><td><code>C</code></td><td>Like <code>A</code> (analogously).</td></tr>
<tr><td>Result</td><td><code>ErrorNo</code></td><td>Error code.
As an unsigned 32-bit value.</td></tr>
<tr><td></td><td>0</td><td>No error.</td></tr>
<tr><td></td><td>1</td><td><code>A</code> exceeded the maximum allowed value.</td></tr>
<tr><td></td><td>2</td><td><code>A</code> undercut the minimum allowed value.</td></tr>
<tr><td></td><td>4</td><td><code>B</code> exceeded the maximum allowed value.</td></tr>
<tr><td></td><td>8</td><td><code>B</code> undercut the minimum allowed value.</td></tr>
<tr><td></td><td>16</td><td><code>C</code> exceeded the maximum allowed value.</td></tr>
<tr><td></td><td>32</td><td><code>C</code> undercut the minimum allowed value.</td></tr>
<tr><td></td><td>3</td><td>File-open error (empty string, file not found etc.).</td></tr>
<tr><td></td><td>12</td><td>File error (checksum could not be determined, file corrupt).</td></tr>
<tr><td>Multi-board Com’d Name</td><td colspan="2">Not available as a multi-board command.</td></tr>
<tr><td>Comments</td><td colspan="2">write_abc_to_file_20b is available even without explicit access rights to a specific RTC6 board.
The board-specific error variables <code>LastError</code> and <code>AccError</code> (see Chapter 6.8 “Error Handling”, page 133) are neither generated nor altered by write_abc_to_file_20b.
ABC values are not outputted with error code ≠ 0 or ≠ 12.
<code>write_abc_to_file_20b(A, B, C)</code> is synonymous with <code>write_abc_to_file(A × 1/16, B, C × 16)</code>.</td></tr>
<tr><td>RTC4→RTC6</td><td colspan="2">New command.</td></tr>
<tr><td>RTC5→RTC6</td><td colspan="2">New command.</td></tr>
<tr><td>Version Info</td><td colspan="2">Available as of DLL 631, OUT 632.</td></tr>
<tr><td>References</td><td colspan="2">read_abc_from_file_20b</td></tr>
</table>

Ctrl Command	write_da_1
Function	See **write_da_x**.
Call	`write_da_1( Value )`
Parameters	`Value` 12-bit output value for the ANALOG OUT1 analog output port. As an unsigned 32-bit value. Bits with higher significance are ignored. `Value` = 0 corresponds to an output value of 0 V. `Value` = 2^{12}–1 corresponds to an output value of 10 V.
Multi-board Com'd Name	**n_write_da_1**
Comments	• –
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode: like **write_da_x**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_da_x**

Delayed Short List Command	write_da_1_list
Function	See **write_da_x_list**.
Call	`write_da_1_list( Value )`
Parameters	`Value` Like **write_da_1**.
Multi-board Com'd Name	**n_write_da_1_list**
Comments	• –
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode: like **write_da_x**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_da_x_list**

Ctrl Command	write_da_2
Function	See **write_da_x**.
Call	`write_da_2( Value )`
Parameters	`Value` 12-bit output value for the ANALOG OUT2 analog output port. As an unsigned 32-bit value. Bits with higher significance are ignored. `Value` = 0 corresponds to an output value of 0 V. `Value` = 2^{12}–1 corresponds to an output value of 10 V.
Multi-board Com'd Name	**n_write_da_2**
Comments	• –
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode: like **write_da_x**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_da_x**

Delayed Short List Command	write_da_2_list
Function	See **write_da_x_list**.
Call	`write_da_2_list( Value )`
Parameters	`Value` Like **write_da_2**.
Multi-board Com'd Name	**n_write_da_2_list**
Comments	• –
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode: like **write_da_x**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_da_x_list**

Ctrl Command	write_da_x
Function	Writes an output value to one of the analog output ports of the RTC6.
Call	`write_da_x( x, Value )`
Parameters	`x` Number of the analog output port. As an unsigned 32-bit value. Allowed values: [1, 2] (1: ANALOG OUT1, 2: ANALOG OUT2).
	`Value` 12-bit output value for the selected analog output port. As an unsigned 32-bit value. Bits with higher significance are ignored. `Value` = 0 corresponds to an output value of 0 V. `Value` = 2^{12}–1 corresponds to an output value of 10 V.
Multi-board Com'd Name	**n_write_da_x**
Comments	• See also Chapter 9.1.4 "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 305. • The output range of the analog output ports is 0 V…10 V. • For `x` < 1 or `x` > 2, **write_da_x** is ignored (**get_last_error** return code `RTC6_PARAM_ERROR`). • **write_da_1** / **write_da_2** can be used alternatively to **write_da_x** (without parameter `x`).
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode, the RTC6 multiplies the specified `Value` by 4.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_da_x_list**

Delayed Short List Command	**write_da_x_list**
Function	Like **write_da_x**, but a list command.
Call	`write_da_x_list( x, Value )`
Parameters	`x` Like **write_da_x**.
	`Value` Like **write_da_x**.
Multi-board Com'd Name	**n_write_da_x_list**
Comments	• For `x` < 1 or `x` > 2, **write_da_x_list** is replaced by a **list_nop** (**get_last_error** return code `RTC6_PARAM_ERROR`). • As of version DLL 602, OUT 602: When the ANALOG OUT output port (for example, for the laser power) is to be executed synchronous to the laser switch times, use `set_laser_power( x-1, Value )`.
RTC4→RTC6	Unchanged functionality. In RTC4 Compatibility Mode: like **write_da_x**.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_da_x**, **set_laser_power**, **write_port_list**

Ctrl Command	write_hi_pos		
Function	Only for some discontinued systems. Writes the specified values to the Flash Memory of the RTC6 board to be used as ASC reference values (= Home-In reference positions, not to confuse with Home-In positions).		
Call	`Result = write_hi_pos ( HeadNo, X1, X2, Y1, Y2 )`		
Parameters	`HeadNo`	Number of the scan head connector. As an unsigned 32-bit value. Allowed values: = 1: Connector for First Scan Head. = 2: Connector for Second Scan Head.	
	`X1`	X1 reference position. In bits. As a signed 32-bit value.	
	`X2`	Like `X1` (analogously).	
	`Y1`	Like `X1` (analogously).	
	`Y2`	Like `X1` (analogously).	
Result	Error code. As an unsigned 32-bit value.		
	0	Kein Fehler.	
	Bit #0	=1:	Wrong `HeadNo`.
	Bit #1	=1:	Wrong sensor position for `X1`.
	Bit #2	=1:	Wrong sensor position for `X2`.
	Bit #3	=1:	Wrong sensor position for `Y1`.
	Bit #4	=1:	Wrong sensor position for `Y2`.
	Bit #5	=1:	Invalid ASC version.
	Bit #6	=1:	Download failed. The values have possibly not been saved.
Multi-board Com'd Name	**n_write_hi_pos**		
Comments	• **write_hi_pos** is used in cases when a scan head is moved from RTC6 board "A" to RTC6 board "B" in order to transfer its Home-In reference positions from RTC6 board "A" to RTC6 board "B". • Use `get_hi_pos( HeadNo )` to read out the Home-In reference positions of RTC6 board "A" (only after **init_rtc6_dll**, see the **get_hi_pos** command description). • **write_hi_pos** is also executed if the scan head is switched off or even a scan head is not attached. • With an ASC type-1 equipped scan head attached, **auto_cal**`(Command = 4)` (or **auto_cal**`(Command = 0)` because this command implicitly executes **auto_cal**`(Command = 4)`) must have been successfully executed at last on the RTC6 board "B". A cross-check with `get_auto_cal(HeadNo)` needs to return 100. Otherwise, **write_hi_pos** would return the error Bit #5 = 1. If required, connect the scan head to RTC6 board "B" and execute **auto_cal**`(HeadNo, 4)` before executing **write_hi_pos**. • **write_hi_pos** takes several hundred µs. During this time, the 10 µs clock cycle of RTC6 boards is interrupted.		

Ctrl Command	write_hi_pos
RTC4→RTC6	New command.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**get_hi_pos**, **get_auto_cal**, **auto_cal**

Ctrl Command	write_image_eth	
Function	Standalone Functionality: Reads out data for automatic booting from a binary file on the PC and saves it to the NAND Memory.	
Prerequisite	RTC6 Software Package ≥ V1.7.0 and BIOS-ETH ≥ 26.	
Call	`Result = write_image_eth( Name )`	
Parameters	`Name`	Name of the binary file. As a pointer to a \0-terminated ANSI string.
Result	`Result`	Error code. Like `Result` of **read_image_eth**. As an unsigned 32-bit value.
Multi-board Com'd Name	**n_write_image_eth**	
Comments	• **write_image_eth** is not executed (**get_last_error** return code `RTC6_BUSY`), if: – the BUSY list execution status is set – the INTERNAL-BUSY list execution status is set • If the `Name` cannot be opened, a **get_last_error** return code `RTC6_PARAM_ERROR` is generated. • During the execution of **read_image_eth** the 10 μs clock cycle of the DSP is interrupted for up to 2 minutes. • **write_image_eth** is only allowed with RTC6 Ethernet Boards. Otherwise, a **get_last_error** return code `RTC6_TYPE_REJECTED` is generated. • See Chapter 16.7 "Standalone Functionality", page 988.	
RTC4→RTC6	New command.	
RTC5→RTC6	New command.	
Version Info	Available as of DLL 618, OUT 618, RBF 623.	
References	**read_image_eth**, **store_program**	

Ctrl Command	write_io_port
Function	Writes a value to the 16-Bit Digital Output Port on the EXTENSION 1 Socket Connector.
Call	`write_io_port( Value )`
Parameters	`Value` 16-bit output value (DIGITAL OUT0…DIGITAL OUT15). As an unsigned 32-bit value. Only the least significant 16 bits are evaluated.
Multi-board Com'd Name	**n_write_io_port**
Comments	• Use **set_io_cond_list** and **clear_io_cond_list** to set or clear individual bits of the 16-Bit Digital Output Port, depending on the state of the 16-Bit Digital Input Port. • **write_io_port_mask** and **write_io_port_mask_list** also allow changing selectable bits of the 16-Bit Digital Output Port. • See also Chapter 9.1.1 "16-Bit Digital Output Port", page 303.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_io_port_list**, **write_io_port_mask**, **write_io_port_mask_list**, **set_io_cond_list**, **clear_io_cond_list**, **get_io_status**

Delayed Short List Command	write_io_port_list
Function	Like **write_io_port**, but a list command.
Call	`write_io_port_list( Value )`
Parameters	`Value` Like **write_io_port**.
Multi-board Com'd Name	**n_write_io_port_list**
Comments	• See **write_io_port**. • As of version DLL 602, OUT 602: When the 16-Bit Digital Output Port (for example, for laser power) is to be executed synchronous to the laser switch times, use `set_laser_power( 3, Value )`.
RTC4→RTC6	Unchanged functionality.
RTC5→RTC6	Unchanged functionality.
Version Info	Available as of DLL 600, OUT 600, RBF 600.
References	**write_io_port**, **set_laser_power**, **write_port_list**

Ctrl Command	write_io_port_mask	
Function	Writes those bits of `Value` specified by the `Mask` parameter to the 16-Bit Digital Output Port on the EXTENSION 1 Socket Connector.	
Call	`write_io_port_mask( Value, Mask )`	
Parameters	`Value`	16-bit output value (DIGITAL OUT0…DIGITAL OUT15). As an unsigned 32-bit value. Only the least significant 16 bits are evaluated.
	`Mask`	16-bit mask (for DIGITAL OUT0…DIGITAL OUT15). As an unsigned 32-bit value. Only the least significant 16 bits are evaluated.
Multi-board Com'd Name	**n_write_io_port_mask**	
Comments	• The `Mask` parameter determines *which* bits of the 16-Bit Digital Output Port are to be altered, the `Value` parameter determines *how* they are altered. All bits of the 16-Bit Digital Output Port that were not set in `Mask` remain unaltered, that is, they are outputted again as previously. • For `Mask` = 0xFFFF, **write_io_port_mask** behaves like **write_io_port**. • See also Chapter 9.1.1 "16-Bit Digital Output Port", page 303.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**write_io_port**, **write_io_port_mask_list**	

Delayed Short List Command	write_io_port_mask_list	
Function	Like **write_io_port_mask**, but a list command.	
Call	`write_io_port_mask_list( Value, Mask )`	
Parameters	`Value`	Like **write_io_port_mask**.
	`Mask`	Like **write_io_port_mask**.
Multi-board Com'd Name	**n_write_io_port_mask_list**	
Comments	• See **write_io_port_mask**. • **write_io_port_mask_list** cannot be used for laser power control, see **write_io_port_list**.	
RTC4→RTC6	New command.	
RTC5→RTC6	Unchanged functionality.	
Version Info	Available as of DLL 600, OUT 600, RBF 600.	
References	**write_io_port_mask**, **write_io_port_list**, **set_laser_power**	

Undelayed Short List Command	**write_port_list**
Function	Writes a value to a digital or analog output port.
Call	`write_port_list( Port, Value, NoDelay )`
Parameters	`Port` Output port. As an unsigned 32-bit value. Allowed values: = 0: ANALOG OUT1 output port. See also Section "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 77. = 1: ANALOG OUT2 output port. See also Section "12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2", page 77. = 2: 8-Bit Digital Output Port (EXTENSION 2 Socket Connector). See also Section "8-Bit Digital Output Port", page 84. = 3: 16-Bit Digital Output Port (EXTENSION 1 Socket Connector). See also Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81. = 4: 2-Bit Digital Output Port (LASER Connector). See also Section "2-Bit Digital Output Port", page 77. > 4: **write_port_list** is not executed (**get_last_error** return code `RTC6_PARAM_ERROR`).
	`Value` Output value. As an unsigned 32-bit value. Allowed values: For `Port` = 0: 12-bit values [0…4,095]. For `Port` = 1: 12-bit values [0…4,095]. For `Port` = 2: 8-bit values [0…255]. For `Port` = 3: 16-bit values [0…65,535]. For `Port` = 4: 2-bit values [0…3]. Out-of-range values are clipped to the boundary values. **get_last_error** return code `RTC6_PARAM_ERROR`.
	`NoDelay` Output delay. As an unsigned 32-bit value. Allowed values: = 0: `Value` is outputted only after `PreviewTime` has expired. > 0: `Value` is outputted immediately.
Multi-board Com'd Name	**n_write_port_list**

Undelayed Short List Command	**write_port_list**
Comments	• `NoDelay` is only relevant, if the SCANahead Functionality on the RTC6 board has been activated by **set_scanahead_params**. • **write_port_list**(`NoDelay` = 0) behaves (depending on `Port`) similar to: – **write_da_x_list** – **write_8bit_port_list** – **write_io_port_list** – **set_laser_pin_out_list**
RTC4→RTC6	New command.
RTC5→RTC6	New command.
Version Info	Available as of DLL 634, OUT 635, RBF 634.
References	**set_scanahead_params**

10.3 RTC6 Commands in other Publications

Some RTC6 commands are published in the header (for example, `RTC6impl.hpp`), which are not part of this manual. These are intended either for a limited group of users or specific hardware.

RTC6 command	Corresponding multi-board command	Comments
check_enduring_wobbel	n_check_enduring_wobbel	Reserved.
enduring_wobbel	n_enduring_wobbel	Reserved.
eth_error_dump	n_eth_error_dump	Reserved.
get_home_4_return_time	n_get_home_4_return_time	Reserved.
get_home_position_4	n_get_home_position_4	Reserved.
home_position_4	n_home_position_4	Reserved.
jump_abs_drill	n_jump_abs_drill	page 510
jump_abs_drill_2	n_jump_abs_drill_2	page 510
jump_abs_drill_3	n_jump_abs_drill_3	Reserved.
jump_rel_drill	n_jump_rel_drill	page 513
jump_rel_drill_2	n_jump_rel_drill_2	page 513
jump_rel_drill_3	n_jump_rel_drill_3	Reserved.
list_jump_rel_ctrl	n_list_jump_rel_ctrl	Reserved.
load_oct_table_no	n_load_oct_table_no	Reserved.
load_zoom_correction_file	n_load_zoom_correction_file	page 568
micro_vector_quad_axis	n_micro_vector_quad_axis	Reserved.
micro_vector_quad_axis_v	n_micro_vector_quad_axis_v	Reserved.
micro_vector_quad_axis_v_2	n_micro_vector_quad_axis_v_2	Reserved.
micro_vector_set_position	n_micro_vector_set_position	Reserved.
move_to	n_move_to	page 597
multi_axis_config	n_multi_axis_config	Reserved.
multi_axis_flags	n_multi_axis_flags	Reserved.

RTC6 command (cont'd.)	Corresponding multi-board command (cont'd.)	Comments (cont'd.)
quad_axis_get_status	**n_quad_axis_get_status**	Reserved.
quad_axis_get_values	**n_quad_axis_get_values**	Reserved.
quad_axis_init	**n_quad_axis_init**	Reserved.
set_enduring_wobbel	**n_set_enduring_wobbel**	Reserved.
set_enduring_wobbel_2	**n_set_enduring_wobbel_2**	Reserved.
set_enduring_wobbel_3	**n_set_enduring_wobbel_3**	Reserved.
set_enduring_wobbel_4	**n_set_enduring_wobbel_4**	Reserved.
set_enduring_wobbel_ctrl	**n_set_enduring_wobbel_ctrl**	Reserved.
set_home_4_return_time	**n_set_home_4_return_time**	Reserved.
set_laser_pulse_form	**n_set_laser_pulse_form**	Reserved.
set_scanahead_line_params_ex	**n_set_scanahead_line_params_ex**	Reserved.
set_scanahead_line_params_ex_list	**n_set_scanahead_line_params_ex_list**	Reserved.
set_zoom	**n_set_zoom**	page 851
set_zoom_list	**n_set_zoom_list**	page 851

10.4 Unsupported RTC4 Commands

RTC4-Ctrl Command	**dsp_start**
Support status	Not supported by the RTC6.
RTC4→RTC6	**dsp_start** is not needed anymore. The DSP starts automatically after the program file is loaded by **load_program_file**.

RTC4-Ctrl Command	**get_xy_pos**
Support status	Not supported by the RTC6.
RTC4→RTC6	Replacement: **get_value**, **get_values**

RTC4-Ctrl Command	**get_xyz_pos**
Support status	Not supported by the RTC6.
RTC4→RTC6	Replacement: **get_value**, **get_values**

RTC4-Ctrl Command	**read_pixel_ad**
Support status	Not supported by the RTC6.
RTC4→RTC6	There is no equivalent RTC6 command.

RTC4-Ctrl Command	**rtc4_count_cards**
Support status	Not supported by the RTC6.
Replaced by	**rtc6_count_cards**

RTC4-Ctrl Command	**select_list**
Support status	Not supported by the RTC6.
RTC4→RTC6	**select_list**(0) can be replaced by **set_extstartpos**(0), **select_list**(1) by **set_extstartpos**(`Mem1`). Thereby, `Mem1` is the memory size of "List 1" (and the absolute start address of "List 2"). After initialization (with **load_program_file**), `Mem1` is 4000, otherwise as set by **config_list**. `Mem1` can be determined (for example, after a board changed "ownership") by **set_start_list_2** and **get_input_pointer**.

RTC4-Ctrl Command	set_list_mode
Support status	Not supported by the RTC6.
RTC4→RTC6	See Chapter 6.5.4 "RTC4-Circular Queue Mode", page 124.

RTC4-Ctrl Command	set_piso_control
Support status	Not supported by the RTC6.
RTC4→RTC6	**set_piso_control** is not needed anymore with the RTC6. For data transfer in accordance with the SL2-100 protocol, the bi-directional communication between the scan system and the RTC5/RTC6 does not depend on the data cable length. For data transfer in accordance with the XY2-100 protocol, the timing of the communication must be further on adjusted to reflect the length of the data cable; but the adjustment is now realized by a jumper at the XY2-100 Converter (Accessory).

RTC4-Ctrl Command	set_wobbel_xy
Support status	Not supported by the RTC6.
Replaced by	**set_wobbel**, **set_wobbel_mode**

RTC4-Ctrl Command	z_out
Support status	Not supported by the RTC6.
RTC4→RTC6	For setting the z value in a 3-axis scan system, **set_defocus** can be used (only together with an RTC6 with enabled Option "3D"). After **load_z_table**(0.0, 1.0, 0.0), **set_defocus**(Z) has the same effect as **z_out**(Z) (see also **set_offset_xyz**).

RTC4-Ctrl Command	z_out_list
Support status	Not supported by the RTC6.
RTC4→RTC6	After **load_z_table**(0.0, 1.0, 0.0), **set_defocus_list**(Z) has the same effect as **z_out_list**(Z) (see also **set_offset_xyz_list**).

10.5 Unsupported RTC5 Commands

RTC5-Ctrl Command	**free_rtc5_dll**
Support status	Not supported by the RTC6.
Replaced by	**free_rtc6_dll**

RTC5-Ctrl Command	**init_rtc5_dll**
Support status	Not supported by the RTC6.
Replaced by	**init_rtc6_dll**

RTC5-Ctrl Command	**rtc5_count_cards**
Support status	Not supported by the RTC6.
Replaced by	**rtc6_count_cards**

11 Demo Programs

- Currently the RTC6 Software Package does not contain demo programs.

12 Troubleshooting

Problem	Remedy
PC does not boot	Switch off the PC and check the following: • Check if the RTC6 board is correctly seated in the PCIe slot. Refer to the instructions in your PC manual. • Check for metal parts that may have fallen into the PC housing during installation. • Check for loose cables or connectors.
RTC6 board does not respond	• Check the driver installation. See Chapter 5.4 "Installing the RTC6 Software", page 93. • Use the included HPGL converter program to check if the board can be properly accessed. If not: Check the cable type and the cable length. If the scan system is controlled by an XY2-100 Converter (Accessory), then check, whether the solder jumpers of the XY2-100 Converter (Accessory) are set appropriate for the cable length. See Figure 14. If yes: Check the RTC6 DLL import declarations in your user program. See Chapter 6.2.2 "Importing Commands", page 98.
User program fails	• Check the driver installation. See Chapter 5.4 "Installing the RTC6 Software", page 93. • Check the RTC6 board initialization in the user program. • Check the RTC6 DLL import declarations in your user program. See Chapter 6.2.2 "Importing Commands", page 98.
Scan head control fails	• Check if the scan head is properly connected to the RTC6 board by the data cable. Make sure to follow the specifications for the data cable. See Chapter 4.5.3 "Data Cables (Accessories)", page 73. • Check the power supply of the scan head. Refer to your scan head operating manual. • Check your user program.
Laser control fails	• Check the interface between the RTC6 board and the laser.
Irregular marking results	• Check the Laser Delays and Scanner Delays. See Chapter 7.2.3 "Notes on Optimizing the Delays", page 158.
Laser does not switch off during Jump Commands	• Check the Laser Delays and Scanner Delays. See Chapter 7.2 "Delay Settings – Coordinating Scan Head Control and Laser Control", page 148.

Additionally, the following commands are helpful for troubleshooting:

- By **get_error** and **get_last_error**, nearly any command can be checked for proper execution, see Chapter 6.8 "Error Handling", page 133.
- By **set_verify**, you can verify that all downloads (commands, tables) were performed error-free, see Section "Download Verification", page 134.
- By **get_value** or **get_values**, you can query specific values returned from the scan-system. By **set_trigger[*]** and **get_waveform**, you can record an entire series of return values, see Section "Status Monitoring and Diagnostics", page 188 (with iDRIVE Scan Systems see also Chapter 8.1 "iDRIVE Functions", page 221).
- By **set_wait** and **get_wait_status**, you can check if program branches (conditional jumps) executed as intended.
- By **get_overrun**, you can check if overruns of the 10 µs clock cycle occurred, see Section "Clock Overruns", page 187.
- By **get_status** or **get_out_pointer**, you can determine which command number the program is currently executing (for example, for "infinite loops" due to a circular argument in the program flow).
- **get_startstop_info** provides information about the Laser Control Signals and possible transmission errors to and from the attached scan system.

If specific outputs from a port have no effect, then check if the user program is performing directly consecutive accesses of that same port. Because so-called short list commands (see Section "Normal List Command, Short List Command, Variable List Command and Multiple List Command", page 335) are typically used for this, one command might overwrite the other's output value within the same 10 µs clock cycle. In this situation, separate both short commands with a (normal) list command, for example, **list_nop** or **list_continue**.

If the execution time (measured by **save_and_restart_timer** and **get_time**) does not correspond with your calculation, check if your user program contains so-called short list commands, which generally do not require their own clock cycle for execution. Another possibility is that additional Scanner Delays were automatically inserted to prevent (improper) overlap of LaserOn and LaserOff, see Section "Automatic Delay Adjustments", page 158.

If the problems persist, contact SCANLAB.

13 Customer Service

13.1 Servicing and Repairs

All RTC6 board servicing and repairs should be performed only at SCANLAB.The warranty expires if the RTC6 board has been altered.

13.2 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 are 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 is not held responsible:

- when the product has been damaged through misuse or improper operation
- for RTC6 board repairs not performed by SCANLAB
- for damage resulting from improper packaging of a product returned to SCANLAB
- if the RTC6 board has been altered
- for consequential damages

13.3 Contacting SCANLAB

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

SCANLAB GmbH
Siemensstr. 2a
82178 Puchheim
Germany

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

info@scanlab.de
www.scanlab.de

13.4 Product Disposal

The RTC6 board can be returned to SCANLAB for a fee to be properly disposed of.

14 Legal

14.1 EU Declaration of Conformity – RTC6 PCIe Board

EU-Konformitätserklärung

für das Produkt:

RTC4 (PCI); RTC4 PCI-Express; RTC5 (PCI); RTC5 PCI-Express; RTC6 PCI-Express

Der Hersteller

SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Deutschland

erklärt, dass das genannte Produkt die einschlägigen Harmonisierungsrechtsvorschriften der Union erfüllt:

- **2014/30/EU -** Richtlinie des EUROPÄISCHEN PARLAMENTS UND DES RATES zur Harmonisierung der Rechtsvorschriften der Mitgliedstaaten über die elektromagnetische Verträglichkeit (EMV-Richtlinie).
- **2011/65/EU -** Richtlinie des EUROPÄISCHEN PARLAMENTS UND DES RATES zur Beschränkung der Verwendung bestimmter gefährlicher Stoffe in Elektro- und Elektronikgeräten (RoHS-Richtlinie).

Folgende harmonisierte Normen wurden angewandt:

- **EN IEC 61000-6-2:2005** – Elektromagnetische Verträglichkeit (EMV) - Teil 6-2: Fachgrundnormen - Störfestigkeit für Industriebereiche
- **DIN EN 55011:2018-05** - Industrielle, wissenschaftliche und medizinische Geräte - Funkstörungen - Grenzwerte und Messverfahren
- **DIN EN IEC 63000:2019-05** - Technische Dokumentation zur Beurteilung von Elektro- und Elektronikgeräten hinsichtlich der Beschränkung gefährlicher Stoffe

Die alleinige Verantwortung für die Ausstellung dieser Konformitätserklärung trägt der Hersteller.

Dieses Dokument zur Kundeninformation wurde elektronisch ausgestellt und ist daher ohne Unterschrift gültig. Eine unterschriebene Ausführung ist Bestandteil der SCANLAB-internen technischen Dokumentation.

EU Declaration of Conformity

for the product:

RTC4 (PCI); RTC4 PCI-Express; RTC5 (PCI); RTC5 PCI-Express; RTC6 PCI-Express

The manufacturer

SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Germany

declares that the product mentioned above complies with the relevant Union harmonization legislation:

- **2014/30/EU -** Directive of the EUROPEAN PARLIAMENT AND OF THE COUNCIL on the harmonisation of the laws of the Member States relating to electromagnetic compatibility (EMC Directive)
- **2011/65/EU** - Directive of the EUROPEAN PARLIAMENT AND OF THE COUNCIL on the restriction of the use of certain hazardous substances in electrical and electronic equipment (RoHS Directive).

The following harmonized standards have been applied:

- **EN IEC 61000-6-2:2005** - Electromagnetic compatibility (EMC) - Part 6-2: Generic standards - Immunity for industrial environments
- **DIN EN 55011:2018-05** - Industrial, scientific and medical equipment – Radio-frequency disturbance characteristics - Limits and methods of measurement
- **DIN EN IEC 63000:2019-05** - Technical documentation for the assessment of electrical and electronic products with respect to the restriction of hazardous substances

This declaration of conformity is issued under the sole responsibility of the manufacturer.

This document, used for customer information, was issued electronically and is therefore valid without a signature. A signed version is part of the SCANLAB internal technical documentation.

SCANLAB GmbH
Puchheim
2021-07-19

14.2 Compliance with FCC Rules

The RTC6 PCIe Board has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules.

These limits are designed to provide reasonable protection against harmful interference when the RTC6 PCIe Board is operated in a commercial environment. The RTC6 PCIe Board generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with this instruction manual, may cause harmful interference to radio communications. Operation of the RTC6 PCIe Board in a residential area is likely to cause harmful interference in which case the user is required to correct the interference at his own expense.

15 Technical Specifications – RTC6 PCIe Board

System Requirements

Windows PC with free PCI-Express slot

Operating system	With RTC6 Software Package ≥ V1.3.0: Microsoft Windows 10, 8, 7 as 32-bit or 64-bit version. *Windows XP and Windows Vista are not supported!*

Dimensions

Length	160 mm
Height	104.5 mm

Interface to the PC

PCI-Express x-1, version 1.0

Scan System Control

Number of RTC6 List Memory areas	Up to 3, configurable
Total capacity of RTC6 List Memory	8,388,608 list positions
Output interval of Microsteps	10 μs
Maximum range for the Image Field coordinates	–524,288…+524,287 (20-bit signed)
Virtual Image Field (for example, for Processing-on-the-fly)	–268,435,456… +268,435,455 (28 bit, signed = 29 bit)[a]

(a) With RTC6 Software Package ≥ 1.4.0.

Interfaces

SCANHEAD Connector

Connector	9-pin D-SUB connector (female)

2. SCANHEAD Socket Connector

Connector	10-pin socket connector. • xy output values only with enabled Option “Second Scan Head Control” • z output values only with enabled Option “3D”
Signal transmission	SL2-100 protocol. Via XY2-100 Converter (Accessory): XY2-100 protocol.

LASER Connector

Connector	15-pin D-SUB connector (female)
Laser Control Signals	LASER1, LASER2, LASERON
• TTL level	5 V, active-HIGH or active-LOW (programmable)
• Max. current load	20 mA
• Reference	GND[a]
	GND2 with Option "DC/DC Converter"

(a) RTC6 PCIe Boards: GND is the PC ground. RTC6 Ethernet Boards: GND is the ground at the POWER connector.

Output port	12-Bit Analog Output Port 1 ANALOG OUT1
• Output voltage range	0 V…10 V
• Resolution	12 Bit
• Max. current load	5 mA
• Reference	GND[a]
Output port	12-Bit Analog Output Port 2 ANALOG OUT2
• Output voltage range	0 V…10 V
• Resolution	12 Bit
• Max. current load	5 mA
• Reference	GND[a]
Input port	2-Bit Digital Input Port
• LOW level	< 0.6 V
• HIGH level	> 2.3 V
• Max. input voltage range	–0.5 V…+5.5 V
• Input resistance (pull-up)	> 4.7 kΩ
• Reference	GND[a]
Output port	2-Bit Digital Output Port
	buffered
• LOW level	< 0.55 V
• HIGH level	> 3.8 V
• Max. current load	20 mA
• Reference	GND[a]
Input ports for external control signals	TTL level
	active-LOW
	/START and /STOP are connected internally to +3.3 V by pull-up resistors (4,7 kΩ), see Figure 18.
• /START	edge sensitive (HIGH → LOW transition)
	HIGH level > 2.3 V
	LOW level < 0.6 V
• /STOP	level sensitive
	HIGH level > 2.3 V
	LOW level < 0.6 V
• Reference	GND[a]
Other signals	
• BUSY OUT	5 V, TTL active-HIGH
	max. 10 mA
	BUSY OUT, page 951, BUSY OUT, page 952, BUSY OUT, page 953 are identical
• +5 V	max. 100 mA
• Reference	GND[a]

EXTENSION 1 Socket Connector

Connector	40-pin socket connector
	output signal level configurable by jumper
Output port	16-Bit Digital Output Port
	buffered
• LOW level	< 0.4 V
• HIGH level	> 2.0 V (3.3 V or 5 V)
• Max. current load	±8 mA
• Reference	GND[a]
LATCH OUT	3.3 V or 5 V, TTL active-HIGH, 5 μs pulse, max. 10 mA
SYNC OUT	3.3 V or 5 V, TTL active-HIGH, square wave signal (5 μs pulse, 10 μs period) max. 10 mA
Input port	16-Bit Digital Input Port
• LOW level	< 0.5 V
• HIGH level	> 2.6 V…24 V
• Max. input voltage range	–0.5 V…+26 V
• Input resistance	> 10 kΩ
• Reference	GND[a]
Other signals	
• BUSY OUT	3.3 V or 5 V, TTL active-HIGH, max. 10 mA
	BUSY OUT, page 951, BUSY OUT, page 952, BUSY OUT, page 953 are identical
• VCC_OUT	3.3 V or 5 V, max. 100 mA
• +5 V	max. 100 mA
• Reference	GND[a]

EXTENSION 2 Socket Connector

Connector	26-pin socket connector
	configurable by jumper
Output port	8-Bit Digital Output Port
	buffered
• LOW level	< 0.4 V
• HIGH level	> 2.0 V
• Max. current load	±8 mA
• Reference	GND[a]
• LATCH signal	5 V, TTL active-HIGH, 5 μs pulse, max. 10 mA
Laser Control Signals	LASERON, LASER1, LASER2 (see LASER Connector, page 951)
Other signals	
• +5 V	max. 100 mA
• Reference	GND[a]

MARKING ON THE FLY Socket Connector

Connector	16-pin socket connector
2 encoder input ports for incremental encoders	ENCODER X(1±,2±) and ENCODER Y(1±,2±), designed for a pair of standardized differential input signals (RS-422) each. HIGH level ≥ 2.0 V LOW level ≤ 0.8 V) f ≤ 4 MHz
Output port	12-Bit Analog Output Port 2 ANALOG OUT2 (see LASER Connector, page 951)
Input ports for external control signals	TTL level active-LOW /START2 and /STOP2 are connected internally to +3.3 V by pull-up resistors (4,7 kΩ).
• /START2	edge sensitive (HIGH → LOW transition) HIGH level > 2.3 V LOW level < 0.6 V
• /STOP2	level sensitive HIGH level > 2.3 V LOW level < 0.6 V
• Reference	GND[(a)]
Other signals	
• BUSY OUT	5 V, TTL active-HIGH max. 10 mA BUSY OUT, page 951, BUSY OUT, page 952, BUSY OUT, page 953 are identical
• +5 V	max. 100 mA
• Reference	GND[(a)]

RS232 Socket Connector

Connector	10-pin socket connector
Input port	RxD
• Max. voltage range	–25 V…+25 V
Output port	TxD
• Max. voltage range	–13 V…+13 V
Reference	GND[(a)]
Baud rate	300…115200

McBSP/ANALOG Socket Connector

Connector	10-pin socket connector
McBSP interface	See Section "McBSP Interface", page 87.
• Transmitter signal level	3.3 V TTL
• Receiver signal level	3.3 V or 5 V TTL
• McBSP mode	Single Phase Frame Single Element per Frame 32 bits per Element Data delay `XDelay` bits Data delay `RDelay` bits
• Reference	GND[(a)]
SPI interface functionality	No
Analog input ports	ANALOG IN0, ANALOG IN1
• Input voltage range	0 V…10 V
• Input resistance	> 5 kΩ
• ADC resolution	12 bit
• Reference	GND[(a)]

STEPPER MOTOR Socket Connector

Connector	10-pin socket connector
Signals for controlling 1 or 2 stepper motors:	
• ENABLE output port	5 V, TTL
• DIRECTION output port	5 V, TTL
• CLOCK output port	5 V, TTL active-HIGH, 5 μs pulse
• SWITCH input port	TTL active-LOW Internally, SWITCH1-input port and SWITCH2-input port are connected to +3.3 V by 10 kΩ pull-up resistors.
• Max. current load	25 mA
• Reference	GND[(a)]

16 Appendix A: RTC6 Ethernet Board

16.1 Product Overview

16.1.1 RTC6 Ethernet Board vs. RTC6 PCIe Board – Usage and Comparison

RTC6 Ethernet Boards and RTC6 PCIe Boards share the same *core functionality*:

- Real-time control of the laser and scan system.
- Transfer of direct commands (control commands) from the computer to the board. These commands are executed immediately.
- Transfer of list commands from the computer to the RTC6 List Memory. These commands execute only after the list is started.
- List execution occurs in real time.
- Calculation of the set position occurs every 10 μs.
- Image Field correction is applied to the set positions.
- The interface to the scan system uses the 20-bit SL2-100 protocol.
- For software development, the total command set for RTC6 Ethernet Boards and RTC6 PCIe Boards is contained in `RTC6DLL.dll`/`RTC6DLLx64.dll`.

RTC6 Ethernet Boards differ in the following aspects:

- Ethernet interface[1] instead of PCIe bus
  - Communication with the computer is by TCP/IP and UDP.
  - RTC6 Ethernet Boards work *without* RTC6 board driver.
  - The board does not need to be plugged into a PC.
  - Real-time clock, see Chapter 16.2.15 "Real-Time Clock", page 977.
  - With RTC6 Software Package ≥ V1.7.0, RTC6 Ethernet Boards can also be used in standalone operation without a PC, see Chapter 16.7 "Standalone Functionality", page 988.

(1) Supports Gigabit Ethernet according to 1000BASE-T.

16.1.2 System Requirements

Caution!

- To avoid interference coupling and emissions, never operate the RTC6 Ethernet Board without shielding (e.g. outside a metal housing).
- The RTC6 Ethernet Board is designed exclusively for industrial use. It is intended for integration in a machine (typically in a laser system) and does **not** fulfill all requirements of a ready-to-use consumer end product. Only operate the RTC6 Ethernet Board after integration in a machine meeting all applicable directives and standards (of your local jurisdiction). It is *not* suitable for use as a toy, in households or inclement environments (e.g. outdoors). The operator must take appropriate precautionary measures to prevent such improper usage.
- Installation and commissioning should only be performed by personnel with sufficient qualifications, e.g. knowledge of electrical equipment safety. Only perform installation and maintenance if power and lasers are switched off.
- Because the RTC6 Ethernet Board is a Class A device capable of generating interference in residential areas, the operator may be required to carry out appropriate measures.

Hardware

- Shielded housing with:
  - Provisions for mechanical mounting, incl. mounting hardware
  - appropriate thermal coupling (cooling) of the board
  - Power outlet
  - Ethernet cable
  - Power supply as specified
  - Cable with connector for powering the RTC6 Ethernet Board
  - Flat ribbon cables for peripheral equipment (laser, scan head, extensions)
- Windows PC
- Gigabit Ethernet (preferred), 10/100 Mbit/s Ethernet

Software

- TCP/IP protocol IPv4
- UDP protocol
- Used ports (default)
  - 63749 (UDP)
  - 63750 (UDP, TCP)

For the development of user programs:

- DLL/Import declarations (contained in RTC6 Software Package)

16.1.3 Options

Same as RTC6 PCIe Boards, see Chapter 2.6 "Options", page 39. Additionally, with RTC6 Ethernet Boards only:

- **Option "LDSA"**

  – Support of Standalone Functionality by laserDESK

  Allows Standalone Functionality-related features to be used in laserDESK.
  Example: Booting an RTC6 Ethernet Board with a Boot Image created under laserDESK.
  Note that Boot Images created under laserDESK cannot be used with RTC6 Ethernet Boards without Option "LDSA". For more information, refer to the laserDESK online help.

