MANUAL

APPLICATIONS

FOR "CUA-USB" AND "CUA-ETH" CONTROL UNITS

NEWSON ENGINEERING NV

Table of Contents

1 LIBRARY RHOTHOR.DLL

1.1 LINKING THE LIBRARY WITH AN APPLICATION

Before the functionalities of the control unit can be used within any application, the library "rhothor.dll" needs to be linked to it. As with all Dynamic Link Libraries, implicit or explicit linking can be used. The installation CD contains the necessary files to perform this linking.

DLL files on the installation CD-ROM:

Path Visual C++	\rhothor\src\dll\vc\
Path Borland C	\rhothor\src\dll\bc\

1.2 IMPLICIT LINKING

When using implicit linking, the DLL is loaded during start-up of the application using that DLL. The application can simply use the library functions like any other function in its source code. This way of linking is achieved as follows:

1. copy "rhothor.dll" to the application's directory or to the windows default DLL directory. Usually this is c:\windows\system32.
2. add library "rhothor.lib" to the project. (*)
3. include "rhothordll.h", the DLL header file, in the source code.

(*) In Borland C++ environments the command IMPLIB is used to create the lib file.

1.3 EXPLICIT LINKING

With explicit linking, applications must make a function call to explicitly load the DLL at run time. To explicitly link to a DLL, an application one must:

1. Call LoadLibrary (*) to load the DLL and obtain a module handle. If successful, the function maps the DLL into the applications address space.
2. Call GetProcAddress (*) from within the application for each function of interest to obtain a function pointer. This pointer can then be used by the application to call the function.
3. Call FreeLibrary (*) when done with the DLL. This function releases the DLL resources.

(*) exact function name may differ with programming language or compiler type

1.4 USING THE LIBRARY FROM WITHIN AN APPLICATION

After linking, the library provides the application methods and properties to access, query, calibrate and control the deflection system.

1.5 ACCESSING FUNCTIONS

Accessing functions provide the means to set-up a connection between application and the laser deflection system. This is achieved using the following function:

summary

```
long rtSelectDevice(char* IP)
```

A distinction must be made between the CUA-USB and CUA-ETH versions of the control system. For the latter, the argument is a string containing the IP-address of the control unit. In the former case the name of the usb device must be used (this can be set in the Rhothor software and is particularly useful for larger set-ups using multiple deflection heads).

When using the device name "USB" the frist free USB rhothor device is addressed. For multihead systems the functions rtGetFirstFreeUSBDevice and rtGetNextFreeUSBDevice can be used to build up a device list.

summary

```
long rtGetFirstFreeUSBDevice(char* Name)
long rtGetNextFreeUSBDevice(char* Name)
```

1.6 Query Functions

Configuration of the control system must be done through the configuration software, see manual "A2G_Cfg". However, the following family of functions allows the application to query configuration settings and status information of the deflection system.

summary

name	parameters	units	range
rtGetCanLink	long Address		
	long* Value		
rtGetCounter	long* Value		
rtGetFieldSize	double* Size	mm	
rtGetFieldSizeZ	double* Size	mm	
rtGetID	char* Name		
rtGetIO	long* Value		
rtGetLaserLink	long Address		
	long* Value		
rtGetMaxSpeed	double* Speed	mm/sec	
rtGetResolvers	double* X	mm	
	double* Y	mm	
rtGetScannerDelay	long* Delay	µsec	
rtGetSerial	long* Serial		
rtGetSetpointFilter	long* TimeConst	µsec	
rtGetStatus	long* Memory	bytes	
rtGetVersion	char* Version		

1.7 Emulate Functions

It is possible to put the rhothor.dll in an emulation mode to generate flash jobs which can be transferred over a PLC to the actual deflection head. The emulation functions allow to parameter the virtual hardware in order to generate the correct flash job. Emulation mode is enabled by addressing a virtual hardware device. The virtual hardware device is addressed through rtSelectDevice(“none”)

summary

name	parameters	units	range
rtSetFieldSize	double Size	mm	

1.8 CALIBRATION FUNCTIONS

The calibration functions allow for custom calibration settings required to compensate for errors induced by the optics in the deflection head. See section 4 for more details on the calibration process and input data.

summary

name	parameters	units
rtAddCalibrationData	const char* FileName	
rtLoadCalibration		
rtLoadCalibrationFile	const char* FileName	
rtResetCalibration		
rtStoreCalibrationFile	const char* FileName	
rtAddCalibrationDataZ	const char* FileName	
rtLoadCalibrationFileZ	const char* FileName	
rtResetCalibrationZ		
rtStoreCalibration		
rtStoreCalibrationFileZ	const char* FileName	
rtStoreCalibrationFile	const char* FileName	
bcSamplePoint	double X	mm
	double Y	mm
	long Row	
	long Col	
	double Sweep	mm
	double* OffsetX	mm
	double* OffsetY	mm

1.9 FLASH FILE FUNCTIONS

The deflection control system is fitted with flash memory. This memory is formatted using a dedicated layout not compatible with standard operating system calls. Therefore the content of the flash memory can only be altered by the use of the library flash file functions.

summary

name	parameters
rtIndexFetch	long Index
rtFileUpload	const char* SrcFile
	const char* FileName
rtFileUploadAtIndex	const char* SrcFile
	const char* FileName
	long Index
rtFileDownload	const char* FileName
	const char* DestFile
rtFormatFlash	
rtEraseFromFlash	const char* FileName
rtGetFileIndex	const char* FileName
	long* Index
rtGetFlashFirstFileEntry	char* Name
	long* Size
rtGetFlashNextFileEntry	char* Name
	long* Size

rtGetFlashMemorySizes	long* Total
	long* Allocated
rtFileClose	
rtFileOpen	long Mode
	char* FileName
rtFileFetch	char* FileName

1.10 CONTROL FUNCTIONS

Control functions allow the application to send commands to the deflection control system during marking. Commands received by the deflection control system are executed. To avoid that the system stalls, a buffer containing the list of consecutive commands is used. This buffering has to be managed by the application and supports two modes of execution: mode LIST_START and mode AUTO_START. Calling control functions prior to selecting an execution mode will generate errors.

Mode LIST_START
This mode is selected by calling “rtListOpen(1)”. In LIST_START mode up to two command lists are used. This means that the application can fill up one list while the other one is being executed. Execution of a list has to be started explicitly by calling function “rtListClose()”. During execution of a list another function call to “rtListOpen(1)” will open a second list for command entry. If the second one is closed before the running one is terminated, all its commands will be moved into the running one.

Mode AUTO_START
This mode is selected by calling “rtListOpen(2)”. In AUTO_START mode only one command list is used. The list acts like a FIFO between application and deflection control system. All marking commands will be automatically sent to the deflection control system and marking will start immediately. When all commands are sent, the application needs to close the list to release the final content for processing.

Mode BOOT_START
This mode is selected by calling “rtListOpen(3)”. In BOOT_START mode a static command list is completely stored on on the boot sector of the control board. This means the command list will be automatically executed whenever the control board boots. This mode is usefull in standalone applications. Due to the size of the boot sector the number of control functions in this command list is limited. This time limitation can easily be avoided by using macro calls “rIndexFetch”.

Mode LOAD_START
This mode is selected by calling “rtListOpen(4)”. In LOAD_START mode a static command list is completely stored on on the local memory of the controller. The application has to fill up the complete list prior to its execution. Due to the size of the local memory, the number of control functions in the list is limited. This time limitation can easily be avoided by using macro calls “rIndexFetch”. A macro call takes up a few bytes of local memory but may contain a virtual endless list of control functions stored in flash memory. Execution of the list is started by calling the function “rtListClose()”. Because the complete list is stored on the local memory iteration statements (rtSetLoop / rtDoLoop) can be used to execute some control functions 1, some or endless times. Execution termination can be forced with the rtAbort function.

List execution mode can be altered and execution can be aborted by the application at any time. In case of the latter, the deflection control system will stop immediately and all remaining commands in the lists will be erased.

During command processing the deflection control system controls the laser through a gate signal. When no list is being executed by the system, or when the system stalls, the laser is switched off.

summary

name	parameters	units	range	gate (*)	listmode (**)
rtAbort				off	n/a
rtArcTo	double X	mm		on	1,2,3,4,F
	double Y	mm			
	double BF		-1…1		
rtArcMoveTo	double X	mm		off	1,2,3,4,F
	double Y	mm			
	double BF				
rtBurst	long Time	µsec		on	1,2,3,4,F
rtCircle	double X	mm		on	1,2,3,4,F
	double Y	mm			
	double Angle	deg			
rtCircleMove	double X	mm		off	1,2,3,4,F
	double Y	mm			
	double Angle	deg			
rtDoLoop				off	3,4
rtElse				off	3,4
rtEndIf				off	3,4
rtIfIO	long Value			off	3,4
	long Mask				
rtIncrementCounter				off	1,2,3,4,F
rtJumpTo	double X	mm		off	1,2,3,4,F
	double Y	mm			
rtJumpTo3D	double X	mm		off	1,2,3,4,F
	double Y	mm			
	double Z	mm			
rtLineTo	double X	mm		on	1,2,3,4,F
	double Y	mm			
rtLineTo3D	double X	mm		on	1,2,3,4,F
	double Y	mm			
	double Z	mm			
rtListClose				n/a	n/a
rtListOpen	long Mode		1,2,3	n/a	n/a
rtMoveTo	double X	mm		off	1,2,3,4,F
	double Y	mm			
rtMoveTo3D	double X	mm		off	1,2,3,4,F
	double Y	mm			
	double Z	mm			
rtOpenCanLink	long Baudrate	Bd		n/a	1,2,3,4,F
rtPulse	double X	mm		pulsed	1,2,3,4,F
	double Y	mm			
rtPulse3D	double X	mm		pulsed	1,2,3,4,F
	double Y	mm			
	double Z	mm			
rtResetCounter				n/a	1,2,3,4,F
rtResetResolver	long Nr			n/a	1,2,3,4,F