  See also **get_rtc_version**, Bit #11.

16.1.4 Labeling

Same as RTC6 PCIe Boards, see Chapter 2.1 "Labeling", page 32.

16.1.5 Type Identification

Same as RTC6 PCIe Boards, see Chapter 2.7 "Jumper Settings and Type Designations", page 41.

16.1.6 Unpacking Instructions and Typical Scope of Delivery

Same as RTC6 PCIe Boards, see Chapter 2.2 "Unpacking Instructions and Typical Scope of Delivery", page 32.

16.1.7 Delivered Software

Same as RTC6 PCIe Boards, see Chapter 2.3 "Delivered RTC6 Software Package", page 32.

16.1.8 Accessories for the RTC6 PCIe Board

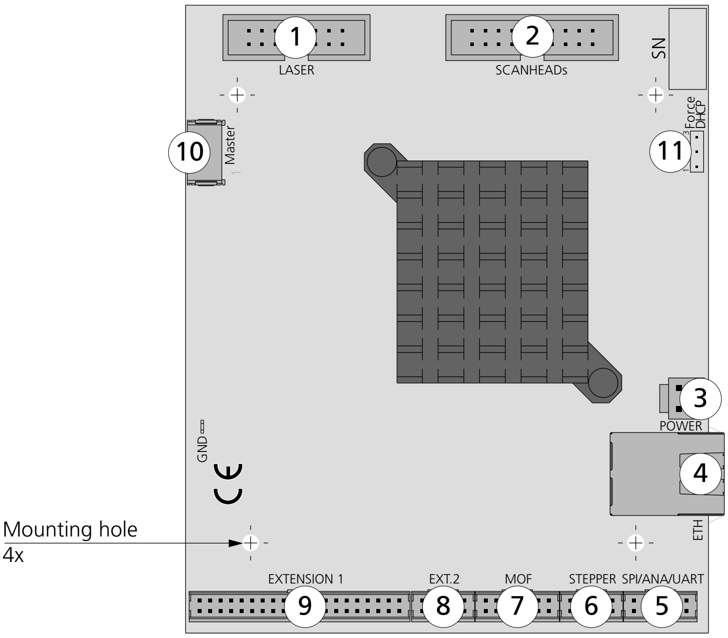
Accessories mentioned in Chapter 2.8 "Accessories for the RTC6 PCIe Board", page 45 for RTC6 PCIe Board is (without suiting adapters) not suitable for RTC6 Ethernet Boards (because of different connector plugs and pitch of the pins).

16.1.9 Supplementary Software

As with RTC6 PCIe Boards, see Chapter 2.9 "Supplementary Software", page 48.

16.2 Layout, Interfaces, Jumper Settings

16.2.1 Layout – Upper Side



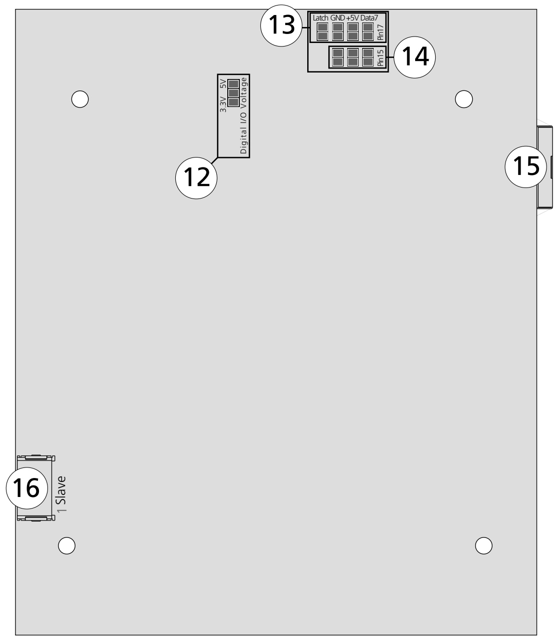
Legend

1. LASER . . . . 16-pin socket connector. To connect the laser. With digital and analog outputs.
   For details, see "LASER Socket Connector", page 961.
2. SCANHEADs. . . . . 20-pin socket connector. To connect the first and second scan head.
   For details, see "SCANHEADs Socket Connector", page 964.
3. POWER . . . . 2-pin socket connector. For the voltage supply.
   For details, see "POWER Socket Connector", page 966.
4. ETH . . . . RJ-48 8P8C connector. For the network connection.
   For details, see Chapter 16.2.7 "ETH Connector", page 966.
5. SPI/ANA/UART . . . . 12-pin socket connector. Hardware interface for several data exchange methods.
   With analog input ports. For details, see "SPI/ANA/UART Socket Connector", page 967.
6. STEPPER. . . . . 10-pin socket connector. For controlling up to two stepper motors.
   For details, see "STEPPER Socket Connector", page 968.
7. MOF . . . . 14-pin socket connector. Hardware interface for encoder pulses, for example, for Processing-on-the-fly applications. For details, see "MARKING ON THE FLY Socket Connector", page 85.
8. EXT. 2 . . . . 10-pin socket connector. With a 8-Bit Digital Output Port.
   For details, see "EXTENSION 2 Socket Connector", page 83.
9. EXTENSION 1 . . . . 40-pin socket connector. With a 16-Bit Digital Output Port and a 16-Bit Digital Input Port. For details, see EXTENSION 1 Socket Connector, page 970.
10. Master. . . . . 6-pin socket connector. To connect with another RTC6 board for the purpose of synchronize clocks. For details, see "Master Socket Connector, Slave Socket Connector", page 973.
11. Force DHCP . . . . Jumper field Jumper field to ignore or not to ignore the saved static IP address.
    For details, see "Jumper Field 'Force DHCP'", page 977.

74

RTC6 Ethernet Board V1.2. Upper side.

16.2.2 Layout – Lower Side



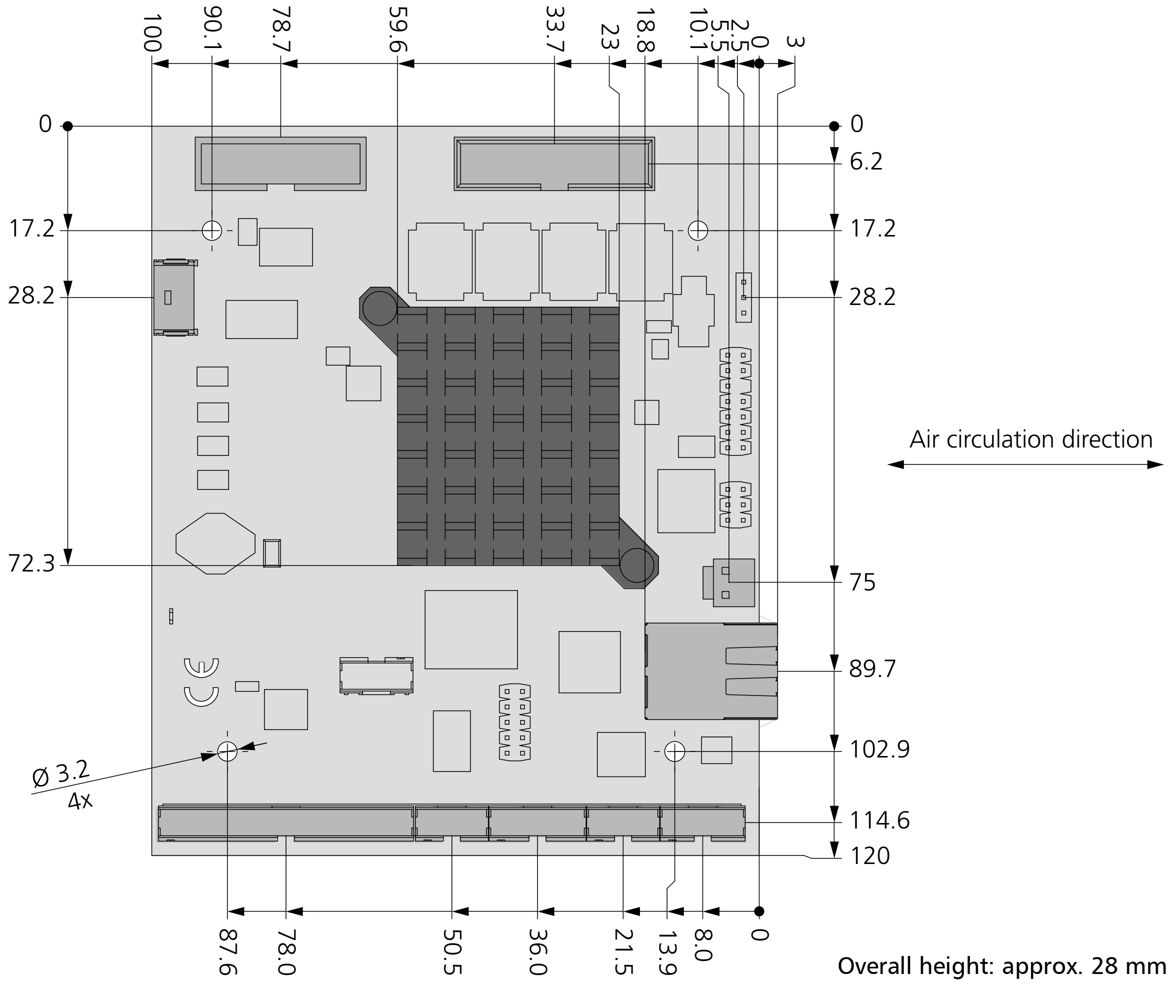
Legend

12. Solder jumper field A . . . . To configure the level of the output signals at the EXTENSION 1 Socket Connector, page 970.
For details, see "Solder Jumper Field A – Configuring Output Signal Level at EXTENSION 1 Socket Connector", page 42.
13. Solder jumper field B . . . . To configure the signal at EXT. 2 Socket Connector pin (09).
For details, see "Solder Jumper Field B – Configuring pin (09) of EXT. 2 Socket Connector", page 975.
14. Solder jumper field C . . . . To configure the signal at EXT. 2 Socket Connector pin (08).
For details, see "Solder Jumper Field C – Configuring pin (08) of EXT. 2 Socket Connector", page 976.
15. ETH connector . . . . See Figure 74.
16. Slave. . . . . 6-pin socket connector. To connect with another RTC6 board for the purpose of synchronize clocks. For details, see "Master Socket Connector, Slave Socket Connector", page 973.

75

RTC6 Ethernet Board V1.2. Lower side.

16.2.3 Dimensions and Connector Positions



Overall height: approx. 28 mm depending on the installed cooling element).

76

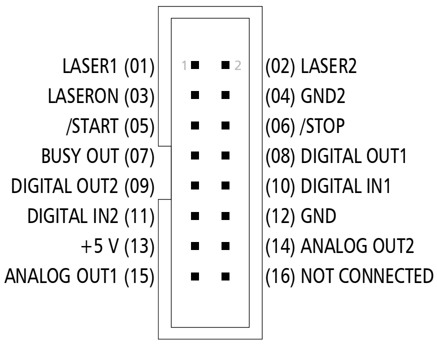
RTC6 Ethernet Board V1.2. Dimensions and connector positions. All dimensions in mm.

16.2.4 LASER Socket Connector

- For technical data see Chapter 16.11 "Technical Specifications – RTC6 Ethernet Board", LASER Socket Connector, page 1040.

The LASER Socket Connector has 16 pins[(1)]. It is located on the upper side of the RTC6 Ethernet Board, see Figure 74, number 1.

The pin-out is shown in Figure 77.



Identical in construction to Würth 61201620621.

77

LASER Socket Connector: pin-out. The pitch of the pins is 2.54 mm. On RTC6 Ethernet Boards without Option "DC/DC Converter", GND and GND2 are identical. On RTC6 Ethernet Boards with Option "DC/DC Converter", GND2 and the laser output signal are galvanically decoupled from GND.

Notice!

- If you want to use the RTC6 Ethernet Board in conjunction with laserDESK, observe the following:
  - laserDESK uses additional pins on the EXTENSION 1 Socket Connector, page 970, for example, to control the laser.
  - Refer to the extended documentation for the LASER connector which can be found in the laserDESK online help (alternatively available from SCANLAB or in laserDESK-zip, which can be downloaded from the SCANLAB website).

(1) With RTC6 PCIe Boards: 15 pin D-SUB connector, female.

Notes

- With RTC6 Ethernet Boards, GND is the ground at the POWER Socket Connector. See Chapter 16.2.6 "POWER Socket Connector", page 966. With the RTC6 PCIe Board, GND is the PC ground.
- A suitable 0.2 m cable (#0116048) is available from SCANLAB, see also Figure 78.

Laser Control Signals

As with RTC6 PCIe Boards, see Chapter 4.6.1 "LASER Connector", page 75. However, note that the pin numbers differ.

Signal	RTC6 Ethernet Board	RTC6 PCIe Board
LASERON	Pin (03)	Pin (02)
LASER1	Pin (01)	Pin (01)
LASER2	Pin (02)	Pin (09)

External Control Signals

As with RTC6 PCIe Boards, see Chapter 4.6.1 "LASER Connector", page 75.

BUSY List Execution Status

As with RTC6 PCIe Boards, see Chapter 4.6.1 "LASER Connector", page 75. However, note that the pin numbers differ.

Signal	RTC6 Ethernet Board	RTC6 PCIe Board
BUSY OUT	Pin (07)	Pin (04)

2-Bit Digital Input Port, 2-Bit Digital Output Port

As with RTC6 PCIe Boards, see Chapter 4.6.1 "LASER Connector", page 75. However, note that the pin numbers differ.

Signal	RTC6 Ethernet Board	RTC6 PCIe Board
DIGITAL IN1	Pin (10)	Pin (13)
DIGITAL IN2	Pin (11)	Pin (06)
DIGITAL OUT1	Pin (08)	Pin (12)
DIGITAL OUT2	Pin (09)	Pin (05)

12-Bit Analog Output Port 1, 12-Bit Analog Output Port 2

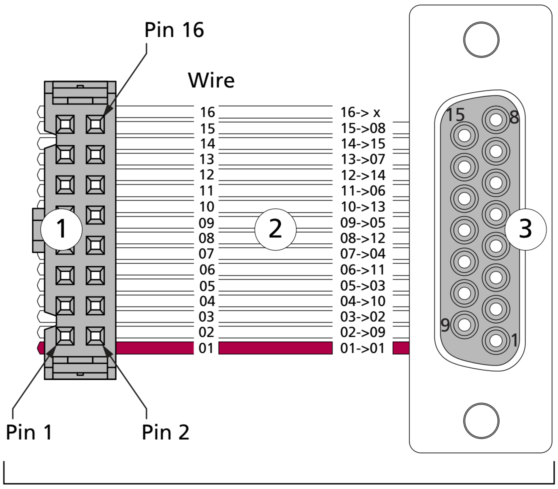
As with RTC6 PCIe Boards, see Chapter 4.6.1 "LASER Connector", page 75. However, note that the pin numbers differ.

Signal	RTC6 Ethernet Board	RTC6 PCIe Board
ANALOG OUT1	Pin (15)	Pin (08)
ANALOG OUT2[a]	Pin (14)	Pin (15)

(a) With the RTC6 PCIe Board the ANALOG OUT2 signal is additionally available at the MARKING ON THE FLY Socket Connector.

Input and Output Wiring

The input and output wiring of the LASER Socket Connector is shown in Figure 79. It is identical to RTC6 PCIe Boards, except the pin numbers.



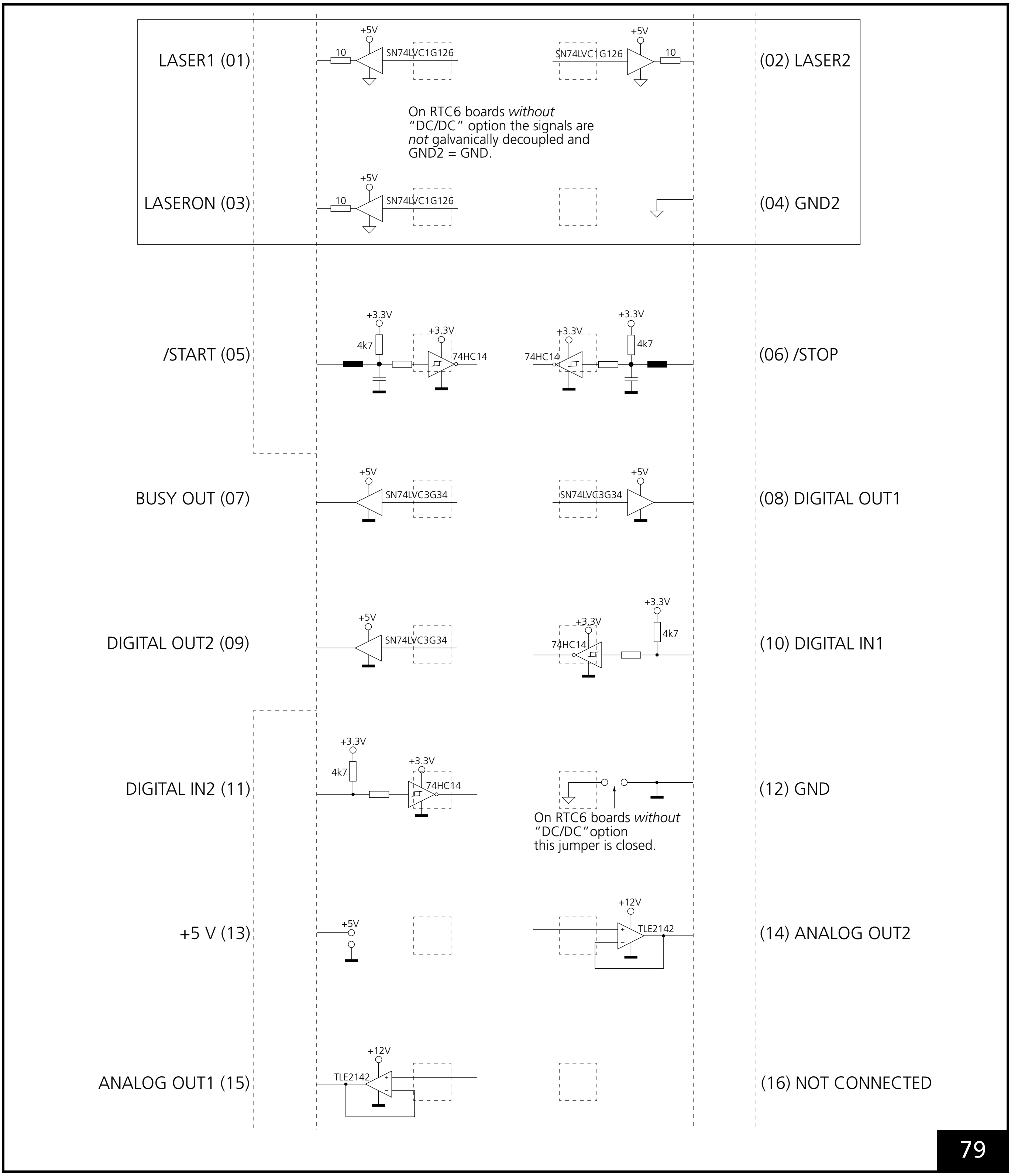
Corresponds functionally to cable #0116048 (l=0.2 m).

Legend

1. Connector, 16-pin female. For LASER Socket Connector, see Figure 77.
   Identical in construction to Würth 61201623021.
2. Flat ribbon cable 26 AWG, 20 conductors, pitch 1.27 mm.
3. Connector, 15-pin D-SUB, female.

78

Cable (proposal). Actual implementation may differ according to customer needs.
The depicted items are not in the standard scope of delivery of the RTC6 Ethernet Board!



LASER Socket Connector (detail, indicated by dashed line): input/output wiring plan.

16.2.5 SCANHEADs Socket Connector

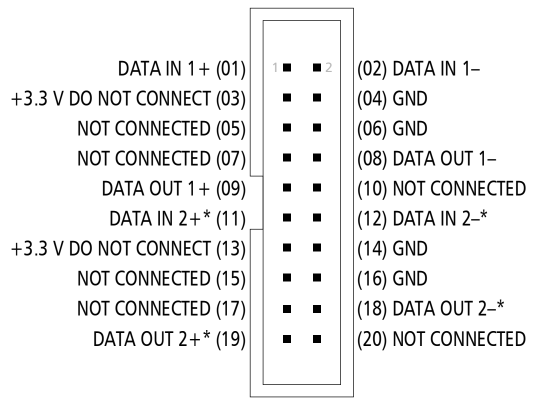
- For technical data see Chapter 16.11 "Technical Specifications – RTC6 Ethernet Board", SCANHEADs Socket Connector, page 1040.

The SCANHEADs Socket Connector has 20 pins. It is located on the upper side of the RTC6 Ethernet Board, see Figure 74, number 2.

The SCANHEADs Socket Connector provides the same signals as the RTC6 PCIe Board:

- pin (01)…pin (10) like SCANHEAD Connector (Connector for First Scan Head)
- pin (11)…pin (20) like 2. SCANHEAD Socket Connector (Connector for Second Scan Head)

The pin-out is shown in Figure 80.



* Signals for the 2nd scan head: option.

Identical in construction to Würth 61202020621.

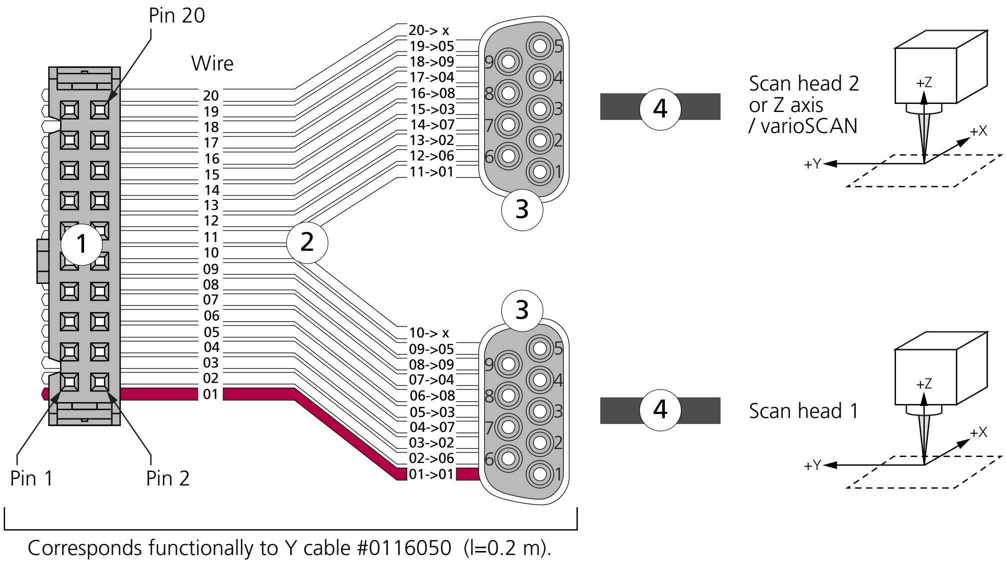
80

SCANHEADs Socket Connector: pin-out. The pitch of the pins is 2.54 mm.

A cabling example is shown in Figure 81 (the shown Y cable is available from SCANLAB with a length of 0.2 m: #0116050).

Scan system control (via an XY2-100 Converter (Accessory), if needed) is identical to that of the RTC6 PCIe Board.

For further information, see Chapter 4.5 "Interfaces to Scan System", page 69.



Legend

1. Connector, 20-pin female. For SCANHEADs Socket Connector, see Figure 80.
   Identical in construction to Würth 61202023021.
2. Flat ribbon cable 26 AWG, 20 conductors, pitch 1.27 mm.
3. Connector, 9-pin D-SUB, female.
4. Data cable for SL2-100 protocol. For example, #0115428.

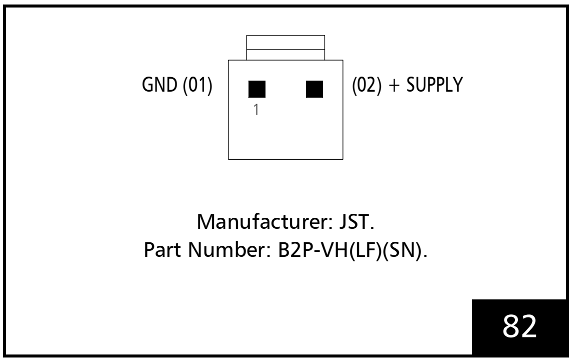
81

Cabling (proposal). Actual implementation may differ according to customer needs.
The depicted items are not in the standard scope of delivery of the RTC6 Ethernet Board!

16.2.6 POWER Socket Connector

The POWER Socket Connector has 2 pins. It is located on the upper side of the RTC6 Ethernet Board, see Figure 74, number 3. It serves to provide the voltage supply for the RTC6 Ethernet Board.

The pin-out is shown in Figure 82.



POWER Socket Connector. The pitch of the pins is 3.96 mm.

Pin	Power supply
(01) GND	Ground.
(02) + Supply	See Chapter 16.11 "Technical Specifications – RTC6 Ethernet Board", page 1039.

Notes

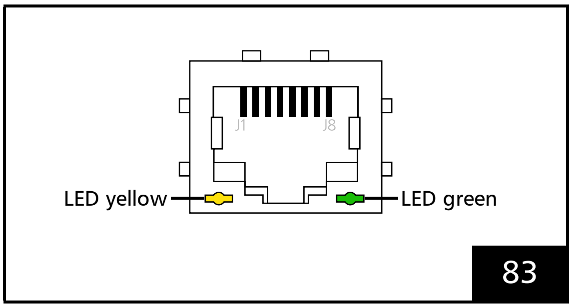
- Required parts for the mating connector are *not included in the scope of delivery*:
  Manufacturer: JST
  1 × Housing, part number VHR-2N
  2 × Contact, part number SVH-41T-P1.1.

16.2.7 ETH Connector

The ETH connector is an 8P8C type RJ connector (colloquially: RJ-45). It is located on the upper side of the RTC6 Ethernet Board, see Figure 74, number 4. It serves to connect[1] the RTC6 Ethernet Board to the network.

For status indication the following LEDs are part of the connector, see Figure 83:

- LED yellow, function "IP address assigned".
  The LED is *permanently* on as soon as the RTC6 Ethernet Board has an assigned IP address (static IP, DHCP or as of BIOS 24 link-local address[2]).
  Note: Without stored static IP address the switched-on LED indicates that the IP address assignment has been carried-out successfully by DHCP. In case if there is a stored static IP address a switched-on LED does not mean that the IP address is valid in the subnet in every case. As soon as the board has been acquired by a PC the LED *flashes* when there is network traffic ("Traffic").
- LED green, function "Link / Traffic".
  The LED is *permanently* on as soon as a connection is established ("Link"). This is usually the case when the Ethernet cable is plugged-in and there is an active router or a LAN adapter at the remote end.
  The LED *flashes* when there is network traffic ("Traffic").



ETH connector. RJ-48 8P8C.

(1) Category 5e (Cat 5e) Ethernet cable recommended.

(2) See page 979.

16.2.8 SPI/ANA/UART Socket Connector

- For technical data see Chapter 16.11 "Technical Specifications – RTC6 Ethernet Board", SPI/ANA/UART Socket Connector, page 1043.

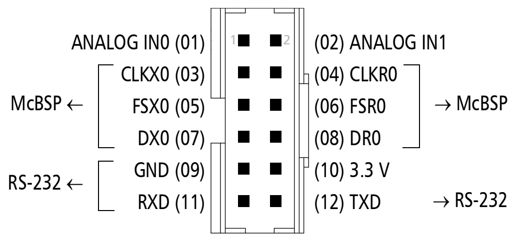
The SPI/ANA/UART Socket Connector has 12 pins. It is located on the upper side of the RTC6 Ethernet Board, see Figure 74, number 5.

The SPI/ANA/UART Socket Connector is a hardware interface intended for analog input ports as well as for the data exchange methods McBSP (Multichannel Buffered Serial Port) and RS-232.

The SPI/ANA/UART Socket Connector corresponds functionally to the:

- McBSP/ANALOG Socket Connector and RS232 Socket Connector of RTC6 PCIe Boards

The relevant pin-outs are shown in Figure 84.



Manufacturer: Amphenol FCI.
Product Number: 98424-F52-12LF.

84

SPI/ANA/UART Socket Connector: pin-out. The pitch of the pins is 2.00 mm.

Notes

- Required parts for the mating connector are *not included in the scope of delivery*:
  Manufacturer: Amphenol FCI
  Series. Minitek®
  1× Housing, part number 90311-012LF
  12× Contact, part number 77138-101LF.

Analog Input Ports

The SPI/ANA/UART Socket Connector provides 2 analog input ports:

- ANALOG IN0
  - See Section "Analog Input Ports", page 89
- ANALOG IN1
  - See Section "Analog Input Ports", page 89

For technical specifications, see SPI/ANA/UART Socket Connector, page 1043.

McBSP Interface

See Section "McBSP Interface", page 87.

RS-232 Interface

With RTC6 Ethernet Boards, the RS232 Socket Connector pins of the RTC6 PCIe Board are integrated to the SPI/ANA/UART Socket Connector[(1)].

Specifications and further information as with the RTC6 PCIe Board, see Chapter 4.6.5 "RS232 Socket Connector", page 86.

The signals are referenced to GND, see Notes, page 961.

(1) The socket connector is labeled ".../UART" (Universal asynchronous receiver/transmitter). Refers to the electronic circuit which generates the data bits and the necessary data frame to be transferred on the RS-232 interface.

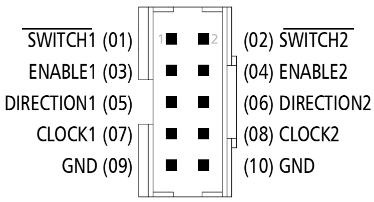
16.2.9 STEPPER Socket Connector

- For technical data see Chapter 16.11 "Technical Specifications – RTC6 Ethernet Board", STEPPER Socket Connector, page 1043.

The STEPPER Socket Connector has 10 pins. It is located on the upper side of the RTC6 Ethernet Board, see Figure 74, number 6.

At the STEPPER Socket Connector signals for controlling up to two stepper motors can be outputted.

The pin-out is shown in Figure 85.



Manufacturer: Amphenol FCI.
Product Number: 98424-F52-10LF.

85

STEPPER Socket Connector: pin-out. The pitch of the pins is 2.00 mm.

All signals are referenced to GND, see Notes, page 961.

Specifications and further information as with the RTC6 PCIe Board, Chapter 4.6.7 "STEPPER MOTOR Socket Connector", page 90.

Notes

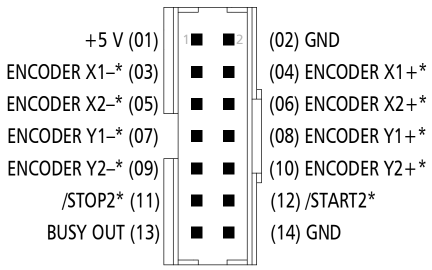
- Required parts for the mating connector are *not included in the scope of delivery*
  Manufacturer: Amphenol FCI
  Series. Minitek®
  1× Housing, part number 90311-010LF
  10× Contact, part number 77138-101LF

16.2.10 MOF Socket Connector

- For technical data see Chapter 16.11 "Technical Specifications – RTC6 Ethernet Board", MOF Socket Connector, page 1042.

The MOF Socket Connector[1] has 14 pins. It is located on the upper side of the RTC6 Ethernet Board, see Figure 74, number 7.

The pin-out is shown in Figure 86.



* Signals are a purchasable option.

Manufacturer: Amphenol FCI.
Product Number: 98424-F52-14LF.

86

MOF Socket Connector: pin-out. The pitch of the pins is 2.00 mm.

Notes

- Required parts for the mating connector are *not included in the scope of delivery*:
  Manufacturer: Amphenol FCI
  Series. Minitek®
  1× Housing, part number 90311-014LF
  14× Contact, part number 77138-101LF.
- RTC6 Ethernet Boards *do not* have a pin for the ANALOG OUT2 signal at their MOF Socket Connector, see Chapter 4.6.4 "MARKING ON THE FLY Socket Connector", page 85.

Encoder Input Ports

As with RTC6 PCIe Board, see Section "Encoder Input Ports", page 85.

External Control Signals

As with RTC6 PCIe Board, see Section "External Control Signals", page 85.

The signals are referenced to GND, see Notes, page 961.

BUSY OUT Status

The BUSY OUT signal at pin (13) is identical to the BUSY OUT signal at the LASER Socket Connector, page 961.
The signal is referenced to GND, see Notes, page 961.

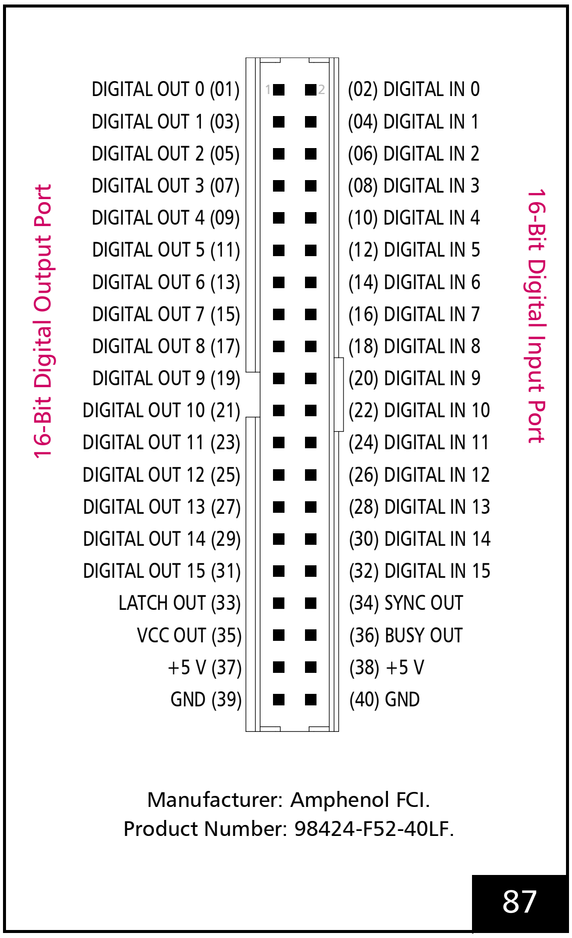
(1) Marking on the Fly (= Processing-on-the-fly).

16.2.11 EXTENSION 1 Socket Connector

- For technical data see Chapter 16.11 "Technical Specifications – RTC6 Ethernet Board", EXTENSION 1 Socket Connector, page 1041.

The EXTENSION 1 Socket Connector has 40 pins. It is located on the upper side of the RTC6 Ethernet Board, see Figure 74, number 9.

The pin-out is shown in Figure 87.



EXTENSION 1 Socket Connector: pin-out. The pitch of the pins is 2.00 mm.

Notes

- Required parts for the mating connector are *not included in the scope of delivery*:
  Manufacturer: Amphenol FCI
  Series. Minitek®
  1× Housing, part number 90311-040LF
  40× Contact, part number 77138-101LF.
- The EXTENSION 1 Socket Connector on RTC6 Ethernet Boards has a 2.00 mm pitch of the pins, whereas RTC6 PCIe Boards have 2.54 mm.

Configuring the Output Signal Level

With the solder jumper field A on the lower side of the RTC6 Ethernet Board, the level of all output signals at the EXTENSION 1 Socket Connector (DIGITAL OUT 0…DIGITAL OUT 15, LATCH_OUT, SYNC_OUT, BUSY OUT, VCC_OUT) can be configured for 5 V or 3.3 V, see Section "Solder Jumper Field A – Configuring Output Signal Level at EXTENSION 1 Socket Connector", page 974.

The configured signal level is stationary outputted at pin (35): signal VCC_OUT. VCC_OUT is referenced to GND, see Notes, page 961.
The maximum current load of the signal is 100 mA.

16-Bit Digital Output Port and 16-Bit Digital Input Port

As with RTC6 PCIe Board, see Section "16-Bit Digital Input Port and 16-Bit Digital Output Port", page 81.

The signals are referenced to GND, see Notes, page 961.

Synchronization of Data Acquisition

As with RTC6 PCIe Board, see Section "Synchronization of Data Acquisition", page 82.

The signals are referenced to GND, see Notes, page 961.

BUSY List Execution Status

The BUSY OUT signal at pin (36) is identical to the BUSY OUT signal at the LASER Socket Connector, page 961.
The signal is referenced to GND, see Notes, page 961.

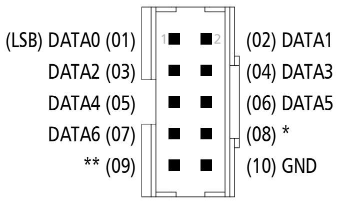
16.2.12 EXT. 2 Socket Connector

- For technical data see Chapter 16.11 "Technical Specifications – RTC6 Ethernet Board", EXT. 2 Socket Connector, page 1042.

The EXT. 2 Socket Connector has 10 pins. It is located on the upper side of the RTC6 PCIe Board, see Figure 74, number 8.

It provides a buffered 8-Bit Digital Output Port: DATA0…DATA7.

The pin-out is shown in Figure 24.



EXT. 2 Socket Connector: pin-out. The pitch of the pins is 2.00 mm.

On RTC6 PCIe Boards the designation is EXTENSION 2.

Apart from the total number of pins, the provided signals differ as well as the pin numbers, see following table.

Signal	RTC6 Ethernet Boards, EXT. 2 Socket Connector 10 pins	RTC6 PCIe Boards, EXTENSION 2 Socket Connector, 26 pins
DATA0	Pin (01)	Pin (01)
DATA1	Pin (02)	Pin (03)
DATA2	Pin (03)	Pin (05)
DATA3	Pin (04)	Pin (07)
DATA4	Pin (05)	Pin (09)
DATA5	Pin (06)	Pin (11)
DATA6	Pin (07)	Pin (13)
*, see in Figure 24	Pin (08)	Pin (15)
**, see in Figure 24	Pin (09)	Pin (17)
GND	Pin (10)	Pin (02)
+5 V	–	Pin (06), (18), (25)
LASER1	–	Pin (22)
LASER2	–	Pin (19)
GND2	–	Pin (23)

Notes

- Required parts for the mating connector are *not included in the scope of delivery*:
  Manufacturer: Amphenol FCI
  Series. Minitek®
  1× Housing, part number 90311-010LF
  10× Contact, part number 77138-101LF.
- The pin (08) and pin (09) are configurable by solder jumpers.
- An RTC6 Ethernet Board *does not* have pins for LASER1 and LASER2 at its EXT. 2 Socket Connector.

Configuration by Solder Jumpers

The pin (08) of the EXT. 2 Socket Connector is configured by the solder jumper field C whereas pin (09) is configured by solder jumper field B. Both jumper fields are on the lower side of the RTC6 Ethernet Board, see Figure 75. For further information, see Section "Solder Jumper Field B – Configuring pin (09) of EXT. 2 Socket Connector", page 975 and Section "Solder Jumper Field C – Configuring pin (08) of EXT. 2 Socket Connector", page 976.

Notes

- If the DATA7 bit (DATA7) is assigned to pin (08), then the full 8-bit output value is available at the output port (pins (1) to pin (08) of the EXT. 2 Socket Connector).
- If pin (08) is set to +5 V (HIGH level), an offset of 128 results for the output value. That is, the output value range is from 128…255.
- If pin (08) ist set to GND (LOW level) the output value range is restricted to 0…127.
- The DATA7 bit can be used for other purposes by assigning it to pin (09).

8-Bit Digital Output Port

As with RTC6 PCIe Board, see Section "8-Bit Digital Output Port", page 84.

16.2.13 Master Socket Connector, Slave Socket Connector

The Master socket connector has 6 pins, see Figure 89. It is located on the upper side of the RTC6 Ethernet Board, see Figure 74, number 10.

The Slave socket connector has 6 pins, see Figure 89. It is located on the lower side of the RTC6 Ethernet Board, see Figure 74, number 16.

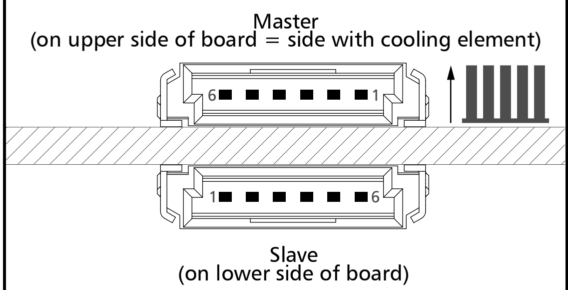
The both socket connectors serve to synchronize the clock cycles of several RTC6 boards.

Connection cables are available from SCANLAB, see Figure 9, page 68. The necessary information for assembling your own cables is shown in Figure 8, page 68.

For further information, see

- Chapter 6.6.3 "Master/Slave Operation", page 127
- Chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 311.

A specific hardware revision of the RTC6 Ethernet Board is required for proper functioning of the master/slave clock synchronization (see label with part number and modification index). If you have an older RTC6 Ethernet Board or are unsure, contact SCANLAB.



Manufacurer: ERNI.
Artikle number: 214013 (2×).

89

Master socket connector and Slave socket connector.
The pitch of the pins is 1.27 mm.

Notice!

- Master/Slave-connected RTC6 Ethernet Boards are destroyed by different potentials. This may even be the case, if the boards are connected to the same power supply but the cabling is unfavourable. Refer to Figure 90 for the recommended wiring.
  To avoid different potentials:
  (1) Make sure that the power supply is switched off.
  (2) First connect one RTC6 Ethernet Board to the power supply, and then connect the other one through a branch of this cable (keep the length as short as possible).
  (5) Only then switch the power supply back on again.
- Always switch off the power supply of all Master/Slave-connected RTC6 Ethernet Boards before disconnecting the cabling from them.

16.2.14 Jumper Settings

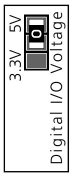
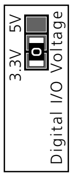
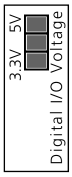
SCANLAB ships RTC6 Ethernet Boards in various jumper configurations. The jumpers can be reconfigured at a later time. See Chapter 2.7 "Jumper Settings and Type Designations", page 41.

Notice!

- Only configure allowed jumper settings. Otherwise, the board gets damaged!

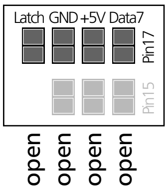
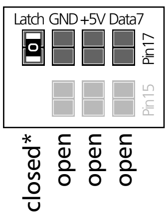
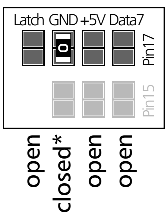
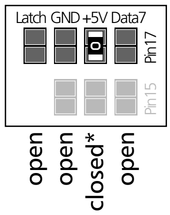
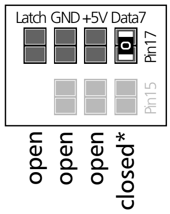
Solder Jumper Field A – Configuring Output Signal Level at EXTENSION 1 Socket Connector

- Position on the board: lower side, see Figure 75, number 12.
- Purpose: to configure the level (5 V or 3.3 V) of all output signals at the EXTENSION 1 Socket Connector, see the following table.
- See also Section "Configuring the Output Signal Level", page 81.

Allowed jumper setting	At the EXTENSION 1 Socket Connector
 closed* open	Output signal level 5 V.
 open closed*	Output signal level 3.3 V.
 open open	No signal output.
*Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!	

Solder Jumper Field B – Configuring pin (09) of EXT. 2 Socket Connector

- Position on the board: lower side, see Figure 75, number 13.
- Purpose: to configure the signal at pin (09) of the EXT. 2 Socket Connector, see the following table.
- See also “Configuration by Solder Jumpers”, page 83.
- Configurations of solder jumper field B and solder jumper field C are independent from each other.
- On the RTC6 Ethernet Board the printed label of solder jumper field B is ‘Pin 17’. This has been chosen deliberately in order to keep consistency with already existing RTC boards. Even though pin (09) is actually configured.

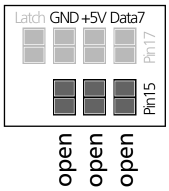
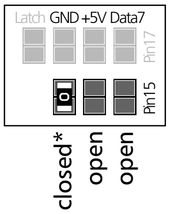
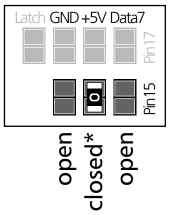
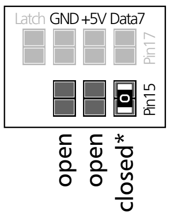
Allowed jumper setting	Output at the EXT. 2 Socket Connector pin (09)
	No signal.
	LATCH signal.
	GROUND (low level).
	+5 V (high level).
	DATA7[a].

*Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!

(a) Synonym: Data Bit #7. MSB of the 8-bit output value.

Solder Jumper Field C – Configuring pin (08) of EXT. 2 Socket Connector

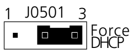
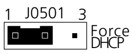
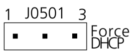
- Position on the board: lower side, see Figure 75, number 14.
- Purpose: to configure the signal at pin (08) of the EXT. 2 Socket Connector, see the following table.
- See also “Configuration by Solder Jumpers”, page 83.
- Configurations of solder jumper field C and solder jumper field B are independent from each other.
- On the RTC6 Ethernet Board the printed label of solder jumper field C is ’Pin 15’. This has been chosen deliberately in order to keep consistency with already existing RTC boards. Even though pin (08) is actually configured.

Allowed jumper setting	**Output at the EXT. 2 Socket Connector** pin (08)
	No signal.
	GROUND (low level).
	+5 V (high level).
	DATA7[a].
*Caution: make sure that *only one* position is closed in this solder jumper field. Other combinations are not allowed and cause damage to the board!	

(a) Synonym: Data Bit #7. MSB of the 8-bit output value.

Jumper Field 'Force DHCP'

- Position on the board: lower side, see Figure 74, number 11.
- Purpose: see the following table.

Allowed jumper setting	**Effect**
 open closed	When the RTC6 Ethernet Board is switched on, an IP address is to be obtained by DHCP:(a) "Force DHCP position" For all configurable network parameters (static IP, pertaining net mask, gateway, UDP and TCP ports), the default settings are used (instead the ones which are saved on the board).
 closed open*	If user-defined network settings are saved, these are used when switching on the RTC6 Ethernet Board.
 open open	Same as "open closed".
*Position as delivered by the factory.	

(a) See also page 979.

16.2.15Real-Time Clock

The RTC6 Ethernet Board features a real-time clock. If the board is without power, the clock continues to run for about > 1 week. With a Hardware Reset, this time is automatically adopted (no **time_update** required).

By **time_update**, the real-time clock is automatically synchronized with the PC time. The accuracy can be fine-tuned by **time_control_eth**.

16.3 Installation and Operation

16.3.1 Hardware Installation

Notice!

- Store the board in an electrostatically neutral environment using the supplied anti-static bag.
- Observe ESD precautions when installing the board.
- Do not touch the electrical contacts of the board.
- Protect the board from moisture, dust, corrosive vapors and mechanical loads.

Proceed as follows to install the RTC6 Ethernet Board in a shielded housing:

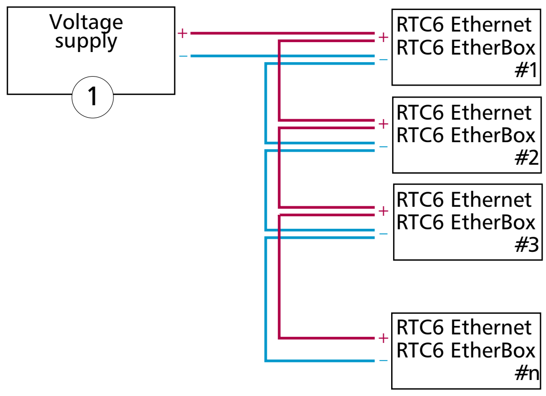
(1) Remove the RTC6 Ethernet Board from its anti-static bag. Do not touch the electrical contacts of the board.

(2) Mount the RTC6 Ethernet Board at the intended location in your shielded housing.

(3) Connect the RTC6 Ethernet Board to:
- your power supply using a power cable
  – Also observe Figure 90 –
- your Ethernet port using an Ethernet cable

(4) Connect the RTC6 Ethernet Board using appropriate cables to:
- the scan head
- the laser

(5) If you want to use the signals at the RTC6 Ethernet Board socket connectors, then attach appropriate cables.



Legend

1. For RTC6 Ethernet Board +11 V…+50 V
   For RTC6 EtherBox +12 V…+30 V

Notice!

- For voltage supply:
  – Use "daisy chain" topology
  – Do not use star topology
- Applies regardless of whether the devices are master/slave connected or not.

90

Recommended cabling for connecting several RTC6 Ethernet Boards/RTC6 EtherBoxes.

16.3.2 Software Installation

For installing and starting-up an RTC6 Ethernet Board and its software proceed analogously as with RTC6 PCIe Boards, Chapter 5 "Installation and Start-Up", page 91. However, step 3, page 91 does not apply:

RTC6 Ethernet Boards do not require the RTC6 board driver.

16.3.3 Connecting to a Network

Connect the RTC6 Ethernet Board with the desired network segment and switch on the supply voltage. The green LED at the ETH connector, see Chapter 16.2.7 "ETH Connector", page 966, should now light up and possibly blink.

If no static IP address is stored on the RTC6 Ethernet Board, a DHCP server is necessary to obtain a dynamic IP address. In this case the yellow LED should light up after a few seconds. This indicates a successful receipt of an IP address.

If in "Force DHCP position" (as of BIOS 24) no IP address has been assigned within 60 seconds after Power-On, then the link-local address 169.254.1.0/16 is automatically used.

If a static IP address is stored on the RTC6 Ethernet Board, the yellow LED immediately lights up after switching on the RTC6 Ethernet Board.

Important: it is possible that the IP configuration stored on the RTC6 Ethernet Board is invalid for the connected network segment. In this case the yellow LED lights up as well. However, the RTC6 Ethernet Board is unresponsive in the network.

16.4 Notes on Migrating Existing and Programming New RTC6 User Programs

RTC6 Ethernet Boards must be registered manually to the RTC6 board management, see Chapter 16.5.3 "About the RTC6 Board Management", page 985 (**init_rtc6_dll** only searches RTC6 PCIe Boards).

After that, RTC6 Ethernet Boards can used generally in the same way as RTC6 PCIe Boards.

In principle, communication via Ethernet is more unsteady compared to a PCIe bus connection within a PC. Therefore, further attention should be paid to potentially occurring errors and to its handling than with RTC6 PCIe Boards (see **get_last_error**).

16.4.1 Finding RTC6 Ethernet Boards in the Network and Querying their Properties

Case 1: IP address of the RTC6 Ethernet Board is known

```
card_no = eth_assign_card_ip(ip, 0); // 0: assign to first free number
Result = select_rtc(card_no);         // If Result equals CardNo: Success
…
```

Case 2: IP address of the RTC6 Ethernet Board is not known

```
result = eth_search_cards(eth_convert_string_to_ip("192.168.250.1"),
             eth_convert_string_to_ip("255.255.255.0"));
if (result != 0)
{
   card_no = eth_assign_card(1, 0); // take first found card, assign to first free number
   result = select_rtc(card_no);
   if (result == card_no)
   {
            ip = eth_get_ip(); // now IP address is known
            .… // do anything
   }
}
```

16.4.2 Example Code (C++): Initialization Covering RTC6 Ethernet Boards

```
//  Abstract
//      A console application to demonstrate how to initialize RTC6 boards
//
//  Comment
//      This application demonstrates how to properly intialize RTC6 boards.
//      It will enumerate all available PCIe boards as well as search for Ethernet boards in a given subnet.
//
//  Platform
//      Win32 - 32-bit Windows
//      x64   - 64-bit Windows
//
//  Necessary Files
//      RTC6impl.h
//      RTC6DLL.dll for Win32 or RTC6DLLx64.dll for x64
//      RTC6DLL.lib for Win32 or RTC6DLLx64.lib for x64
//      RTC6DAT.dat
//      RTC6OUT.out for PCIe boards and/or RTC6ETH.out for Ethernet boards
//      RTC6RBF.rbf
//      Cor_1to1.ct5 or any other correction file
//
//  Compiler
//      - tested with Microsoft C++ Compiler 19.00.24215.1.

#include "RTC6impl.hpp"

#include <iostream>
#include <array>
#include <string>
#include <conio.h>

using namespace std;

// RTC error codes
const UINT ERROR_NO_ERROR = 0U;
const UINT ERROR_NO_CARD = 1U;
const UINT ERROR_VERSION_MISMATCH = 256U;

// Use current working directory as path
const char* PROGRAM_FILE_PATH = nullptr;
// Use Cor_1to1.ct5 in current working directory as correction file
const char* CORRECTION_FILE_PATH = "./Cor_1to1.ct5";

// Subnet to use for Ethernet board search
const char* ETH_SEARCH_IP = "192.168.250.253";
const char* ETH_SEARCH_NETMASK = "255.255.255.0";


bool LoadProgramAndCorrectionFile(UINT card)
{
   // DAT, ETH/OUT and RBF need to be in the current working directory
   const auto loadProgram = n_load_program_file(card, PROGRAM_FILE_PATH);
   if (loadProgram != ERROR_NO_ERROR)
   {
      cout << "n_load_program_file for card " << card << " failed with error code: " << loadProgram << endl;
      return false;
   }

   // Acquire board for further access
   const auto acquire = acquire_rtc(card);
   if (acquire != card)
   {
      cout << "acquire_rtc for card " << card << " failed with error code: " << acquire << endl;
      return false;
   }
```

```
   // Load 2D correction file as table 1
   const auto loadCorrection = n_load_correction_file(card, CORRECTION_FILE_PATH, 1, 2);
   if (loadCorrection != ERROR_NO_ERROR)
   {
      cout << "n_load_correction_file for card " << card << " failed with error code: " << loadCorrection << endl;
      return false;
   }

   // select_cor_table( 1, 0 ); is done internally. Call select_cor_table, if you want a different setting.

   const auto serialNumber = n_get_serial_number(card);
   cout << "Initialized card " << card << " (SN " << serialNumber << ")" << endl;

   return true;
}

bool InitializeRTC()
{
   // Initialize DLL
   const auto initDLL = init_rtc6_dll();
   if (initDLL != ERROR_NO_ERROR)
   {
      if (initDLL & ERROR_VERSION_MISMATCH)
      {
         // Version mismatch can happen if a board has been previously used with a different software version.
         // This error can be fixed by loading the current program file on to the board.
      }
      else if (initDLL & ERROR_NO_CARD)
      {
         // The RTC6 DLL will return ERROR_NO_CARD if no PCIe board has been found.
         // We can still use Ethernet boards if available.
      }
      else
      {
         cout << "init_rtc6_dll failed with error code: " << initDLL << endl;
         return false;
      }
   }

   // Initialize PCIe boards
   const auto foundCards = rtc6_count_cards();
   for (auto card = 1; card <= foundCards; card++)
   {
      if (!LoadProgramAndCorrectionFile(card))
      {
         return false;
      }
   }
```

```
   // Search for and initialize Ethernet boards
   const auto ethSearchIP = eth_convert_string_to_ip(ETH_SEARCH_IP);
   const auto ethSearchNetmask = eth_convert_string_to_ip(ETH_SEARCH_NETMASK);
   const auto foundEthCards = eth_search_cards(ethSearchIP, ethSearchNetmask);
   if (foundEthCards > 0)
   {
      for (auto searchCard = 1; searchCard <= foundEthCards; searchCard++)
      {
         array<UINT, 16> cardInfo;
         eth_get_card_info_search(searchCard, (UINT)cardInfo.data());

         const auto serialNumber = cardInfo[1];

         // Param 0 automatically assigns the board to the next free index
         const auto card = eth_assign_card(searchCard, 0);
         if (card <= 0)
         {
            cout << "eth_assign_card for SN " << serialNumber << " failed with error code: " << card;
            return false;
         }

         if (!LoadProgramAndCorrectionFile(card))
         {
            return false;
         }
      }
   }

   // At least one PCIe or Ethernet board available
   return (foundCards > 0) || (foundEthCards > 0);
}

int main(int argc, char* argv[])
{
   if (!InitializeRTC())
   {
      return 1;
   }

   // Use boards for marking etc...

   _getch();
   return 0;
}
```

16.5 RTC6 Ethernet Board Commands and Functions

16.5.1 Notes on Working with IP Addresses

With the RTC6 commands, all IP addresses (always IPv4, IPv6 is not supported) are specified as decimal values in Big Endian format ("Big Endian byte order").

For example, the IP address "192.168.250.1" must be specified as "33204416", see following table.

Format	IP address	Hex value	Decimal value
Little Endian	`192.168.250.1` (a)	`0×C0A8FA01`	`3232299521`
Big Endian	`1.250.168.192`	`0×01FAA8C0`	`33204416`

(a) Usual dotted decimal notation.

eth_convert_ip_to_string converts the IP address in Big Endian byte order to usual dotted decimal notation.

eth_convert_string_to_ip converts the IP address in usual dotted decimal notation to Big Endian byte order.

16.5.2 About Searching for RTC6 Ethernet Boards

To search for all RTC6 Ethernet Boards available in the network, **eth_search_cards** can be used. If the search is to be limited to a certain address range only, then **eth_search_cards_range** is to be used.

The data of all RTC6 Ethernet Boards which have been answered within a configurable timeout (see **eth_set_search_cards_timeout**) are registered to the search result list (see below).

By a card search, RTC6 Ethernet Boards with unknown IP address can be identified in the network (for example, because the have received it dynamically by a DHCP server).

The found RTC6 Ethernet Boards are registered in a temporary list which is the search result list.

Index	Record **in the** search result list
1	Information(a) on RTC6 Ethernet Board 1
2	Information(a) on RTC6 Ethernet Board 2
n	Information(a) on RTC6 Ethernet Board n

(a) IP address, serial number, connection status, etc., see **eth_get_card_info_search**.

For a specified search result list index **eth_get_card_info_search** returns the available information on the RTC6 Ethernet Board, whereas **eth_get_ip_search** returns only the IP address and **eth_get_serial_search** only the serial number.

On the one hand the number of found RTC6 Ethernet Boards is already returned by **eth_search_cards**. On the other hand **eth_found_cards** also returns it at any time as well (without the need to perform the search in the network again).

Several subsequent calls of **eth_search_cards** can deliver different results, depending how many RTC6 Ethernet Boards are available in the network and in which chronological order the answers come in.

16.5.3 About the RTC6 Board Management

RTC6 boards are addressed by a unique number under which they must be entered in the RTC6 board management.

The RTC6 board management is an RTC6 DLL-internal list consisting of 255 possible RTC6 board records, see following table.

Index	RTC6 board **record**
1	RTC6 PCIe Board 1
2	RTC6 PCIe Board 2
3	RTC6 PCIe Board 3
4	"No card"
5	Information on RTC6 Ethernet Board[a]
6	Information on RTC6 Ethernet Board[a]
7	"No card"
…	…
42	Information on RTC6 Ethernet Board[a]
43	"No card"
…	…
255	"No card"

(a) IIP address, serial number, connection status, etc., see eth_get_card_info_search.

At the beginning of the list are RTC6 PCIe Boards, if any are present in the PC.

RTC6 PCIe Boards are automatically searched by **init_rtc6_dll** and consecutively numbered. The numbering cannot be changed. **rtc6_count_cards** only returns the number of RTC6 PCIe Boards found.

RTC6 Ethernet Board, on the other hand, must be entered manually into the RTC6 board management:

- If the IP address is known, **eth_assign_card_ip** can be used to enter a card at any index between **rtc6_count_cards** + 1…255.
- If the IP address is unknown, by **eth_assign_card** an RTC6 Ethernet Board from the search result list, see Chapter 16.5.2 "About Searching for RTC6 Ethernet Boards", page 984, can be entered at an arbitrary index. However, neither an RTC6 PCIe Board nor an RTC6 Ethernet Board must be registered at this index

eth_max_card returns the highest index where an RTC6 Ethernet Board in the RTC6 board management is registered.

In contrast, **eth_count_cards** returns the total number of entered RTC6 Ethernet Boards.

By **get_card_type** the registered board type can be queried:

- 0 = "No card"
- 1 = RTC6 PCIe Board
- 2 = RTC6 Ethernet Board

16.5.4 Checking the Connection to the RTC6 Ethernet Board

In general, communication via Ethernet is more unreliable than PCIe bus connections within PCs (simple example: Ethernet cable is not plugged in). By **eth_check_connection** it can be simply checked, whether the RTC6 Ethernet Board responds and therefore, the Ethernet connection still exists.

The behavior of the RTC6 Ethernet Board in case of Ethernet Link Loss can be set with **eth_configure_link_loss**. In `Mode` = 0…2, among other things, Bit #15 of **get_startstop_info** is set with an Ethernet Link Loss (detection time: < 100 ms). Bit #15 is set back to 0 not until **get_startstop_info** is called.

16.5.5 Command Set for the RTC6 Ethernet Board

- See Chapter "Control Commands for RTC6 Ethernet Boards", page 328.

16.6 Safe Startup and Shutdown Sequences

To assure safety during startup, switch on the components of the laser system in the following order:

(1) Switch on the network PC.

(2) Switch on the power supply for the RTC6 Ethernet Board.

(3) Start the control software.

(4) Switch on any required peripheral devices.

(5) Switch on the power supply for the scan system.

(6) Switch on the laser.

To assure safety during shutdown, switch off the components of the laser system in exactly the reverse order:

(1) Switch off the laser.

(2) Switch off the power supply for the scan system.

(3) Switch off the peripheral devices.

(4) Terminate the control software.

(5) Switch off the power supply for the RTC6 Ethernet Board.

(6) Shut down the network PC.

Caution!

- When the PC switches on or off, the RTC6 Ethernet Board board output ports levels might briefly fluctuate, resulting in unintended changes to Laser Control Signals. The above-mentioned startup and shutdown sequences must therefore be strictly followed. Otherwise, the laser might briefly, unexpectedly and dangerously switch on.
- Always start up the PC and control software prior to turning on the scan system. And switch the scan system back off prior to shutting down the control software and PC. Otherwise, unintended scan system motions might occur. The laser must always be switched on last and switched off first. Otherwise, there is the risk that the laser beam might be deflected in an arbitrary direction.

16.7 Standalone Functionality

- In contrast to PC operation, the aim of the Standalone Operation Mode is that the RTC6 Ethernet Board operates independently without a connected PC. Among other things, for this purpose, the list commands to be processed, all correction tables to be used and the control commands to be executed during automatic booting[1] must be stored in the NAND Memory.

Notice!

- The following requirements must be met for Standalone Operation Mode:
  - **(1)** At least BIOS version 0x26 (BIOS-ETH 26) is installed[a] on the RTC6 Ethernet Board. See Chapter 16.7.1 "Upgrading BIOS-ETH", page 989.
  - **(2)** The user program uses RTC6 Software Package ≥ 1.7.0, that is, a combination of
    - ≥ DLL 618
    - ≥ ETH 618
    - ≥ RBF 623
    - ≥ DAT 603

(a) See `RTC6conf.exe` and `RTC6BIOSETH_42.out`.

- Then the RTC6 Ethernet Board can be configured to boot automatically[1] (depending on the data stored in NAND Memory by **store_program**(`Mode`) after a Hardware Reset. After that, it is in one of the following states:
  - "Normal PC Operation State"
  - "Standalone Basic State"
  - "Standalone Full State"
- The "Normal PC Operation State" is achieved by:
  - BIOS < BIOS-ETH 26
  - ≥ BIOS-ETH 26 and **set_eth_boot_control**(0)
  - ≥ BIOS-ETH 26 and **store_program**(1)

  In these cases, **load_program_file** must be called for further operation.
- The "Standalone Basic State" is achieved by:
  - BIOS ≥ BIOS-ETH 26, **store_program**(0) and **set_eth_boot_control**(1)

  A **load_program_file** call is not required. See also Chapter 16.7.4 "Boot Image", page 991. The procedure for "Standalone Basic State" is described in Chapter 16.7.2 "Preparing the "Standalone Basic State"", page 989.
- The "Standalone Full State" is achieved by:
  - BIOS ≥ BIOS-ETH 26, **store_program**(2) and **set_eth_boot_control**(1)

  Compared to the "Standalone Basic State" it is also no longer necessary to load correction files. The same applies to loading of the list commands which have been stored upon **store_program**(2). Furthermore, control commands for configuration[2] can be executed automatically as required. See also Chapter 16.7.4 "Boot Image", page 991. The procedure for "Standalone Full State" is described in Chapter 16.7.3 "Preparing the "Standalone Full State"", page 990. In "Standalone Full State", the RTC6 Ethernet Board can process its list independently and without a connected PC after an /START.
- The NAND Memory contents can be written to the PC as a so-called "Boot Image" by **read_image_eth** as a binary file and copied to any number of RTC6 Ethernet Boards by **write_image_eth**, see Chapter 16.7.4 "Boot Image", page 991.
- Details on automatic booting can be found in Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994.
- RTC4 *SCAN*alone Board users must observe the safety notice on the encoder counter direction, see Notice!, page 59.

(1) In this Chapter, "automatic booting" means that in addition to the actual booting, additional data is read out from the NAND Memory.

(2) See Chapter 16.7.5 "Control Commands Allowed for Automatic Booting", page 992.

16.7.1 Upgrading BIOS-ETH

Proceed as follows to upgrade the BIOS of an RTC6 Ethernet Board with ≤ BIOS-ETH 26 to BIOS-ETH 42 only (no subsequent Standalone Operation Mode):

(1) Carry out a Hardware Reset.

(2) Call **load_program_file** by specifying the relevant data from RTC6 Software Package ≥ V1.7.0.

(3) Call **store_program**(`Mode` = 1).
The NAND Memory content is erased.

(4) Call **set_eth_boot_control**(0).
Thus, the board does *not* boot automatically[1] after a Hardware Reset (no matter what is stored in NAND Memory).

(5) Run `RTC6conf.exe`[2] and install `RTC6BIOSETH_42.out` via **FLASH BIOS** button.
Thus, the upgrade is completed.

(6) Carry out a Hardware Reset.
- The RTC6 Ethernet Board *does not* boot automatically[1]
- Is now ready for PC operation

16.7.2 Preparing the "Standalone Basic State"

Proceed as follows to put the board to "Standalone Basic State":

(1) Prerequisite: Chapter 16.7.1 "Upgrading BIOS-ETH", page 989 has been carried out.

(2) Call **load_program_file** by specifying the relevant data from RTC6 Software Package ≥ V1.7.0.

(3) Call **store_program**(`Mode`= 0).
Data for the "Standalone Basic State", see page 872, is saved to the NAND Memory.

(4) Call **set_eth_boot_control**(`Ctrl` = 1).
Thus, the board boots automatically after a Hardware Reset[1][3].

(5) Carry out a Hardware Reset.
- The RTC6 Ethernet Board boots automatically[1]
- Is now ready for PC operation
- Subsequently, you do *not* need to call **load_program_file**

Notes

- The actual RTC6 files no longer need to be provided in the user program. An accidentally wrong RTC6 DLL version is automatically detected, because a wrong RTC6 DLL leads to a **get_last_error** return code `RTC6_VERSION_MISMATCH` (whereas a **load_program_file** does not, as long as RTC6 files are compatible with the RTC6 DLL).

(1) See Footnote, page 988.

(2) See also `RTC6conf.pdf`.

(3) This is also the default setting with "new" (= **set_eth_boot_control**(`Ctrl` = 0) has never been executed) RTC6 Ethernet Boards.

16.7.3 Preparing the "Standalone Full State"

Proceed as follows to put the board to "Standalone Full State":

(1) Prerequisite: Chapter 16.7.2 "Preparing the "Standalone Basic State"", page 989 has been carried out.

(2) Verify that the RTC6 Ethernet Board works properly in PC operation.

(3) Work out a suitable sequence of control commands for automatic booting(1).
- The allowed control commands are listed in Chapter 16.7.5 "Control Commands Allowed for Automatic Booting", page 992, separated according to Boot Phase 1 and Boot Phase 2.
- Call **eth_boot_dcmd** before each of these commands.
- Pay attention to the correct order, for example,
  - **config_list** should be called before loading list commands
  - if **set_scanahead_params** is used, a SCANahead System must already be connected and switched on
  - if an "Automatic Laser Control" is used, it may only be activated after **set_scanahead_params**

(4) Test the result of step 3:
execute it in PC operation in exactly the same way as it is to be executed later in Standalone Operation Mode.

(5) Call **load_program_file** in order to initialize the RTC6 Ethernet Board.

(6) Load all required control commands, list commands and correction files.

(7) Emergency provision in case an error occurs during saving: call **set_eth_boot_control**(0). This prevents automatic booting.

(8) Do one of the following:
- For Remote Interface Mode only – call **store_program**(`Mode` = 3).
- Otherwise – call **store_program**(`Mode` = 2).

(9) If an error occurred:
Repeat step 8.

(10) If the process has been completed without errors: Call **set_eth_boot_control**(`Ctrl` = 1). This activates automatic booting.

(11) Carry out a Hardware Reset.
- The RTC6 Ethernet Board is now in "Standalone Full State"
- See also Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994

Notes

- In "Standalone Full State" the RTC6 Ethernet Board is already configured with the control commands for automatic booting(2) and provided with all correction tables. If list commands have been stored, they are ready to be executed automatically after an /START.

(1) See Footnote, page 988.

(2) See Chapter 16.7.5 "Control Commands Allowed for Automatic Booting", page 992.

16.7.4 Boot Image

- Prerequisites: see page 988.

The data in the NAND Memory can be written to the PC as "boot image" and copied from there to any RTC6 Ethernet Boards (for example, of several production machines):

- **read_image_eth**,
  see Creating a Boot Image on the PC
- **write_image_eth**,
  see Copying Boot Image to Board(s)

The properties of boot images are shown in Table 7 (RTC6 Software Package V1.7.0, approximate values):

Table 7: Properties of Boot Images

Boot Image	File size	Reading duration[a]	Writing duration [b]
for "Standalone Basic State"	2 MB	a few seconds	a few seconds
for "Standalone Full State"	160 MB … 264 MB	up to 2 minutes	up to 2 minutes

(a) With **read_image_eth**.

(b) With **write_image_eth**.

Creating a Boot Image on the PC

- Prerequisites: see page 988.

read_image_eth(`Name`) read out the contents of the NAND Memory and writes it in binary to the "`Name`" file on the PC. The user program must have write permission for this file. For reading duration, see Table 7.
See also Section "Procedure after an Transmission Abortion", page 991.

Copying Boot Image to Board(s)

- Prerequisites: see page 988.

write_image_eth(`Name`) reads the file "`Name`" and writes its content to the NAND Memory. The user program must have read permission for this file. For writing duration, see Table 7.

(1) Call **set_eth_boot_control**(0).
Thus, the board *does not* boot automatically after a Hardware Reset (recommended as automatic booting might be faulty).

(2) Call **write_image_eth**(`Name`) and do not interrupt the process! Otherwise, observe Section "Procedure after an Transmission Abortion", page 991.

(3) If the process went without errors, call **set_eth_boot_control**(1). Otherwise, repeat step 2.

(4) Carry out a Hardware Reset.
Provided the same options are enabled: This RTC6 Ethernet Board boots exactly the same way as the RTC6 Ethernet Board from which the boot image has been taken.

Procedure after an Transmission Abortion

If the transmission is aborted, for example, due to an Ethernet connection interruption, the RTC6 Ethernet Board probably remains in an INTERNAL-BUSY list execution status state. The aborted process is not continued, even if the connection is re-established:

(1) Call either **stop_execution** or execute a /STOP in order to release the board from this INTERNAL-BUSY list execution status state.

(2) Call **read_image_eth**(`Name`) (once again).
Thus, the board does *not* boot automatically after a Hardware Reset.

(3) Carry out a Hardware Reset.

(4) Call **load_program_file**.

(5) If you
- create a boot image on the PC:
  call **read_image_eth**(`Name`) again.
- copy a boot image to the board:
  call **write_image_eth**(`Name`).

16.7.5 Control Commands Allowed for Automatic Booting

- Only certain control commands are allowed in each boot phase, see table 8 and Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994:
  - Boot Phase 1
  - Boot Phase 2
- They meet one or more of the following criteria:
  - Requires no communication with the PC
  - Does not send a response to the PC
  - There is no corresponding list command
- Exceptions are, for example,
  - **set_free_variable**, see Comments there
  - **set_auto_laser_control** or **set_scanahead_params**
    - They check the connected scan systems at point in time of the call
    - They save exclusively the parameters to be finally set only in case of success
  - **set_jump_mode** only sets the tuning numbers `VA1`...`JB2` during automatic booting and does not check whether they match the scan system.
  - The table containing the Jump Delay values `[JumpTable<No>]`, see page 228 and **load_jump_table_offset**, must have been previously saved via **create_dat_file**.
- Upon (later) automatic booting, the connected scan systems are *not* checked anymore!
- Boot Phase 2-control commands are control commands which:
  - Necessarily need to be executed after a Boot Phase 1-control command
  - Require functioning peripherals, for example, an i*DRIVE* scan system(1) with "Automatic Laser Control". Important: users must make sure themselves that the periphery is actually working at runtime

(1) See Glossary entry on page 28.

Table 8: Control commands allowed for automatic booting (alphabetically)

Boot Phase 1

bounce_supp
config_laser_signals
config_list
eth_boot_timeout
eth_set_remote_tgm_format
home_position
home_position_xyz
mcbsp_init
set_control_mode
set_controlpreview_compensation_ctrl
set_extstartpos
set_fly_tracking_error
set_free_variable
set_jump_mode
set_laser_control
set_laser_mode
set_matrix(`HeadNo` = 4)
set_max_counts
set_mcbsp_freq
set_mcbsp_out_ptr
set_offset_xyz(`HeadNo` = 4)
set_pulse_picking_length
set_rot_center
set_scanahead_params
set_scanahead_speed_control
set_timelag_compensation
simulate_encoder
uart_config

Boot Phase 2

control_command
init_fly_2d
select_cor_table
set_auto_laser_control
set_dsp_mode
set_hi
start_loop

16.7.6 Automatic Booting – Process in Detail

(1) After a Hardware Reset, the RTC6 Ethernet Board tries to read data for automatic booting from the NAND Memory.(1)
During this time the yellow LED flashes dimmed with approx. 1 Hz until the end of the initialization.

- Case 1: There are *no* data present (after **store_program**(Mode= 1)) or **set_eth_boot_control**(0) has been carried out.
  The initialization is completed without these data. Afterwards, the RTC6 Ethernet Board is in "Normal PC Operation State", page 988.
- Case 2: There are only data for "Standalone Basic State", see page 872 (after **store_program**(Mode= 0)). The RTC6 Ethernet Board is initialized (as by **load_program_file**). Afterwards, it is immediately ready for PC operation = "Standalone Basic State".
- Case 3: There are also data for "Standalone Full State", see page 872 (after **store_program**(Mode= 2)). Then step 2 is executed.

(2) The RTC6 Ethernet Board

- switches on the BUSY Pin
- reads the "remaining" data for automatic booting to "Standalone Full State", see page 872) from the NAND Memory.(1)
- The subsequent boot steps 3...6 are two-phase.

(3) In Boot Phase 1 (duration: some seconds):

- The correction file(s) are read out
- Boot Phase 1-control commands are executed (in the order in which they have been saved). If such commands are stored more than once, they are executed correspondingly often, with newer data overwriting older data.

(4) After Boot Phase 1:(1)

- Initialization ceases
- The BUSY Pin is switched off
- Do not switch the laser on yet!
- Switch on scan systems and other peripheral devices
- Make sure peripheral devices are ready for operation. Only then, trigger Boot Phase 2 by a /START
- If your peripheral devices are always ready for operation after a certain time, you can specify a waiting time (TimeOut) with the Boot Phase 1-control command **eth_boot_timeout**. After its expiry, initialization continues automatically (= without an explicit /START). With a /START within the waiting time, Boot Phase 2 starts immediately. Outside the waiting time, an /START is ignored as long as Boot Phase 2 is still running (see BUSY Pin!) The /START is automatically enabled (similar to **set_control_mode**(Bit #0 = 1)). Further /STARTs after an /STOP must be enabled by users themselves by **set_control_mode**(Bit #3 = 1).

(1) Status == 1 of **eth_get_standalone_status**

(5) In Boot Phase 2:(1)

- The BUSY Pin is switched on
- The RTC6 List Memory is read out (duration: approx. 1 minute)
- Boot Phase 2-control commands are executed.
- Initialization ceases
- The BUSY Pin is switched off

(6) After Boot Phase 2:

- If an error has occurred(2) (further operation of the board may not be possible), the LED continues to flash at about 4 Hz
- When the initialization is successfully completed(3), the dimmed LED flashing stops. The LED returns to its normal state, see Chapter 16.2.7 "ETH Connector", page 966
- The target state of the RTC6 Ethernet Board indicates **eth_get_standalone_status**(&`Mode` =1…2).
- At this point, it would be a suitable point in time to also switch on the laser (observe the notices on laser safety in Chapter 3.2 "Laser Safety", page 65) as well as other peripherals.
- The RTC6 Ethernet Board waits endlessly for an /START, which starts the processing of the list
- The RTC6 Ethernet Board is ready for PC operation as well

(1) **eth_get_standalone_status**(&`Status` == 1)

(2) **eth_get_standalone_status**(&`Error` = 1)

(3) **eth_get_standalone_status**(&`Status` =0)

16.8 "High Performance Mode"

With RTC6 Ethernet Boards, **eth_set_high_performance_mode** switches on (off) the

- "High Performance Mode"

"High Performance Mode" is only effective with list commands. If it is switched on (`Mode` = 1), the RTC6 DLL no longer waits for a response from the RTC6 Ethernet Board after each telegram.

Instead, the RTC6 DLL now acknowledges several telegrams at a time. Therefore, more list commands per time can be written into the RTC6 List Memory (= "higher list download performance").

The RTC6 DLL always waits until the RTC6 Ethernet Board has acknowledged all previous list commands with:

- Control command calls
- **set_end_of_list**
- **list_return**

To ensure that all list commands are written to the RTC6 List Memory, the list command block should always be terminated by **set_end_of_list** or **list_return** when:

- A list is started with
  - /Slave-START
  - /START
- A list command block is not directly followed by a control command

However, a significantly higher rate can only be achieved, if you:

- Write correspondingly larger list command blocks
- Avoid that by calling a control command the list command blocks are confirmed prematurely

16.9 Remote Interface Mode

Remote Interface Mode serves to control RTC6 Ethernet Boards without RTC6 DLL[1].