rtScanCanLink	long Address			n/a	1,2,3,4,F
	long Node				
	long Index				
	long SubIndex				
rtSendUartLink	char* Data			n/a	1,2,3,4,F
rtSetCanLink	long Node			n/a	1,2,3,4,F
	long Index				
	long SubIndex				
	char* Data				
rtSetIO	long Value			off	1,2,3,4,F
	long Mask				
rtSetImageMatrix	double a11			n/a	1,2,3,4,F
	double a12				
	double a21				
	double a22				
rtSetImageMatrix	double a11			n/a	1,2,3,4,F
	double a12				
	double a21				
	double a22				
	double a31				
	double a32				
rtSetImageOffsXY	double X	mm		n/a	1,2,3,4,F
	double Y	mm			
rtSetImageOffsRelXY	double X	mm		n/a	1,2,3,4,F
	double Y	mm			
rtSetImageOffsZ	double Z			n/a	1,2,3,4,F
rtSetJumpSpeed	double Speed	mm/sec		n/a	1,2,3,4,F
rtSetLaser	bool OnOff			OnOff	1,2,3,4,F
rtSetLaserLink	long Address				1,2,3,4,F
	long Value				
rtSetLaserTimes	long GateOnDelay	µsec		off	1,2,3,4,F
	long GateOffDelay	µsec			
rtSetLoop	long Count			off	3,4
rtSetMatrix	double a11			off	1,2,3,4,F
	double a12				
	double a21				
	double a22				
rtSetMaxSpeed	double Speed	mm/sec			1,2,3,4,F
rtSetOffsIndex	long Index				1,2,3,4,F
rtSetOffsXY	double X	mm		off	1,2,3,4,F
	double Y	mm			
rtSetOffsZ	double Z	mm		off	1,2,3,4,F
rtSetOscillator	long Nr		1,2,3	off	1,2,3,4,F
	double Period	µsec	<=1000000		
	double PulseWidth	µsec			
rtSetOTF	long Nr				1,2,3,4,F
	bool On				
rtSetResolver	long Nr		1,2	off	1,2,3,4,F

	double StepSize	mm			
	double Range	mm			
rtSetResolverRange	long Nr				1,2,3,4,F
	double Range	mm			
rtSetResolverTrigger	long Nr				1,2,3,4,F
	double Position	mm			
	long IO				
rtSetSpeed	double Speed	mm/sec		n/a	1,2,3,4,F
rtSetVarBlock	long i				1,2,3,4,F
	char Data				
rtSetWobble	double Diam	mm			1,2,3,4,F
	long Freq	Hz			
rtSleep	long Time	µsec		off	1,2,3,4,F
rtWaitCanLink	long Address				1,2,3,4,F
	long Value				
	long long Mask				
rtWaitIO	long Value			off	1,2,3,4,F
	long Mask				
rtWaitPosition	double Window				1,2,3,4,F
rtWaitResolver	long Nr		1,2	off	1,2,3,4,F
	double TriggerPos	mm			
	long TriggerMode		1,2		

(*) Legend on gate signals

n/a:	not applicable
off:	gate off
on:	gate on
pulse:	off - gate on during 5 µsec - off

(**) legend on supported list modes

n/a:	not applicable
1:	usable in list mode 1 – LIST_START
2:	usable in list mode 2 – AUTO_START
3:	usable in list mode 3 – BOOT_START
4:	usable in list mode 4 – LOAD_START
F:	usable in flash job creation (rtFileOpen)

All control functions return an error code (data type long). For most of them, their values are as listed below. See the appendix for deviating return codes.

name	value	description
ERR_OK	-1	control command added to the list successfully
ERR_BUSY	2	the control system is processing a job
ERR_JOB	3	there is no open list, call "rtListOpen"
ERR_HARDWARE	5	the deflection control system is not responding
ERR_MEMORY	15	can't allocate memory
ERR_DATA	13	the control command holds invalid parameters
ERR_IMPLEMENTATION	23	the command is not supported

ERR_LASER_LOW	42	insufficient laser power
ERR_TRACKING	43	system is not running
ERR_FILE_IN_USE	47	file in use in bootstart

1.11 CONDITIONAL FUNCTIONS

Statements in the list are normally executed sequentially in the order in which they have been added to the execution list. This is called sequential execution and this is the default program flow. In order to generate stand-alone scripts (ListMode 3 and 4) some conditional statements and iterative statements exist to modify the program flow.

summary

name	parameters	units	range	gate (*)	listmode (**)
rtDoLoop				off	3,4
rtDoWhile				off	3,4
rtElse				off	3,4
rtElseIfIO	Value			off	3,4
	Mask				
rtEndIF				off	3,4
rtIfIO	Value			off	3,4
	Mask				
rtWhileIO	Value			off	3,4
	Mask				
rtSetLoop	LoopCounter			off	3,4
rtSuspend				off	3,4

rtWhileIO … rtDoWhile Statements

Executes a series of statements as long as a given condition is True. The condition is evaluated through the state of one or more IO’s. The rtWhileIO … rtDoWhile can be nested to any level. Each rtDoWhile() matches the most recent rtWhileIO. When rtWhileIO(0,0) is used the loop will run for ever (or untill the program is aborted through rtAbort)

rtSetLoop … rtDoLoop Statements

Repeats a series of statements a number of times. When the loop counter is set to 0 the loop will run for ever (or untill the program is aborted through rtAbort) The rtSetLoop … rtDoLoop statement cannot be nested.

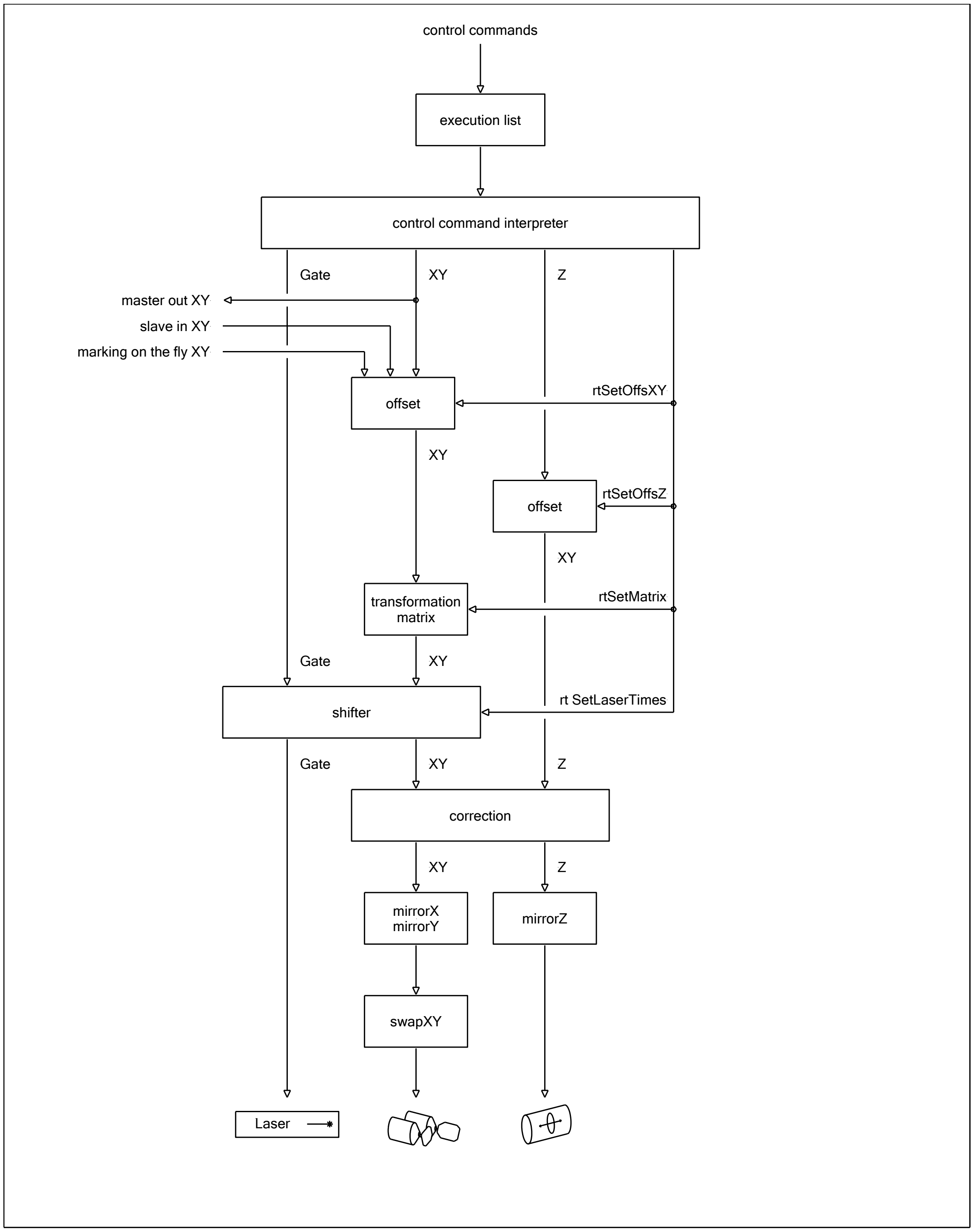
rtIfIO … [rtElseIfIO …] [rtElse …] rtEndIf Statements

Conditionally executes a group of statements depending of the value of a condition. The condition is evaluated through the state of one or more IO’s.

rtSuspend

Suspend system. System can be resumed through Ethernet commando.