For this purpose Command Telegrams are sent via the network (UDP) to the RTC6 Ethernet Board.

A C++ Remote Interface Wrapper that simplifies usage is included in RTC6 Software Package.

In this Chapter:

16.9.1 Prerequisites for Remote Interface Mode

- RTC6 Software Package ≥ V1.16.3
- The RTC6 Ethernet Board is equipped with ≥ BIOS-ETH 40, see also Chapter 16.7.1 "Upgrading BIOS-ETH", page 989.
- The RTC6 Ethernet Board is configured for Standalone Operation Mode in "Standalone Full State", see Chapter 16.7 "Standalone Functionality", page 988.
- In Boot Phase 1, **eth_set_remote_tgm_format** is called.

16.9.2 Restrictions in Remote Interface Mode

- A conventional RTC6 DLL-based connection to the RTC6 Ethernet Board cannot be established. Therefore, RTC6 Commands that require a connection then deliver **get_error** `RTC6_ACCESS_DENIED` or **eth_get_error** not_acquired.
- RTC6 Ethernet Board can only still be searched for:
  - Via RTC6 DLL by **eth_search_cards**
- Most RTC6 Commands are available in a corresponding functionality as Remote Commands, see:
  - Chapter 16.9.9 "Remote Commands", page 1005
  - `rtc6_rif_wrapper.h`
- Some RTC6 Commands are *not* available in a corresponding functionality as Remote Commands, see:
  - Chapter 16.9.10 "RTC6 Commands not Available as Remote Commands", page 1028

(1) Or, without Linux Shared Library, see Subfolder Linux, page 34.

16.9.3 Enabling Remote Interface Mode

(1) Configure the RTC6 Ethernet Board for Standalone Operation Mode in "Standalone Full State", see Chapter 16.7.3 "Preparing the "Standalone Full State"", page 990. Use **store_program**(`Mode` = 3) in step 8.

(2) Make sure that **eth_set_remote_tgm_format** is called in Boot Phase 1, see step 3 in Chapter 16.7.3 "Preparing the "Standalone Full State"", page 990.

(3) After each Hardware Reset, the RTC6 Ethernet Board then boots in "Standalone Full State" and is then automatically set to Remote Interface Mode.

16.9.4 Disabling Remote Interface Mode – Temporarily

The Remote Interface Mode can be disabled temporarily – until the next Hardware Reset.

In this state, the RTC6 Ethernet Board can be controlled via the conventional RTC6 DLL-based(1) connection.

- Do one of the following:
  - Send a Command Telegram that meets the following specification:
    - The *first and only* `uint32_t` value in TGM_PL_CMD_RAW is `0x0D15AB1E` (1)
  - Call **wrapper function disable_remote_interface**
  - Click the corresponding button in `RTC6conf.exe`
  - Interrupt booting into the "Standalone Full State". During this time (about 10 s with **store_program**(`Mode` = 3) in step 8; otherwise, approx. 60 s), call **load_program_file**. This aborts booting. Afterwards, a conventional RTC6 DLL-based connection to the RTC6 Ethernet Board can be established.

(1) If the Remote Interface Mode is activated: the first `uint32_t` value TGM_PL_ANSW_RAW is `0xD15AB1ED`.

16.9.5 Disabling Remote Interface Mode – Permanently

The RTC6 Ethernet Board is reset to "Standalone Full State" when the Remote Interface Mode is permanently disabled.

In this state, the RTC6 Ethernet Board can be controlled via the conventional RTC6 DLL-based[1] connection.

(1) Disable Remote Interface Mode temporarily as described in Chapter 16.9.4 "Disabling Remote Interface Mode – Temporarily", page 998.

(2) Do one of the following:

- Call **eth_set_remote_tgm_format**(0) in Boot Phase 1 (or remove **eth_set_remote_tgm_format**(1) from Boot Phase 1)
- Set up "Standalone Basic State", see Chapter 16.7.2 "Preparing the "Standalone Basic State"", page 989 (`store_program`(0))
- Switch off the Standalone Operation Mode at all, see Chapter 16.7 "Standalone Functionality", page 988 (`store_program`(1) or `set_eth_boot_control`(0)

16.9.6 Telegram Format RAW[(1)]

In this Chapter:

Command Telegram – Structure

(Telegram *to* the RTC6 Ethernet Board)

Command Telegram	
TGM_HEADER	Header data
TGM_PL_CMD_RAW	Payload 1 Remote Control Com mand *OR* 1…n Remote List Comman ds[(a)]
TGM_MAX_SIZE[(b)]	≤ 1.400 byte

(a) The corresponding RTC6 Commands are stored in RTC6 List Memory in this order.

(b) In `telegrams.h`. Allowed length of [TGM_HEADER + TGM_PL_CMD_RAW].

Answer Telegram – Structure

(Telegram *from* the RTC6 Ethernet Board)

Answer Telegram	
TGM_HEADER	Header data
TGM_PL_ANSW_RAW	Payload
TGM_MAX_SIZE[(a)]	≤ 1.400 byte

(a) In `telegrams.h`. Allowed length of [TGM_HEADER + TGM_PL_CMD_RAW].

Notes

- The RTC6 Ethernet Board responses to each received Command Telegram with exactly 1 Answer Telegram.

(1) enum TGM_FORMAT, enumeration constant RAW in `telegrams.h`.

TGM_HEADER

(Header data of Command Telegram or Answer Telegram)

- TGM_HEADER is structure element of:
  - Command Telegram
  - Answer Telegram

Data type (cont'd)	Name (cont'd)	Description (cont'd)
uint32_t (a)	length	Length of TGM_PL_CMD_RAW or TGM_PL_ANSW_RAW. In bytes.
uint32_t	version	Version number. To date: 0x01000000 (TGM_VERSION in telegrams.h). Must fit version stored in BIOS.
uint32_t	seqnum	Sequence number. For flow control. Answer Telegrams always contain the same seqnum value. The RTC6 Ethernet Board only executes Command Telegrams with seqnum > last received seqnum. If seqnum = last received seqnum, then only the last response telegram is sent again. Observe page 1034.
uint32_t	type	Telegram type. 0 TYPE_COMMAND = Command Telegram 1 TYPE_ANSWER = Answer Telegram
uint32_t	format	Telegram format. Must match the value set by **eth_set_remote_tgm_format**.

(a) Synonym: unsigned int. Unsigned 32-bit value: [0…+(23^2–1)].

TGM_PL_CMD_RAW

(Payload of Command Telegram in Telegram Format RAW)

- TGM_PL_CMD_RAW defines the Command Telegram payload data format
- Observe the further information in Command Telegram – Structure, page 1000. et seqq.

Data type	Name	Description
uint32_t	1st ID	ID(a) of 1st Remote Command. *OR:* 0x12345678, see page 1034.
n × uint32_t	Parameter 1st ID	Command Telegram Parameter of 1st ID(b).
uint32_t	2. ID	…
n × uint32_t	Parameter 2. ID	…
…	…	…

(a) IDs of Remote Control Commands, see page 1006 et seqq.
IDs of Remote List Commands, see page 1016 et seqq.

(b) Command Telegram Parameter of Remote Control Commands, see page 1006 et seqq.
Command Telegram Parameter of Remote List Commands, see page 1016 et seqq..

TGM_PL_ANSW_RAW

(Payload of Answer Telegram in Telegram Format RAW)

- TGM_PL_ANSW_RAW defines the Answer Telegram payload data format

Data type	Name	Description
uint32_t	LastError 1st ID	LastError of 1st ID. See **get_last_error**.
uint32_t	1st ID	ID(a) of 1st Remote Commands.
n × uint32_t	Answer 1st ID	Answer Telegram Parameter of 1st ID(b).
uint32_t	LastError 2. ID	…
uint32_t	2. ID	…
n × uint32_t	Answer 2. ID	…
…	…	…

(a) IDs of Remote Control Commands, see page 1006 et seqq.
IDs of Remote List Commands, see page 1016 et seqq.

(b) Answer Telegram Parameter of Remote Control Commands, see page 1006 et seqq.
Answer Telegram Parameter of Remote List Commands, see page 1016 et seqq.

16.9.7 Error Codes for LastError Occurring in Remote Interface Mode Only

get_error Bit #31 (invalid Command Telegram) is set in Remote Interface Mode only.

If the Command Telegram TGM_HEADER is faulty, the 1st ID is not evaluated. Then, LastError 1st ID is the only uint32_t value in TGM_PL_ANSW_RAW.

Several Error Bits can be set at the same time. They can be queried by:

- Remote Control Command 148 R_DC_GET_REMOTE_ERROR
- **wrapper function get_remote_error**

Error Bit	Description	Corresponding Enumeration Constant in enum TGM_ERROR[a]
Bit #0 = 0 … Bit #6 = 0	No error.	ERROR_NO_ERROR
Bit #0 = 1	Invalid size of Command Telegram. Must be: > TGM_HEADER[a] or ≤ TGM_MAX_SIZE[a].	ERROR_TGM_SIZE
Bit #1 = 1	version[a] value in Command Telegram is not compatible with BIOS. Make sure that C++ Remote Interface Wrapper and BIOS are from the same RTC6 Software Package.	ERROR_HEADER_VERSION
Bit #2 = 1	Invalid length[a] value. The received telegram size does not match the size specified in TGM_PL_CMD_RAW.	ERROR_HEADER_LENGTH
Bit #3 = 1	Invalid type[a] value. Must be: 0 with a Command Telegram.	ERROR_HEADER_TYPE
Bit #4 = 1	Invalid format[a] value. Must be: Same value as previously set by **eth_set_remote_tgm_format**.	ERROR_HEADER_FORMAT
Bit #5 = 1	Invalid Remote Command-ID. Must be: Valid Remote Command-ID from enum REMOTE_ID[a].	ERROR_PAYLOAD_UNKNOWN_CMD_ID
Bit #6 = 1	Invalid payload. Must be: 1 Remote Control Command *or* 1…n Remote List Command(s).	ERROR_PAYLOAD_INVALID

(a) See telegrams.h.

Examples

(1) TGM_PL_CMD_RAW contains several Remote Control Commands and therefore, is invalid

– Call

`Payload (R_DC_GOTO_XYZ, …, R_DC_SET_MARK_SPEED, …)`

– Answer

(`LastError` Bit #31, `R_DC_SET_MARK_SPEED`)

– Comment

- `76 R_DC_GOTO_XYZ` is not executed
- `72 R_DC_SET_MARK_SPEED` is not executed
- `148 R_DC_GET_REMOTE_ERROR` returns Bit #6 = 1

(2) TGM_PL_CMD_RAW contains Remote List Commands as well as Remote Control Commands and therefore, is invalid

– Call

`Payload (R_LC_JUMP_XY_ABS, …, R_LC_END_OF_LIST, R_DC_EXECUTE_LIST_POS, …)`

– Answer

(`LastError` Bit #31, `R_DC_EXECUTE_LIST_POS`)

– Comment

- The corresponding RTC6 Commands are not stored in RTC6 List Memory
- `15 R_DC_EXECUTE_LIST_POS` is not executed
- `148 R_DC_GET_REMOTE_ERROR` returns Bit #6 = 1

16.9.8 Sending Command Telegrams and Receiving Answer Telegrams

- Command Telegrams must be sent via UDP to the RTC6 Ethernet Board. The UPD port is the same as for conventional transmission(1).
- The concrete implementation depends on the operating system and the programming language (refer to associated socket APIs) and therefore, cannot be described here. In Folder Demo Files, you can find a code example (C++) for Winsock API and Linux socket API.
- With UDP, lost packets are not automatically detected. In addition, the transmission sequence is not guaranteed. Therefore, you should explicitly wait for and check the corresponding Answer Telegram after each Command Telegram.
- Include as many Remote List Commands as possible in a Command Telegram. This way you reduce the total number of telegrams.
- You have to implement flow control checks (which are normally performed by the RTC6 DLL(2)) manually. Example: The RTC6 List Memory is to be filled and the Answer Telegram gets lost. This does not mean that the RTC6 Commands have not been stored. In this case, query the input pointer position manually. If you want to be sure that the RTC6 Commands have been stored without error: reset the input pointer and send the Command Telegram again.

16.9.9 Remote Commands

- Remote Control Command are:
  – ID 0…ID 255
- Remote List Command are:
  – ID 256…ID 511
- Most Remote Commands have a functional equivalent in exactly one RTC6 command.
- With their parameters and internal handling, some Remote Commands even cover several RTC6 Commands. Example: `ID 312 R_LC_MARK_XYZT_ABS` covers both **mark_abs** and **timed_mark_abs_3d** (**mark_abs** is internally handled as **timed_mark_abs_3d** with `Z` = 0x7FFFFFFF and `T` = 0).
- Important: Remote List Commands are not buffered. Therefore, statements (in other Chapters) about RTC6 Commands triggering a flush do not apply in Remote Interface Mode context.
- `UINT` and `LONG` type designations correspond to those from the import declarations for C/C++.

`UINT`	`typedef uint32_t`
`LONG`	`typedef int32_t`

- Other data types for Remote Commands are:

`CHAR`	8-Bit single character.
`DOUBLE`	64-bit IEEE floating point.

- The Answer Telegram does not distinguish between (normal) “result” and “returned parameter values”. A “returned parameter value” immediately follows the (normal) “result”.

(1) `UDPexcl` of **eth_set_port_numbers**, `UDPexcl` of **eth_get_port_numbers**.

(2) Or, Linux Shared Library, see Subfolder Linux, page 34.

Remote Control Commands

- See also Remote List Commands, page 1016.

ID	Remote Control Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
0	R_DC_NO_COMMAND	–	Reserved.	–
1	R_DC_CONFIG_LIST	UINT Mem1 UINT Mem2	config_list(Mem1, Mem2).	–
2	R_DC_GET_CONFIG_LIST	–	get_config_list().	–
3	R_DC_SET_START_LIST_POS	UINT ListNo UINT Pos	set_start_list_pos(ListNo, Pos). For UINT ListNo = 1: set_start_list_1(). For UINT ListNo = 2: set_start_list_2(). For UINT Pos = 0: set_start_list(ListNo).	–
4	R_DC_GET_INPUT_POINTER	–	get_input_pointer().	UINT Pos
5	R_DC_GET_LIST_SPACE	–	get_list_space().	UINT ListSpace
6	R_DC_LOAD_LIST_POS	UINT ListNo UINT Pos	load_list(ListNo, Pos).	UINT NumberOfList
7	R_DC_LOAD_SUB	UINT Index	load_sub(Index).	–
8	R_DC_LOAD_CHAR	UINT Char	load_char(Char).	–
9	R_DC_LOAD_TEXT_TABLE	UINT Index	load_text_table(Index).	–
10	R_DC_SET_EXTSTART_POS	UINT Pos	set_extstartpos(Pos).	–
11	R_DC_SET_MAX_COUNT	UINT Counts	set_max_counts(Counts).	–
12	R_DC_GET_COUNTS	–	get_counts().	UINT Counts
13	R_DC_STORE_TIMESTAMP	–	store_timestamp_counter().	–
14	R_DC_SIMULATE_EXT_START	–	simulate_ext_start_ctrl().	–
15	R_DC_EXECUTE_LIST_POS	UINT ListNo UINT Pos	execute_list_pos(ListNo, Pos). For UINT ListNo = 1: execute_list_1(). For UINT ListNo = 2: execute_list_2(). For UINT Pos = 0: execute_list(ListNo).	–
16	R_DC_STOP_EXECUTION	–	stop_execution().	–
17	R_DC_SIMULATE_EXT_STOP	–	simulate_ext_stop().	–
18	R_DC_EXTERN_START_DELAY	LONG Delay UINT EncoderNo	set_ext_start_delay(Delay, EncoderNo).	–
19	R_DC_AUTO_CHANGE_POS	UINT Pos	auto_change_pos(Pos). For UINT Pos = 0: auto_change().	–
20	R_DC_START_LOOP	–	start_loop().	–
21	R_DC_QUIT_LOOP	–	quit_loop().	–
22	R_DC_PAUSE_LIST	–	pause_list().	–
23	R_DC_RESTART_LIST	–	restart_list().	–
24	R_DC_RELEASE_WAIT	–	release_wait().	–
25	R_DC_GET_WAIT_STATUS	–	get_wait_status().	UINT WaitStatus

Remote Control Commands, page 1006

ID	Remote Control Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
26	R_DC_SET_PAUSE_LIST_COND	UINT Mask1 UINT Mask0 UINT Mode	For UINT Mode = 0: set_pause_list_cond(Mask1, Mask0). For UINT Mode = 1: set_pause_list_not_cond(Mask1, Mask0).	-
27	R_DC_GET_TIME	-	get_time().	DOUBLE TimerValue
28	R_DC_GET_LAP_TIME	-	get_lap_time().	DOUBLE TimerValue
29	R_DC_SET_CONTROL_MODE	UINT Mode	set_control_mode(Mode).	-
30	R_DC_READ_STATUS	-	read_status().	UINT Listenstatus
31	R_DC_GET_STATUS	-	get_status(&Status, &Pos).	UINT Status UINT Pos
32	R_DC_GET_STARTSTOP_INFO	-	get_startstop_info().	UINT Info
33	R_DC_GET_OVERRUN	-	get_overrun().	UINT NoOfOverruns
34	R_DC_CONTROL_COMMAND	UINT Head UINT Axis UINT Data	control_command(Head, Axis, Data).	-
35	R_DC_GET_VALUE	UINT Signal	get_value(Signal).	UINT Value
36	R_DC_GET_VALUES	UINT SignalPtr[4]	get_values(SignalPtr, &ResultPtr).	UINT ResultPtr [4]
37	R_DC_GET_HEAD_STATUS	UINT HeadNo	get_head_status(Head).	UINT Statuswort
38	R_DC_GET_HEAD_PARA	UINT HeadNo UINT ParaNo	get_head_para(HeadNo, ParaNo).	DOUBLE HeadPara
39	R_DC_GET_TABLE_PARA	UINT TableNo UINT ParaNo	get_table_para(TableNo, ParaNo).	DOUBLE TablePara
40	R_DC_GET_Z_DISTANCE	LONG X LONG Y LONG Z	get_z_distance(X, Y, Z).	LONG ZDistance
41	R_DC_GET_GALVO_CONTROLS	UINT InPtr[5] = {X, Y, Z, Defocus, Zoom}	get_galvo_controls(InPtr, OutPtr).	UINT OutPtr[4] = {XA, YA, XB, YB}
42	R_DC_SET_LASER_CONTROL	UINT Ctrl	set_laser_control(Ctrl).	-
43	R_DC_SET_LASER_MODE	UINT Mode	set_laser_mode(Mode).	-
44	R_DC_ENABLE_LASER	-	enable_laser().	-
45	R_DC_DISABLE_LASER	-	disable_laser().	-
46	R_DC_LASER_SIGNAL_ON	-	laser_signal_on().	-
47	R_DC_LASER_SIGNAL_OFF	-	laser_signal_off().	-
48	R_DC_SET_QSWITCH_DELAY	UINT Delay	set_qswitch_delay(Delay).	-
49	R_DC_SET_LASER_TIMING	UINT HalfPeriod UINT PulseLength	set_laser_pulses_ctrl(HalfPeriod, PulseLength).	-
50	R_DC_SET_FIRST_PULSE_KILLER	UINT Length	set_firstpulse_killer(Length).	-
51	R_DC_SET_LASER_PIN_OUT	UINT Pins	set_laser_pin_out(Pins).	-
52	R_DC_GET_LASER_PIN_IN	-	get_laser_pin_in().	UINT LaserPinIn

Remote Control Commands, page 1006

ID	Remote Control Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
53	R_DC_SET_STANDBY_TIMING	UINT HalfPeriod UINT PulseLength	set_standby(HalfPeriod, PulseLength).	-
54	R_DC_GET_STANDBY	-	get_standby(&HalfPeriod, &PulseLength).	UINT HalfPeriod UINT PulseLength
55	R_DC_PULSE_PICKING	UINT No	set_pulse_picking(No).	-
56	R_DC_PULSE_PICKING_LENGTH	UINT Length	set_pulse_picking_length(Length).	-
57	R_DC_CONFIG_LASER_SIGNALS	UINT Config	config_laser_signals(Config).	-
58	R_DC_SET_AUTO_LASER_CONTROL	UINT Ctrl UINT Value UINT Mode UINT MinValue UINT MaxValue	set_auto_laser_control(Ctrl, Value, Mode, MinValue, MaxValue).	UINT ErrorCode
59	R_DC_SET_AUTO_LASER_PARAMS	UINT Ctrl UINT Value UINT MinValue UINT MaxValue	ErrorCode = set_auto_laser_params(Ctrl, Value, MinValue, MaxValue)	-
60	R_DC_SPOT_DISTANCE	DOUBLE Dist	spot_distance_ctrl(Dist).	-
61	R_DC_SET_ENCODER_SPEED	UINT EncoderNo DOUBLE Speed DOUBLE Smooth	set_encoder_speed_ctrl(EncoderNo, Speed, Smooth).	-
62	R_DC_SET_DEFAULT_PIXEL	UINT PulseLength	set_default_pixel(PulseLength).	-
63	R_DC_SET_PORT_DEFAULT	UINT Port UINT Value	set_port_default(Port, Value).	-
64	R_DC_WRITE_8BIT_PORT	UINT Value	write_8bit_port(Value).	-
65	R_DC_WRITE_IO_PORT_MASK	UINT Value UINT Mask	write_io_port_mask(Value, Mask). For UINT Mask = 0xFFFF: write_io_port(Value).	-
66	R_DC_GET_IO_STATUS	-	get_io_status().	UINT IOStatus
67	R_DC_READ_IO_PORT	-	read_io_port().	UINT IOPort
68	R_DC_READ_IO_PORT_BUF	UINT Index	read_io_port_buffer(Index, &Value, &XPos, &YPos, &Time).	UINT CurrentIndex UINT Value LONG XPos LONG YPos UINT Time
69	R_DC_WRITE_DA_X	UINT x UINT Value	write_da_x(x, Value). For UINT x = 1: write_da_1(Value). For UINT x = 2: write_da_2(Value).	-
70	R_DC_READ_ANALOG_IN	-	read_analog_in().	UINT AnalogValue
71	R_DC_SET_JUMP_SPEED	DOUBLE Speed	set_jump_speed_ctrl(Speed).	-
72	R_DC_SET_MARK_SPEED	DOUBLE Speed	set_mark_speed_ctrl(Speed).	-
73	R_DC_SET_DEFOCUS	LONG Shift	set_defocus(Shift).	-
74	R_DC_SET_DEFOCUS_OFFSET	LONG Shift	set_defocus_offset(Shift).	-

Remote Control Commands, page 1006

ID	Remote Control Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
75	R_DC_SET_DELAY_MODE	UINT VarPoly UINT DirectMode3D UINT EdgeLevel UINT MinJumpDelay UINT JumpLengthLimit	set_delay_mode(VarPoly, DirectMove3D, EdgeLevel, MinJumpDelay, JumpLengthLimit).	-
76	R_DC_GOTO_XYZ	LONG X LONG Y LONG Z	goto_xyz(X, Y, Z). For LONG Z = 0x7FFFFFFF: goto_xy(X, Y).	-
77	R_DC_SET_OFFSET_XYZ	UINT HeadNo LONG X LONG Y LONG Z UINT at_once	set_offset_xyz(HeadNo, XOffset, YOffset, ZOffset , at_once). For LONG Z = 0x7FFFFFFF: set_offset(HeadNo, XOffset, YOffset, at_once).	-
78	R_DC_HOME_POSITION_XYZ	LONG XHome LONG YHome LONG ZHome	home_position_xyz(XHome, YHome, ZHome). For LONG ZHome = 0x7FFFFFFF: home_position(XHome, YHome).	-
79	R_DC_TIME_UPDATE	UINT DateTime[8] = {Year, Month, Day, YearDay, WeekDay, Hours, Minutes, Seconds} see 1, page 1015	time_update() at point in time UINT DateTime[8].	-
80	R_DC_TIME_CONTROL_ETH	DOUBLE PPM	time_control_eth(PPM).	-
81	R_DC_SET_SERIAL	UINT No UINT Step	set_serial_step(No, Step).	-
82	R_DC_SELECT_SERIAL_SET	UINT No	select_serial_set(No).	-
83	R_DC_GET_SERIAL	-	get_serial().	DOUBLE Current-SerialNo
84	R_DC_GET_LIST_SERIAL	-	get_list_serial(&Set).	DOUBLE LastMarked-SerialNo UINT Set
85	R_DC_SET_ZOOM	UINT Zoom	set_zoom(Zoom).	-
86	R_DC_SET_MATRIX	UINT HeadNo DOUBLE M11 DOUBLE M12 DOUBLE M21	set_matrix(HeadNo, M11, M12, M21, M22, at_once) .	DOUBLE M22 UINT at_once
87	R_DC_SET_ANGLE	UINT HeadNo DOUBLE Angle UINT at_once	set_angle(HeadNo, Angle, at_once).	-
88	R_DC_SET_SCALE	UINT HeadNo DOUBLE Scale UINT at_once	set_scale(HeadNo, Scale, at_once).	-
89	R_DC_SIMULATE_ENCODER	UINT EncoderNo	simulate_encoder(EncoderNo).	-
90	R_DC_READ_ENCODER	-	read_encoder(&Encoder0_0, &Encoder1_0, &Encoder0_1, &Encoder1_1).	LONG Encoder0_0 LONG Encoder1_0 LONG Encoder0_1 LONG Encoder1_1

Remote Control Commands, page 1006

ID	Remote Control Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
91	R_DC_GET_ENCODER	-	get_encoder(&Encoder0, &Encoder1).	LONG Encoder0 LONG Encoder1
92	R_DC_INIT_FLY_2D	LONG OffsetX LONG OffsetY UINT No	init_fly_2d(OffsetX, OffsetY, No).	-
93	R_DC_GET_FLY_2D_OFFSET	-	get_fly_2d_offset(&OffsetX, &OffsetY).	LONG OffsetX LONG OffsetY
94	R_DC_GET_MARKING_INFO	-	get_marking_info().	UINT MarkingInfo
95	R_DC_ROT_CENTER	LONG X LONG Y	set_rot_center(X, Y).	-
96	R_DC_SET_MCBSP_FREQ	UINT Freq	set_mcbsp_freq(Freq).	UINT mcbsp_freq
97	R_DC_MCBSP_INIT	UINT XDelay UINT RDelay	mcbsp_init(XDelay, RDelay).	-
98	R_DC_SET_MCBSP_MODE	UINT Axis DOUBLE Scale	For UINT Axis = 1: set_mcbsp_x(Scale). For UINT Axis = 2: set_mcbsp_y(Scale). For UINT Axis = 4: set_mcbsp_rot(Resolution) with Resolution = DOUBLE Scale. For UINT Axis = 0 and DOUBLE Scale = does not matter: set_mcbsp_matrix().	-
99	R_DC_SET_MCBSP_MODE_GLOBAL	UINT Axis DOUBLE Scale	For UINT Axis = 1 set_mcbsp_global_x(Scale). For UINT Axis = 2 set_mcbsp_global_y(Scale). For UINT Axis = 4 set_mcbsp_global_rot(Resolution) with Resolution = DOUBLE Scale. For UINT Axis = 0 and DOUBLE Scale = does not matter: set_mcbsp_global_matrix().	-
100	R_DC_SET_MCBSP_IN	UINT Mode DOUBLE Scale	set_mcbsp_in(Mode, Scale).	-
101	R_DC_SET_MULTI_MCBSP_IN	UINT Ctrl UINT P UINT Mode	set_multi_mcbsp_in(Ctrl, P, Mode).	-
102	R_DC_READ_MCBSP	UINT No	read_mcbsp(No).	UINT mcbsp_value
103	R_DC_READ_MULTI_MCBSP	UINT No	read_multi_mcbsp(No).	UINT mcbsp_value
104	R_DC_APPLY_MCBSP	UINT HeadNo UINT at_once	apply_mcbsp(HeadNo, at_once).	-
105	R_DC_GET_SCANAHEAD_PARAMS	UINT HeadNo	get_scanahead_params(HeadNo, &Preview, &Amax, &Vmax).	UINT Error UINT Preview UINT Amax UINT Vmax
106	R_DC_SCANAHEAD_ACTIVATE_AUTO	UINT Mode	activate_scanahead_autodelays(Mode).	UINT CurrentMode
107	R_DC_SET_SCANAHEAD_LASER_SHIFT	LONG dLasOn LONG dLasOff	set_scanahead_laser_shifts(dLasOn, dLasOff).	-
108	R_DC_SET_SCANAHEAD_LINE_PARAMS	UINT CornerScale UINT EndScale UINT AccScale UINT JumpScale	set_scanahead_line_params (CornerScale, EndScale, AccScale).	-

Remote Control Commands, page 1006

ID	Remote Control Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
109	R_DC_SET_SCANAHEAD_PARAMS	UINT Mode UINT HeadNo UINT TableNo UINT Preview UINT Vmax DOUBLE Amax	set_scanahead_params(Mode, HeadNo, TableNo, Preview, Vmax, Amax).	UINT Error
110	R_DC_SET_SCANAHEAD_SPEED_CTRL	UINT Mode	set_scanahead_speed_control(Mode).	-
111	R_DC_STEPPER_INIT	UINT No UINT Period LONG Dir LONG Pos UINT Tol UINT Enable UINT WaitTime	stepper_init(No, Period, Dir, Pos, Tol, Enable, WaitTime).	-
112	R_DC_STEPPER_ENABLE	LONG Enable1 LONG Enable2	stepper_enable(Enable1, Enable2).	-
113	R_DC_STEPPER_DISABLE_SWITCH	LONG Disable1 LONG Disable2	stepper_disable_switch(Disable1, Disable2).	-
114	R_DC_STEPPER_CTRL	LONG Period1 LONG Period2	stepper_control(Period1, Period2).	-
115	R_DC_STEPPER_NO_ABS	UINT No LONG Pos UINT WaitTime	stepper_abs_no(No, Pos, WaitTime).	-
116	R_DC_STEPPER_NO_REL	UINT No LONG dPos UINT WaitTime	stepper_rel_no(No, dPos, WaitTime).	-
117	R_DC_STEPPER_ABS	LONG Pos1 LONG Pos2 UINT WaitTime	stepper_abs(Pos1, Pos2, WaitTime).	-
118	R_DC_STEPPER_REL	LONG dPos1 LONG dPos2 UINT WaitTime	stepper_rel(dPos1, dPos2, WaitTime).	-
119	R_DC_GET_STEPPER_STATUS	-	get_stepper_status(&Status1, &Pos1, &Status2, &Pos2).	UINT Status1 LONG Pos1 UINT Status2 LONG Pos2
120	R_DC_SET_SKY_WRITING_PARA	DOUBLE Timelag LONG LaserOnShift UINT Nprev UINT Npost	set_sky_writing_para(Timelag, LaserOnShift, Nprev, Npost). For UINT Nprev, UINT Npost ≥ 65,535: set_sky_writing(Timelag, LaserOnShift).	-
121	R_DC_SET_SKY_WRITING_ANGLE	DOUBLE Limit	set_sky_writing_limit(Limit).	-
122	R_DC_SET_SKY_WRITING_MODE	UINT Mode	set_sky_writing_mode(Mode).	-
123	R_DC_UART_CONFIG	UINT BaudRate	uart_config(BaudRate).	UINT RealBaudRate
124	R_DC_RS232_WRITE_DATA	UINT Data	rs232_write_data(Data).	-
125	R_DC_RS232_WRITE_TEXT	CHAR Text[N] see 2, page 1015	rs232_write_text(pData).	-

Remote Control Commands, page 1006

ID	Remote Control Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
126	R_DC_RS232_READ_DATA	–	rs232_read_data().	UINT RS232Data
127	R_DC_BOUNCE_SUPP	UINT Length	bounce_supp(Length).	–
128	R_DC_SET_HI	UINT HeadNo DOUBLE GalvoGainX DOUBLE GalvoGainY LONG GalvoOffsetX LONG GalvoOffsetY	set_hi(HeadNo, GalvoGainX, GalvoGainY, GalvoOffsetX, GalvoOffsetY).	–
129	R_DC_SET_JUMP_MODE	LONG Flag UINT Length LONG VA1 LONG VA2 LONG VB1 LONG VB2 LONG JA1 LONG JA2 LONG JB1 LONG JB2	set_jump_mode(Flag, Length, VA1, VA2, VB1, VB2, JA1, JA2, JB1, JB2).	–
130	R_DC_SELECT_COR_TABLE	UINT HeadA UINT HeadB	select_cor_table(HeadA, HeadB).	–
131	R_DC_SET_FREE_VAR	UINT VarNo UINT Value	set_free_variable(No, Value).	–
132	R_DC_GET_FREE_VAR	UINT VarNo	get_free_variable(No).	UINT Value
133	R_DC_SET_MCBSP_OUT_PTR	UINT Num UINT SignalPtr[Num]	set_mcbsp_out_ptr(Number, SignalPtr).	–
134	R_DC_PERIODIC_TOGGLE	UINT Port UINT Mask UINT P1 UINT P2 UINT Count UINT Start	periodic_toggle(Port, Mask, P1, P2, Count, Start).	–
135	R_DC_ENDURING_WOBBEL_1	UINT CenterX UINT CenterY UINT CenterZ UINT LimitHi UINT LimitLo DOUBLE ScaleX DOUBLE ScaleY DOUBLE ScaleZ	Reserved.	–
136	R_DC_ENDURING_WOBBEL_2	UINT CenterX UINT CenterY UINT CenterZ UINT LimitHi UINT LimitLo DOUBLE ScaleX DOUBLE ScaleY DOUBLE ScaleZ	Reserved.	–
137	R_DC_STOP_TRIGGER	–	stop_trigger().	–
138	R_DC_MEASUREMENT_STATUS	–	measurement_status(&Busy, &Pos).	UINT Busy UINT Pos

Remote Control Commands, page 1006

ID	Remote Control Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
139	R_DC_GET_WAVEFORM	UINT Channel UINT Offset UINT Number	get_waveform_offset(Channel, Offset, Number, Ptr).	UINT Index UINT Data[Number] see 3, page 1015
140	R_DC_SET_VERIFY	UINT Verify	set_verify(Verify).	UINT OldVerify
141	R_DC_GET_HEX_VERSION	–	get_hex_version().	UINT HexVersion
142	R_DC_GET_RTC_VERSION	–	get_rtc_version().	UINT RTCVersion
143	R_DC_GET_BIOS_VERSION	–	get_bios_version().	UINT BiosVersion
144	R_DC_GET_SERIAL_NUMBER	–	get_serial_number().	UINT SerialNumber
145	R_DC_SET_INPUT_POINTER	UINT Pos	set_input_pointer(Pos).	–
146	R_DC_GET_LIST_POINTER	–	get_list_pointer(&ListNo, &Pos).	UINT ListNo UINT Pos
147	R_DC_EXECUTE_AT_POINTER	UINT Pos	execute_at_pointer(Pos).	–
148	R_DC_GET_REMOTE_ERROR	–	A corresponding RTC6 command does not exist because get_error Bit #31 can only occur in Remote Interface Mode.	UINT Error see Error Bit, page 1003
149	R_DC_SET_LASER_PULSE_SYNC	UINT Mode UINT Delay	set_laser_pulse_sync(Mode, Delay).	–
150	R_DC_GET_TIMESTAMP_LONG	–	get_timestamp_long(&TimeStampL, &TimeStampH).	UINT TimestampLow UINT TimestampHigh
151	R_DC_CLEAR_FLY_OVERFLOW	UINT Mode	clear_fly_overflow_ctrl(Mode).	–
152	R_DC_GET_TEMPERATURE	–	get_temperature().	DOUBLE Temperature
153	R_DC_SET_MCBSP_OUT_OIE	UINT Signal1 UINT Signal2	set_mcbsp_out_oie_ctrl(Signal1, Signal2).	–
154	R_DC_LOAD_CORRECTION_FILE	UINT Offset UINT Size CHAR Data[Size]	load_correction_file(…). Siehe 4, page 1015	UINT Error
155	R_DC_NUMBER_OF_COR_TABLES	UINT Number	number_of_correction_tables(Number).	–
156	R_DC_LOAD_Z_TABLE	DOUBLE A DOUBLE B DOUBLE C UINT No	load_z_table_no(A, B, C, No).	UINT Error
157	R_DC_LOAD_OCT_TABLE	DOUBLE A DOUBLE B UINT No	Reserved.	UINT Error

Remote Control Commands, page 1006

ID	Remote Control Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
158	R_DC_SET_TIMELAG_COMPENSATION	UINT HeadNo UINT TimelagXY UINT TimelagZ	set_timelag_compensation(HeadNoXY, TimelagXY, TimelagZ).	-
159	R_DC_SET_ENCODER_FILTER	UINT Length UINT Mode	set_encoder_filter_ctrl(Length, Mode).	-
160	R_DC_GET_MASTER_SLAVE	-	get_master_slave().	UINT MasterSlaveStatus
161	R_DC_GET_SYNC_STATUS	-	get_sync_status().	UINT SyncStatus
162	R_DC_MASTER_SLAVE_CONFIG	UINT Config	master_slave_config(Config).	-
163	R_DC_GET_STATIC_IP	-	eth_get_static_ip(&Ip, &NetMask, &Gateway).	UINT Ip UINT Netmask UINT Gateway
164	R_DC_SET_STATIC_IP	UINT Ip UINT Netmask UINT Gateway	eth_set_static_ip(Ip, NetMask, Gateway).	UINT Error
165	R_DC_GET_PORT_NUMBERS	-	eth_get_port_numbers(&UDPsearch, &UDPexcl, &TCP).	UINT UDPsearch UINT UDPexcl UINT TCP
166	R_DC_SET_PORT_NUMBERS	UINT UDPsearch UINT UDPexcl UINT TCP	eth_set_port_numbers(UDPsearch, UDPexcl, TCP).	UINT Error
167	R_DC_CONFIGURE_LINK_LOSS	UINT Mode	eth_configure_link_loss(Mode).	-
168	R_DC_SET_SKY_WRITING_MIN_SPEED	DOUBLE Speed	set_sky_writing_min_speed_ctrl(Speed).	-
169	R_DC_SET_CTRL_PREVIEW_COMP	UINT ControlPreview UINT Mode	set_controlpreview_compensation_ctrl(ControlPreview, Mode).	-
170	R_DC_SET_FLY_TRACKING_ERROR	UINT TrackingErrorX UINT TrackingErrorY	set_fly_tracking_error(TrackingErrorX, TrackingErrorY).	-
…	…	…	…	…
255	-	-	-	-

Remote Control Commands, page 1006

Notes

(1) On 79 R_DC_TIME_UPDATE.

```
// Example code for Windows (C++)

time64_t dt( _time64( NULL ) );
tm today;
(void) _localtime64_s( &today, &dt
UINT DateTime[8]

DateTime[ E_YEAR    ] = today.tm_year + 1900;

// 1-12
DateTime[ E_MONTH   ] = today.tm_mon + 1;

// 1…31
DateTime[ E_DAY     ] = today.tm_mday;

// 0-364/365
DateTime[ E_YEARDAY ] = today.tm_yday;

// 1-7. 1 = Sunday
DateTime[ E_WEEKDAY ] = today.tm_wday + 1;

// 0-23
DateTime[ E_HOURS   ] = today.tm_hour;

// 0-59
DateTime[ E_MINUTES ] = today.tm_min;

// 0-59
DateTime[ E_SECONDS ] = today.tm_sec;
```

(2) On 125 R_DC_RS232_WRITE_TEXT, CHAR Text[N].
You must select N so that the telegram total length does not exceed TGM_MAX_SIZE. If necessary, distribute the text to several Remote Commands. You must terminate each of the texts by "\0".

(3) On 139 R_DC_GET_WAVEFORM.
Number of UINT Data[Number] must be ≤ 343(1).
Otherwise, LastError 1st ID (in TGM_PL_ANSW_RAW) returns RTC6_PARAM_ERROR.
You need to read out larger amounts of data by several 139 R_DC_GET_WAVEFORM calls, see **get_waveform_offset** implementation in rtc6_rif_wrapper.cpp.

(4) On 154 R_DC_LOAD_CORRECTION_FILE.
In Remote Interface Mode, **load_correction_file** must be implemented in several steps, see also rtc6_rif_wrapper.cpp.
In step 1, the entire contents of the correction file must be read in and sent to the RTC6 Ethernet Board. The amount of data is too large for a single Command Telegram. Therefore, this must be done by several Command Telegrams.
UINT Offset specifies the byte offset within the file. UINT Size specifies the number of bytes of the current block. This is followed by n Bytes of data (n = Size).
Step 2 follows not until the entire correction file content has been loaded to the RTC6 Ethernet Board. For this UINT Offset must be set to –1 (UINT_MAX, see telegrams.h). In UINT Size the correction file number (No) must be written to the lower 16 bits and the dimension (Dim) to the upper 16 bits. No further data follow in step 2.
The call corresponds to
load_correction_file(Name = file path, No, Dim).

(1) Derivation: 4 byte per 1 waveform value.
[5 × 4 byte TGM_HEADER] +
[1 × 4 byte LastError 1st ID] +
[1 × 4 byte Remote Command 1st ID] +
[343 × 4 byte] = [≤ 1.400 byte TGM_MAX_SIZE].

Remote List Commands

- See also Remote Control Commands, page 1006.

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
256	R_LC_NO_COMMAND	–	list_nop().	–
257	R_LC_CONTINUE	–	list_continue().	–
258	R_LC_NEXT	–	list_next().	–
259	R_LC_END_OF_LIST	–	set_end_of_list().	–
260	R_LC_LIST_RETURN	–	list_return().	–
261	R_LC_REPEAT	–	list_repeat().	–
262	R_LC_UNTIL	UINT Number	list_until(Number).	–
263	R_LC_LIST_JUMP_POS	UINT Pos	list_jump_pos(Pos).	–
264	R_LC_LIST_JUMP_REL	LONG Pos	list_jump_rel(Pos).	–
265	R_LC_SET_WAIT	UINT WaitWord	set_wait(WaitWord).	–
266	R_LC_LONG_DELAY	UINT Delay	long_delay(Delay).	–
267	R_LC_SET_EXTSTART_POS	UINT Pos	set_extstartpos_list(Pos).	–
268	R_LC_SIMULATE_EXTERN_START	LONG Delay UINT EncoderNo	simulate_ext_start(Delay, EncoderNo).	–
269	R_LC_LIST_CALL	UINT Pos UINT Number UINT AbsCall	For UINT AbsCall = 1: list_call_abs_repeat(Pos, Number). For UINT Number = 0 or 1: list_call_abs(Pos). For UINT AbsCall = 0: list_call_repeat(Pos, Number). For UINT Number = 0 or 1: list_call(Pos).	–
270	R_LC_SUB_CALL	UINT Index UINT Number UINT AbsCall	For UINT AbsCall = 1: sub_call_abs_repeat(Index, Number). For UINT Number = 0 or 1: sub_call_abs(Index). For UINT AbsCall = 0: sub_call_repeat(Index, Number). For UINT Number = 0 or 1: sub_call(Index).	–
271	R_LC_IF_COND	UINT Mask1 UINT Mask0 UINT Mode	For UINT Mode = 0: if_cond(Mask1, Mask0). For UINT Mode = 1: if_not_cond(Mask1, Mask0).	–
272	R_LC_IF_PIN_COND	UINT Mask1 UINT Mask0 UINT Mode	For UINT Mode = 0: if_pin_cond(Mask1, Mask0). For UINT Mode = 1: if_not_pin_cond(Mask1, Mask0).	–
273	R_LC_CLEAR_IO_COND	UINT Mask1 UINT Mask0 UINT MaskClear	clear_io_cond_list(Mask1, Mask0, MaskClear).	–

Remote List Commands, page 1016

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
274	R_LC_SET_IO_COND	UINT Mask1 UINT Mask0 UINT MaskSet	set_io_cond_list(Mask1, Mask0, MaskSet).	–
275	R_LC_SWITCH_IOPORT	UINT MaskBits UINT BitShifts	switch_ioport(MaskBits, ShiftBits).	–
276	R_LC_LIST_CALL_COND	UINT Mask1 UINT Mask2 UINT Pos UINT AbsCall	For UINT AbsCall = 1: list_call_abs_cond(Mask1, Mask0, Pos). For UINT AbsCall = 0: list_call_cond(Mask1, Mask0, Pos).	–
277	R_LC_SUB_CALL_COND	UINT Mask1 UINT Mask2 UINT Index UINT AbsCall	For UINT AbsCall = 1: sub_call_abs_cond(Mask1, Mask0, Index). For UINT AbsCall = 0: sub_call_cond(Mask1, Mask0, Index).	–
278	R_LC_LIST_JUMP_POS_COND	UINT Mask1 UINT Mask0 UINT Pos	list_jump_pos_cond(Mask1, Mask0, Pos).	–
279	R_LC_LIST_JUMP_REL_COND	UINT Mask1 UINT Mask0 LONG Pos	list_jump_rel_cond(Mask1, Mask0, Pos).	–
280	R_LC_CONFIG_LASER_SIGNALS	UINT Config	config_laser_signals_list(Config).	–
281	R_LC_LASER_SIGNAL_ON	–	laser_signal_on_list().	–
282	R_LC_LASER_SIGNAL_OFF	–	laser_signal_off_list().	–
283	R_LC_SET_LASER_TIMING	UINT HalfPeriod UINT PulseLength	set_laser_pulses(HalfPeriod, PulseLength).	–
284	R_LC_SET_FIRST_PULSE_KILLER	UINT Length	set_firstpulse_killer_list(Length).	–
285	R_LC_SET_QSWITCH_DELAY	UINT Delay	set_qswitch_delay_list(Delay).	–
286	R_LC_SET_LASER_PIN_OUT	UINT Pins	set_laser_pin_out_list(Pins).	–
287	R_LC_PULSE_PICKING	UINT No	set_pulse_picking_list(No).	–
288	R_LC_SET_STANDBY_TIMING	UINT HalfPeriod UINT PulseLength	set_standby_list(HalfPeriod, PulseLength).	–
289	R_LC_SET_AUTO_LASER_PARAMS	UINT Ctrl UINT Value UINT MinValue UINT MaxValue	ErrorCode = set_auto_laser_params_list(Ctrl, Value, MinValue, MaxValue).	–
290	R_LC_LASER_POWER	UINT Port UINT Power	set_laser_power(Port, Power).	–
291	R_LC_SPOT_DISTANCE	DOUBLE Dist	spot_distance(Dist).	–
292	R_LC_SET_ENCODER_SPEED	UINT Encoder DOUBLE Speed DOUBLE Smooth	set_encoder_speed(EncoderNo, Speed, Smooth).	–
293	R_LC_SET_DEFAULT_PIXEL	UINT PulseLength	set_default_pixel_list(PulseLength).	–
294	R_LC_SET_PORT_DEFAULT	UINT Port UINT Value	set_port_default_list(Port, Value).	–

Remote List Commands, page 1016

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
295	R_LC_SET_PIXEL_LINE	UINT Channel UINT HalfPeriod DOUBLE dX DOUBLE dY DOUBLE dZ	set_pixel_line_3d(Channel, HalfPeriod, dX, dY, dZ). For DOUBLE dZ = 0: set_pixel_line(Channel, HalfPeriod, dX, dY).	–
296	R_LC_SET_N_PIXEL_DATA	UINT PortOutValue1 UINT PortOutValue2 UINT Number	set_n_pixel(PortOutValue1, PortOutValue2, Number). For UINT Number = 1: set_pixel(PortOutValue1, PortOutValue2).	–
297	R_LC_WRITE_8BIT_PORT	UINT Value	write_8bit_port(Value).	–
298	R_LC_WRITE_IO_PORT_MASK	UINT Value UINT Mask	write_io_port_mask_list(Value, Mask). For UINT Mask = 0xFFFF write_io_port_list(Value).	–
299	R_LC_READ_IO_PORT	–	read_io_port_list().	–
300	R_LC_WRITE_DA_X	UINT x UINT Value	write_da_x_list(x, Value). For UINT x = 1: write_da_1_list(Value). For UINT x = 2: write_da_2_list(Value).	–
301	R_LC_SET_MARK_SPEED	DOUBLE Speed	set_mark_speed(Speed).	–
302	R_LC_SET_JUMP_SPEED	DOUBLE Speed	set_jump_speed(Speed).	–
303	R_LC_SET_SCANNER_DELAYS	UINT Jump UINT Mark UINT Polygon	set_scanner_delays(Jump, Mark, Polygon).	–
304	R_LC_SET_LASER_DELAYS	LONG LaserOnDelay UINT LaserOffDelay	set_laser_delays(LaserOnDelay, LaserOffDelay).	–
305	R_LC_SET_DEFOCUS	LONG Shift	set_defocus_list(Shift).	–
306	R_LC_SET_DEFOCUS_OFFSET	LONG Shift	set_defocus_offset_list(Shift).	–
307	R_LC_SET_ZOOM	UINT Zoom	set_zoom_list(Zoom).	–
308	R_LC_SET_OFFSET_XYZ	UINT HeadNo LONG XOffset LONG YOffset LONG ZOffset UINT at_once	set_offset_xyz_list(HeadNo, XOffset, YOffset, ZOffset, at_once). For LONG ZOffset = 0x7FFFFFFF: set_offset_list(HeadNo, XOffset, YOffset, at_once).	–
309	R_LC_SET_MATRIX	UINT HeadNo UINT Ind1 UINT Ind2 DOUBLE Mij UINT at_once	set_matrix_list(HeadNo, Ind1, Ind2, Mij, at_once).	–
310	R_LC_SET_ANGLE	UINT HeadNo DOUBLE Angle UINT at_once	set_angle_list(HeadNo, Angle, at_once).	–
311	R_LC_SET_SCALE	UINT HeadNo DOUBLE Scale UINT at_once	set_scale_list(HeadNo, Scale, at_once).	–

Remote List Commands, page 1016

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
312	R_LC_MARK_XYZT_ABS	LONG X LONG Y LONG Z DOUBLE T	timed_mark_abs_3d(X, Y, Z, T). For LONG Z = 0x7FFFFFFF: timed_mark_abs(X, Y, T). For DOUBLE T = 0: mark_abs_3d(X, Y, Z). For LONG Z = 0x7FFFFFFF, DOUBLE T = 0: mark_abs(X, Y).	–
313	R_LC_MARK_XYZT_REL	LONG dX LONG dY LONG dZ DOUBLE T	timed_mark_rel_3d(dX, dY, dZ, T). For LONG dZ = 0: timed_mark_rel(dX, dY, T). For DOUBLE T = 0: mark_rel_3d(dX, dY, dZ). For LONG dZ = 0 and DOUBLE T = 0: mark_rel(dX, dY).	–
314	R_LC_ARC_XYAT_ABS	LONG X LONG Y DOUBLE Angle DOUBLE T	timed_arc_abs(X, Y, Angle, T). For DOUBLE T = 0: arc_abs(X, Y, Angle).	–
315	R_LC_ARC_XYAT_REL	LONG dX LONG dY DOUBLE Angle DOUBLE T	timed_arc_rel(dX, dY, Angle, T). For DOUBLE T = 0: arc_rel(dX, dY, Angle).	–
316	R_LC_ARC_XYZA_ABS	LONG X LONG Y LONG Z DOUBLE Angle	arc_abs_3d(X, Y, Z, Angle).	–
317	R_LC_ARC_XYZA_REL	LONG dX LONG dY LONG dZ DOUBLE Angle	arc_rel_3d(dX, dY, dZ, Angle).	–
318	R_LC_SET_ELLIPSE	UINT A UINT B DOUBLE Phi0 DOUBLE Phi	set_ellipse(a, b, Phi0, Phi).	–
319	R_LC_ELLIPSE_ABS	LONG X LONG Y DOUBLE Alpha	mark_ellipse_abs(X, Y, Alpha).	–
320	R_LC_ELLIPSE_REL	LONG dX LONG dY DOUBLE Alpha	mark_ellipse_rel(dX, dY, Alpha).	–
321	R_LC_JUMP_XY_ABS	LONG X LONG Y	jump_abs(X, Y).	–
322	R_LC_JUMP_XY_REL	LONG dX LONG dY	jump_rel(dX, dY).	–
323	R_LC_JUMP_XYZT_ABS	LONG X LONG Y LONG Z DOUBLE T	timed_jump_abs_3d(X, Y, Z, T). For LONG Z = 0x7FFFFFFF: timed_jump_abs(X, Y, T). For DOUBLE T = 0: jump_abs_3d(X, Y, Z).	–
324	R_LC_JUMP_XYZT_REL	LONG dX LONG dY LONG dZ DOUBLE T	timed_jump_rel_3d(dX, dY, dZ, T). For LONG dZ = 0: timed_jump_rel(dX, dY, T). For DOUBLE T = 0: jump_rel_3d(dX, dY, dZ).	–

Remote List Commands, page 1016

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
325	R_LC_JUMP_XY_DRILL_ABS	LONG X LONG Y UINT DrillTime LONG XOff LONG YOff	jump_abs_drill_2(X, Y, DrillTime, XOff, YOff). For LONG XOff = LONG YOff = 0: jump_abs_drill(X, Y, DrillTime).	–
326	R_LC_JUMP_XY_DRILL_REL	LONG dX LONG dY UINT DrillTime LONG XOff LONG YOff	jump_rel_drill_2(dX, dY, DrillTime, XOff, YOff). For LONG XOff = LONG YOff = 0: jump_rel_drill(dX, dY, DrillTime).	–
327	R_LC_SET_VECTOR_CONTROL	UINT Ctrl UINT Value	set_vector_control(Ctrl, Value).	–
328	R_LC_LASER_ON_PULSES_PARA	UINT Period UINT Pulses UINT P	para_laser_on_pulses_list(Period, Pulses, P). For UINT Pulses > 65,535: laser_on_list(Period). For UINT P = 2^{32}–1: laser_on_pulses_list(Period, Pulses).	–
329	R_LC_MARK_XYZP_ABS	LONG X LONG Y LONG Z UINT P	para_mark_abs_3d(X, Y, Z, P). For LONG Z = 0x7FFFFFFF: para_mark_abs(X, Y, P).	–
330	R_LC_MARK_XYZP_REL	LONG dX LONG dY LONG dZ UINT P	para_mark_rel_3d(dX, dY, dZ, P). For LONG dZ = 0: para_mark_rel(dX, dY, P).	–
331	R_LC_MARK_XYZPT_ABS	LONG X LONG Y LONG Z UINT P DOUBLE T	timed_para_mark_abs_3d(X, Y, Z, P, T). For LONG Z = 0x7FFFFFFF: timed_para_mark_abs(X, Y, P, T).	–
332	R_LC_MARK_XYZPT_REL	LONG dX LONG dY LONG dZ UINT P DOUBLE T	timed_para_mark_rel_3d(dX, dY, dZ, P, T). For LONG dZ = 0: timed_para_mark_rel(dX, dY, P, T).	–
333	R_LC_JUMP_XYZP_ABS	LONG X LONG Y LONG Z UINT P	para_jump_abs_3d(X, Y, Z, P). For LONG Z = 0x7FFFFFFF: para_jump_abs(X, Y, P).	–
334	R_LC_JUMP_XYZP_REL	LONG dX LONG dY LONG dZ UINT P	para_jump_rel_3d(dX, dY, dZ, P). For LONG dZ = 0: para_jump_rel(dX, dY, P).	–
335	R_LC_JUMP_XYZPT_ABS	LONG X LONG Y LONG Z UINT P DOUBLE T	timed_para_jump_abs_3d(X, Y, Z, P, T). For LONG Z = 0x7FFFFFFF: timed_para_jump_abs(X, Y, P, T).	–

Remote List Commands, page 1016

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
336	R_LC_JUMP_XYZPT_REL	LONG dX LONG dY LONG dZ UINT P DOUBLE T	timed_para_jump_rel_3d(dX, dY, dZ, P, T). For LONG dZ = 0: timed_para_jump_rel(dX, dY, P, T).	-
337	R_LC_MICRO_VECTOR_ABS_3D	LONG X LONG Y LONG Z LONG LasOn LONG LasOff	micro_vector_abs_3d(X, Y, Z, LasOn, LasOff). For LONG Z = 0x7FFFFFFF: micro_vector_abs(X, Y, LasOn, LasOff).	-
338	R_LC_MICRO_VECTOR_REL_3D	LONG dX LONG dY LONG dZ LONG LasOn LONG LasOff	micro_vector_rel_3d(dX, dY, dZ, LasOn, LasOff). For LONG dZ = 0: micro_vector_rel(dX, dY, LasOn, LasOff).	-
339	R_LC_MARK_CHAR	UINT Char UINT AbsCall	For UINT AbsCall = 1: mark_char_abs(Char). For UINT AbsCall = 0: mark_char(Char).	-
340	R_LC_SELECT_CHAR_SET	UINT No	select_char_set(No).	-
341	R_LC_MARK_TEXT	CHAR Text[N] siehe 1, page 1027 UINT AbsCall	For UINT AbsCall = 1: mark_text_abs(Text). For UINT AbsCall = 0: mark_text(Text).	-
342	R_LC_MARK_SERIAL	UINT Mode UINT Digits UINT AbsCall	For UINT AbsCall = 1: mark_serial_abs(Mode, Digits). For UINT AbsCall = 0: mark_serial(Mode, Digits).	-
343	R_LC_SET_SERIAL	UINT No UINT Step	set_serial_step_list(No, Step).	-
344	R_LC_SELECT_SERIAL_SET	UINT No	select_serial_set_list(No).	-
345	R_LC_TIME_FIX_F_OFF	UINT FirstDay UINT Offset	time_fix_f_off(FirstDay, Offset). For UINT FirstDay = 0: time_fix(). For UINT Offset = 0: time_fix_f(FirstDay).	-
346	R_LC_MARK_DATE	UINT Part UINT Mode UINT AbsCall	For UINT AbsCall = 1: mark_date_abs(Part, Mode). For UINT AbsCall = 0: mark_date(Part, Mode).	-
347	R_LC_MARK_TIME	UINT Part UINT Mode UINT AbsCall	For UINT AbsCall = 1: mark_time_abs(Part, Mode). For UINT AbsCall = 0: mark_time(Part, Mode).	-
348	R_LC_SET_WOBBEL_MODE_PHASE	UINT Transversal UINT Longitudinal DOUBLE Freq LONG Mode DOUBLE Phase	set_wobbel_mode_phase(Transversal, Longitudinal, Freq, Mode, Phase).	-
349	R_LC_SET_WOBBEL_MODE	UINT Transversal UINT Longitudinal DOUBLE Freq LONG Mode	set_wobbel_mode(Transversal, Longitudinal, Freq, Mode). For LONG Mode = 0: set_wobbel(Transversal, Longitudinal, Freq).	-
350	R_LC_SET_WOBBEL_DIR	LONG dX LONG dY	set_wobbel_direction(dX, dY).	-

Remote List Commands, page 1016

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
351	R_LC_SET_WOBBEL_CONTROL	UINT Ctrl UINT Value UINT MinValue UINT MaxValue	set_wobbel_control(Ctrl, Value, MinValue, MaxValue).	-
352	R_LC_SET_WOBBEL_VECTOR	DOUBLE dTrans DOUBLE dLong UINT Period DOUBLE dPower	set_wobbel_vector(dTrans, dLong, Period, dPower).	-
353	R_LC_SET_WOBBEL_OFFSET	LONG OffsetTrans LONG OffsetLong	set_wobbel_offset(OffsetTrans, OffsetLong).	-
354	R_LC_SET_SKY_WRITING_PARA	DOUBLE Timelag LONG LasrOnShift UINT Nprev UINT Npost	set_sky_writing_para_list(Timelag, LaserOnShift, Nprev, Npost). For UINT Nprev = UINT Npost = 65,535: set_sky_writing_list(Timelag, LaserOnShift).	-
355	R_LC_SET_SKY_WRITING_ANGLE	DOUBLE Limit	set_sky_writing_limit_list(Limit).	-
356	R_LC_SET_SKY_WRITING_MODE	UINT Mode	set_sky_writing_mode_list(Mode).	-
357	R_LC_SET_CONTROL_MODE	UINT Mode	set_control_mode_list(Mode).	-
358	R_LC_STORE_ENCODER	UINT Pos	store_encoder(Pos).	-
359	R_LC_START_TIMER	-	save_and_restart_timer().	-
360	R_LC_STORE_TIMESTAMP	-	store_timestamp_counter().	-
361	R_LC_WAIT_FOR_TIMESTAMP	UINT Counter UINT Mode	wait_for_timestamp_counter_mode(TimeStampCounter , Mode). For UINT Mode > 2: wait_for_timestamp_counter(TimeStampCounter).	-
362	R_LC_SET_TRIGGER	UINT Period UINT Signal1 UINT Signal2 UINT Signal3 UINT Signal4	set_trigger4(Period, Signal1, Signal2, Signal3, Signal4). For UINT Signal3 and/or UINT Signal4 unzulässig: set_trigger(Period, Signal1, Signal2).	-
363	R_LC_EXTERN_START_DELAY	LONG Delay UINT EncoderNo	set_ext_start_delay_list(Delay, EncoderNo).	-
364	R_LC_SET_FLY	DOUBLE Scale UINT Axis UINT No	For UINT Axis = 1 and UINT No = does not matter: set_fly_x(ScaleX) with ScaleX = DOUBLE Scale. For UINT Axis = 2 and UINT No = does not matter: set_fly_y(ScaleY) with ScaleY = DOUBLE Scale. For UINT Axis = 3 and UINT No = EncoderNo: set_fly_z(ScaleZ, EncoderNo) with [ScaleZ = DOUBLE Scale, UINT No] and EncoderNo = UINT No. For UINT Axis = 4 and UINT No = does not matter: set_fly_rot(Resolution) with Resolution = DOUBLE Scale.	-
365	R_LC_ROT_CENTER	LONG X LONG Y	set_rot_center_list(X, Y).	-
366	R_LC_SET_FLY_2D	DOUBLE ScaleX DOUBLE ScaleY	set_fly_2d(ScaleX, ScaleY).	-

Remote List Commands, page 1016

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
367	R_LC_FLY_RETURN_Z	LONG X LONG Y LONG Z	fly_return_z(X, Y, Z). For LONG Z = 0x7FFFFFFF: fly_return(X, Y).	–
368	R_LC_SET_FLY_LIMITS	LONG Xmin LONG Xmax LONG Ymin LONG Ymax	set_fly_limits(Xmin, Xmax, Ymin, Ymax).	–
369	R_LC_SET_FLY_LIMITS_Z	LONG Zmin LONG Zmax	set_fly_limits_z(Zmin, Zmax).	–
370	R_LC_IF_FLY_OVERFLOW	LONG Mode UINT Axis	For UINT Axis = 1: if_fly_x_overflow(Mode). For UINT Axis = 2: if_fly_y_overflow(Mode). For UINT Axis = 3: if_fly_z_overflow(Mode).	–
371	R_LC_IF_NOT_FLY_OVERFLOW	LONG Mode UINT Axis	For UINT Axis = 1: if_not_fly_x_overflow(Mode). For UINT Axis = 2: if_not_fly_y_overflow(Mode). For UINT Axis = 3: if_not_fly_z_overflow(Mode).	–
372	R_LC_CLEAR_FLY_OVERFLOW	UINT Mode	clear_fly_overflow(Mode).	–
373	R_LC_PARK_POSITION	UINT Mode LONG X LONG Y	park_position(Mode, X, Y).	–
374	R_LC_PARK_RETURN	UINT Mode LONG X LONG Y	park_return(Mode, X, Y).	–
375	R_LC_ACTIVATE_FLY_ENC_XY	DOUBLE ScaleX DOUBLE ScaleY LONG EncX LONG EncY	activate_fly_xy_encoder(ScaleX, ScaleY, EncX, EncY). For LONG EncX = LONG EncY = 0: activate_fly_xy(ScaleX, ScaleY).	–
376	R_LC_ACTIVATE_FLY_ENC_2D	DOUBLE ScaleX DOUBLE ScaleY LONG EncX LONG EncY	activate_fly_2d_encoder(ScaleX, ScaleY, EncX, EncY). For LONG EncX = LONG EncY = 0: activate_fly_2d(ScaleX, ScaleY).	–
377	R_LC_WAIT_FOR_ENCODER_MODE	LONG Value UINT Encoder UINT Mode	wait_for_encoder_mode(Value, EncoderNo, Mode). UINT Mode = 0: wait_for_encoder(Value, EncoderNo).	–
378	R_LC_WAIT_FOR_ENCODER_RANGE	LONG EncXmin LONG EncXmax LONG EncYmin LONG EncYmax UINT Mode	wait_for_encoder_in_range_mode(EncXmin, EncXmax, EncYmin, EncYmax, Mode). For UINT Mode = 0: wait_for_encoder_in_range(EncXmin, EncXmax, EncYmin, EncYmax).	–
379	R_LC_IF_NOT_ACTIVATED	–	if_not_activated().	–
380	R_LC_RANGE_CHECKING	UINT Head UINT Mode UINT Data	range_checking(HeadNo, Mode, Data).	–

Remote List Commands, page 1016

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
381	R_LC_SET_FLY_POS	DOUBLE Scale UINT Axis	For UINT Axis = 1: set_fly_x_pos(ScaleX) with ScaleX = DOUBLE Scale. For UINT Axis = 2: set_fly_y_pos(ScaleY) with ScaleY = DOUBLE Scale. For UINT Axis = 4: set_fly_rot_pos(Resolution) with DOUBLE Scale = Resolution.	-
382	R_LC_WAIT_FOR_MCBSP	UINT Axis LONG Value LONG Mode	wait_for_mcbsp(Axis, Value, Mode).	-
383	R_LC_SET_MCBSP_IN	UINT Mode DOUBLE Scale	set_mcbsp_in_list(Mode, Scale).	-
384	R_LC_SET_MULTI_MCBSP_IN	UINT Ctrl UINT P UINT Mode	set_multi_mcbsp_in_list(Ctrl, P, Mode).	-
385	R_LC_SET_MCBSP_MODE	UINT Axis DOUBLE Scale	For UINT Axis = 1: set_mcbsp_x_list(Scale). For UINT Axis = 2: set_mcbsp_y_list(Scale). For UINT Axis = 4: set_mcbsp_rot_list(Resolution) with Resolution = DOUBLE Scale). For UINT Axis = 0 and DOUBLE Scale = does not matter: set_mcbsp_matrix_list().	-
386	R_LC_APPLY_MCBSP	UINT HeadNo UINT at_once	apply_mcbsp_list(HeadNo, at_once).	-
387	R_LC_SET_MCBSP_MODE_GLOBAL	UINT Axis DOUBLE Scale	For UINT Axis = 1: set_mcbsp_global_x_list(Scale). For UINT Axis = 2: set_mcbsp_global_y_list(Scale). For UINT Axis = 4: set_mcbsp_global_rot_list(Resolution) with Resolution = DOUBLE Scale. For UINT Axis = 0 and DOUBLE Scale = does not matter: set_mcbsp_global_matrix_list().	-
388	R_LC_SET_FLY_1	UINT Axis UINT Mode DOUBLE Scale	set_fly_1_axis(Axis, Mode, Scale).	-
389	R_LC_FLY_RETURN_1	UINT Axis LONG RetPos1	fly_return_1_axis(Axis, RetPos1).	-
390	R_LC_WAIT_FOR_1	LONG Value UINT Mode LONG WaitMode UINT LaserMode	wait_for_1_axis(Value, Mode, WaitMode, LaserMode).	-
391	R_LC_ACTIVATE_FLY_1	UINT Axis UINT Mode DOUBLE Scale LONG Offset	activate_fly_1_axis(Axis, Mode, Scale, Offset).	-
392	R_LC_PARK_POSITION_1	UINT Mode UINT Axisv LONG ParkPos	park_position_1_axis(Mode, Axis, ParkPos).	-

Remote List Commands, page 1016

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
393	R_LC_PARK_RETURN_1	UINT Mode UINT Axis LONG RetPos	park_return_1_axis(Mode, Axis, RetPos).	-
394	R_LC_SET_FLY_2	UINT Axis1 UINT Mode1 DOUBLE Scale1 UINT Axis2 UINT Mode2 DOUBLE Scale2	set_fly_2_axes(Axis1, Mode1, Scale1, Axis2, Mode 2, Scale2).	-
395	R_LC_FLY_RETURN_2	UINT Axis2 LONG RetPos1 UINT Axis2 LONG RetPos2	set_fly_2_axes(Axis1, Mode1, Scale1, Axis2, Mode 2, Scale2).	-
396	R_LC_WAIT_FOR_2	UINT ModeX LONG MinValueX LONG MaxValueX UINT ModeY LONG MinValueY LONG MaxValueY LONG WaitMode UINT LaserMode	wait_for_2_axes(ModeX, MinValueX, MaxValueX, ModeY, MinValueY, MaxValueY, WaitMode, LaserMode).	-
397	R_LC_ACTIVATE_FLY_2	UINT ModeX DOUBLE ScaleX LONG OffsetX UINT ModeY DOUBLE ScaleY LONG OffsetY	activate_fly_2_axes(ModeX, ScaleX, OffsetX, ModeY, ScaleY, OffsetY).	-
398	R_LC_PARK_POSITION_2	UINT Mode LONG ParkPosX LONG ParkPosY	park_position_2_axes(Mode, ParkPosX, ParkPosY).	-
399	R_LC_PARK_RETURN_2	UINT Mode LONG RetPosX LONG RetPosY	park_return_2_axes(Mode, RetPosX, RetPosY).	-
400	R_LC_SET_FLY_3	UINT ModeX DOUBLE ScaleX UINT ModeY DOUBLE ScaleY UINT ModeZ DOUBLE ScaleZ	set_fly_3_axes(ModeX, ScaleX, ModeY, ScaleY, ModeZ, ScaleZ).	-
401	R_LC_FLY_RETURN_3	LONG RetPosX LONG RetPosY LONG RetPosZ	fly_return_3_axes(RetPosX, RetPosY, RetPosZ).	-
402	R_LC_RS232_WRITE_TEXT	CHAR Text[N] siehe 2, page 1027	rs232_write_text_list(pData) with pData = CHAR Text[N].	-
403	R_LC_SET_MCBSP_OUT	UINT Signal1 UINT Signal2	set_mcbsp_out(Signal1, Signal2).	-

Remote List Commands, page 1016

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
404	R_LC_PERIODIC_TOGGLE	UINT Port UINT Mask UINT P1 UINT P2 UINT Count UINT Stop	periodic_toggle_list(Port, Mask, P1, P2, Count, Start).	-
405	R_LC_SET_FREE_VAR	UINT VarNo UINT Value	set_free_variable_list(No, Value) with No = UINT VarNo.	-
406	R_LC_SELECT_COR_TABLE	UINT HeadA UINT HeadB	select_cor_table_list(HeadA, HeadB).	-
407	R_LC_SET_DELAY_MODE	UINT VarPloly UINT DirectMove3D UINT EdgeLevel UINT MinJumpDelay UINT JumpLengthLimit	set_delay_mode_list(VarPoly, DirectMove3D, EdgeLevel, MinJumpDelay, JumpLengthLimit).	-
408	R_LC_SCANAHEAD_ACTIVATE_AUTO	LONG Mode	activate_scanahead_autodelays_list(Mode).	-
409	R_LC_SET_SCANAHEAD_LASER_SHIFT	LONG dLasOn LONG dLasOff	set_scanahead_laser_shifts_list(dLasOn, dLasOff).	-
410	R_LC_SET_SCANAHEAD_LINE_PARAMS	UINT CornerScale UINT EndScale UINT AccScale UINT JumpScale	Reserved.	-
411	R_LC_STEPPER_ENABLE	LONG Enable1 LONG Enable2	stepper_enable_list(Enable1, Enable2).	-
412	R_LC_STEPPER_CTRL	LONG Period1 LONG Period2	stepper_control_list(Period1, Period2).	-
413	R_LC_STEPPER_NO_ABS	UINT No LONG Pos	stepper_abs_no_list(No, Pos).	-
414	R_LC_STEPPER_NO_REL	UINT No LONG dPos	stepper_rel_no_list(No, dPos).	-
415	R_LC_STEPPER_ABS	LONG Pos1 LONG Pos2	stepper_abs_list(Pos1, Pos2).	-
416	R_LC_STEPPER_REL	LONG dPos1 LONG dPos2	stepper_rel_list(dPos1, dPos2).	-
417	R_LC_STEPPER_WAIT	UINT No	stepper_wait(No).	-
418	R_LC_CAMMING	UINT FirstPos UINT NPos UINT EncoderNo UINT Ctrl DOUBLE Scale	camming (FirstPos, NPos, EncoderNo, Ctrl, Scale, Code).	-
419	R_LC_SET_JUMP_MODE	LONG Flag	set_jump_mode_list(Flag).	-
420	R_LC_ENDURING_WOBBEL	-	Reserved.	-
421	R_LC_FLY_PREDICTION	UINT Prediction UINT PredictionY	fly_prediction(PredictionX, PredictionY).	-

Remote List Commands, page 1016

ID	Remote List Command	Command Telegram Parameter	Functionality	Answer Telegram Parameter
422	R_LC_WAIT_FOR_TIMESTAMP_LONG	UINT WaitCounterL UINT WaitCounterH UINT MaxWaitTime UINT Mode	wait_for_timestamp_counter_long(WaitCounterL, WaitCounterH, MaxWaitTime, Mode).	–
423	R_LC_SET_TRIGGER_8	UINT Period UINT Signals[8]	set_trigger8(Period, Signal1, Signal2, Signal3, Signal4, Signal5, Signal6, Signal7, Signal8).	–
424	R_LC_SET_WOBBEL_VECTOR_2	DOUBLE dTrans DOUBLE dLong UINT Period DOUBLE dPower DOUBLE dPower2 UINT Ctrl	set_wobbel_vector_2(dTrans, dLong, Period, dPower, dPower2, Ctrl).	–
425	R_LC_SET_MCBSP_OUT_PTR	UINT Num UINT SignalPtr[Num]	set_mcbsp_out_ptr_list(Number, SignalPtr).	–
426	R_LC_SET_MCBSP_OUT_OIE	UINT Signal1 UINT Signal2	set_mcbsp_out_oie_list(Signal1, Signal2).	–
427	R_LC_WRITE_PORT	UINT Port UINT Value UINT NoDelay	write_port_list(Port, Value, NoDelay).	–
428	R_LC_FLY_DISABLE	–	fly_disable_list( )	–
429	R_LC_INIT_FLY_2D	LONG OffsetX LONG OffsetY UINT No	init_fly_2d_list(OffsetX, OffsetY, No).	–
430	R_LC_SET_DEFOCUS_2	LONG Shift UINT Mode	set_defocus_2_list(Shift, Mode).	–
…	…	…	…	…
511	–	–	–	–

Remote List Commands, page 1016

Notes

(1) On 341 R_LC_MARK_TEXT, CHAR Text[N].

You must select N so that the telegram total length does not exceed TGM_MAX_SIZE. If necessary, distribute the text to several Remote Commands. You must terminate each of the texts by “\0”.

(2) On 402 R_LC_RS232_WRITE_TEXT, CHAR Text[N].

You must select N so that the telegram total length does not exceed TGM_MAX_SIZE. If necessary, distribute the text to several Remote Commands. You must terminate each of the texts by “\0”.

16.9.10 RTC6 Commands not Available as Remote Commands

RTC6 command	Reason, Alternative, Comment
acquire_rtc	In Remote Interface Mode not relevant: Access rights.
auto_cal	In Remote Interface Mode not relevant: Only for some discontinued systems.
copy_dst_src	In Remote Interface Mode not relevant: Accessing the file system. RTC6 List Memory fragmentation is to be avoided in Remote Interface Mode.
create_dat_file	In Remote Interface Mode not relevant: Accessing the file system.
eth_assign_card	In Remote Interface Mode not relevant: Search for RTC6 boards.
eth_assign_card_ip	In Remote Interface Mode not relevant: RTC6 board management. If needed: To be implemented by the user.
eth_boot_dcmd	In Remote Interface Mode not relevant: Standalone Operation Mode is prerequisite, page 997.
eth_boot_timeout	In Remote Interface Mode not relevant: Standalone Operation Mode is prerequisite, page 997.
eth_check_connection	To be implemented by the user. Send any control command and check answer.
eth_convert_ip_to_string	If needed: To be implemented by the user.
eth_convert_string_to_ip	If needed: To be implemented by the user.
eth_count_cards	In Remote Interface Mode not relevant: RTC6 board management. If needed: To be implemented by the user.
eth_found_cards	In Remote Interface Mode not relevant: Search for RTC6 boards.
eth_get_card_info	In Remote Interface Mode not relevant: RTC6 board management.
eth_get_card_info_search	In Remote Interface Mode not relevant: Search for RTC6 boards.
eth_get_com_timeouts	Deprecated.
eth_get_com_timeouts_auto	To be implemented by the user. Refer to C++ Remote Interface Wrapper: **eth_set_com_timeouts_auto**.
eth_get_error	To be implemented by the user. Refer to C++ Remote Interface Wrapper: **get_network_error**.

RTC6 command (cont'd.)	Reason, Alternative, Comment (cont'd.)
eth_get_ip	In Remote Interface Mode not relevant: RTC6 board management.
eth_get_ip_search	In Remote Interface Mode not relevant: Search for RTC6 boards.
eth_get_last_error	To be implemented by the user. Refer to C++ Remote Interface Wrapper: **get_network_error**.
eth_get_serial_search	In Remote Interface Mode not relevant: Search for RTC6 boards.
eth_get_standalone_status	In Remote Interface Mode not relevant: Standalone Operation Mode is prerequisite, page 997.
eth_max_card	In Remote Interface Mode not relevant: RTC6 board management. If needed: To be implemented by the user.
eth_remove_card	In Remote Interface Mode not relevant: RTC6 board management. If needed: To be implemented by the user.
eth_search_cards	In Remote Interface Mode not relevant: Search for RTC6 boards.
eth_search_cards_range	In Remote Interface Mode not relevant: Search for RTC6 boards.
eth_set_com_timeouts	Deprecated.
eth_set_com_timeouts_auto	To be implemented by the user. Refer to C++ Remote Interface Wrapper: **eth_set_com_timeouts_auto**.
eth_set_high_performance_mode	If needed: To be implemented by the user.
eth_set_remote_tgm_format	In Remote Interface Mode not relevant: Standalone Operation Mode is prerequisite, page 997.
eth_set_search_cards_timeout	In Remote Interface Mode not relevant: Search for RTC6 boards.
free_rtc6_dll	In Remote Interface Mode not relevant: RTC6 DLL.
get_auto_cal	In Remote Interface Mode not relevant: Only for some discontinued systems.
get_card_type	In Remote Interface Mode not relevant: only RTC6 Ethernet Boards are supported.
get_char_pointer	In Remote Interface Mode not relevant: Accessing the file system. RTC6 List Memory fragmentation is to be avoided in Remote Interface Mode.