2 FUNCTIONAL OVERVIEW - INTERNAL CONTROLLER



During marking, the deflection control system receives control commands through a list which are interpreted and executed by a command interpreter. In this interpreter, X-, Y- and Z- co-ordinates are generated together with a gate signal for laser control. Before being fed to the regulators, the coordinates pass through a number of functions which are capable of handling transformations such as offset, rotation and image correction. Special function blocks handle additional coordinate processing which are used when the deflection control system is marking on-the-fly or when it is operated in a master-slave mode. A shifter function block handles the required gate on and off delays of the gate signal.

3 LASER

CUA control units can interface with nearly every kind of laser. The triggering of the laser is synchronized with steering of the motors. The deflection control system uses its IO pins to interface with external machine components. IO pins 5,6 and 7 are intended to be used for controlling the laser. Linked to each of those pins is an oscillator that can be used to generate complex laser signals. Nearly any desired laser signal can be generated by combining those oscillators with the gate signal. Like all IOs, IO 5, 6 and 7 are compatible with the RS485 standard. Some additional conversion circuitry may be needed to physically connect the signals from the deflection control system to the laser.

Overview laser signals/functions
1. gate with separately setable laser on and laser off delays
2. gated burst for external laser triggering
3. separate gate for first pulse suppression
4. pulse with modulation to set laser power

When a laser is configured for internal triggering or works in CW-mode, only a gate signal needs to be supplied by the deflection control system. The laser frequency can be set through an additional serial link between the application computer and the laser. Any of the three IO's on the deflection control system can be used to generate the gate signal. Before the gate signal is available on the connector, the application needs to set the output state of the selected pin by calling "rtSetIO". Output state of all IO's are reset on power-up. The motors will move but the gate signal will not be transmitted to the laser when a marking is started without prior setting of the pins output state. Running markings without activating the main laser can be used on applications that have an additional lineout laser. They are used as simulation runs. Because the motors still move, the alignment laser shows where the actual marking would be.

When the laser is configured for external triggering the deflection control system has to generate burst signals to activate the laser. For the generation of laser signals the system comprises three programmable oscillators. The use of those oscillators allows the application to alter the laser parameters without communication overhead. To generate an external trigger signal the preferable port is IO7. This pin is linked with oscillator 3, which is synchronized with rising flanks of gate. Frequency and pulse width of the oscillator needs to be set by calling "rtSetOscillator(3,..)" before the gate signal is activated.

How to set-up a laser for external triggering ?
1. connect IO7 to the external trigger of the laser
2. configure IO7 as "OUT * Gate * OSC3" (*)
3. use "rtSetIO(64,64) in the command stream to set the output state of IO7
4. use "rtSetOscillator(3,...)" in the command stream to set laser frequency

(*) use configuration software, see manual "A2G_Cfg"

On a diode pumped solid state laser, the energy is modulated by controlling the pulse repetition rate and diode current. When the laser is configured for external triggering the pulse repetition is set by configuring oscillator 3. On most lasers the diode current can be set through an analog interface. A deflection control system does not have analog outputs but it can use its oscillators to generate pulse width modulated signals. External low pass filters can then be used to convert those to an analog voltage. In this topology the diode current on the laser can be changed as easy as its trigger frequency. Changing oscillator settings during the command stream, changes the diode current. IO5 is the preferable pin for external power control.

Laser activation and power control are handled over a single input on common CO2 lasers. There is no analog or separate trigger input. The laser is activated by applying a modulated signal. The pulse width of the applied signal controls the lasers output power. Removing the signal from the power control input turns the laser off. On some lasers wake up pulses need to be fed into their power control input when they are not activated. Gate controlled multiplexing between oscillator 1 and oscillator 2 is available on IO5. The application can configure oscillator 1 to set the laser power for marking. Oscillator 2 can be set to generate the wake up signal.

How to set-up a laser for external power control ?

1. connect IO5 through conversion electronics with analog power pin of the laser
2. configure IO5 as "OUT * OSC1" (*)
3. use “rtSetIO(16,16)” in the command stream to set the output state of IO5
4. use "rtSetOscillator(1,...)" in the command stream to set laser power

(*) use configuration software, see manual “A2G_Cfg”

4 CALIBRATION

Deflection systems suffer from positional errors resulting from different sources.

Common error sources:

- *linearity error of the motor*
- *pin cushion distortion*
- *f-theta distortions*
- *alignment problems*
- *...*

CUA control units handle these errors by adding counter errors during movement of the laser beam. Those counter errors are stored in local DSP memory for quick access. The CUA control allocates a predefined memory block which contains the calibration table. This table is automatically loaded in the DSP memory on bootup. You can setup the default calibration table through rhothor.exe. Alternatively calling “rtLoadCalibrationFile” will replace the current calibration table with the calibration data from a file stored on the harddisk.

In addition to previous mentioned error sources 3D-marking systems have to cope with other issues. If a 3D-deflection system is being used without a flat field f-theta lens the obvious error source is the shape of the focal area. This area is spherical instead of flat. With or without a flat field f-theta lens the size of the marking changes with the distance between focal area and mirror surfaces of the deflection system.

3D-error sources

- *focal area is spherical (*)*
- *optical amplification is a linear function of focal distance*

Calibration challenges are not limited to motor motions. Also laser power can vary as a function of motor positions. The deflection control system uses up to 3 oscillators to control the laser. Any of those oscillators can be set to be mapped. In that case the pulse width of that oscillator is modulated based upon the mapping data.

Laser error sources

- *mirror reflectivity changes with angle*
- *diameter laser spot changes with position*
- *absorption of laser light in f-theta lens changes with position*

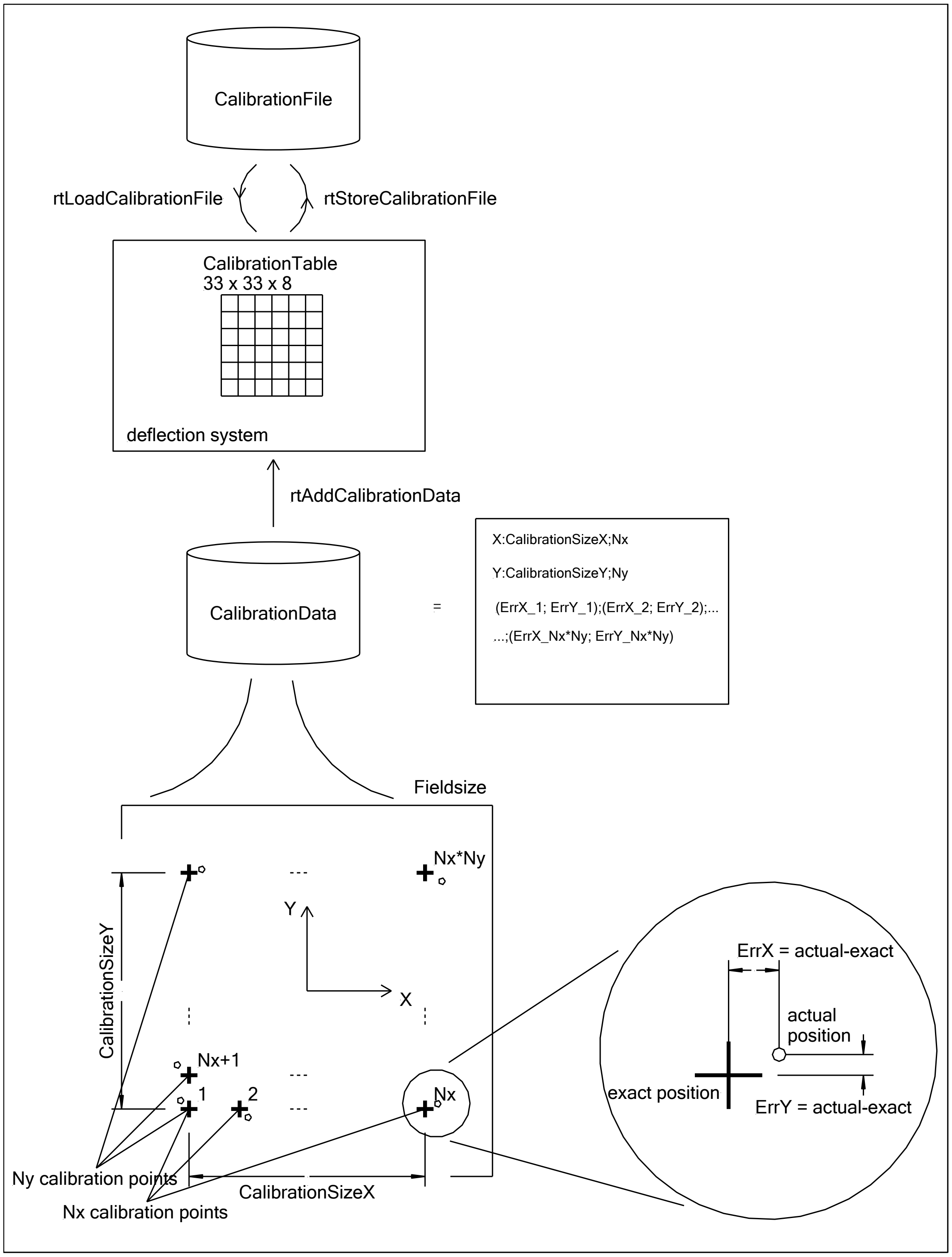
The control unit offers solutions for all those error sources. The calibration algorithm is based on look-up tables. The XY-marking area is divided by 32*32 equal sized sub fields. The look-up table holds offset values for each edge point of those fields. By grouping adjacent edges, the calibration table holds data for 33*33 edge points.

for each edge point 8 values are stored:

1. *OffsetX for Z=0*
2. *dOffsetX/dZ*
3. *OffsetY for Z=0*
4. *dOffsetY/dZ*
5. *OffsetZ for Z=0*
6. *dOffsetZ/dZ*
7. *PWMfactor for Z=0 (*)*
8. *PWMfactor/dZ*

4.1 2D-CALIBRATION

Schematical flow of a 2D-calibration



Calibrating a deflection system starts by defining the calibration size. Generally the calibration will not be done over the complete marking area, but rather a restricted portion of it. The field size is the total range the deflection system can cover whereas the calibration size is the part of the coverage that is desired to be used.

Marking a grid of crosses and measuring their position offsets is a common way to gather distortion data. The number of those crosses can be chosen freely. The measured distortion data has to be linked into a file. The current calibration can then be updated with this distortion file by calling library function “rtAddCalibrationData”. Generating a calibration is an iterative process. After the first run the measured offsets will likely to be large. Adding those offsets to the current calibration will significantly change the motor positions and movement meaning that distortion in those new positions is likely to be different. Therefore the calibration run needs to be repeated a few times. Not all the runs need to be done with the same calibration size or the same number of calibration points. It is a common practice to start the first calibration run with 3*3 calibrations points and increase the number of calibration points as the iteration progresses.

Steps for making a calibration file:

1. call library function “rtResetCalibration” or “rtLoadCalibrationFile” to initialize the calibration
2. define the calibration size (< field size)
3. define the number of calibration points 3*3, 5*5, ...
4. mark and measure the calibration points
5. link measurements in a file
6. call library function "rtAddCalibrationData" to update the calibration
7. repeat from step 3 if necessary
8. call library function "rtStoreCalibrationFile" to store the new calibration to disc.
9. call library function “rtStoreCalibration” to store the calibration as default calibration on the controller board

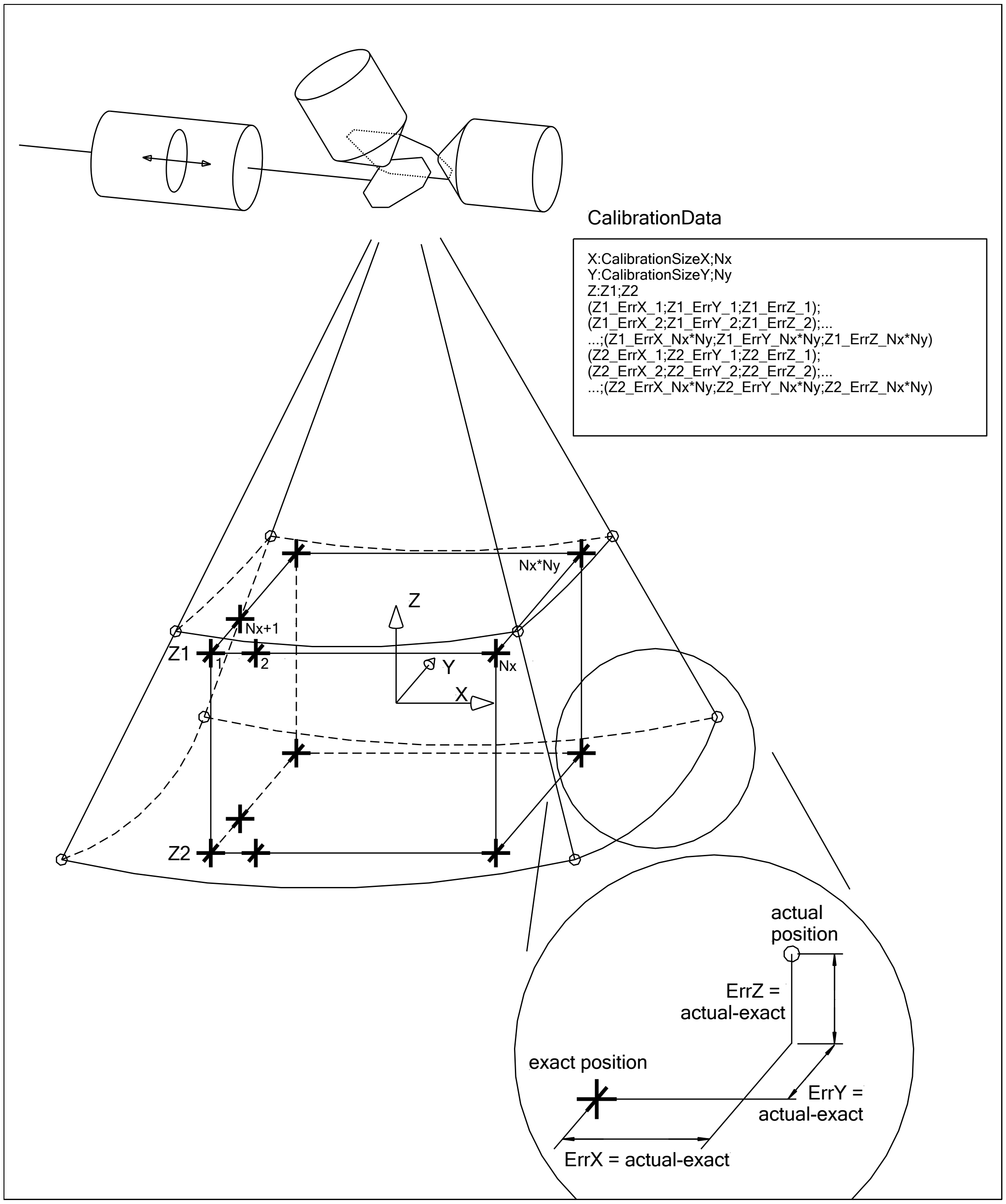
4.2 3D Calibration

The control unit supports true 3D-marking. If a deflection system fitted with a Z-motor is used as a full 3D-system, it needs to be calibrated on 2 Z positions.

Steps for making a 3D-calibration file:

1. call library function “rtResetCalibration” or “rtLoadCalibrationFile” to initialise the calibration
2. define the calibration size (< field size)
3. define the number of calibration points 3*3, 5*5, ...
4. go to the first Z position, Z1
5. mark and measure the calibration points at Z1
6. go to the second Z position, Z2
7. mark and measure the calibration points at Z2
8. link measurements in a file
9. call library function "rtAddCalibrationData" to update the calibration
10. repeat from step 3 if necessary
11. call library function "rtStoreCalibrationFile" to store the new calibration to disc.
12. call library function “rtStore Calibration” to store the calibration as default calibration on the controller board

Format 3D-CalibrationData



CalibrationData

```
X:CalibrationSizeX;Nx
Y:CalibrationSizeY;Ny
Z:Z1;Z2
(Z1_ErrX_1;Z1_ErrY_1;Z1_ErrZ_1);
(Z1_ErrX_2;Z1_ErrY_2;Z1_ErrZ_2);...
...;(Z1_ErrX_Nx*Ny;Z1_ErrY_Nx*Ny;Z1_ErrZ_Nx*Ny)
(Z2_ErrX_1;Z2_ErrY_1;Z2_ErrZ_1);
(Z2_ErrX_2;Z2_ErrY_2;Z2_ErrZ_2);...
...;(Z2_ErrX_Nx*Ny;Z2_ErrY_Nx*Ny;Z2_ErrZ_Nx*Ny)
```

5 BEACON™ CALIBRATION SYSTEM

In general generating calibration data is done by first marking a number of crosses. In a second step those crosses are measured by a camera using image processing. BEACON™ calibration system combines lasering and image processing in a single device. It allows calibration without the need of a camera and image processing software.

On 3D-deflection systems the BEACON™ calibration system has also an auto focus function. The library function "bcSamplePoint3D" returns offset values for X, Y and Z. The offset Z value is the distance between the surface of the BEACON™ calibration system and the focal plane.

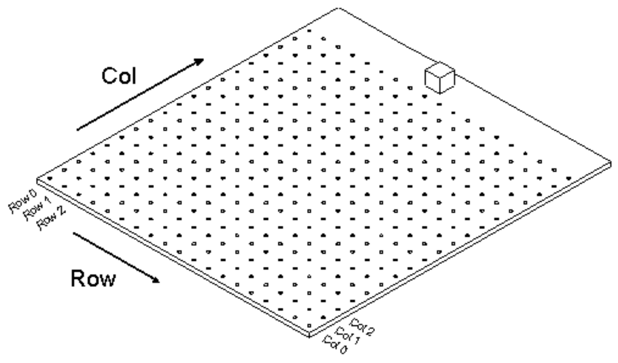
The BEACON™ calibration system has one or many sample holes depending on the type. Every sample hole can be seen as a basic camera system. Images of all those camera systems are added to form a single compound picture. All measurements are done on this picture.

A measurement cycle using a BEACON™ calibration system is started by calling library function "bcSamplePoint" or "bcSamplePoint3D". Using synchronized marking and measuring, position deviation between laser beam and sample hole is obtained. To avoid actual marking lines on the surface of the BEACON™ calibration system, the laser should be set at minimal output power. The ideal laser for a BEACON™ calibration system is a CW laser. If the machine uses a Q-switched laser, a high frequency should be selected. A high frequency will reduce pulse energy and helps to avoid actual marking on the surface of the BEACON™ calibration system.