RTC6 command (cont'd.)	Reason, Alternative, Comment (cont'd.)
get_dll_version	In Remote Interface Mode not relevant: RTC6 DLL.
get_error	To be implemented by the user. Refer to C++ Remote Interface Wrapper: **get_error**.
get_hi_data	In Remote Interface Mode not relevant: Only for some discontinued systems.
get_hi_pos	In Remote Interface Mode not relevant: Only for some discontinued systems.
get_jump_table	In Remote Interface Mode not relevant: Accessing the file system.
get_last_error	To be implemented by the user. Refer to C++ Remote Interface Wrapper: **get_last_error**.
get_mcbsp_list	No function.
get_out_pointer	**get_status**
get_rtc_mode	If needed: To be implemented by the user. In addition: The line RTC4→RTC6 or RTC5→RTC6 must be observed with each RTC6 command.
get_sub_pointer	In Remote Interface Mode not relevant: Accessing the file system. RTC6 List Memory fragmentation is to be avoided in Remote Interface Mode.
get_text_table_pointer	In Remote Interface Mode not relevant: Accessing the file system. RTC6 List Memory fragmentation is to be avoided in Remote Interface Mode.
get_transform	To be implemented by the user. Using, for example, CalibrationLibrary.
get_transform_offset	To be implemented by the user. Using, for example, CalibrationLibrary.
init_rtc6_dll	In Remote Interface Mode not relevant: RTC6 DLL.
load_auto_laser_control	In Remote Interface Mode not relevant: Accessing the file system.
load_disk	In Remote Interface Mode not relevant: Accessing the file system. RTC6 List Memory fragmentation is to be avoided in Remote Interface Mode.
load_fly_2d_table	In Remote Interface Mode not relevant: Accessing the file system.
load_jump_table	In Remote Interface Mode not relevant: Accessing the file system.

RTC6 command (cont'd.)	Reason, Alternative, Comment (cont'd.)
load_jump_table_offset	In Remote Interface Mode not relevant: Accessing the file system.
load_position_control	In Remote Interface Mode not relevant: Accessing the file system.
load_program_file	In Remote Interface Mode not relevant: Standalone Operation Mode is prerequisite, page 997 and therefore, has been already executed.
load_stretch_table	In Remote Interface Mode not relevant: Accessing the file system.
load_varpolydelay	In Remote Interface Mode not relevant: Accessing the file system.
load_zoom_correction_file	In Remote Interface Mode not relevant: Accessing the file system.
move_to	In Remote Interface Mode not relevant: Special command.
read_abc_from_file	To be implemented by the user. Using, for example, CalibrationLibrary.
read_abc_from_file_20b	To be implemented by the user. Using, for example, CalibrationLibrary.
read_image_eth	In Remote Interface Mode not relevant: Standalone Operation Mode is prerequisite, page 997.
read_user_data	No function.
regulation3	In Remote Interface Mode not relevant: Special command.
release_rtc	In Remote Interface Mode not relevant: Access Rights.
reset_error	To be implemented by the user. Refer to C++ Remote Interface Wrapper: **reset_error**.
rs232_config	**uart_config**
rtc6_count_cards	In Remote Interface Mode not relevant: Only RTC6 Ethernet Boards are supported.
save_disk	In Remote Interface Mode not relevant: Accessing the file system. RTC6 List Memory fragmentation is to be avoided in Remote Interface Mode.
select_rtc	In Remote Interface Mode not relevant: Access Rights. In Remote Interface Mode not relevant: Multi-board commands.

RTC6 command (cont'd.)	Reason, Alternative, Comment (cont'd.)
send_user_data	No function.
set_char_pointer	In Remote Interface Mode not relevant: Accessing the file system. RTC6 List Memory fragmentation is to be avoided in Remote Interface Mode.
set_char_table	**set_text_table_pointer**
set_dsp_mode	In Remote Interface Mode not relevant: Only for some discontinued systems.
set_duty_cycle_table	In Remote Interface Mode not relevant: Special command.
set_eth_boot_control	In Remote Interface Mode not relevant: Standalone Operation Mode is prerequisite, page 997.
set_jump_table	In Remote Interface Mode not relevant: Accessing the file system.
set_laser_timing	**set_laser_pulses**
set_laser_timing_table	In Remote Interface Mode not relevant: Special command.
set_list_jump	**list_jump_pos**
set_rtc4_mode	If needed: To be implemented by the user. In addition: The line RTC4→RTC6 must be observed with each RTC6 command.
set_rtc5_mode	If needed: To be implemented by the user. In addition: The line RTC5→RTC6 must be observed with each RTC6 command.
set_rtc6_mode	In Remote Interface Mode is standard case.
set_softstart_level	No function.
set_softstart_level_list	No function.
set_softstart_mode	No function.
set_softstart_mode_list	No function.

RTC6 command (cont'd.)	Reason, Alternative, Comment (cont'd.)
set_sub_pointer	In Remote Interface Mode not relevant: Accessing the file system. RTC6 List Memory fragmentation is to be avoided in Remote Interface Mode.
set_text_table_pointer	In Remote Interface Mode not relevant: Accessing the file system. RTC6 List Memory fragmentation is to be avoided in Remote Interface Mode.
store_program	In Remote Interface Mode not relevant: Standalone Operation Mode is prerequisite, page 997.
sync_slaves	No function.
transform	To be implemented by the user. Using, for example, CalibrationLibrary.
upload_transform	To be implemented by the user. Using, for example, CalibrationLibrary.
verify_checksum	In Remote Interface Mode not relevant: Accessing the file system. In Remote Interface Mode not relevant: Standalone Operation Mode is prerequisite, page 997 (**load_program_file** verifies the checksum there).
write_abc_to_file	To be implemented by the user. Using, for example, CalibrationLibrary.
write_abc_to_file_20b	To be implemented by the user. Using, for example, CalibrationLibrary.
write_hi_pos	In Remote Interface Mode not relevant: Only for some discontinued systems.
write_image_eth	In Remote Interface Mode not relevant: Standalone Operation Mode is prerequisite, page 997.

16.9.11 Software Development Notes

- The initial `seqnum` should be set[1] at user program start. For this purpose send a Command Telegram with 0x12345678 as 1st `uint32_t` value in TGM_PL_CMD_RAW. Then, the RTC6 Ethernet Board returns the last executed `seqnum`.

(1) In C++ Remote Interface Wrapper, the initial `seqnum` is set automatically.

16.9.12 About C++ Remote Interface Wrapper

The total functionality from

- `rtc6_rif_wrapper.cpp`
  (Source file for C++ Remote Interface Wrapper)
- `rtc6_rif_wrapper.h`
  (Header file for C++ Remote Interface Wrapper)
- `telegrams.h`
  (Header file for `rtc6_rif_wrapper.h`. Defines the Command Telegram structure as well as Remote Command IDs)

is referred to in this document as:

- C++ Remote Interface Wrapper

The C++ Remote Interface Wrapper offers the basic logic for:

- Generating Command Telegrams
- Sending[(1)(2)] Command Telegrams
- Receiving of Command Telegrams
- Retry handling
- A simple error detection

For sending and receiving Command Telegrams to work, you must provide a platform-specific implementation for the following virtual socket functions, see Figure 91 as well as the example implementation `example_1.cpp` from RTC6 Software Package:

- `send`
- `recv`
- `select`

The C++ Remote Interface Wrapper provides a similar interface to `RTC6impl.h` and `RTC6impl.hpp`.

Signatures of the wrapper functions (in `rtc6_rif_wrapper.h`) and RTC6 DLL are largely the same.

Only in the C++ Remote Interface Wrapper are available:

- wrapper function **get_remote_error**
- wrapper function **disable_remote_interface**
- wrapper function **get_network_error**

Not available in the C++ Remote Interface Wrapper:

- RTC6 command **eth_get_last_error**
- RTC6 command **eth_get_error**

  Replacement is the wrapper function get_network_error with 2 predefined error codes which can be set by C++ Remote Interface Wrapper:
  - `NETWORK_ERR_TIMEOUT`
    A timeout occurred while receiving a Answer Telegram, see also **eth_set_com_timeouts_auto**.
  - `NETWORK_ERR_WRONG_ID`
    The wrapper function IDs do not match in Answer Telegram and Command Telegram.

  You can define the remaining error codes yourself and correspond to the last return value of `send` / `recv` / `select`.

Notes

- **In C++ Remote Interface Wrapper, the initial `seqnum` is set automatically.**

(1) Command Telegrams are to be sent to the RTC6 Ethernet Board via UDP. The port to be used is the same as for conventional transmission (**eth_get_port_numbers**(`UDPexcl`), **eth_set_port_numbers**(`UDPexcl`)).

(2) Only 1 list command (not several) per Command Telegram.

To Use **C++ Remote Interface Wrapper**:

(1) Create a derived class of NetworkAdapter.

(2) Create a datagram socket (UDP) with the IP address of the desired RTC6 Ethernet Board and port 63750[(1)].

(3) Implement the platform specific socket functionality in the virtual functions `send` / `recv` / `select`.

(4) Create an RTC object and pass NetworkAdapter implementation.

```
// Pseudo code (not complete).

// Send specified amount of bytes.
// Return 0 if successful, a negative error code otherwise.
   virtual int32_t send(const char* buf, size_t len) = 0;

// Receive up to len bytes and write them into buf.
// Return 0 if successful, a negative error code otherwise.
// If the received telegram should be discarded return a positive value.
   virtual int32_t recv(char* buf, size_t len) = 0;

// Determine the read status. Return the number of read operations available.
// Return 0 if no data is available after timeout_us microseconds,
// a negative error code otherwise.
   virtual int32_t select(long timeout_us) = 0;
```

91

See step 3 as well as explanations, page 1035.

16.10 Legal

16.10.1 EU Declaration of Conformity – RTC6 Ethernet Board

EU-Konformitätserklärung

für das Produkt:

RTC4 Ethernet, RTC6 Ethernet

Der Hersteller

SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Deutschland

erklärt, dass das genannte Produkt die einschlägigen Harmonisierungsrechtsvorschriften der Union erfüllt:

- **2011/65/EU -** Richtlinie des EUROPÄISCHEN PARLAMENTS UND DES RATES zur Beschränkung der Verwendung bestimmter gefährlicher Stoffe in Elektro- und Elektronikgeräten (RoHS-Richtlinie).

Folgende harmonisierte Normen wurden angewandt:

- **DIN EN IEC 63000:2019-05** - Technische Dokumentation zur Beurteilung von Elektro- und Elektronikgeräten hinsichtlich der Beschränkung gefährlicher Stoffe

Die alleinige Verantwortung für die Ausstellung dieser Konformitätserklärung trägt der Hersteller.

Dieses Dokument zur Kundeninformation wurde elektronisch ausgestellt und ist daher ohne Unterschrift gültig. Eine unterschriebene Ausführung ist Bestandteil der SCANLAB-internen technischen Dokumentation.

EU Declaration of Conformity

for the product:

RTC4 Ethernet, RTC6 Ethernet

The manufacturer

SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Germany

declares that the product mentioned above complies with the relevant Union harmonization legislation:

- **2011/65/EU** - Directive of the EUROPEAN PARLIAMENT AND OF THE COUNCIL on the restriction of the use of certain hazardous substances in electrical and electronic equipment (RoHS Directive).

The following harmonized standards have been applied:

- **DIN EN IEC 63000:2019-05** - Technical documentation for the assessment of electrical and electronic products with respect to the restriction of hazardous substances

This declaration of conformity is issued under the sole responsibility of the manufacturer.

This document, used for customer information, was issued electronically and is therefore valid without a signature. A signed version is part of the SCANLAB internal technical documentation.

SCANLAB GmbH
Puchheim

2021-07-19

16.10.2Compliance with FCC Rules

The RTC6 Ethernet Board has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules.

These limits are designed to provide reasonable protection against harmful interference when the RTC6 Ethernet Board is operated in a commercial environment. The RTC6 Ethernet Board generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with this instruction manual, may cause harmful interference to radio communications. Operation of the RTC6 Ethernet Board in a residential area is likely to cause harmful interference in which case the user is required to correct the interference at his own expense.

16.10.3TI-RTOS

TI-RTOS is used in the RTC6 Ethernet Board.

16.11 Technical Specifications – RTC6 Ethernet Board

System Requirements

Windows PC with Ethernet interface (Gigabit Ethernet (preferred), 10/100 Mbit/s Ethernet)

Operating system	Microsoft Windows 10, 8, 7 as 32 bit or 64 bit version.

Dimensions (without plugged-in plug connectors)

Length	120 mm
Width	103 mm
Height	Approx. 30 mm

Other Connections and Specifications

Voltage supply	+11 V…+50 V
Maximum power consumption, no peripherals attached	< 10 W

In order to minimize power loss, a low input voltage should be selected.

Operating temperature

• with natural convection	10 °C…50 °C
• with forced convection	10 °C…60 °C

The user must ensure sufficient cooling of the RTC6 Ethernet Board.

Standalone functionality	Yes, see Chapter 16.7 "Standalone Functionality", page 988.

Interface to the Network

Ethernet	Gigabit Ethernet (preferred), 10/100 Mbit/s Ethernet
IP address obtaining	• By a DHCP server • Link-local address (as of BIOS 24)[a] • Static IP address (storable on the board)
Required local ports[b]	• 63749 (UDP) • 63750 (UDP, TCP)

(a) See page 979.

(b) Configurable by **eth_set_port_numbers**.

Scan System Control

Number of RTC6 List Memory areas	Up to 3, configurable
Total capacity of the RTC6 List Memory	8,388,608 list positions
Output interval of Microsteps	10 μs
Maximum value range for the Image Field coordinates	–524,288…+524,287 (20 bit, signed)
Virtual Image Field, e.g. for Processing-on-the-fly	–268,435,456…+268,435,455 (28 bit, signed = 29 bit)[a]

(a) With RTC6 Software Package ≥ 1.4.0.

Interfaces to Scan Systems

SCANHEADs Socket Connector

Connector	20 pin socket connector with 2.54 mm pitch of the pins[a]. The signals for the first scan head are outputted at pin (01)…pin (10). The signals for the second scan head are outputted at pin (11)…pin (20): • xy output values only with enabled Option “Second Scan Head Control” • z output values only with enabled Option “3D”
Signal transmission	SL2-100 protocol. Via XY2-100 Converter (Accessory): XY2-100 protocol

(a) *Not* as with RTC6 PCIe Board.

LASER Socket Connector

Connector	16 pin socket connector with 2.54 mm pitch of the pins[a]
Laser Control Signals	LASER1, LASER2, LASERON
• TTL level	5 V, active-HIGH or active-LOW programmable
• Max. current load	20 mA
• Reference	GND - boards without Option “DC/DC Converter”. GND2 - boards with Option “DC/DC Converter”
Analog output ports	ANALOG OUT1, ANALOG OUT2[b]
• Output voltage range	0 V…10 V
• Resolution	12 Bit
• Max. current load	5 mA
• Reference	GND
Digital input ports	2 Bits
• LOW level	< 0.6 V
• HIGH level	> 2.3 V
• Max. input voltage range	–0.5 V…+5.5 V
• Input resistance (pull-up)	> 4.7 kΩ
• Reference	GND
Digital output port	2 Bits, buffered
• LOW level	< 0.55 V
• HIGH level	> 3.8 V
• Max. current load	20 mA
• Reference	GND

(a) *Not* as with RTC6 PCIe Board.

(b) With RTC5 boards and RTC6 PCIe Boards, the ANALOG OUT2 signal is also available at the MARKING ON THE FLY Socket Connector.

LASER Socket Connector (cont'd)

Input ports for external start and stop signals	TTL active-LOW, internally connected to +3.3 V by pull-up resistors (4.7 kΩ)
• /START	Edge sensitive
• /STOP	Level sensitive
• Reference	GND
Other signals	
• BUSY OUT	5 V, TTL active-HIGH, Max. 10 mA, Reference GND
• +5 V	Max. 100 mA
• Reference	GND

EXTENSION 1 Socket Connector

Connector	40 pin socket connector with 2.00 mm pitch of the pins[(a)]. Output signal level is configurable by solder jumper field A
Digital output port	16 Bits, buffered
• LOW level	< 0.4 V
• HIGH level	> 2.0 V (3.3 V or 5 V)
• Max. current load	±8 mA
• Reference	GND
• LATCH-Signal	3.3 V or 5 V, TTL active-HIGH, 5 μs pulse, max. 10 mA
Digital input port	16 Bit, protected
• LOW level	< 0.5 V
• HIGH level	> 2.6 V…24 V
• Max. input voltage range	–0.5 V…+26 V
• Input resistance	> 10 kΩ
• Reference	GND
• SYNC-Signal	3.3 V or 5 V, TTL active-HIGH, square wave signal (5 μs pulse, 10 μs period), max. 10 mA
Other signals	
• BUSY OUT	3.3 V or 5 V, TTL active-HIGH, max. 10 mA
• VCC	3.3 V or 5 V, max. 100 mA
• +5 V	Max. 100 mA
• Reference	GND

(a) With RTC6 PCIe Boards: 2.54 mm pitch of the pins.

EXT. 2 Socket Connector

Connector	10 pin socket connector with 2.00 mm pitch of the pins. pin (09)[(a)] is configurable by solder jumper field B. pin (08)[(b)] is configurable by solder jumper field C
Digital output port	8 Bits, buffered
• LOW level	< 0.4 V
• HIGH level	> 2.0 V
• Max. current load	±8 mA
• Reference	GND
• LATCH-Signal	5 V, TTL active-HIGH, 5 μs pulse, max. 10 mA
Laser Control Signals	None[(c)(d)]
Other signals	
• +5 V	Max. 100 mA
• Reference	GND

(a) pin (09) of RTC6 Ethernet Boards = pin (17) of RTC6 PCIe Boards.
(b) pin (08) of RTC6 Ethernet Boards = pin (15) of RTC6 PCIe Boards.
(c) *Not* as with RTC6 PCIe Board.
(d) With RTC6 PCIe Boards: LASER1 and LASER2.

MOF Socket Connector[(a)]

Connector	14 pin socket connector with 2.00 mm pitch of the pins[(a)]
2 encoder input ports for incremental encoders	ENCODER X(1±, 2±) and ENCODER Y(1±, 2±), designed for 1 pair of standardized differential input signals (RS-422) each. HIGH level ≥ 2.0 V LOW level ≤ 0.8 V f ≤ 4 MHz
Analog output port	None[(b)(c)]
Input ports for external start and stop signals	/START2 /STOP2 Like /START, /STOP of LASER Socket Connector, page 1040
Other signals	
• BUSY OUT	Identical with BUSY OUT on LASER Socket Connector, page 1040
• +5 V	Max. 100 mA
• Reference	GND

(a) With RTC6 PCIe Boards: 16 pin socket connector with 2.54 mm pitch of the pins. Furthermore, the printed label on the board is MARKING ON THE FLY.
(b) *Not* as with RTC6 PCIe Board.
(c) The RTC6 Ethernet Board has no pin for the ANALOG OUT2 signal on its MOF Socket Connector.

SPI/ANA/UART Socket Connector

With RTC6 Ethernet Boards, the McBSP/ANALOG Socket Connector pins and RS232 Socket Connector pins of the RTC6 PCIe Board are integrated into the SPI/ANA/UART Socket Connector.

Connector	10 pin socket connector with 2.00 mm pitch of the pins.
Analog input ports	ANALOG IN0, ANALOG IN1
• Input voltage range	0 V…10 V
• Input impedance	> 5 kΩ[a]
• ADC resolution	12 Bit
• Reference	GND[b]
Pins for RS-232	
Input port	RxD
• Voltage range	Max. –25 V…+25 V
Output port	TxD
• Voltage range	Max. –13 V…+13 V
Reference	GND
Baud rate	300…115,200

Pins for McBSP, see also Section "McBSP Interface", page 87

• Transmitter signal level	3.3 V TTL
• Receiver signal level	3.3 V or 5 V TTL
• McBSP mode	Single Phase Frame Single Element per Frame 32 Bits per Element Data delay `XDelay` bits Data delay `RDelay` bits
• Reference	GND
SPI interface functionality	No.

(a) *Not* as with RTC6 PCIe Board.

(b) See Notes, page 961.

STEPPER Socket Connector

Connector	10 pin socket connector with 2.00 mm pitch of the pins.

Signals for controlling up to two stepper motors:

• Output ports ENABLE1, ENABLE2, DIRECTION1, DIRECTION2	5 V, TTL
• Output ports CLOCK1, CLOCK2	5 V, TTL active-HIGH, 5 μs pulse
• Input ports SWITCH1, SWITCH2	TTL active-LOW, internally connected to +3.3 V by pull-up resistors (10 kΩ)
• Max. current load	25 mA
• Reference	GND

17 Appendix B: UFPM Extension Board

The UFPM Extension Board[1] has been developed for RTC6 PCIe Boards[2].

The assembly is shown in Figure 92, dimensions and details in Figure 93.

(1) For example, #0137980, #0140965, #0152803.

(2) Cannot be uses with RTC6 Ethernet Boards for mechanical reasons.

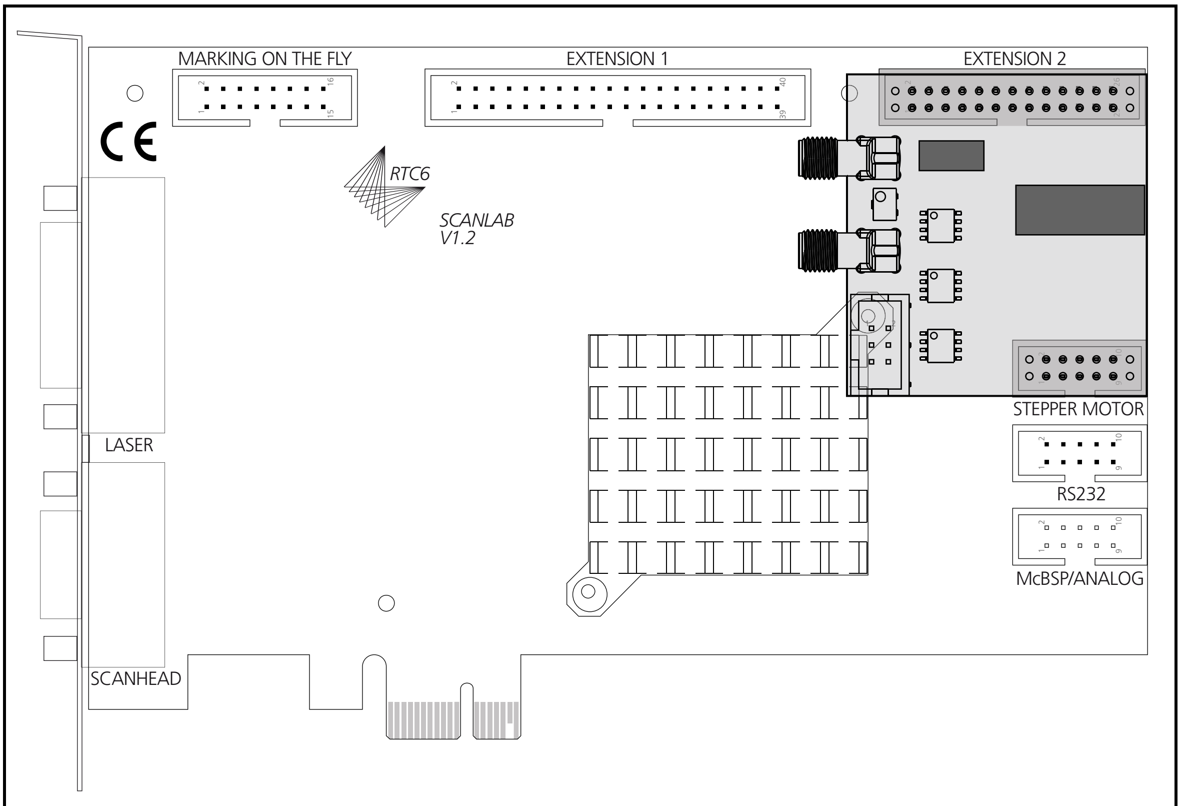
The UFPM Extension Board converts 8-bit digital signals into analog voltage values using a fast digital-to-analog converter.

It is recommended, if:

- *Analog* controlled lasers are operated *and*
- Pixel output frequencies ≥ 100 kHz[3][4] are to be achieved (see **set_pixel_line**)

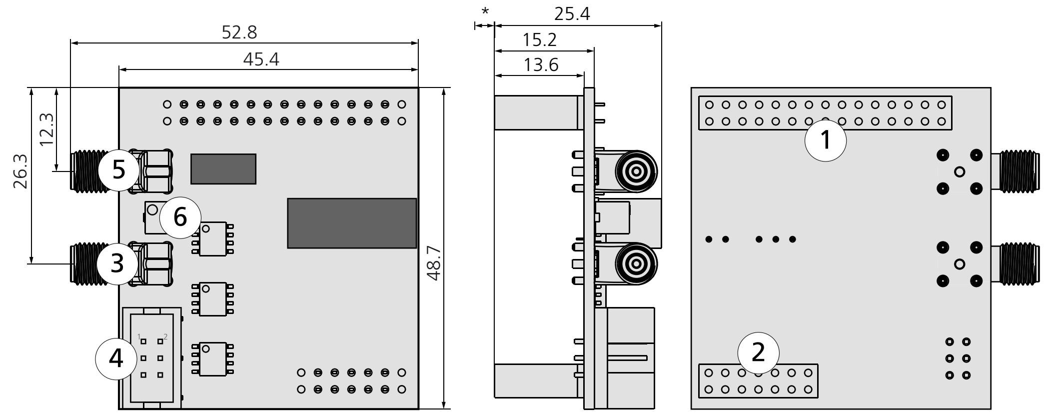
(3) Option "UFPM" is mandatory for pixel output frequencies 800 kHz…3,2 MHz.

(4) The UFPM Extension Board also supports pixel output frequencies < 100 kHz, of course.



92

UFPM Extension Board: Assembly with the RTC6 PCIe Board.
Note: The Second Scan Head Slot Cover (Accessory) (#0115132) cannot be used.



* 3 mm (distance to the RTC6 PCB surface when mounted).

Legend

1 . . . .J1. 30-pin socket for connecting to the 26-pin EXTENSION 2 Socket Connector of the RTC6 PCIe Board. The outer 4 pins of connector J1 are unconnected and only serve to prevent misaligned insertion of the board.
2 . . . .J2. 14-pin socket for connecting to the 10-pin STEPPER MOTOR Socket Connector of the RTC6 PCIe Board. Socket J2 only serves to facilitate mechanical retention to the RTC board.
3 . . . .J3. SMA connector with analog output port ANALOG OUTA.
4 . . . .J4. 6-pin socket connector with analog output port ANALOG OUTA and ANALOG OUTB.
5 . . . .J5. SMA connector with analog output port ANALOG OUTB.
6 . . . .P3. Potentiometer for fine-tuning the 5 V amplitude of analog output ports ANALOG OUTA and ANALOG OUTB.

93

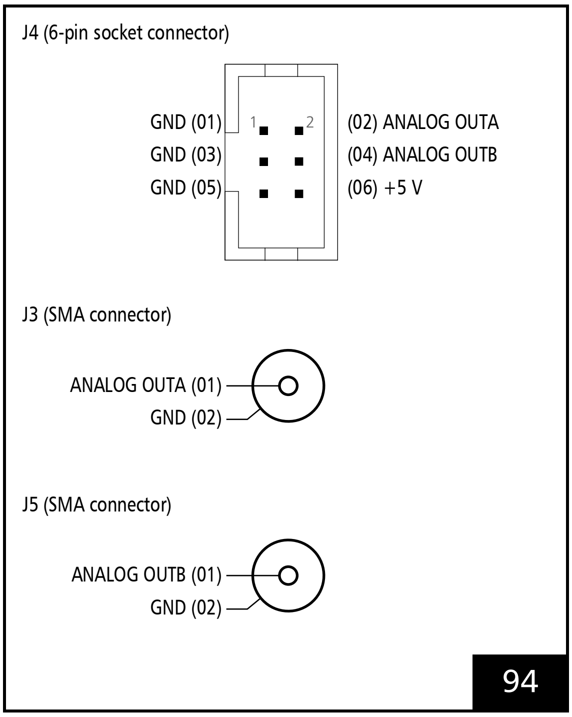
UFPM Extension Board: dimensions and details. All dimensions in mm.

The UFPM Extension Board can be used to control the pixel-to-pixel variation in laser power via analog voltage values with 8 bit resolution.
Output voltage range: 0 V…5 V.

For this purpose, the outputs must be sent to Port = 3 (8-Bit Digital Output Port at the EXTENSION 2 Socket Connector), see **set_pixel_line**.

For all Pixel Output Modes including their extensions, the laser pulse duration must be specified before the beginning of the pixel line by **set_laser_pulses**, **set_laser_pulses_ctrl** or **set_laser_timing** with at least 1/64 μs duration (even if the pulse duration is not used for laser control). Otherwise, no pulses are outputted at LASER1.

The UFPM Extension Board adapts the LATCH signal duration to the pixel frequency (approx. `HalfPeriod`...5 μs). It synchronously converts the 8-bit digital values to analog values in the range 0 V...5 V and makes them available at its analog output ports ANALOG OUTA and ANALOG OUTB. The connector pin-outs on the UFPM Extension Board are shown in Figure 94. Potentiometer P3 lets you fine-tune the exact voltage amplitude, see Figure 93.



UFPM Extension Board: connector pin-outs.

Software Requirements

- The UFPM Extension Board is supported by default by the RTC6 Software Package package.

Hardware Requirements

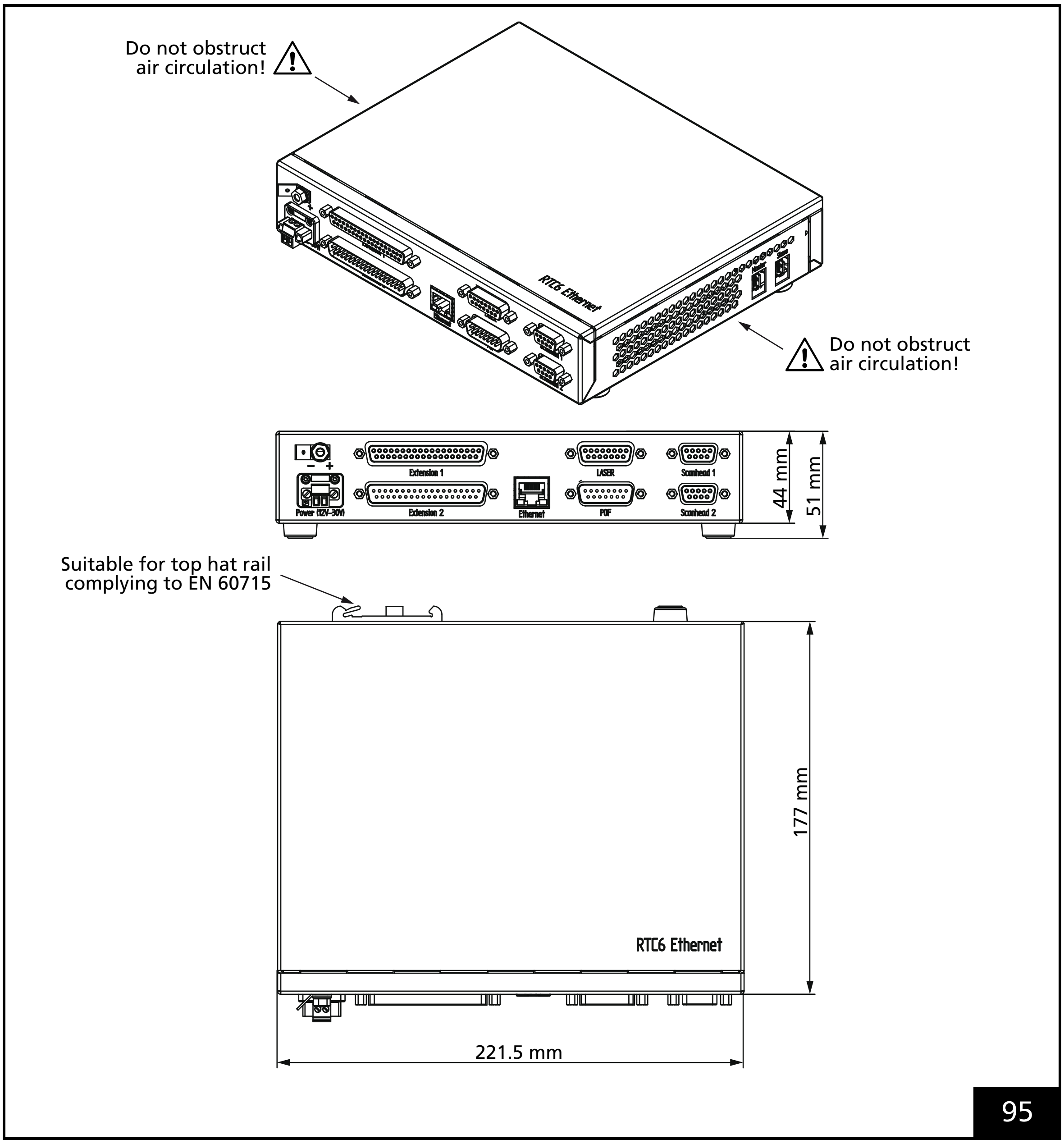
- The pixel values are outputted as 8-bit digital values at the 8-Bit Digital Output Port of the RTC6 PCIe Board, EXTENSION 2 Socket Connector. To enable outputting of (complete) pixel values as well as the latch signal at the EXTENSION 2 Socket Connector, the following solder jumper on the RTC6 PCIe Board must be closed(1):
  - DATA7 in solder jumper field C (= the DATA7 signal is outputted at pin 15 of the EXTENSION 2 Socket Connector)
  - LATCH in solder jumper field B (= the LATCH signal is outputted at pin 17 of the EXTENSION 2 Socket Connector)

Technical Specifications

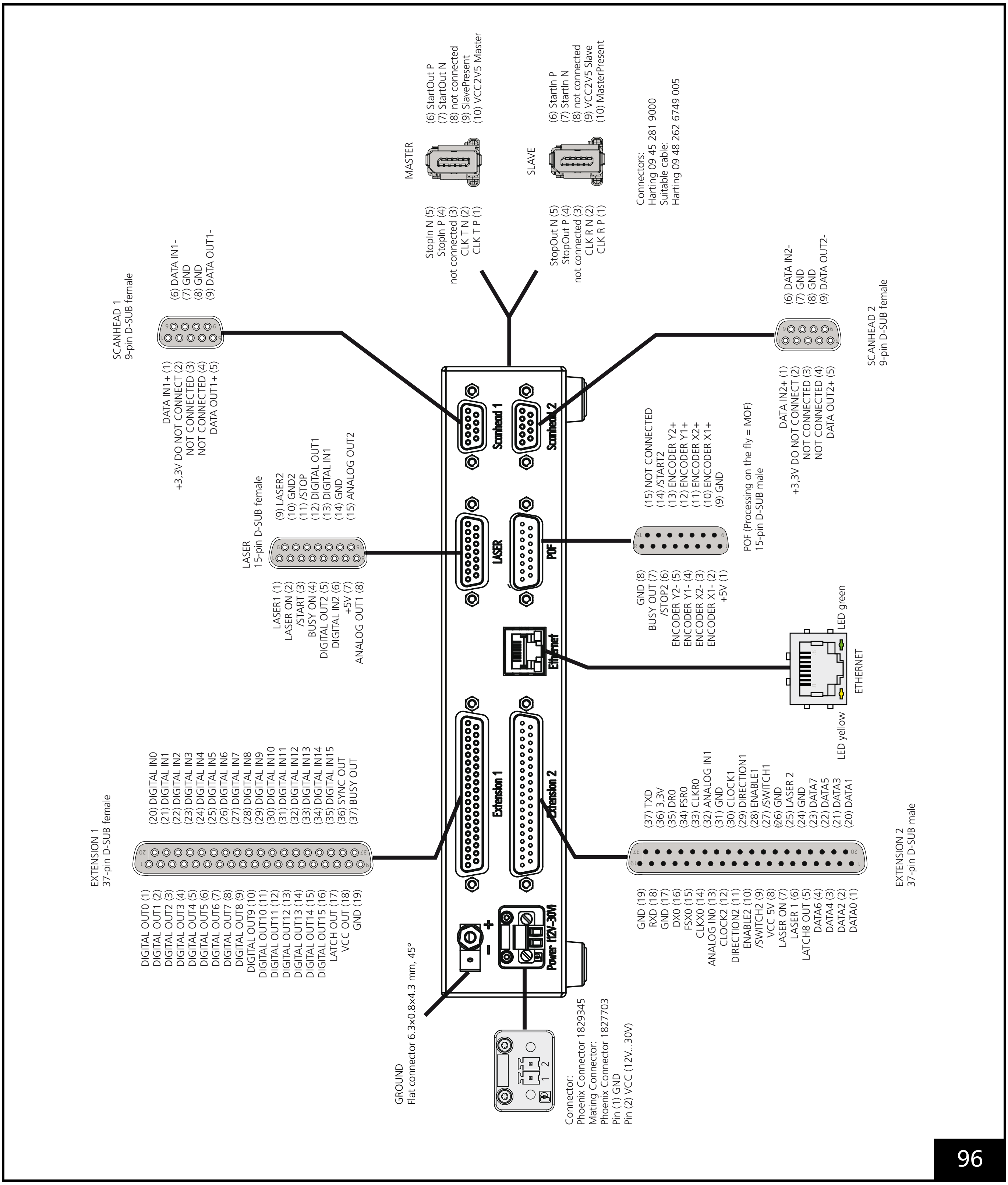
- Dimensions
  - Length — 52.8 mm
  - Width — 48.7 mm
- Analog output ports ANALOG OUTA, ANALOG OUTB
  - Connectors — 1 6-pin socket connector, 2 SMA connectors (coaxial connectors)
  - Output voltage range — 0 V...5 V
  - Resolution — 8 bits
  - Max. current load — 5 mA
  - Reference — GND

(1) Corresponds to RTC6 PCIe Boards TYPE n24.