5.1 2D CALIBRATION

Steps for 2D-calibration using a BEACON™ calibration system

1. place the BEACON™ calibration system beneath the deflection system
2. call library function “rtResetCalibration” or “rtLoadCalibrationFile” to initialize the calibration
3. run a small job to set the laser parameters, use high laser frequencies for accurate measurements (>10kHz)
4. define the calibration size (< field size and should fit on the BEACON™ measuring hole grid)
5. define the number of calibration points 3*3, 5*5, ... (should fit on the BEACON™ measuring hole grid)
6. for each calibration point define row/column of the appropriate measuring hole on the BEACON™ system (row=0 and column=0 is the top left measuring hole) and use these parameters in the function “bcSamplePoint“ to measure the point offsets
7. link all measurements in a file
8. call library function "rtAddCalibrationData" to update the calibration
9. repeat from 7 if necessary
10. call library function "rtStoreCalibrationFile" to store the new calibration to disc.



NOTE:
When measuring a calibration point, the apropriate measuring hole has to be selected. This is the measuring hole that fits the calibration point. A measuring hole is defined by a row and column number. When the USB connector lays at the right hand side then row 0 defines the top row and column 0 defines the left column.

5.2 FLAWLESS STICHING

Often an XY table is used when the size of the marking is larger than the marking area of the deflection system. The marking is split into parts and is processed sequentially by moving the workpiece using the table. A common problem lies in aligning the axis system of the deflection system with the axis system of this table. Aligning the deflection system with the XY-table can be done automatically by using a single cell on a BEACON™ calibration system.

For the BEACON™ every sample point on its surface is the same. During the calibration, the XY table is used to move one sample hole off the BEACON to the different calibration positions. All calibration points are measured using this one sample hole. Consequently the ideal positions of the calibration points are layed out using the axis system of the XY-table. After processing a complete calibration cycle the X- and Y-axis of the deflection system lies parallel to the X and Y system of the XY table. Stitching of marking fields is flawless.

The offset values returned by the functions “bcSamplePoint” or “bcSamplePoint3D” need to be logged. When all the calibration points are measured the calibration tables of the deflection system can be updated by calling “rtAddCalibrationData”. Care must be taken that only the image distortion and axis alignment errors are calibrated out of the deflection system. At the start of the calibration, the sample hole of the BEACON must be placed exactly beneath the zero point of the deflection system. Otherwise the axis system of the deflection system will also be shifted by the calibration. Measuring the position accuracy of the BEACON prior to the calibration can be done by calling “bcSamplePoint(0,0,Row,Col,Sweep,X,Y) ”. The return values can be used to offset the table before starting the actual calibration measurements. Row and Col should be set on the center value.

Steps for 2D-calibration using a BEACON™ calibration system and an XY table

1. put the BEACON™ calibration system on the XY table (*)
2. call library function “rtResetCalibration” or “rtLoadCalibrationFile” to initialise the calibration
3. run a small job to set the laser parameters
4. use the XY table to position the BEACON™'s centre sample point beneath the zero position of the deflection system
5. measure the offset using library function bcSamplePoint(0,0,Row,Col,Sweep,&OffsX,&OffsY)
6. if the returned offset is too high, use it as a table offset and repeat from step 4
7. define the calibration size (< field size)
8. define the number of calibration points 3*3, 5*5, ...
9. use the XY table to position BEACON™'s centre sample point to the first point (X1,Y1) that must be calibrated
10. measure this point offset using library function bcSamplePoint(X1,Y1,Row,Col,Sweep,&OffsX,&OffsY)
11. log the measurement
12. use the XY table to position BEACON™'s centre sample point to the next point (Xi,Yi) that must be calibrated
13. measure this point offset using library function bcSamplePoint(Xi,Yi,Row,Col,Sweep,&OffsX,&OffsY)
14. log the measurement
15. repeat from step 12 for all the points that need to be calibrated
16. link measurements in a file
17. call library function "rtAddCalibrationData" to update the calibration
18. repeat from step 8 if necessary
19. call library function "rtStoreCalibrationFile" to store the new calibration to disc.

(*) BEACON™'s top surface needs to be in the FOCAL plane of the deflection system.

5.3 3D Calibration

The calibration table concept supports full 3D-calibration. To achieve this the planar calibration has to be executed on 2 Z levels.

Steps for 3D-calibration using a BEACON™ calibration system and a Z axis

11. place the BEACON™ calibration system beneath the deflection system
12. call library function “rtResetCalibration” or “rtLoadCalibrationFile” to initialise the calibration
13. run a small job to set the laser parameters
14. define the calibration size (< field size)
15. define the number of calibration points 3*3, 5*5, ...
16. go to the first Z position, Z1
17. for each calibration point call bcSamplePoint3D(Xi,Yi,Z1,Row,Col,Sweep,&OffsX,&OffsY,&OffsZ) to measure its offsets
18. go to the second Z position, Z2
19. for each calibration point call bcSamplePoint3D(Xi,Yi,Z2,Row,Col,Sweep,&OffsX,&OffsY,&OffsZ) to measure its offsets
20. link all measurements in a file
21. call library function "rtAddCalibrationData" to update the calibration
22. repeat from step 5 if necessary
23. call library function "rtStoreCalibrationFile" to store the new calibration to disc.

6 MARKING-ON-THE-FLY

Marking-on-the-Fly is a common used way to increase machine performance. Moving parts that have to be marked do not need to be mechanically stopped before processing. Being able to perform Marking-on-the-Fly does not only reduce idle times between markings but also reduces overall machine complexity. The control unit provides several Marking-on-the-Fly solutions. All those solutions are supported for both X- and Y-axis.

Marking-on-the-Fly solutions
1. constant speed
2. resolver AB
3. absolute offset

Marking on the fly must be enabled using the configuration software, see manual “A2G_Cfg”.

6.1 CONSTANT SPEED

In this mode the speed of the part that is being marked is assumed to be constant during the complete marking. Before the marking can be started, this speed has to be communicated to the deflection control system.

6.2 RESOLVER AB

This mode allows complete freedom of the part's movement during his marking. The position of the part is monitored using a 2 phase resolver. This resolver is connected with the deflection control system over 2 designated IO's. One IO is used for the resolvers A phase the other one for the resolvers B phase. Every flank received from these IO's corresponds with a movement equal to a step size. The phase relation between the resolvers A and B signals determines the direction of the movement. The step size can be set using the library function "rtSetResolver".

6.3 ABSOLUTE OFFSET (XY2-100)

This mode is the XY2-100 compatible mode. The deflection control system receives X and Y offset values from an XY2-100 compatible output device. See data sheet XY2-100

7 MASTER-SLAVE

Master-Slave marking is a configuration wherein several deflection systems are doing the same marking. The master is processing the vector stream and controls the laser. The X and Y co-ordinates of the generated set points are broadcasted by the master control unit towards all connected slaves. Rotation and scaling of those co-ordinates can be set independently for each member of the master-slave network. Each member has its own local parameters. Those parameters have to be set prior to the marking.

Operations done by the Master operation during marking

1. fetch and process command
2. generate XYZ co-ordinates
3. broadcast XY co-ordinates (*)
4. rotate and scale (local transformation matrix)
5. add offset (local offset vector)
6. add calibration data (local calibration tables)
7. steer motors and laser

(*) only X and Y co-ordinate is broadcasted by the master

Operations done by Slave deflection system during marking

1. receive XY co-ordinates from master
2. rotate and scale (local transformation matrix)
3. add offset (local offset vector)
4. add calibration data (local calibration tables)
5. steer motors

During the marking the XY co-ordinates are broadcasted by the master towards all the slaves through the Master-Slave network. One master can have an unlimited number of slaves. The connection between those control units is done by 2 twisted pair cables. They start at the master then they pass slave type 1 systems to end at a slave type 2 system. This wiring is needed to have a correctly terminated network, see drawing.

Master-Slave network

master:	*IO1/2 terminated*
slave type 1:	*IO5/6 no termination*
slave type 2:	*IO1/2 terminated*

Master-Slave operation can also be used in 3D-systems. Because the master does not broadcast the Z co-ordinates, marking instructions like "rtLineTo3D" have to be avoided. However, each Master-Slave network element can have a local Z offset and has full operational use of its calibration tables.

A Master-Slave operation is commonly used to mark a raster of similar components. The number of components in such a raster is likely to change. An obvious reason could be that the raster is not always complete. The raster can comprise missing or defect parts that don't have to be marked. Laser light however is routed towards all the deflection systems by the use of beam splitters. Mounting shutters before each deflection system or providing a beam dumps for each deflection system provides a mean to switch on and off markings on the raster.

8 LOCAL FLASH MEMORY

CUA control units are fitted with flash memory. The systems can therefore be operated without computer. Instead of data streaming, the image data is stored on the local flash. The deflection control system uses a dedicated file management system to handle the contents on the flash. The data stored in this memory is not accessible over the operation system. Accessing the data is only possible through the library. The library comprises functions that allow an application to store and retrieve data from the flash memory. Files on the flash are created using the functions rtFileOpen and rtFileClose. When a file is opened all function calls to the library are not executed but logged on the flash.

control function macros

Control functions can be stored to flash as control function macros. When, during processing of control functions, a function call “rtFileFetch (mymacro)” is encountered, the deflection control system will replace this with the contents from the file “mymacro”. Recursive calls of “rtFileFetch” are not supported. This means that the control functions stored in macros may not contain a function call “rtFileFetch”. The control functions “rtListOpen”, “rtListClose”, “rtSetLoop” and “rtDoLoop” are also not allowed in control function macros. Beside said functions all control functions listed in chapter 1 are allowed to be used in the macros.

Library function calls to create a control function macro on flash comprising a single line. The line can afterwards be marked by calling the control function rtFileFetch(mymacro).

```
rtFileOpen(“mymacro”);
rtSetSpeed(1000);
rtJumpTo(0,0);
rtLineTo(10,0);
rtFileClose();
```

autoexec

When the system is operated without central application computer, it must be able to start list processing automatically. Control functions can be stored to flash under the bootsector. When the deflection control system is powered on, it scans the flash if this bootsector is filled in with a bootscript. When the file is found, its execution list is started automatically. The function call sequence in this file must start by calling “rtListOpen(3)” and end with a call to “rtListClose()”.

Library function calls to create the file “autoexec” on flash

```
rtListOpen(3);
rt…
rtListClose();
```

Local flash is commonly used when the deflection system is operated in an environment without dedicated application computer. However control code stored in local flash can also be combined with standard command streaming. The application can use the local flash to store recurrent marking data prior to the actual marking. During the marking the command stream can be just the function call "rIndexFetch". This minimises the bandwidth between application computer and controller during marking.

9 Appendix A: Rhothor.DLL Library Functions

9.1.1 bcSamplePoint

long bcSamplePoint(double X, double Y, long Row, long Col, double Sweep, double* OffsetX, double* OffsetY)

This method measures the position deviation of a calibration point at co-ordinate (X,Y) using a Beacon device. Row and Col determine the position of the measuring hole on the Beacon device, used for calibration. Row=0, Col=0 is the top left measuring hole. Sweep length determines the line length used during measuring process. This value should be larger then 3mm. Too large values can result in invalid measurements.

Parameters

X	X co-ordinate exact position calibration point
Y	Y co-ordinate exact position calibration point
Row	row of the measuring cell on the Beacon device
Col	column of the measuring cell on the Beacon device
Sweep	sweeping length used during the measuring process on the Beacon device.
OffsetX	measured X error = measured actual X position - exact X position calibration point
OffsetY	measured Y error = measured actual Y position - exact Y position calibration point

Deviating return codes

ERR_HARDWARE, 5	no Beacon device available
ERR_LASER, 42	laser power is set too low

9.1.2 rtAbort

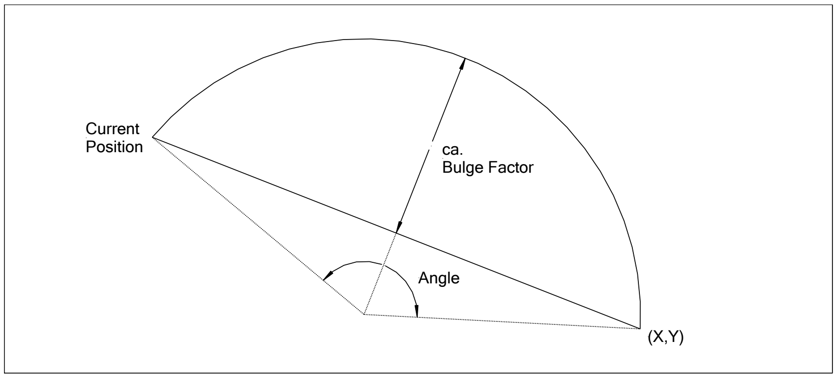
long rtAbort()

This function aborts a running command list.

9.1.3 rtArcTo

long rtArcTo(double X, double Y, double BF)

This function marks an arc, starting from the current position to co-ordinate (X, Y) using the BF as bulge factor. The bulge is the tangent of 1/4 the included angle for an arc segment, made negative if the arc goes clockwise from the start point to the end point.



Parameters

X	target X co-ordinate in mm
Y	target Y co-ordinate in mm
BF	bulge factor, 0 indicates a straight segment, 1 is a semicircle

9.1.4 rtArcMoveTo

long rtArcMoveTo(double X, double Y, double BF)

Same as rtArcTo, but without enabling the gate signal for the laser.

9.1.5 rtAddCalibrationData

long rtAddCalibrationData(const char* FileName)

When a system is calibrated a number of calibration points have to be measured. The results of those measurements have to be grouped in calibration data as a list of error values. With this method the gathered calibration data can be added to the current calibration.

Parameters

FileName	0 terminated string containing the full file name and path

9.1.6 rtAddCalibrationDataZ

long rtAddCalibrationDataZ(const char* FileName)

Same as rtAddCalibrationData but for Z-axis calibration.

9.1.7 rtBurst

long rtBurst(long Time)

This command adds a burst command to the list. During execution of a burst the gate signal is activated while the motors are standing still.

Parameters

Time	duration of burst in µsec

9.1.8 rtCircle

long rtCricle(double X, double Y, Angle)

Marks a circular path of length Angle from the current position around a center point defined by X and Y.

Parameters

X	center X coordinate of the circle
Y	center Y coordinate of the circle
Angle	The angle travelled around the center point

9.1.9 rtCircleMove

long rtCircleMove(double X, double Y, double Angle)

Same as rtCircle, but without enabling the gate signal for the laser.

9.1.10 rtCharDef

long rtCharDef(long Ascii)

This function is the opening statement of a character definition. All marking commands following this statement define the shape of the character to be defined. The following example illustrates the definition of the letter "A":

```
rtFontDef("custom");
rtCharDef(65);
rtJumpTo(0,0);
rtLineTo(2.5,5);
rtLineTo(5,0);
rtSetImageOffsRelXY(0,5);
rtCharDef(66)
...
rtFontDefEnd();
```

When defining characters it is customary to start at position (0,0) and use relative image offsets to indicate the starting position of the next character. In the above example, "rtSetImageOffsRelXY" defines the point at which the following character in a string would start. See "rtSetVarBlock" and "rtVarBlockFetch" on how to mark strings.

Parameters

Ascii	Ascii code of the character to be defined.

9.1.11 rtDoLoop

long rtDoLoop()

This is the closing statement for an rtSetLoop instruction and starts the execution of the loop when called.

9.1.12 rtDoWhile

long rtDoWhile()

This is the closing statement for an rtWhileIO instruction.

9.1.13 rtEraseFromFlash

long rtEraseFromFlash(char* FileName)

This command deletes an existing file from the flash.

Parameters

FileName	The name of the file to be erased from the flash.

9.1.14 rtElse

long rtElse()

Else-statement for the "rtIfIO" function.

9.1.15 rtElseIfIO

long rtElseIfIO(long Value, long Mask)

Introduces a IO state condition to be tested if the previous IO state conditional test has failed

Parameters

Mask	defines bitwise which Pins are to be checked.
Value	defines bitwise the condition of the input pins defined by Mask.

9.1.16 rtEndIf

long rtEndIf()

Statement for terminating the conditional IO commands rtIfIO and rtElse.

9.1.17 rtFileClose

long rtFileClose()

This command is the closing statement to the "rtFileOpen" function and stores the macro file to the flash memory when called.

9.1.18 rtFileCloseAtHost

long rtFileCloseAtHost()

This is an alternative closing statement to the function "rtFileOpen" and stores the macro file locally instead of on the control system's flash memory.

9.1.19 rtFileCloseAtIndex

long rtFileCloseAtIndex(long Index)

Closing statement for the method "rtFileOpen" with the option of storing the macro file at a specific index location.

Parameters

Index	index of target location to store the file

9.1.20 rtFileDownload

long rtFileDownload(char* FileName, char* DestFile)

This command allows a user to retrieve a macro stored on the internal flash memory of the control system and store in on a computer.

Parameters

FileName	name of the macro on the flash memory
DestFile	path of the destination file

9.1.21 rtFileFetch

long rtFileFetch(char* FileName)

This command executes all the commands from the macro stored in a file on the flash memory. When a function call “rtFileFetch (mymacro)” is encountered, the deflection system will replace this with the contents of the flash file.

Parameters

FileName	the name of the file to be executed from the flash.

9.1.22 rtFileOpen

long rtFileOpen(char* FileName)

This command creates a new macro file on the flash. All list commands called after a rtFileOpen will be stored in a file on the flash memory instead of being executed by the deflection system. Call function rtFileClose to close the macro file. Subsequently the file can be loaded and executed from the flash by calling the function rtFileFetch. FileName size is limited to 255 characters.

Parameters

FileName	the name of the file to be created on the flash.

9.1.23 rtFileUpload

long rtFileUpload(char* SrcFile, char* FileName)

Upload a file directly to the flash memory.

Parameters

SrcFile	path of the file
FileName	name of the file on the control system

9.1.24 rtFileUploadAtIndex

long rtFileUploadAtIndex(long Index)

The methods stores a file on the control systems internal flash memory at a specified index location.

Parameters

Index	index of target location

9.1.25 rtFontDef

long rtFontDef(char* Name)

In order to facilitate the marking of recurrent characters or symbols (such as, for example, text) one can make use of a font definition statement. Such a statement contains a series of character definitions which are assigned to ASCII codes - see "rtCharDef".

Parameters

Name	name of the font to be created

9.1.26 rtFontDefEnd

long rtFontDefEnd()

Ending statement for "rtFontDef".

9.1.27 rtFormatFlash

long rtFormatFlash()

This command start the process of preparing the flash, including setting up an empty file system.

9.1.28 rtGetCanLink

long rtGetCanLink(long Address, long* Value)

This method gets a value from the CAN interface. The address parameter determines which of the two internal CAN controllers is to be used. Each controller contains 8 bytes, addressed from 0-7 and 7-15, respectively.

Parameters

Address	address of the internal CAN controller
Value	pointer to the location where the value is to be stored.

9.1.29 rtGetCounter

long rtGetCounter(long* Value)

This method is closely related to "rtResetCounter" and returns the current counter value of the list.

9.1.30 rtGetFieldSize

long rtGetFieldSize(double* Size)

Parameter

Size	fieldsize as set in the configuration software, see manual “A2G_Cfg”, in mm.