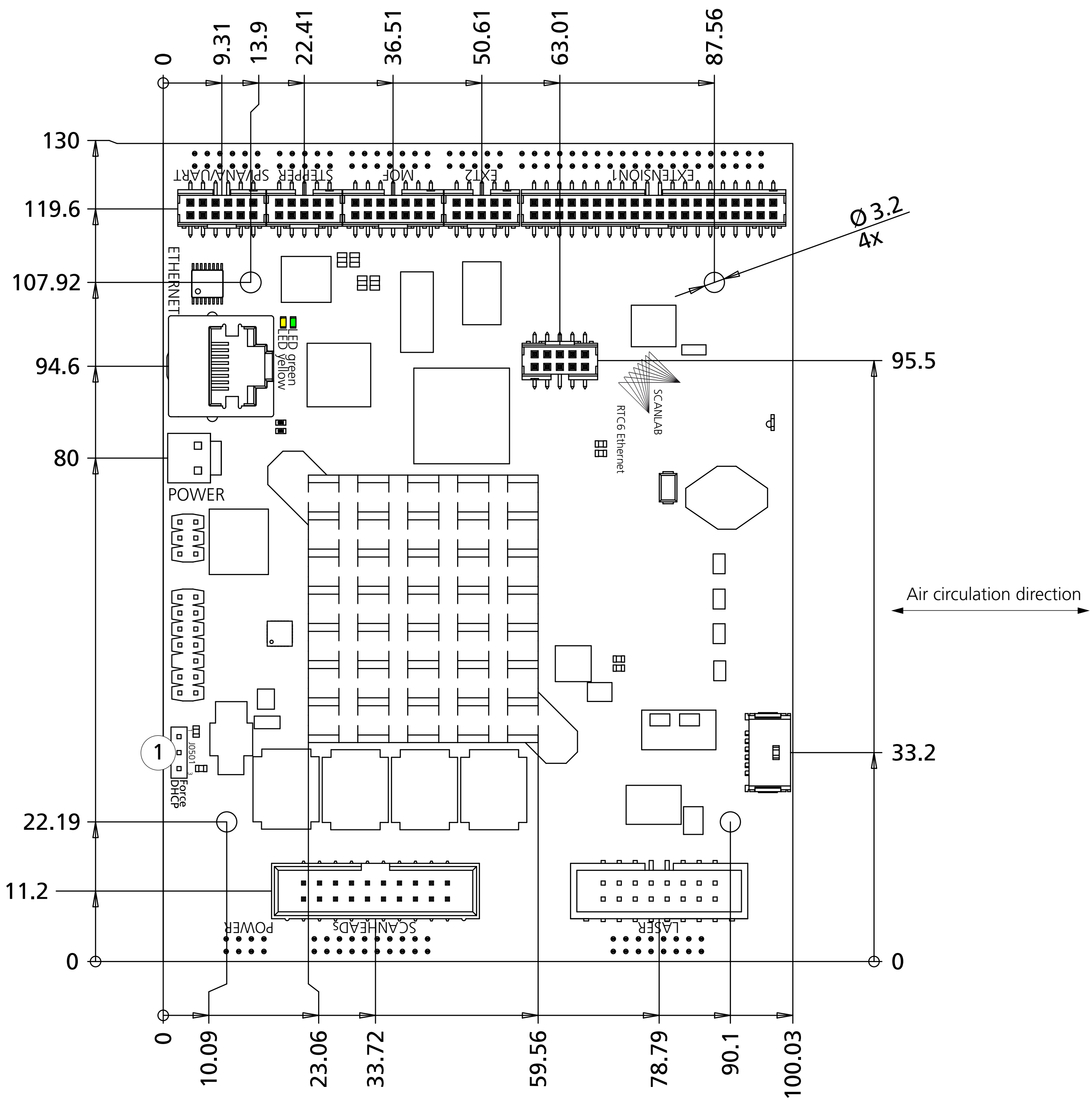
18 Appendix C: RTC6 EtherBox



RTC6 EtherBox. Dimensions. For example, #0148520.



RTC6 EtherBox: pin-outs.



Legend

1. Force DHCP. . . . . Jumper field Jumper field to ignore or not to ignore the saved static IP address. Details like "Jumper Field 'Force DHCP'", page 977.

97

RTC6 EtherBox: Installed RTC6 Ethernet Board. Upper side. Dimensions and connector positions. All dimensions in mm.

18.1 Mounting

Notice!

- The RTC6 board will be irreversibly destroyed if overheated. Make sure that the perforated sides of the RTC6 EtherBox are never closed or covered.

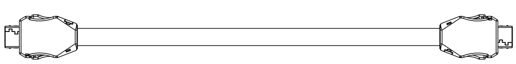
- Use the top-hat rail bracket on the backside, see Figure 95, to hang the RTC6 EtherBox in a top hat rail. alternatively, place the device free.

18.2 Cabling

Caution!

- Even when both RTC6 EtherBoxes are connected to the same power supply, different potentials can result of unfavorable cabling and can destroy the boards. To avoid them:
  – Connect one RTC6 EtherBox to the power supply. Then connect the other RTC6 EtherBox by a cable junction to the same power supply.
  – Keep the connection between the RTC6 EtherBoxes as short as possible.
  – First do the cabling of the components correctly, then switch on the power supply.
  – Switch off the power supply before disconnect the cabling.

- Power is supplied via a Phoenix connector, see Figure 96. Use the delivered Phoenix mating connector (manufacturer’s article number: 1827703).
- SCANLAB recommends grounding the device. To do so, connect the grounding to the flat connector (6.3×0.8×4.3 mm).
- Connect the PC to the Ethernet connector via an Ethernet cable.
- Connect your desired devices to the respective connector. Observe the pin-outs of the connectors, see Figure 96.
- For the electrical integration of the RTC6 EtherBox, use only the connections on the outside of the housing. Do not use any pins on the built-in RTC6 Ethernet Board (see Figure 97) that are not connected by SCANLAB to connectors on the housing of the RTC6 EtherBox.
- *Do not* connect the master/slave connector of the RTC6 EtherBox to that of separate RTC6 boards because the pin-outs are different.
- If you supply multiple RTC6 Ethernet Boards/RTC6 EtherBoxes with voltage from only one source, you must implement the cabling as shown in Figure 90.
- If you use several RTC6 EtherBoxes together(1), take care of a correct master/slave connection. A suitable cable is available from SCANLAB, see Figure 98.



#0149964 l = 200 mm

98

SCANLAB Master/Slave connecting cable for RTC6 EtherBoxes.

(1) See also Chapter 6.6.3 ”Master/Slave Operation”, page 127.

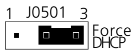
18.3 Installation and Operation

- When installing the RTC6 Ethernet Board, a static IP address must be obtained. For that, you have to set the jumper field **Force DHCP** on the RTC Ethernet board.

  (1) Make sure that the RTC6 EtherBox is disconnected from the power supply.

  (2) Unscrew the two Torx screws on the back of the RTC6 EtherBox and remove the housing cover.

  (3) Set the jumper field **'Force DHCP'**, see Figure 97, to "Force DHCP position":



"Force DHCP position"

  (4) Put the housing cover back on and screw the two Torx screws from step 2.

  (5) Connect the RTC6 Ethernet Board to the power supply back on again and a network, see also Chapter 16.3.3 "Connecting to a Network", page 979.

  (6) More information about installation and use of the RTC6 Ethernet Board (inside the RTC6 EtherBox) can be found in the main part of the RTC6 Manual.

18.4 Legal

18.4.1 EU Declaration of Conformity – RTC6 EtherBox

EU-Konformitätserklärung

für das Produkt:

RTC6 EtherBox

Der Hersteller

SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Deutschland

erklärt, dass das genannte Produkt die einschlägigen Harmonisierungsrechtsvorschriften der Union erfüllt:

- **2011/65/EU -** Richtlinie des EUROPÄISCHEN PARLAMENTS UND DES RATES zur Beschränkung der Verwendung bestimmter gefährlicher Stoffe in Elektro- und Elektronikgeräten (RoHS-Richtlinie).

Folgende harmonisierte Normen wurden angewandt:

- **DIN EN IEC 63000:2019-05** - Technische Dokumentation zur Beurteilung von Elektro- und Elektronikgeräten hinsichtlich der Beschränkung gefährlicher Stoffe

Die alleinige Verantwortung für die Ausstellung dieser Konformitätserklärung trägt der Hersteller.

Dieses Dokument zur Kundeninformation wurde elektronisch ausgestellt und ist daher ohne Unterschrift gültig. Eine unterschriebene Ausführung ist Bestandteil der SCANLAB-internen technischen Dokumentation.

EU Declaration of Conformity

for the product:

RTC6 EtherBox

The manufacturer

SCANLAB GmbH, Siemensstr. 2a, 82178 Puchheim, Germany

declares that the product mentioned above complies with the relevant Union harmonization legislation:

- **2011/65/EU** - Directive of the EUROPEAN PARLIAMENT AND OF THE COUNCIL on the restriction of the use of certain hazardous substances in electrical and electronic equipment (RoHS Directive).

The following harmonized standards have been applied:

- **DIN EN IEC 63000:2019-05** - Technical documentation for the assessment of electrical and electronic products with respect to the restriction of hazardous substances

This declaration of conformity is issued under the sole responsibility of the manufacturer.

This document, used for customer information, was issued electronically and is therefore valid without a signature. A signed version is part of the SCANLAB internal technical documentation.

SCANLAB GmbH
Puchheim

2021-07-19

18.4.2 Confirmation of Electromagnetic Compatibility (EMC)

Confirmation of electromagnetic compatibility (EMC)

The product:

RTC6 EtherBox

has been verified for electromagnetic compatibility (EMC) as follows:

☒	An identical system has been tested and meets the requirements of the harmonized standards **EN 55011:2016** **EN 61000-6-2:2005 +AC:2005** **EN 61000-3-2:2014** **EN 61000-3-3:2013**
☐	A technically comparable system has been tested and meets the requirements of the harmonized standards [name]
☐	The EMC relevant components of the system were tested individually or in a different configuration and met the requirements of the harmonized standards [name]
☐	An identical system was tested in an EMC-compatible housing and in this condition met the requirements of the harmonized standards [name]

Remarks:

./.

SCANLAB GmbH
Puchheim

2023-07-06

This document was issued electronically and is therefore valid without signature.

07/2023

18.4.3 Compliance with FCC Rules

The RTC6 EtherBox has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules.

These limits are designed to provide reasonable protection against harmful interference when the RTC6 EtherBox is operated in a commercial environment. The RTC6 EtherBox generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with this instruction manual, may cause harmful interference to radio communications. Operation of the RTC6 EtherBox in a residential area is likely to cause harmful interference in which case the user is required to correct the interference at his own expense.

18.5 Technical Specifications – RTC6 EtherBox

Dimensions

See Figure 96.

Weight

1.4 kg

Electrical Specifications

Voltage supply[a]	+12 V…+30 V
Maximum power consumption, no peripherals attached	< 12 W

(a) Power supply not included.

19 Appendix D: RTC6 Software Packages – History

RTC6 Software Package Version	DLL (a)	OUT (a)	ETH (a)	RBF (a)	DAT (a)	BIOS (b)	BIOS-ETH (c)	Date
1.3.0	606	606	*(d)	611	601	21	*	2017-09-11
1.3.1	607	607	607	611	601	22	*	2017-11-09
1.3.2	608	608	608	611	601	22	22	2018-01-23
1.3.3	608	608	608	612	601	22	22	2018-04-19
1.4.1	609	609	609	614	601	23	23	2018-08-03
1.4.2	610	610	610	615	602	23	23	2018-11-15
1.4.4	611	611	611	615	603	23	24	2019-03-01
1.5.0	614	614	614	619	603a	23	25	2019-07-26
1.5.2	615	615	615	621	603a	23	25	2019-09-11
1.6.0	616	616	616	622	603a	23	25	2019-11-25
1.6.1	617	617	617	623	603a	23	25	2020-02-07
1.7.0	618	618	618	623	603a	23	26	2020-03-13
1.7.1	618	618	618	623	603a	23	26	2020-03-19
1.7.3	619	619	619	624	603a	23	27	2020-06-19
1.7.4	620	620	620	625	603a	23	27	2020-07-10
1.7.5	621	621	621	625	603a	23	27	2020-07-24
1.7.6	622	622	622	625	603a	23	28	2020-10-02
1.7.7	623	623	623	626	603a	23	28	2020-12-11
1.7.8	624	624	624	627	603a	23	28	2021-01-22
1.7.9	625	625	625	628	603a	23	28	2021-02-05
1.7.10	625	626	626	628	603a	23	28	2021-02-26
1.7.11	626	627	627	629	603a	23	28	2021-04-30
1.7.12	627	628	628	629	603a	23	29	2021-07-09
1.9.0	628	629	629	630	603a	23	30	2021-09-07
1.10.0	629	630	630	631	603a	23	31	2021-10-15
1.11.0	630	631	631	631	603a	23	32	2021-11-12
1.12.0	631	632	632	632	604	23	33	2021-12-22

RTC6 Software Package Version	DLL (a)	OUT (a)	ETH (a)	RBF (a)	DAT (a)	BIOS (b)	BIOS-ETH (c)	Date
1.13.0	632	633	633	633	604	23	33	2022-02-18
1.14.1	634	635	635	634	604	23	35	2022-07-22
1.15.0	635	636	636	635	604	23	36	2022-09-16
1.15.2	637	637	637	636	604	23	36	2022-11-24
1.15.3	638	638	638	636	604	23	36	2022-12-22
1.15.4	638	639	639	637	604	23	36	2023-01-20
1.15.5	639	640	640	637	604	23	37	2023-02-15
1.16.0	640	641	641	637	604	23	38	2023-03-01
1.16.1	643	645	645	639	604	23	39	2023-07-21
1.16.2	643	646	646	639	604	23	39	2023-09-15
1.16.3	643	646	646	639	604	23	40	2023-09-29
1.17.0	644	647	647	640	604	23	40	2023-11-24
1.18.0	646	649	649	641	604	23	42	2024-03-04

(a) See Table page 25.

(b) RTC6BIOSOUT_*.out, see page 35.

(c) RTC6BIOSETH_*.out, see page 35.

(d) Cannot be updated by users or version cannot be read out.

20 Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards

- For usage, see **control_command**.
- Example:
  `control_command( Data` = 0501_H)
  that is, $Code_{HIGH}$ = 05 (**SetMode**) and
  $Code_{LOW}$ = 01 requests that the *iDRIVE* scan system should change the data type to be transmitted to "actual position". The *iDRIVE* scan system evaluates the request and – if successful – subsequently transmits the "actual position" to the RTC board (notation in this manual is "Signal 0501_H" in this case).
- The following reference is used in several RTC manuals and is thus generic.

$Code_{HIGH}$ (hex)	
05	**SetMode** **SetMode** selects the data signal to be returned from the scan system for the specified axis. See also Chapter 8.1.2 "Configuring the Data Signal Transmission Behavior of the Scan System", page 222. Each $Code_{LOW}$ parameter value corresponds to a particular data type. Default setting: $Code_{LOW}$ = 00_H (XY2-100 status word). See also "On SetMode", Seite 1098. See also "On SetMode, SetControlDefinitionMode and SetEchoMode", Seite 1098. *iDRIVE* scan systems with XY2-100 interface transmit a signed 16-bit value. *iDRIVE* scan systems with SL2-100 interface transmit an signed 20-bit value. For example, excelli*SCAN*-scan heads. *iDRIVE* scan systems with XY2-100 Enhanced interface transmit a signed 16-bit value. With these, the XY2-100 Converter (Accessory) converts (by multiplying by 16) the value to a signed 20-bit value. In this document: 0500_H; 0501_H; 0502_H; 0503_H; 0504_H; 0505_H; 0506_H; 0507_H; 0508_H; 0509_H; 0510_H; 0511_H; 0512_H; 0513_H; 0514_H; 0515_H; 0516_H; 0517_H; 0518_H; 0519_H; $051A_H$; $051B_H$; $051C_H$; $051D_H$; $051E_H$; $051F_H$; 0520_H; 0521_H; 0522_H; 0523_H; 0524_H; 0525_H; 0526_H; 0527_H; 0528_H; 0529_H; $052A_H$; $052B_H$; $052C_H$; $052D_H$; $052E_H$; $052F_H$; 0530_H; 0531_H; 0532_H; 0533_H; 0534_H; 0535_H; 0536_H; 0537_H; 0538_H; 0539_H; $053A_H$; $053B_H$; $053C_H$; $053D_H$; $053E_H$; $053F_H$; 0549_H; $055E_H$; 0564_H; 0565_H; 0574_H;

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	00	As of scan system firmware ≥ 2001. XY2-100 status word		
		16-bit protocol	20-bit protocol	
		Bit #15 (MSB)	Bit #19 (MSB)	Like [Bit #7|Bit #11]. = 1: The scan system servo control is ready for input data ("PowerOK", actually corresponds to a "ServoOK"). For scan systems with sensors as well as scan systems with interlock: there is also no interlock error.
		Bit #14	Bit #18	Like [Bit #6|Bit #10]. = 1: The system temperature within optimal range ("TempOK"). For scan systems with sensors: there is also no warning.
		Bit #13	Bit #17	= 1.
		Bit #12	Bit #16	Like [Bit #4|Bit #8]. = 1: The y axis position errors are within allowed range (PosAck Signal).
		Bit #11	Bit #15	Like [Bit #3|Bit #7]. = 1: The x axis position errors are within allowed range (PosAck Signal).
		Bit #10	Bit #14	Like [Bit #2|Bit #6]. = 1. Only intelli*WELD*: Reserved. For scan systems with sensors for automatic self calibration: Reserved.
		Bit #9	Bit #13	= 0.
		Bit #8	Bit #12	= 1.
		Bit #7	Bit #11	Like [Bit #15|Bit #19].
		Bit #6	Bit #10	Like [Bit #14|Bit #18].
		Bit #5	Bit #9	= 1.
		Bit #4	Bit #8	Like [Bit #12|Bit #16].
		Bit #3	Bit #7	Like [Bit #11|Bit #15].
		Bit #2	Bit #6	Like [Bit #10|Bit #14].
		Bit #1	Bit #5	= 0.
		Bit #0	Bit #4	= 1.
		–	Bit #3	= 0.
		–	Bit #2	= 0.
		–	Bit #1	= 0.
		–	Bit #0	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code_{HIGH} (hex)</th><th>Code_{LOW} (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">01</td><td colspan="3">As of scan system firmware ≥ 2001.
Actual position (angular position of galvanometer scanners)</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>Unit with 16-bit protocol: In bits.
Value range with 16-bit protocol: [–32.768…+32.767].
Unit with 20-bit protocol: In bits.
Value range with 20-bit protocol: –524.288…+524.287.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code_{HIGH} (hex)</th><th>Code_{LOW} (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">02</td><td colspan="3">As of scan system firmware ≥ 2001.
Set position (angular position of galvanometer scanners)
As of scan system firmware ≥ 5050.
With excelliSCAN scan heads:
Precalculated set position (angular position of galvanometer scanners).
Refer to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual, Chapter 2.1.6 "SCANahead System Timing", page 20 and Figure 7.</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>Unit with 16-bit protocol: In bits.
Value range with 16-bit protocol: [–32.768…+32.767].
Unit with 20-bit protocol: In bits.
Value range with 20-bit protocol: –524.288…+524.287.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">03</td><td colspan="3">As of scan system firmware ≥ 2001.
Position error
(= set position 0502$_H$ – actual position 0501$_H$)
As of scan system firmware ≥ 5050.
With excelli*SCAN* scan heads:
Trajectory Error. See Glossary entry.</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>Unit with 16-bit protocol: In bits.
Value range with 16-bit protocol: [–32.768…+32.767].
Unit with 20-bit protocol: In bits.
Value range with 20-bit protocol: –524.288…+524.287.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">04</td><td colspan="3">As of scan system firmware ≥ 2001.
Actual current (output stage current)</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>Unit with 16-bit protocol: In mA.
Value range with 16-bit protocol: [–32.768…+32.767].
Unit with 20-bit protocol: In 16^{-1} mA.
Value range with 20-bit protocol: –524.288…+524.287.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">05</td><td colspan="3">As of scan system firmware ≥ 2001.
Relative galvanometer scanner control
Not intelli*cube*.</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>Unit with 16-bit protocol: In per mille.
Value range with 16-bit protocol: [–1,000…+1,000].
Unit with 20-bit protocol: In 16^{-1} per mille.
Value range with 20-bit protocol: [–16,000…+16,000].
The return value 16 entspricht 1‰.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">06</td><td colspan="3">As of scan system firmware ≥ 2001.
Actual velocity (angular velocity)</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>Unit with 16-bit protocol: In Bit/ms.
Value range with 16-bit protocol: [–32.768…+32.767].
Unit with 20-bit protocol: In Bit/ms.
Value range with 20-bit protocol: –524.288…+524.287.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	07	Reserved.		
		16-bit protocol	20-bit protocol	–
		Bit #0 … Bit #15	Bit #0 … Bit #19	–

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	08	Reserved.		
		16-bit protocol	20-bit protocol	–
		Bit #0 … Bit #15	Bit #0 … Bit #19	–

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	09	Reserved.		
		16-bit protocol	20-bit protocol	–
		Bit #0 … Bit #15	Bit #0 … Bit #19	–

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">10</td><td colspan="3">Reserved.</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td>–</td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>–</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">11</td><td colspan="3">Reserved.</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td>–</td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>–</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	12	As of scan system firmware ≥ 2001. Only intelli*WELD* with vario*SCAN*$_{de}$ FCi. Not: intelli*WELD* with vario*SCAN* FC: Actual z axis position Can only be read out from the channel of the scan head connector to which the z axis is connected.		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #0 … Bit #19	Unit with 16-bit protocol: In bits. Value range with 16-bit protocol: [–32.768…+32.767]. Unit with 20-bit protocol: In bits. Value range with 20-bit protocol: –524.288…+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	13	Reserved.		
		16-bit protocol	20-bit protocol	–
		Bit #0 … Bit #15	Bit #0 … Bit #19	–

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	14	As of scan system firmware ≥ 2001. Temperature of galvanometer scanner		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #0 … Bit #19	Unit with 16-bit protocol: In 10^{-1} °C. Value range with 16-bit protocol: [–32.768…+32.767]. Unit with 20-bit protocol: In 160^{-1} °C. Value range with 20-bit protocol: –524.288…+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	15	As of scan system firmware ≥ 2001. Servo board temperature As of scan system firmware ≥ 5050. Axis 1 (X-axis): Temperature of the servo board Axis 2 (Y-axis): Temperature of the servo add-on board for evaluating the encoder signals		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #0 … Bit #19	Unit with 16-bit protocol: In 10^{-1} °C. Value range with 16-bit protocol: [–32.768…+32.767]. Unit with 20-bit protocol: In 160^{-1} °C. Value range with 20-bit protocol: –524.288…+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	16	As of scan system firmware ≥ 2001. AGC voltage (position detector supply voltage)		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #0 … Bit #19	Unit with 16-bit protocol: In 10^{-2} V. Value range with 16-bit protocol: [–32.768…+32.767]. Unit with 20-bit protocol: In 1600^{-1} V. Value range with 20-bit protocol: –524.288…+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	17	As of scan system firmware ≥ 2001. DSP core supply voltage		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #0 … Bit #19	Unit with 16-bit protocol: In 10^{-2} V. Value range with 16-bit protocol: [–32.768…+32.767]. Unit with 20-bit protocol: In 1600^{-1} V. Value range with 20-bit protocol: –524.288…+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	18	As of scan system firmware ≥ 2001. DSP IO voltage		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #0 … Bit #19	Unit with 16-bit protocol: In 10^{-2} V. Value range with 16-bit protocol: [–32.768…+32.767]. Unit with 20-bit protocol: In 1600^{-1} V. Value range with 20-bit protocol: –524.288…+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">19</td><td colspan="3">As of scan system firmware ≥ 2001.
Analog section voltage</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>Unit with 16-bit protocol: In 10^{-2} V.
Value range with 16-bit protocol: [–32.768…+32.767].
Unit with 20-bit protocol: In 1600^{-1} V.
Value range with 20-bit protocol: –524.288…+524.287.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">1A</td><td colspan="3">As of scan system firmware ≥ 2001.
Analog-digital converter supply voltage</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>Unit with 16-bit protocol: In 10^{-2} V.
Value range with 16-bit protocol: [–32.768…+32.767].
Unit with 20-bit protocol: In 1600^{-1} V.
Value range with 20-bit protocol: –524.288…+524.287.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	1B	As of scan system firmware ≥ 2001. AGC current (PD supply current)		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #0 … Bit #19	Unit with 16-bit protocol: In mA. Value range with 16-bit protocol: [–32.768…+32.767]. Unit with 20-bit protocol: In 16^{-1} mA. Value range with 20-bit protocol: –524.288…+524.287.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	1C	As of scan system firmware ≥ 2066. Hardware version of servo board		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #0 … Bit #19	16-bit protocol: 2nnn = DSCB 5021 = ISB1 5031 = ISB2 6011 = microISB 20-bit protocol: 2nnn×16 = DSCB 5021×16 = ISB1 5031×16 = ISB2 6011×16 = microISB

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	1D	As of scan system firmware ≥ 2001. Relevant for scan systems with galvanometer scanner heating. Relative heating output of the corresponding galvanometer scanner heater Not relevant for intelli*cube*. Not relevant for dyn*AXIS* XS. Not relevant for scan systems with water-cooled galvanometer scanners.		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: In per mille. Value range with 16-bit protocol: [0…1,000]. Unit with 20-bit protocol: In per mille. Value range with 20-bit protocol: [0…1,000].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	1E	As of scan system firmware ≥ 2001. Serial number, lower 16 bits		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0…65,535]. Unit with 20-bit protocol: None. Value range with 20-bit protocol: [0…65,535].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	1F	As of scan system firmware ≥ 2001. Serial number, upper 16 bits		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0…65,535]. Unit with 20-bit protocol: None. Value range with 20-bit protocol: [0…65,535].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	20	As of scan system firmware ≥ 2001. SCANLAB article number, lower 16 bits		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0…65,535]. Unit with 20-bit protocol: None. Value range with 20-bit protocol: [0…65,535].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	21	As of scan system firmware ≥ 2001. SCANLAB article number, upper 16 bits		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0…65,535]. Unit with 20-bit protocol: None. Value range with 20-bit protocol: [0…65,535].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	22	As of scan system firmware ≥ 2001. Version number of scan system firmware		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: None. Value range with 16-bit protocol: [0…65,535]. Unit with 20-bit protocol: None. Value range with 20-bit protocol: [0…65,535].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="4">05</td><td rowspan="4">23</td><td colspan="3">As of scan system firmware ≥ 2001.
Calibration angle
Mechanical scan angle (±) for 96% of the maximum or minimum control value. With 16-bit protocol ±31457 bit. With 20-bit protocol ±503316 bit.</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #4
…
Bit #19</td><td>Unit with 16-bit protocol: In 10^{-3} degrees.
Value range with 16-bit protocol: [0…65,535].
Unit with 20-bit protocol: In 10^{-3} degrees.
Value range with 20-bit protocol: [0…65,535].</td></tr>
<tr><td>–</td><td>Bit #0
…
Bit #3</td><td>= 0.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="4">05</td><td rowspan="4">24</td><td colspan="3">As of scan system firmware ≥ 2001.
Aperture</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #4
…
Bit #19</td><td>Unit with 16-bit protocol: In mm.
Value range with 16-bit protocol: [0…65,535].
Unit with 20-bit protocol: In mm.
Value range with 20-bit protocol: [0…65,535].</td></tr>
<tr><td>–</td><td>Bit #0
…
Bit #3</td><td>= 0.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{\text{HIGH}}$ (hex)</th><th>Code$_{\text{LOW}}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="4">05</td><td rowspan="4">25</td><td colspan="3">As of scan system firmware ≥ 2001.
Wavelength</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #4
…
Bit #19</td><td>Unit with 16-bit protocol: In nm.
Value range with 16-bit protocol: [0…65,535].
Unit with 20-bit protocol: In nm.
Value range with 20-bit protocol: [0…65,535].</td></tr>
<tr><td>–</td><td>Bit #0
…
Bit #3</td><td>= 0.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{\text{HIGH}}$ (hex)</th><th>Code$_{\text{LOW}}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="5">05</td><td rowspan="5">26</td><td colspan="3">As of scan system firmware ≥ 2078.
Tuning number</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #8
…
Bit #15</td><td>Bit #12
…
Bit #19</td><td>Start setting.</td></tr>
<tr><td>Bit #0
…
Bit #7</td><td>Bit #4
…
Bit #11</td><td>Current setting.</td></tr>
<tr><td>–</td><td>Bit #0
…
Bit #3</td><td>= 0.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	27	As of scan system firmware ≥ 2078. Only after `control_command(Data = 17FFH)`: number of the temporarily stored data type		
		16-bit protocol	20-bit protocol	
		Bit #8 … Bit #15	Bit #12 … Bit #19	= 0.
		Bit #0 … Bit #7	Bit #4 … Bit #11	Cached setting.
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>$\text{Code}_{\text{HIGH}}$ (hex)</th><th>Code_{LOW} (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">28</td><td colspan="3">As of scan-system firmware ≥ 2060.
Diagnostic flags, lower 16 bits</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td>–</td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>–</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>$\text{Code}_{\text{HIGH}}$ (hex)</th><th>Code_{LOW} (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">29</td><td colspan="3">As of scan-system firmware ≥ 2060.
Diagnostic flags, upper 16 bits</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td>–</td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>–</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	2A	As of scan system firmware ≥ 2001. Stop event code Value range with 16-bit protocol: [0…65,535]. Value range with 20-bit protocol: [0…65,535].		
		16-bit protocol	20-bit protocol	
		(hex)	(hex)	
		0001	00010	The galvanometer scanner has reached a critical edge position.
		0002	00020	AD converter error.
		0003	00030	Temperature in scan system above max. allowed value.
		0004	00040	External power supply voltages have dropped below the allowed value.
		0005	00050	Flags are not valid.
		0006 … 000C	00060 … 000C0	Reserved. … Reserved.
		000D	000D0	Reserved.
		000E	000E0	Position Acknowledge time out (set position not reached for long time).
		000F	000F0	Reserved.
		0014	00140	Reserved.
		0015	00150	Reserved.
		0016	00160	Reserved.
		0017	00170	Reserved.
		0018	00180	Reserved.
		0019	00190	Reserved.
		001A	001A0	Reserved.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	2B	As of scan-system firmware ≥ 2060. Flags from last stop event, lower 16 bits		
		16-bit protocol	20-bit protocol	–
		Bit #0 … Bit #15	Bit #0 … Bit #19	–

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	2C	As of scan-system firmware ≥ 2060. Flags from last stop event, upper 16 bits		
		16-bit protocol	20-bit protocol	–
		Bit #0 … Bit #15	Bit #0 … Bit #19	–

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	2D	Reserved.		
		16-bit protocol	20-bit protocol	–
		Bit #0 … Bit #15	Bit #0 … Bit #19	–

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	2E	Reserved.		
		16-bit protocol	20-bit protocol	–
		Bit #0 … Bit #15	Bit #0 … Bit #19	–

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	2F	As of scan-system firmware ≥ 2061. Running time (seconds)		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: In s. Value range with 16-bit protocol: [0…59]. Unit with 20-bit protocol: In s. Value range with 20-bit protocol: [0…59].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	30	As of scan-system firmware ≥ 2061. Running time (minutes)		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: In min. Value range with 16-bit protocol: [0…59]. Unit with 20-bit protocol: In min. Value range with 20-bit protocol: [0…59].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	31	As of scan-system firmware ≥ 2061. Running time (hours)		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: In h. Value range with 16-bit protocol: [0…23]. Unit with 20-bit protocol: In h. Value range with 20-bit protocol: [0…23].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	32	As of scan-system firmware ≥ 2061. Running time (days)		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: In d. Value range with 16-bit protocol: [0…65,535]. Unit with 20-bit protocol: In d. Value range with 20-bit protocol: [0…65,535].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	33	As of scan system firmware ≥ 206x. Only intelli*WELD*: 3.3 V sensor board operating voltage		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: In 10^{-3} V. Value range with 16-bit protocol: [0…7,000]. Unit with 20-bit protocol: In 10^{-3} V. Value range with 20-bit protocol: [0…7,000].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	34	As of scan system firmware ≥ 206x. Only intelli*WELD*: Sensor board operating temperature		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: In 0.1 °C. Value range with 16-bit protocol: [0…65,535]. Unit with 20-bit protocol: In 0.1 °C. Value range with 20-bit protocol: [0…65,535].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	35	As of scan system firmware ≥ 206x. Only intelli*WELD*: Purge gas flow rate		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: [(purge gas flow rate – 4 l/min) * 109,89 min/l]. Value range with 16-bit protocol: [0…2,250]. Unit with 20-bit protocol: [purge gas flow rate – 4 l/min) * 109,89 min/l]. Value range with 20-bit protocol: [0…2,250].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	36	As of scan system firmware ≥ 206x. [Temperature of mirror 2 + 26.6 °C]		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: In 0.05 °C. Value range with 16-bit protocol: [0…1,992]. Unit with 20-bit protocol: In 0.05 °C. Value range with 20-bit protocol: [0…1,992].
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="4">05</td><td rowspan="4">37</td><td colspan="3">As of scan system firmware ≥ 206x.
[Temperature of mirror 1 + 26.6 °C]</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #4
…
Bit #19</td><td>Unit with 16-bit protocol: In 0.05 °C.
Value range with 16-bit protocol: [0…1,992].
Unit with 20-bit protocol: In 0.05 °C.
Value range with 20-bit protocol: [0…1,992].</td></tr>
<tr><td>–</td><td>Bit #0
…
Bit #3</td><td>= 0.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="4">05</td><td rowspan="4">38</td><td colspan="3">As of scan system firmware ≥ 206x.
Only intelliWELD:
Protective-window temperature</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #4
…
Bit #19</td><td>Unit with 16-bit protocol: In 0,044 °C.
Value range with 16-bit protocol: [0…2,250].
Unit with 20-bit protocol: In 0,044 °C.
Value range with 20-bit protocol: [0…2,250].</td></tr>
<tr><td>–</td><td>Bit #0
…
Bit #3</td><td>= 0.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="4">05</td><td rowspan="4">39</td><td colspan="3">As of scan system firmware ≥ 206x.
Only intelliWELD:
Collimator temperature</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #4
…
Bit #19</td><td>Unit with 16-bit protocol: In 0.05 °C.
Value range with 16-bit protocol: [0…2,250].
Unit with 20-bit protocol: In 0.05 °C.
Value range with 20-bit protocol: [0…2,250].</td></tr>
<tr><td>–</td><td>Bit #0
…
Bit #3</td><td>= 0.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="4">05</td><td rowspan="4">3A</td><td colspan="3">As of scan system firmware ≥ 206x.
Only intelliWELD:
Galvanometer scanner mount temperature</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #4
…
Bit #19</td><td>Unit with 16-bit protocol: In 0.05 °C.
Value range with 16-bit protocol: [0…2,250].
Unit with 20-bit protocol: In 0.05 °C.
Value range with 20-bit protocol: [0…2,250].</td></tr>
<tr><td>–</td><td>Bit #0
…
Bit #3</td><td>= 0.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	3B	As of scan system firmware ≥ 206x. Only intelli*WELD*: [Coolant-flow rate + 0.93 l × min^{-1}]		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: In 0,00525 l × min-1. Value range with 16-bit protocol: [0…2,250]. Unit with 20-bit protocol: [0…2,250]. Value range with 20-bit protocol: In 0,00525 l × min-1.
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system		
05	3C	As of scan system firmware ≥ 206x. Only intelli*WELD*: Protective-window scattered light value		
		16-bit protocol	20-bit protocol	
		Bit #0 … Bit #15	Bit #4 … Bit #19	Unit with 16-bit protocol: In 0.00444 V. Value range with 16-bit protocol: [0…2,250]. Unit with 20-bit protocol: [0…2,250]. Value range with 20-bit protocol: In 0.00444 V.
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system			
05	3D	As of scan system firmware ≥ 206x. Only intelli*WELD*: Flags sensor board Value range with 16-bit protocol: [0…65,535]. Value range with 20-bit protocol: [0…65,535].			
		16-bit protocol	20-bit protocol		
		Bit #15 (MSB)	Bit #19 (MSB)	= 1:	Protective-window temperature value not in [0…1,882].
		Bit #14	Bit #18	= 1:	Temperature of mirror 1 value not in [0…2,250].
		Bit #13	Bit #17	= 1:	Temperature of mirror 2 value not in [0…2,250].
		Bit #10	Bit #14	= 1:	Emerging-beam-opening temperature value not in [0…1,200]. Additionally, an INTERLOCK error is initiated.
		Bit #3	Bit #7	= 1:	Protective-window scattered light value not in [0…2,250].
		Bit #2	Bit #6	= 1:	Coolant-flow rate value not in [750…2,250].
		Bit #1	Bit #5	= 1:	Galvanometer-mount temperature value not in [0…1,600]. Additionally, an INTERLOCK error is initiated.
		Bit #0	Bit #4	= 1:	Collimator temperature value not in [0…2,000]. Additionally, an INTERLOCK error is initiated.
		–	Bit #0 … Bit #3	= 0. … = 0.	

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	3E	Reserved.		
		16-bit protocol	20-bit protocol	–
		Bit #0 … Bit #15	Bit #0 … Bit #19	–

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	3F	As of scan system firmware ≥ 2072. Set scaling factor for position values		
		16-bit protocol	20-bit protocol	
		Bit #2 … Bit #15	Bit #6 … Bit #19	Reserved.
		Bit #0 … Bit #1	Bit #4 … Bit #5	Scaling factor = $1/2^n$. 16-bit protocoln = 2 × Bit #1 + Bit #0. 20-bit protocol n = 2 × Bit #5 + Bit #4.
		–	Bit #0 … Bit #3	= 0.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	49	As of scan system firmware ≥ 5000. Supply voltage (usually 30 V or 48 V)		
		16-bit protocol	20-bit protocol	–
		Bit #0 … Bit #15	Bit #0 … Bit #19	–

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	$Code_{LOW}$ (hex)	Data type transmitted by the scan system		
05	5E	As of scan system firmware ≥ 5000. Number of transitions "regular operation → fault condition"		
		16-bit protocol	20-bit protocol	–
		Bit #0 … Bit #15	Bit #0 … Bit #19	–

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system
05	64	As of scan system firmware ≥ 5100. Refer to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual, Chapter 2.3 "Advanced Settings for "Spot Distance Control"", page 24: `TimeShift` relative to the SCANLAB default offset.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	Code$_{LOW}$ (hex)	Data type transmitted by the scan system
05	65	As of scan system firmware ≥ 5100. Refer to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual, Chapter 2.3 "Advanced Settings for "Spot Distance Control"", page 24: `TimeShift` in total.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th>Code$_{LOW}$ (hex)</th><th colspan="3">Data type transmitted by the scan system</th></tr>
<tr><td rowspan="3">05</td><td rowspan="3">74</td><td colspan="3">As of scan system firmware ≥ 51*7 + < 5090.
For High-Bandwidth Return Channel Multiplexing. For use in conjunction with Code$_{HIGH}$ (hex) = 65$_H$ and Code$_{HIGH}$ (hex) = 66$_H$. See corresponding Chapter incl. example code.</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td>–</td></tr>
<tr><td>Bit #0
…
Bit #15</td><td>Bit #0
…
Bit #19</td><td>–</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code$_{HIGH}$ (hex)</th><th></th></tr>
<tr><td>0A</td><td>
As of scan system firmware ≥ 2078.

UpdatePermanentMemory

UpdatePermanentMemory causes the scan system to set the current servo behavior as the startup behavior for subsequent restarts or resets.
See also Chapter 8.1.8 ”Configuring the Start Behavior”, page 232.

As of scan system firmware ≥ 5050.

Only excelliSCAN:

excelliSCAN scan heads do not support UpdatePermanentMemory.
<table>
<tr><th>Code$_{LOW}$ (hex)</th><th></th></tr>
<tr><td>00</td><td>Only allowed value.</td></tr>
</table>
</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><td>Code$_{HIGH}$
(hex)</td><td colspan="4"></td></tr>
<tr><td rowspan="8">0E</td><td colspan="4">As of scan system firmware ≥ 206x.