9.1.31 rtGetFileIndex

long rtGetFileIndex(char* FileName)

Parameters

FileName	name of the file for which the index is to be returned

9.1.32 rtGetFlashFirstFileEntry

long rtGetFlashFirstFileEntry(char* FileName, long* AllocatedSize)

This command starts a file search in the file system on the flash and returns the file information of the first file. If the file system is empty FileName will contain an empty string. A file name should never be larger then 255 characters.

Parameters

FileName	pointer to the buffer that will receive the filename
AllocatedSize	allocated byte size of the file in the flash memory

9.1.33 rtGetFlashNextFileEntry

long rtGetFlashNextFileEntry(char* FileName, long* AllocatedSize)

This command returns the file information of the next found file in the file system on the flash. If no next file is found FileName will contain an empty string. Use rtGetFlashFirstFileEntry before using this function. Use this function to cycle through the complete file system. If no more files are available FileName contains an empty string.

Parameters

FileName	pointer to the buffer that will receive the filename
AllocatedSize	allocated byte size of the file in the flash memory

9.1.34 rtGetFlashMemorySizes

long rtGetFlashMemorySizes (long* Total, long* Allocated)

This method retrieves the free memory size on the flash.

Parameter

Total	Total available flash memory, expressed in bytes
Allocated	Allocated flash memory, expressed in bytes

9.1.35 rtGetIO

long rtGetIO(long* Value)

Query the current status of all the system input and output signals on the X7 digital I/O connector, as one number. Inputs and outputs signals are either active (1) or inactive (0). The system signals contribute to the Value parameter as follows:

Bit location	Signal	Contribution to IO value if active
Bit 6	IO7	64 (0x20)
Bit 5	IO6	32 (0x10)
Bit 4	IO5	16 (0x0f)
Bit 3	IO4	8 (0x08)
Bit 2	IO3	4 (0x04)
Bit 1	IO2	2 (0x02)
Bit 0	IO1	1 (0x01)

IO Value number is the sum of the contribution of all active signals:
If Value=2, IO2 signal is active, and all other signal are inactive.
If Value=83, signals IO7(64), IO5(16), IO2(2) and IO1(1) are active (64+16+2+1=83), and all other signals are inactive.

Parameter

Value	7 least significant bits holds the data of the 7 IO pins

9.1.36 rtGetLaserLink

long rtGetLaserLink(long Address, long* Value)

Allows to retrieve status information from an external laser using the LaserLink interface. Refer to rtSetLaserLink and the LaserLink specifications manual for more information.

Parameters

Address	a command identifier for the LaserLink interface
Value	pointer to a variable storing the return value

9.1.37 rtGetMaxSpeed

long rtGetMaxSpeed(double* Speed)

Parameters

Speed	maximum speed as set in the configuration software, see manual “A2G_Cfg” in mm/sec

9.1.38 rtGetResolvers

long rtGetResolvers(double* X, double* Y)

This method returns the offsets integrated by the resolvers connected with the deflection control system.

9.1.39 rtGetScannerDelay

long rtGetScannerDelay(long* Delay)

Parameters

Delay	scanner delay as set in the configuration software, see manual “A2G_Cfg”, in µsec

9.1.40 rtGetSerial

long rtGetSerial(long* Serial)

Returns the serial number of the laser deflection system.

Parameters

Serial	serial number of the deflection control system.

9.1.41 rtGetSetpointFilter

long rtGetSetpointFilter(long* TimeConst)

Parameters

TimeConst	time constant of setpoint low pass filter in µsec

9.1.42 rtGetStatus

long rtGetStatus(long* Memory)

This method retrieves the current status of the rhothor™ deflection control system. Use this method to retrieve the resources used by the system. This control function has deviating return codes.

Parameters

Memory	total size of the execution lists expressed in bytes

Deviating return codes

ERR_OK, -1	system is currently idle
ERR_BUSY, 2	system is currently processing a list

9.1.43 rtGetVersion

long rtGetVersion(char* Version)

With this method the version of the DLL can be retrieved. The application needs to set pointer “Version” to allocated memory prior to calling the function.

Parameters

Version	version string, terminated with a 0 byte

9.1.44 rtIfIO

long rtIfIO(long Value, long Mask)

Conditional statement for checking the state of IO pins. Useful when interfacing and synchronising with external devices. See functions "rtElse" and "rtEndIf".

Parameters

Mask	defines bitwise which Pins are to be checked.
Value	defines bitwise the condition of the input pins defined by Mask.

9.1.45 rtIndexFetch

long rtIndexFetch(long Index)

Loads a job file by index as opposed to loading it by name. See "rtFileFetch".

Parameters

Index	index of the file location

9.1.46 rtJumpTo

rtJumpTo(double X, double Y)

This command generates a jump to co-ordinate (X, Y).

Parameters

X	target X co-ordinate in mm
Y	target Y co-ordinate in mm

9.1.47 rtJumpTo3D

rtJumpTo3D(double X, double Y, double Z)

This command generates a jump to co-ordinate (X, Y, Z). This function should only be used with 3D deflection systems.

Parameters

X	target X co-ordinate in mm
Y	target Y co-ordinate in mm
Z	target Z co-ordinate in mm

9.1.48 rtLineTo

rtLineTo(double X, double Y)

This command marks a line from the current position to co-ordinate (X, Y).

Parameters

X	target X co-ordinate in mm
Y	target Y co-ordinate in mm

9.1.49 rtLineTo3D

rtLineTo3D(double X, double Y, double Z)

This command marks a line from the current position to co-ordinate (X, Y, Z). This function should only be used with 3D deflection systems.

Parameters

X	target X co-ordinate in mm
Y	target Y co-ordinate in mm

9.1.50 rtListClose

long rtListClose()

This method closes an open command list. Its behavior depends on the selected list execution mode.

Mode LIST_START
Calling “rtListClose()” will start the execution of the command list. If there is another list still running when this functions is called, the list data is moved into the running list. This list concatenation allows batch processing by the application without idle times at list swaps.

Mode AUTO_START
In this mode the list is used as a fifo between application and deflection control system. In principle, every command added to the list is all set for execution. However commands are grouped into packages to optimise communication speed. Each call to a control function is added to a package. When a package is full it is added to the execution list. Calling “rtListClose()” will force the commands stored in the last package into the execution list.

9.1.51 rtListOpen

long rtListOpen(long Mode)

This method opens a command list. It is to be called before using control commands. Parameter “Mode” defines the list execution mode.

Parameters

Mode=1	Sets execution mode to LIST_START. The deflection system will execute all commands appended to the command list after “rtListClose()” is called.
Mode=2	Sets execution mode to AUTO_START. The deflection system will execute the commands from the command list automatically. The user only appends execution commands to the list.
Mode=3	Set execution mode to BOOT_START. The command list will be written as bootscript after “rtListClose()” is called.
Mode=4	Set execution mode to LOAD_START. The command list will be completely loaded into the volatile memory of the controller board.

9.1.52 rtLoadCalibration

long rtLoadCalibration()

Stores the current calibration as the default in the control system flash memory.

9.1.53 rtLoadCalibrationFile

long rtLoadCalibrationFile(const char* FileName)

This method loads a calibration file into the rhothor™ deflection control system. The current calibration is overwritten.

Parameters

FileName	0 terminated string containing the full file name and path

9.1.54 rtLoadCalibrationFileZ

rtLoadCalibrationFileZ(const char* FileName)

Same as rtLoadCalibrationFile but for Z-axis calibration.

9.1.55 rtMoveTo

long rtMoveTo(double X, double Y)

This command generates a linear movement starting from the current position to co-ordinate (X, Y) using the marking speed. The movement is done with the laser switched off.

Parameters

X	target X co-ordinate in mm
Y	target Y co-ordinate in mm

9.1.56 rtMoveTo3D

long rtMoveTo3D(double X, double Y, double Z)

This command generates a linear movement starting from the current position to co-ordinate (X, Y, Z) using the marking speed. The movement is done with the laser switched off. This function should only be used with 3D deflection systems.

Parameters

X	target X co-ordinate in mm
Y	target Y co-ordinate in mm
Z	target Z co-ordinate in mm

9.1.57 rtParse

long rtParse(char* Cmd)

This method allows to send marking commands to the Rhothor deflection system in text format and can be used, for example, for command interpreting.

Parameters

Cmd	command string

9.1.58 rtPulse

long rtPulse(double X, double Y)

This command starts a linear movement from the current position to co-ordinate (X, Y) using the marking speed. Afterwards the laser is switched on for 5 µsec. If the command is followed by a marking command the laser is not switched off on termination of the “rtPulse” command.

Parameters

X	target X co-ordinate in mm
Y	target Y co-ordinate in mm

9.1.59 rtPulse3D

long rtPulse3D(double X, double Y, double Z)

This command starts a linear movement from the current position to co-ordinate (X, Y, Z) using the marking speed. Afterwards the laser is switched on for 5 µsec. If the command is followed by a marking command the laser is not switched off on termination of the “rtPulse3D” command. This function should only be used with 3D deflection systems.

Parameters

X	target X co-ordinate in mm
Y	target Y co-ordinate in mm
Z	target Z co-ordinate in mm

9.1.60 rtResetCalibration

long rtResetCalibration

Calling this method resets the calibration tables in the deflection control system. Table entries for image distortion are loaded with 0. Table entries for power mapping are pre-set to 100%.

9.1.61 rtResetCalibrationZ

long rtResetCalibrationZ()

The same as rtResetCalibration but for Z-axis calibration data.

9.1.62 rtResetCounter

long rtResetCounter()

Resets the list counter to zero. The list counter increments for every subsequent command issued and can be polled in applications where status information about a job is required. See rtGetCounter for more information.