SetControlDefinitionMode

SetControlDefinitionMode causes the scan system to transmit information about a specific tuning (or the corresponding control algorithm) over the selected status channel.
See also “On SetMode, SetControlDefinitionMode and SetEchoMode”, Seite 1098.</td></tr>
<tr><td>Code$_{LOW}$
(hex)</td><td colspan="3">Data type transmitted by the scan system</td></tr>
<tr><td rowspan="6">00
01
02
03</td><td colspan="3">As of scan system firmware ≥ 206x.

Tuning number</td></tr>
<tr><td>16-bit protocol</td><td>20-bit protocol</td><td></td></tr>
<tr><td>Bit #15
(MSB)</td><td>Bit #19
(MSB)</td><td>= 0: Tuning available.
= 1: Tuning not available.</td></tr>
<tr><td>Bit #4
…
Bit #14</td><td>Bit #8
…
Bit #18</td><td>Reserved.</td></tr>
<tr><td>Bit #0
…
Bit #3</td><td>Bit #4
…
Bit #7</td><td>Tuning type
= 0: Vector tuning.
= 1: Jump tuning.
= 2: Reserved.
= 3: Reserved.
= 4: As of scan system firmware ≥ 5050.
*SCAN*ahead servo control.</td></tr>
<tr><td>–</td><td>Bit #0
…
Bit #3</td><td>= 0.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	
11	As of scan system firmware ≥ 2078. **SelectControlDefinition** Only scan systems equipped with several tunings (or corresponding servo algorithms). **SelectControlDefinition** causes the scan systems to switch to a certain tuning. See also Chapter 8.1.4 "Selecting the Tuning (Dynamics Setting)", page 225. $Code_{LOW}$ = 00…03 specifies the tuning number. As of scan system firmware ≥ 5050. Only excelli*SCAN*: excelli*SCAN* scan heads do not support **SelectControlDefinition**. With excelli*SCAN* scan heads, there is only one tuning.

$Code_{LOW}$ (hex)	
00	Tuning 0 (= default setting)
01	Tuning 1
02	Tuning 2
03	Tuning 3

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><th>Code_{HIGH} (hex)</th><th colspan="2"></th></tr>
<tr><td rowspan="7">12</td><td colspan="2">As of scan system firmware ≥ 2072.

SetPositionScale

SetPositionScale causes the scan system to switch to a certain scaling factor. The scan system servo electronics multiplies the actual position values received from the RTC board by this scaling factor. The actual position values sent back to the RTC board by the scan system are automatically divided by the same scaling factor. In this way, set position and actual position are always scaled to match each other. Important: You cannot detect a scaling factor ≠ 1 by comparing set position and actual position.
See also Chapter 8.1.7 "Configuring the Effective Calibration", page 231.
Important: The scaling factor needs to be changed in 2 steps: 1. Execute **SetPositionScale** with Code_{LOW} = 83_H. 2. Execute **SetPositionScale** with Code_{LOW} = [00_H...03_H] (= the desired scaling factor). See also "On SetPositionScale", Seite 1099.

As of scan system firmware ≥ 5050.

Only *SCAN*ahead systems:

*SCAN*ahead systems do not support **SetPositionScale**. With *SCAN*ahead systems, the effective calibration cannot be changed.</td></tr>
<tr><td>Code_{LOW} (hex)</td><td></td></tr>
<tr><td>00</td><td>The scaling factor is set to the value 1/1 (= no scaling; default setting)</td></tr>
<tr><td>01</td><td>The scaling factor is set to the value 1/2</td></tr>
<tr><td>02</td><td>The scaling factor is set to the value 1/4</td></tr>
<tr><td>03</td><td>The scaling factor is set to the value 1/8</td></tr>
<tr><td>83</td><td>Make ready for scaling factor change.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)		
15	As of scan system firmware ≥ 2066. **SetPosAcknowledgeLevel** **SetPosAcknowledgeLevel** causes the scan system to switch to a certain PosAck limit value. See also Chapter 8.1.6 "Configuring the PosAck Limit Value", page 231. See also "On SetPosAcknowledgeLevel", Seite 1099.	
	$Code_{LOW}$ (hex)	
	00 … FF	Desired PosAck limit value in LSB16, 16-bit protocol. In bits.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)		
17	As of scan system firmware ≥ 2078. **Store/RestoreTransmissionMode** **Store/RestoreTransmissionMode** causes with $Code_{LOW} = FF_H$ the scan system to cache the data type currently selected for transmission. It can be reinstated at a later time then with $Code_{LOW} = 00_H$. See also `Data` = 0527_H. See also "On Store/RestoreTransmissionMode", Seite 1099.	
	$Code_{LOW}$ (hex)	
	FF	Store temporarily
	00	Reinstate

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

$Code_{HIGH}$ (hex)	
21	As of scan system firmware ≥ 2078. **SetEchoMode** **SetEchoMode** verifies whether data transfer is intact. See also Chapter 8.1.9 "Fault Diagnosis and Functional Test", page 232. **SetEchoMode** passes an 8-bit value to the scan system with $Code_{LOW}$. As a consequence, the scan system transmits a 16-bit value (16-bit protocol) or 20-bit value (20-bit protocol) on the corresponding status channel to the RTC board. If the data transmission has been error-free, with this: • The upper 8 bits and $Code_{LOW}$ are identical • With the 16-bit value, the lower 8 bits / with the 20-bit value, the next lower 8 bits and the complementary value of $Code_{LOW}$ (NOT $Code_{LOW}$) are identical Example with 16-bit protocol: For `control_command(1, 1, 0x210A)` and if data transfer is error-free, `(get_value(1) AND 0xFFFF)` returns 0x0AF5. Example with 20-bit protocol: For `control_command(1, 1, 0x210A)` and if data transfer is error-free, `(get_value(1) AND 0xFFFF0)` returns 0x0AF50. See also "On SetMode, SetControlDefinitionMode and SetEchoMode", Seite 1098.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!

Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	
40	As of scan system firmware ≥ 5102 + ≥ 5112. Only excelli*SCAN*: **ResetTrAck** **ResetTrAck** causes the scan system to reset the TrAck Signal bits. Refer to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual, Chapter 2.1.3 "TrAck Signal", page 16. For more information, for example, about configuring the TrAck limit value, see the corresponding scan system manual. Note: The TrAck limit value cannot be saved for subsequent new starts or resets. excelli*SCAN* scan heads do not support **UpdatePermanentMemory**. Code$_{LOW}$ (hex): 3A — Only excelli*SCAN*: Reset TrAck Signal bits.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	
50	As of scan system firmware ≥ 5100. Only excelli*SCAN*: Refer to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual, Chapter 2.3 "Advanced Settings for "Spot Distance Control"", page 24: dT_{high}.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Code$_{HIGH}$ (hex)	
51	As of scan system firmware ≥ 5100. Only excelli*SCAN*: Refer to "excelliSCAN Scan Heads – Functional Principle of SCANahead Servo Control and Operation by RTC6 Boards" Manual, Chapter 2.3 "Advanced Settings for "Spot Distance Control"", page 24: dT_{low}.

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><td>Code_{HIGH} (hex)</td><td colspan="2"></td></tr>
<tr><td rowspan="5">65</td><td colspan="2">As of scan system firmware ≥ 51*7 + < 5090.

For High-Bandwidth Return Channel Multiplexing. For use in conjunction with Code_{HIGH} (hex) = 66_H and 0574_H. See corresponding Chapter incl. example code.</td></tr>
<tr><td>Code_{LOW} (hex)</td><td></td></tr>
<tr><td>0N</td><td>Sets High-Bandwidth Return Channel Multiplexing block length to N+1 that is, highest High-Bandwidth Return Channel Multiplexing index = N. In other words: number of High-Bandwidth Return Channel Multiplexing values the iDRIVE scan system has to transmit.

Allowed values: 1; 2; 3; 4; 6; 8; 12; 16 (= integer divisor of 192 ≥ 16).</td></tr>
<tr><td>8N</td><td>Sets the next to-be-configured High-Bandwidth Return Channel Multiplexing index to N.
In other words: Defines the position where the signal is transmitted.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

<table>
<tr><td>Code_{HIGH} (hex)</td><td colspan="2"></td></tr>
<tr><td rowspan="3">66</td><td colspan="2">As of scan system firmware ≥ 51*7 + < 5090.

For High-Bandwidth Return Channel Multiplexing. For use in conjunction with Code_{HIGH} (hex) = 65_H and 0574_H. See corresponding Chapter incl. example code.</td></tr>
<tr><td>Code_{LOW} (hex)</td><td></td></tr>
<tr><td>NN</td><td>Sets the return value for the current index to NN = writes the signal value to this index.

For NN, you can insert the same Code_{LOW} values as for SetMode, see Code_{LOW} = 00_H (XY2-100 status word), Code_{LOW} = 01_H (Actual position (angular position of galvanometer scanners)) etc.</td></tr>
</table>

Notice! Generic information. May be incomplete or even incorrect for your scan system.
Always consult the dedicated scan system manual!
Module Rev.1.0.6 en-US

Change Index		
Module Rev.1.0.5 en-US	0536_H	Not only with intelli*WELD*.
Module Rev.1.0.5 en-US	0537_H	Not only with intelli*WELD*.
Module Rev.1.0.6 en-US	0500_H	[Bit #15|Bit #19]. Editorial enhancement.
Module Rev.1.0.6 en-US	0500_H	[Bit #14|Bit #18]. Editorial enhancement.

Comments (RTC6)

- The scan system firmware a.bb.c is read as abbc, for example, 5100 corresponds to version 5.10.0, and 5054 to version 5.05.4.

- On **SetMode**, **SetControlDefinitionMode** and **SetEchoMode**
  - The data type selected by `control_command`(Code$_{HIGH}$ = 05$_H$, 0E$_H$ or 21$_H$ or `Data` = 1700$_H$) is transmitted until another data type is selected.
  - Data transmitted from the scan system to the RTC board can be queried by:
    - **get_value**
    - **get_values**
    - **set_trigger[*]**
    - **get_waveform**

    The first data after switching the data source are transmitted only after a short time delay due to the serial transmission times. Therefore, a delay time of up to 60 s may occur between the switchover time and the first output of the new data type. **control_command** always automatically inserts a waiting time of 60 μs after the data source is switched (the previously mentioned commands always return correct values).
  - All data returned from the scan system to the RTC6 are passed over as signed 20-bit values. This applies even if the RTC6 DLL is set to RTC4 Compatibility Mode and even for scan systems without SL2-100 interface, controlled by an XY2-100 Converter (Accessory). Queried data returned by **get_value**, **get_values** or **get_waveform** are nevertheless generally transferred to the PC as signed 32-bit values (for data evaluation, see comments for **get_value**).
  - For scan systems with integrated SL2-100 interface, **get_head_status** queries the XY2-100 status word regardless of settings made by `control_command`(Code$_{HIGH}$ = 05$_H$, 0E$_H$ or 21$_H$ or `Data` = 1700$_H$). It is returned in addition to the selected data. In contrast, if iDRIVE Scan Systems (*without* an integrated SL2-100 interface) are controlled by an XY2-100 Converter (Accessory), **get_head_status** returns the XY2-100 status word only if this signal has been previously selected to be returned from the scan system by **control_command**, see also Section "Reading Out Data", page 222.
  - After a reset or power-up of the scan system, it can take around 5 seconds for data to be returned from the scan system (see also **get_value**). After a reset or power-up of the scan system, always the XY2-100 status word is returned.

- On **SetMode**
  - Set position (angular position) values returned by the scan system correspond to the effective output values `Sample<AX…BY>_Out` with **set_trigger[*]**. Additionally, the 20-bit set position values returned from a z axis in RTC5 Compatibility Mode (and in RTC4 Compatibility Mode from the x axis and y axis, too) are scaled by a factor 16 relating to the therefore specified 16-bit coordinate values.
  - The angle bit-values (actual position, set position, position error) or angle bit/ms-values (actual speed) returned to the RTC6 can be converted into °-values or °/ms-values by the scan system's calibration angle. The calibration angle can be read out by `Data` = 0523$_H$.
  - Exact values for the internal voltages referred to in the table can vary for different versions of the scan system.
  - intelli*WELD* scan systems can be supplied with various number of sensors. Therefore, observe the manual of the respective scan system. This manual lists the command codes for currently available sensors.

- On **SetPositionScale**
  - If the scaling factor for scaling the position values is changed, then the user program calibration factor for calculating RTC6 control values (in bits) from the desired Image Field position values (in mm), see Chapter 7.3.2 "Image Field Size and Image Field Calibration", page 172, needs to be subjected to an inverse proportional scaling. In addition, the jump speed and mark speed should be adjusted.
  - The currently set scaling factor can be queried with `Data` = $053F_H$.

- On **SetPosAcknowledgeLevel**
  - The limit value must be specified related to a 16-bit position range.
  - The default start behavior is for the scan system to set the limit value to 0.28% of the *full* angular range of an axis (LSB16 or 1/65536, $Code_{LOW}$ = 183 = $B7_H$) after every power-up or reset.
    If other limit values are desired, they must be separately set for each axis ($Code_{HIGH}$ = 15_H). SCANLAB recommends setting only limit values ($Code_{LOW}$) > 14_H (that is, 0.03% of the full position range). Lower values can lead to frequent system safety shutdowns due to Position Acknowledge time outs (set position not reached for an excessive time).

- On **Store/RestoreTransmissionMode**
  - `control_command( Data` = 1700_H) has no effect if a power-up or reset was executed after the most recent execution of `control_command( Data` = $17FF_H$).

21 Appendix F: Change Index

The following are changes in this manual due to the technical evolution of the product as well as significant editorial changes.

In this Chapter:

Changes to Document Revision 1.0.0 en-US from Document Revision 0.0

Where	Why & What
Global	Initial Document-Revision. Document Revision • 1.0.0 en-US applies to RTC6 Software Package • V1.5.0[a]
Appendix F: Change Index, page 1100	

(a) See also RTC6_RevisionHistory_<SW-V>.pdf.

Changes to Document Revision 1.0.1 en-US from Document Revision 1.0.0 en-US

Where	Why & What
Global	Document Revision • 1.0.1 en-US applies to RTC6 Software Package • V1.5.2[a]
eth_get_com_timeouts, page 393	Software change.[a] RTC6 DLL-only settings are now possible even without access to an RTC6 Ethernet Board.
eth_set_com_timeouts, page 408	Same as **eth_get_com_timeouts**.
load_disk, page 543	Software change.[a] Has now a version control.
range_checking, page 622	Software change.[a] `Mode` = 2 added: a **simulate_ext_stop** is passed on to all slave boards.
save_disk, page 647	Same as **load_disk**.
set_wobbel_vector, page 847	Editorial enhancement. To activate “Freely definable wobbel shape” by **set_wobbel_control** and `set_wobbel_mode( Mode = 3 )`, see page 849.
sync_slaves, page 881	Software change.[a] Has no function anymore.
Appendix F: Change Index, page 1100	

(a) See also RTC6_RevisionHistory_<SW-V>.pdf.

Changes to Document Revision 1.0.2 en-US from Document Revision 1.0.1 en-US

Where	Why & What
Global	Document Revision • 1.0.2 en-US applies to RTC6 Software Package • V1.6.0[(a)]
Section “Folder iSCANCfg”, page 33	Software change.[(a)] `iSCANcfg6.exe` is replaced in the RTC6 Software Package by the generic `iSCANcfg.exe`.
Chapter 7.4.1 ”Enabling, Activating and Switching Laser Control Signals”	Editorial change. Figure 51, page 190.
control_command, page 373	Editorial change. Description of “PowerOK” and “TempOK” has been changed, see XY2-100 status word. Furthermore, the description of 0528_H and 0529_H has been removed from the document (these data signal types are not intended for users).
get_error, page 431	Editorial change. error bit `Bit #15` no longer “reserved”, see page 433.
Chapter 16.2.13 ”Master Socket Connector, Slave Socket Connector”	Editorial change. New safety notice on Master/Slave-connected RTC6 Ethernet Boards, see page 973.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.3 en-US from Document Revision 1.0.2 en-US

Where	Why & What
Global	Document Revision • 1.0.3 en-US applies to RTC6 Software Package • V1.6.1[(a)]
Global	Editorial change. The previously used term "Online Positioning" (due to the introduction of "Global Online Positioning") now reads - where applicable - "Local Online Positioning", for example, in Chapter 8.3.1 ""Local Online Positioning"", page 237.
Chapter 8.3.2 ""Global Online Positioning"", page 240	Editorial enhancement. Description of "Global Online Positioning".
Chapter 8.4 "Wobbel Mode"	Editorial enhancement. Section "Example Code (C++)", page 242.
Chapter 8.6.12 ""Fly Extension" Commands", page 268	Editorial enhancement. Description of "Fly Extension" Commands.
activate_fly_1_axis, page 344	Software change.[(a)] New command.
activate_fly_2_axes, page 345	Software change.[(a)] New command.
control_command, page 373	Editorial change. The description of $052B_H$ and $052C_H$ has been removed from the document (these data signal types are not intended for users).
fly_return_1_axis, page 421	Software change.[(a)] New command.
fly_return_2_axes, page 422	Software change.[(a)] New command.
fly_return_3_axes, page 423	Software change.[(a)] New command.
get_startstop_info, page 462	Software change.[(a)] New Bit #14.
park_position_1_axis, page 612	Software change.[(a)] New command.
park_position_2_axes, page 613	Software change.[(a)] New command.
park_return_1_axis, page 616	Software change.[(a)] New command.
park_return_2_axes, page 617	Software change.[(a)] New command.
set_fly_1_axis, page 696	Software change.[(a)] New command.
set_fly_2_axes, page 697	Software change.[(a)] New command.
set_fly_3_axes, page 701	Software change.[(a)] New command.
set_mcbsp_global_matrix, page 747	Software change.[(a)] New command.
set_mcbsp_global_matrix_list, page 748	Software change.[(a)] New command.
set_mcbsp_global_rot, page 749	Software change.[(a)] New command.
set_mcbsp_global_rot_list, page 749	Software change.[(a)] New command.
set_mcbsp_global_x, page 750	Software change.[(a)] New command.

Where (cont'd.)	Why & What (cont'd.)
set_mcbsp_global_x_list, page 750	Software change.[a] New command.
set_mcbsp_global_y, page 751	Software change.[a] New command.
set_mcbsp_global_y_list, page 751	Software change.[a] New command.
store_timestamp_counter, page 873	Software change.[a] New command.
store_timestamp_counter_list, page 873	Software change.[a] New command.
wait_for_1_axis, page 912	Software change.[a] New command.
wait_for_2_axes, page 914	Software change.[a] New command.
wait_for_timestamp_counter, page 922	Software change.[a] New command.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.4 en-US from Document Revision 1.0.3 en-US

Where	Why & What
Global	Document Revision • 1.0.4 en-US applies to RTC6 Software Package • V1.7.0[a]
Section "Folder RTC6 Tools", page 35	Software change.[a] New file. ≥ BIOS-ETH 26 is a prerequisite for using the Standalone Functionality.
eth_boot_dcmd, page 382	Software change.[a] New command.
set_eth_boot_control, page 691	Software change.[a] New command.
eth_boot_timeout, page 382	Software change.[a] New command.
set_free_variable, page 715	Software change.[a] Changed command.
read_image_eth, page 628	Software change.[a] New command.
store_program, page 872	Software change.[a] New command.
write_image_eth, page 934	Software change.[a] New command.
Chapter 16.2.15 "Real-Time Clock", page 977	Editorial enhancement. Description of the RTC6 Ethernet Board real-time clock.
Chapter 16.7 "Standalone Functionality", page 988	Software change.[a] New functionality for RTC6 Ethernet Boards: Standalone Functionality.
Chapter 18 "Appendix C: RTC6 EtherBox", page 1047	Editorial enhancement. Description of the RTC6 EtherBox.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.5 en-US from Document Revision 1.0.4 en-US

Where	Why & What
Global	Document Revision • 1.0.5 en-US applies to RTC6 Software Package • V1.7.5[a]
Chapter 4.6.1 "LASER Connector", page 75	Editorial enhancement. Note for laserDESK users on external documents.
eth_boot_timeout, page 382	Editorial change. Command name is **eth_boot_timeout** (not: **set_eth_boot_timeout**).
eth_get_com_timeouts_auto, page 394	Software change.[a] New command.
eth_set_com_timeouts, page 408	Software change.[a] Changed command.
eth_set_com_timeouts_auto, page 409	Software change.[a] New command.
get_head_status, page 439	Editorial change. Description corrected.
para_mark_abs, page 605	Software change.[a] Changed command.
wait_for_timestamp_counter_mode, page 924	Software change.[a] New command.
Chapter 16.2.4 "LASER Socket Connector", page 961	Editorial enhancement. Note for laserDESK users on external documents.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.6 en-US from Document Revision 1.0.5 en-US

Where	Why & What
Global	Document Revision • 1.0.6 en-US applies to RTC6 Software Package • V1.7.6[(a)]
Section "Folder RTC6 Files", page 34	Software change.[(a)] Other internal structure of the RTC6 Software Package as well as more files.
get_startstop_info, page 462	Software change.[(a)] New Bit #15.
set_laser_delays, page 729	Editorial change. Default setting after **load_program_file** corrected.
wait_for_encoder_mode, page 919	Editorial change. Allowed value range of `Value` corrected.
wait_for_encoder_in_range_mode, page 918	Editorial change. Allowed value range of `EncXmin` corrected.
eth_configure_link_loss, page 386	Software change.[(a)] New command.
Chapter 16.5.4 "Checking the Connection to the RTC6 Ethernet Board", page 986	Editorial enhancement.
Chapter 16.10.3 "TI-RTOS", page 1038	Editorial enhancement. The RTC6 Ethernet Board uses 3rd party software.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.7 en-US from Document Revision 1.0.6 en-US

Where	Why & What
Global	Document Revision • 1.0.7 en-US applies to RTC6 Software Package • V1.7.7[(a)]
load_program_file, page 556	Editorial enhancement.
Chapter 18 "Appendix C: RTC6 EtherBox", page 1047	Hardware change. Figure 95, Figure 96.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.8 en-US from Document Revision 1.0.7 en-US

Where	Why & What
Global	Document Revision • 1.0.8 en-US applies to RTC6 Software Package • V1.7.8[(a)]
Chapter 8.12 "Time Measurements", page 290	Editorial enhancement. 64-bit "Timestamp Counter".
get_timestamp_long, page 474	Software change.[(a)] New command.
wait_for_timestamp_counter_long, page 923	Software change.[(a)] New command.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.9 en-US from Document Revision 1.0.8 en-US

Where	Why & What
Global	Document Revision • 1.0.9 en-US applies to RTC6 Software Package • V1.7.9[(a)]
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 1055	Editorial enhancement.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.10 en-US from Document Revision 1.0.9 en-US

Where	Why & What
Global	Document Revision • 1.0.10 en-US applies to RTC6 Software Package • V1.7.11[(a)]
Chapter 7.4.11 "Pulse Synchronization Mode", page 217	Software change.[(a)]
Chapter 8.2 "Coordinate Transformations", page 233	Editorial enhancement. Notes on data recording, page 235.
eth_get_last_error, page 398	Software change.[(a)] New Bit 27.
regulation3, page 637	Software change.[(a)] New command.
set_defocus_list, page 676	Editorial enhancement. New comment, page 676.
set_duty_cycle_table, page 684	Software change.[(a)] New command.
set_laser_pulse_sync, page 734	Software change.[(a)] New command.
set_laser_timing_table, page 738	Software change.[(a)] New command.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.11 en-US from Document Revision 1.0.10 en-US

Where	Why & What
Global	Document Revision • 1.0.11 en-US applies to RTC6 Software Package • V1.7.11[(a)]
Chapter 18 "Appendix C: RTC6 EtherBox", page 1047	Product renamed from "RTC6 Ethernet Plug-in+Box" to "RTC6 EtherBox". Removed: #0145963, #0146152, #0146155, #0146368, #0146371. Added: #0148520. #0148519.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.12 en-US from Document Revision 1.0.11 en-US

Where	Why & What
Global	Document Revision • 1.0.12 en-US applies to RTC6 Software Package • V1.7.12[(a)]
Section “Folder RTC6 Tools”, page 35	Software change.[(a)] BIOS-ETH 29.
Chapter 16.7.1 ”Upgrading BIOS-ETH”, page 989	Software change.[(a)] BIOS-ETH 29.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.13 en-US from Document Revision 1.0.12 en-US

Where	Why & What
Global	Document Revision • 1.0.13 en-US applies to RTC6 Software Package • V1.7.12[(a)]
control_command, page 373	Editorial change. $052A_H$: Stop Event Code $000D0_H$ is not intended for users.
Chapter 14.1 ”EU Declaration of Conformity – RTC6 PCIe Board”, page 948	Editorial change. Replaces Chapter “Compliance with EC Directive for Electromagnetic Compatibility (EMC)”.
Chapter 16.10.1 ”EU Declaration of Conformity – RTC6 Ethernet Board”, page 1037	Editorial change. Replaces Chapter “Compliance with EC Directive for Electromagnetic Compatibility (EMC)”.
Chapter 18.4.1 ”EU Declaration of Conformity – RTC6 EtherBox”, page 1052	Editorial enhancement.
Chapter 18.4.3 ”Compliance with FCC Rules”, page 1054	Editorial enhancement.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.14 en-US from Document Revision 1.0.13 en-US

Where	Why & What
Global	Document Revision • 1.0.14 en-US applies to RTC6 Software Package • V1.9.0[(a)]
Section "Subfolder Linux", page 34	Software change.[(a)] New subfolder.
Section "Folder RTC6 Tools", page 35	Software change.[(a)] BIOS-ETH 30.
clear_fly_overflow, page 367	Editorial change. `Mode = 63` (not = 15).
clear_fly_overflow_ctrl, page 367	Software change.[(a)] New command.
eth_get_standalone_status, page 402	Software change.[(a)] New command.
get_rtc_version, page 456	Software change.[(a)] Bit #11.
set_sky_writing, page 802	Editorial enhancement. RTC5→RTC6.
set_sky_writing_list, page 804	Editorial enhancement. RTC5→RTC6.
set_sky_writing_para, page 808	Editorial enhancement. RTC5→RTC6.
set_sky_writing_para_list, page 810	Editorial enhancement. RTC5→RTC6.
Chapter 16.1.3 "Options", page 957	New Option "LDSA" (RTC6 Ethernet Boards only).
Chapter 16.7.1 "Upgrading BIOS-ETH", page 989	Software change.[(a)] BIOS-ETH 30.
Chapter 16.7.6 "Automatic Booting – Process in Detail", page 994	Editorial enhancement.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 1055	Software change.[(a)] V1.9.0.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.15 en-US from Document Revision 1.0.14 en-US

Where	Why & What
Global	Document Revision • 1.0.15 en-US applies to RTC6 Software Package • V1.9.0[a]
Chapter 2.10 "Notes for RTC4 Users", page 49	Editorial enhancement. Encoder counter direction of RTC4 boards, see Notice!, page 59.
control_command, page 373	Editorial change. Content moved to Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards", page 1057.
list_repeat, page 533	Editorial enhancement. New comment, page 533.
Chapter 16.3.1 "Hardware Installation", page 978	Editorial enhancement. Figure 90: voltage supply of multiple RTC6 Ethernet Boards/RTC6 EtherBoxes.
Chapter 18.5 "Technical Specifications – RTC6 EtherBox", page 1054	Editorial enhancement.
Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards", page 1057	Editorial enhancement.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.16 en-US from Document Revision 1.0.15 en-US

Where	Why & What
Global	Document Revision • 1.0.16 en-US applies to RTC6 Software Package • V1.11.0[a]
Chapter 1 "About this Manual", page 25	Editorial change.
Section "Folder RTC6 Tools", page 35	Software change.[a] BIOS-ETH 32.
acquire_rtc, page 342	Editorial enhancement. New comment, page 343.
set_wobbel_control, page 839	Software change.[a] `Ctrl` = 7, `Ctrl` = 8.
Chapter 16.7.1 "Upgrading BIOS-ETH", page 989	Software change.[a] BIOS-ETH 32.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 1055	Software change.[a] V1.10.0, V1.11.0.
Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards", page 1057	Editorial change. Module Rev.1.0.1 en-US.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.17 en-US from Document Revision 1.0.16 en-US

Where	Why & What
Global	Document Revision • 1.0.17 en-US applies to RTC6 Software Package • V1.11.0(a)
Chapter 2.8.6 "Slot Cover with 15-pin D-SUB Connector and 9-pin D-SUB Connector", page 47	Editorial enhancement. #0130209.
Chapter 16.3.1 "Hardware Installation", page 978	Editorial change. Figure 90.
Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards", page 1057	Editorial change. Module Rev.1.0.2 en-US.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.18 en-US from Document Revision 1.0.17 en-US

Where	**Why & What**
Global	Document Revision • 1.0.18 en-US applies to RTC6 Software Package • V1.12.0[(a)]
Chapter 1 "About this Manual", page 25	Editorial change.
Section "Folder RTC6 Tools", page 35	Software change.[(a)] BIOS-ETH 33.
Section "Checking the z axis Calibration", page 175	Editorial enhancement. Notes, page 176 and RTC6 commands from V1.12.0.
get_temperature, page 471	Software change.[(a)] New command.
load_z_table_20b, page 565	Software change.[(a)] New command.
load_z_table_no_20b, page 567	Software change.[(a)] New command.
read_abc_from_file_20b, page 625	Software change.[(a)] New command.
set_wobbel_control, page 839	Software change.[(a)] With `Ctrl` = 8, now alternation occurs with every Microstep (no longer with every **set_wobbel_vector** call.
set_wobbel_vector_2, page 851	Software change.[(a)] New command.
write_abc_to_file_20b, page 928	Software change.[(a)] New command.
Chapter 16.7.1 "Upgrading BIOS-ETH", page 989	Software change.[(a)] BIOS-ETH 33.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 1055	Software change.[(a)] V1.12.0.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.19 en-US from Document Revision 1.0.18 en-US

Where	Why & What
Global	Document Revision • 1.0.19 en-US applies to RTC6 Software Package • V1.13.0[(a)]
Chapter 1 "About this Manual", page 25	Editorial change.
Section "Folder RTC6 Tools", page 35	Software change.[(a)] BIOS-ETH 33.
Chapter 7.3.4 "3D Image Field", page 176	Editorial enhancement. ABC values from the `*_ReadMe.txt` file of correction file are to be interpreted as 16-bit values.
load_z_table_20b, page 565	Editorial change. Allowed value range for `A`, `B`, `C`.
load_z_table_no_20b, page 567	Editorial change. Allowed value range for `A`, `B`, `C`.
set_mcbsp_out_ptr, page 760	Software change.[(a)] `Number` Bit 31.
set_mcbsp_out_ptr_list, page 762	Software change.[(a)] New command.
set_wobbel_vector, page 847	Software change.[(a)] Calculation for the variation of the current laser power P after `set_wobbel_mode`(`Mode` = 2): formula, page 848 (instead of: P = P0 × (1 + `dPower` × n).
set_wobbel_vector, page 847	Software change.[(a)] Calculation for the variation of the current laser power P after `set_wobbel_mode`(`Mode` = 3): formula, page 849 (instead of: P = P100 × (factor + `dPower` × n)).
Chapter 16.7.1 "Upgrading BIOS-ETH", page 989	Software change.[(a)] BIOS-ETH 33.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 1055	Software change.[(a)] V1.13.0.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.0.20 en-US from Document Revision 1.0.19 en-US

Where	Why & What
Global	Document Revision • 1.0.20 en-US applies to RTC6 Software Package • V1.14.1[(a)]
Chapter 1 "About this Manual", page 25	Editorial change.
Section "Folder RTC6 Tools", page 35	Software change.[(a)] BIOS-ETH 35.
Chapter 4.4 "Master Socket Connector, Slave Socket Connector", page 68	Hardware change. #0149963 (previously: #0117241).
eth_set_high_performance_mode, page 410	Software change.[(a)] New command.
get_error, page 431	Software change.[(a)] Error bit Bit #17 no longer "reserved", see page 433.
get_list_pointer, page 447	Editorial enhancement. New comment, page 447.
set_laser_control, page 726	Editorial enhancement. Further clarification with Bit #3 and Bit #4.
set_mcbsp_out_oie_ctrl, page 758	Software change.[(a)] New command.
set_mcbsp_out_oie_list, page 759	Software change.[(a)] New command.
set_vector_control, page 833	Software change. `Ctrl` = 8.
wait_for_encoder_mode, page 919	Editorial change. **wait_for_encoder_mode** is category "Normal List Command" (not: "Multiple List Command").
write_port_list, page 937	Software change.[(a)] New command.
Chapter 16.7.1 "Upgrading BIOS-ETH", page 989	Software change.[(a)] BIOS-ETH 35.
Chapter 16.8 ""High Performance Mode"", page 996	Software change.[(a)] New functionality for RTC6 Ethernet Boards: "High Performance Mode".

Where (cont'd.)	Why & What (cont'd.)
Chapter 18.2 "Cabling", page 1050	Hardware change. #0149964 (previously: #0148519).
Chapter 18.5 "Technical Specifications – RTC6 EtherBox", page 1054	Weight, page 1054.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 1055	Software change.(a) V1.14.1.
Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards", page 1057	Editorial change. Module Rev.1.0.6 en-US.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.1.0 en-US from Document Revision 1.0.20 en-US

Where	Why & What
Global	Document Revision • 1.1.0 en-US applies to RTC6 Software Package • 1.16.3[(a)]
Chapter 1 "About this Manual", page 25	Software change.[(a)]
Chapter 1.2 "Related Documents", page 26	Editorial enhancement. StreamParser-DLL – Application Programming Interface Manual.
Section "Folder RTC6 Tools", page 35	Software change.[(a)] BIOS 23, BIOS-ETH 40.
Chapter 5.5 "Changing RTC6 PCIe Board BIOS", page 94	Editorial enhancement.
Chapter 6.4.3 "List Execution Status", page 111	Editorial enhancement. HEAD BUSY list execution status.
Chapter 8.13 "SCANahead Functionality", page 291	Editorial enhancement.
Section "Input Ports for External Encoder Signals", page 321	Software change.[(a)]
eth_config_waveform_streaming_ctrl, page 385	Software change.[(a)] New command.
eth_set_remote_tgm_format, page 412	Software change.[(a)] New command.
get_error, page 431	Editorial enhancement. Bit #31.
get_transform, page 475	Software change.[(a)] See Version Info, page 475.
get_transform_offset, page 476	Software change.[(a)] New command.
get_waveform, page 485	Software change.[(a)] See Version Info, page 485.
get_waveform_offset, page 486	Software change.[(a)] See Version Info, page 486.
init_fly_2d_list, page 505	Software change.[(a)] New command.
set_auto_laser_control, page 660	Software change.[(a)] See Version Info, page 664. M = +64.
set_auto_laser_params, page 665	Editorial enhancement. Result.

Where (cont'd.)	Why & What (cont'd.)
set_ellipse, page 685	Software change.(a) See Version Info, page 685.
set_encoder_filter_ctrl, page 686	Software change.(a) New command.
set_fly_1_axis, page 696	Editorial enhancement. Parameter for scaling factor ignored for Mode 8, 12, 16, page 696.
set_fly_2_axes, page 697	Editorial enhancement. Parameters for scaling factor ignored for Mode 8, 12, 16, page 697.
set_fly_3_axes, page 701	Editorial enhancement. Parameters for scaling factor ignored for Mode 8, 12, 16, page 701.
set_scanahead_params, page 795	Editorial change. With error code 3, no **get_last_error** error is set.
set_trigger, page 822	Editorial enhancement. New comment, page 830.
set_trigger8, page 832	Software change.(a) New command.
store_program, page 872	Software change.(a) See Version Info, page 872.
Chapter 16.7.1 "Upgrading BIOS-ETH", page 989	Software change.(a) BIOS-ETH 40.
Chapter 16.8 ""High Performance Mode"", page 996	Software change.(a) Further clarification.
Chapter 16.9 "Remote Interface Mode", page 997	Software change.(a) New functionality for RTC6 Ethernet Boards: Remote Interface Mode.
Chapter 18.4.2 "Confirmation of Electromagnetic Compatibility (EMC)", page 1053	Editorial enhancement.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 1055	Software change.(a) V1.16.3.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.

Changes to Document Revision 1.1.1 en-US from Document Revision 1.1.0 en-US

Where	Why & What
Global	Document Revision • 1.1.1 en-US applies to RTC6 Software Package • 1.18.0[a]
Chapter 1 "About this Manual", page 25	Software change.
Section "Subfolder Linux", page 34	Software change.[a] Package for Debian Bookworm (Debian Stretch no longer supported).
Section "Folder RTC6 Tools", page 35	Software change.[a] BIOS 23, BIOS-ETH 42.
Section "Sky Writing Mode 4", page 166	Software change.[a] New functionality for RTC6 boards: Sky Writing Mode 4.
Section "Sky Writing with Minimum Mark Speed", page 171	Software change.[a]
Section "2D Position-Dependent Laser Control", page 215	Software change.[a] New functionality for RTC6 boards: 2D Position-Dependent Laser Control.
Chapter 8.6.10 "Tracking Error Compensation of Encoder Values for Processing-on-the-fly Applications", page 266	Software change.[a]
fly_disable_list, page 418	Software change.[a] New command.
fly_prediction, page 419	Software change.[a] New command.
get_marking_info, page 449	Editorial change. Text of `Bit #17` and `Bit #18` was mixed up.
load_position_control_2d_ctrl, page 554	Software change.[a] New command.
set_auto_laser_control, page 660	Software change.[a] See Version Info, page 664. M = +128.
set_auto_laser_control, page 660	Editorial enhancement.[a] Comment, page 662.
set_controlpreview_compensation_ctrl, page 673	Software change.[a] New command.

Where (cont'd.)	Why & What (cont'd.)
set_defocus_2_list, page 677	Software change.[a] New command.
set_encoder_filter_ctrl, page 686	Software change.[a] See Version Info, page 687.
set_fly_tracking_error, page 707	Software change.[a] Command now with functionality.
set_short_cmd_mode_ctrl, page 801	Software change.[a] New command.
set_short_cmd_mode_list, page 801	Software change.[a] New command.
set_sky_writing_min_speed_ctrl, page 805	Software change.[a] New command.
set_sky_writing_mode, page 806	Software change.[a] `Mode` > 3 (Sky Writing Mode 4).
set_sky_writing_mode_list, page 807	Software change.[a] `Mode` > 3 (Sky Writing Mode 4).
upload_transform, page 909	Editorial change. The user program must provide a memory area of size 528,524 bytes on the PC (not: 528,520 bytes), page 910.
Section "RTC6 Commands in other Publications", page 939	Editorial enhancement.
Chapter 16.7.1 "Upgrading BIOS-ETH", page 989	Software change.[a] BIOS-ETH 42.
Section "Remote Control Commands", page 1006	Software change.[a] New: ID `158`…ID `170`.
Section "Remote List Commands", page 1016	Software change.[a] New: ID `428`, ID `429`, ID `430`.
Chapter 16.9.10 "RTC6 Commands not Available as Remote Commands", page 1028	Editorial enhancement.
Chapter 19 "Appendix D: RTC6 Software Packages – History", page 1055	Software change.[a] V1.18.0.
Chapter 20 "Appendix E: iDRIVE Scan Systems – Control Commands and Signals Transmitted to RTC Boards", page 1057	Editorial change. Module Rev.1.0.6 en-US.
Appendix F: Change Index, page 1100	

(a) See also `RTC6_RevisionHistory_<SW-V>.pdf`.