9.1.63 rtResetResolver

long rtResetResolver(long Nr)

Resets a resolver back to zero.

Parameters

Nr	Resolver identifier

9.1.64 rtSelectDevice

long rtSelectDevice(char* IP)

Call this method to address an ethernet control device through its IP. If the function is called with “USB” as string value, a connected USB control device is searched. This method should be called before using any other DLL method in your application. Call this method at the start of the application or when the control system needs a reset.

Parameters

IP	poointer to string containing IP nr or “USB”

9.1.65 rtScanCanLink

long rtScanCanLink(long Address, long Node, long Index, long SubIndex)

Parameters

Address	internal CAN controller address
Node	node identifier
Index	CANOpen object dictionary index
SubIndex	CANOpen object dictionary sub-index

9.1.66 rtSendUartLink

long rtSendUartLink(char* Data)

This method send data over the UART link.

Parameters

Data	pointer to the string to be transmitted followed by a NULL character

9.1.67 rtSetCanLink

long rtSetCanLink(long Node, long Index, long SubIndex, char* Data)

Send data over the CAN link interface. See the CANOpen specifications for more information.

Parameters

Node	receiving node
Index	object dictionary index
SubIndex	subindex of the object dictionary
Data	pointer to the string to be transmitted followed by a NULL character

9.1.68 rtSetHover

long rtSetHover(long Time)

The method is used in combination with "rtSetLead" and specifies the amount of time that the deflection system hovers above the lead position before continuing.

Parameters

Time	time to hover in µs

9.1.69 rtSetImageMatrix

long rtSetImageMatrix(double a11, double a12, double a21, double a22)

Set matrix coefficients for local coordinate transformations.
The resulting image transformation matrix is:

$$\begin{pmatrix} a11 & a12 & 0 \\ a21 & a21 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Parameters

a11, a12, a21, a22

matrix coefficients

9.1.70 rtSetImageMatrix3D

long rtSetImageMatrix3D(double a11, double a12, double a21, double a22, double a31, double a32)

Set matrix coefficients for local coordinate transformations.
The resulting image transformation matrix is:

$$\begin{pmatrix} a11 & a12 & 0 \\ a21 & a21 & 0 \\ a31 & a32 & 1 \end{pmatrix}$$

Parameters

a11, a12, a21, a22, a31, a 33

matrix coefficients

9.1.71 rtSetImageOffsXY

long rtSetImageOffsXY(double X, double Y)

Sets an absolute image offset.

Parameters

X	X-axis offset
Y	Y-axis offset

9.1.72 rtSetImageOffsRelXY

long rtSetImageOffsXY(double X, double Y)

Sets an image offset with respect to the current position.

Parameters

X	X-axis offset
Y	Y-axis offset

9.1.73 rtSetImageOffsZ

long rtSetImageOffsZ(double Z)

Sets an absolute image offset in the Z axis.

Parameters

Z	Z-axis offset

9.1.74 rtSetIO

long rtSetIO(long Value, long Mask)

This command sets the output signal levels on the X7 digital I/O connector:
bit 0 of “Value” is copied to the output bit of IO1 only when bit 0 of “Mask” is true,
bit 1 of “Value” is copied to the output bit of IO2 only when bit 1 of “Mask” is true,….

The output value of an IO-pin is visible when the pin is configured as an output. This command has no effect on IO-pins configured as an input or used by special function blocks. The output values of IO-pins 5,6 and 7 needs to be set before their laser signals are available.

Parameters

Value	the new output values for the IO pins
Mask	defines bitwise which IO pins are to be changed

9.1.75 rtSetJumpSpeed

long rtSetJumpSpeed(double Speed)

Call this method to set the jumping speed. The speed value remains in force until a next call of rtSetJumpSpeed. Functions "JumpTo" and "JumpTo3D" use this value.

Parameters

Speed	jump speed in mm/sec, Speed must be set lower than or equal to maximum speed

9.1.76 rtSetLaser

long rtSetLaser(bool OnOff)

During command processing the laser is controlled together with the deflection motors. With this command the laser can be switched on continuously. When the laser is switched on using this command, it remains activated until explicitly switched off by calling “rtSetLaser(false)”.

Parameters

OnOff=true	turn the laser continuously on.
OnOff=false	switch to command-driven laser.

9.1.77 rtSetLaserLink

long rtSetLaserLink(long Address, long Value)

This method applies to external laser control using the LaserLink interface and allows the user to set the laser parameters. More more information regarding the availble commands refer to the LaserLink datasheet.

Parameters

Address	a command identifier for the LaserLink interface
Value	value to be placed on the LaserLink output

9.1.78 rtSetLaserTimes

long rtSetLaserTimes(long GateOnDelay, long GateOffDelay)

During marking, the command interpreter of the deflection control system generates co-ordinates together with a gate signal in a synchronised fashion. A shifter allows the gate signal to be shifted in time related to the co-ordinate stream. The method sets the time values for this shifter. Positive values for will postpone gate output related to co-ordinate output, while negative values will shift the gate signal before the co-ordinates. Both values can be positive or negative and can be set independent of each other.

Parameters

GateOnDelay	sets the time shift for rising edges of gate
GateOffDelay	sets the time shift of the falling edges of gate

9.1.79 rtSetLead

long rtSetLead(long Time)

Certain applications require a constant marking speed to prevent unwanted or excessive laser-material interaction. This function allows the user to specify a lead time to allow the deflection system to accellerate and decellerate prior to marking to ensure a constant laser exposure.

Parameters

Time	lead time in µs

9.1.80 rtSetLoop

long rtSetLoop(long Repetitions)

Opening statement for a loop. Any commands following the rtSetLoop and preceeding the rtDoLoop commands are repeated. The number of repetitions are set by the Repetitions variable.

Parameters

Repetitions	Number of times to repeat the loop; 0 means endless loop

9.1.81 rtSetMatrix

long rtSetMatrix(double a11, double a12, double a21, double a22)

This command sets the transformation matrix on the deflection system. An application can rotate and stretch images in the XY- plane using the transformation matrix. The setting remains valid until the next call of “rtSetMatrix”.

Transformation on co-ordinates
Xout = a11*Xin + a12*Yin
Yout = a21*Xin + a22*Yin
Zout = Zin

Example: rotation
a11=cosine(rotation angle)
a12=-sine(rotation angle)
a21=sine(rotation angle)
a22=cosine(rotation angle)

9.1.82 rtSetMaxSpeed

long rtSetMaxSpeed(double Speed)

Sets the maximum allowed speed of the deflection system.

Parameters

Speed	maximum speed of the deflection system in mm/sec.

9.1.83 rtSetMinGatePeriod

long rtSetMinGatePeriod(long Time)

This method allows to the user to specify the minimum allowable gate period for laser triggering. Usefull in pulsed laser operation where the laser must regenerate before re-triggering.

Parameters

Time	Minimum gate period in µs

9.1.84 rtSetOTF

long rtSetOTF(long Number, bool On)

Method used to disable the resolver dependent-offset used during on-the-fly marking. This is useful when the laser is off and it is temporarily not required to track the movement of the resolver. Once enabled again the deflection systems jumps to the current resolver position.

Parameters

Number	identifier of the resolver (1 or 2)
On	TRUE/FALSE - enable or disable OTF

9.1.85 rtSetOffsIndex

long rtSetOffsIndex(long Index)

The CUA control unit has a register capable of storing up to 8 different offset presets which can be written using ethernet commands. Using this function in a marking job selects the desired index to be used for the marking of any consecutive commands.

Parameters

Index	offset index (0-7)

9.1.86 rtSetOffsXY

long rtSetOffsXY(double X, double Y)

This method sets the XY-offset vector on the deflection control system. The offset remains valid until the next call of “rtSetOffsXY”.

Transformation on co-ordinates

Xout = Xin + X

Yout = Yin + Y

Zout = Zin

9.1.87 rtSetOffsZ

long rtSetOffsZ(double Z)

This method sets the Z-offset for the co-ordinates. The offset remains valid until the next call of “rtSetOffsZ”.

Transformation on co-ordinates
Xout = Xin
Yout = Yin
Zout = Zin + Z

9.1.88 rtSetOscillator

long rtSetOscillator(long Nr, double Period, double PulseWidth)

This method sets the parameter of one of the 3 internal oscillator signals.

Parameters

Nr	identifies the oscillator. Can be 1, 2 or 3.
Period	µs; range [0.030, 1000000]
PulseWidth	µs; range [0.015, Period-0.015]

9.1.89 rtSetPulseBulge

long rtSetPulseBulge(double Factor)

This method is useful in applications where a pulsed laser is used for rapid grid marking. Due to the finite track delay of the deflection head, marking errors are likely to occur particularly in regions where rapid changes in position are required (corners, zig-zag patterns, etc.). Adjusting the pulse bulge allows the user to pre-compensate for such tracking errors.

Parameters

Factor	bulge factor of the desired bulge

9.1.90 rtSetResolver

long rtSetResolver(long Nr, double StepSize, double Range)

A single phase or dual phase resolver can be used for measuring position movements when a deflection system is marking the workpiece on the fly. One signal pair for each phase is used to connect the resolver. Any signal change messages a position change to the deflection control system. An integrator adds or subtracts “StepSize” with every change. The range of the integration lies within -“Range”/2 and “Range”/2. The integrator overflows to -“Range”/2 and underflows to “Range”/2. Its value is used as an offset value during the marking. Calling this command does not only declares “StepSize” and “Range” but also clears the integrator value. Two resolvers can be connected to the deflection system.
The method can also be called with “Range” set to zero. In this case the integration range is unlimited, from -32768*FieldSize to 32767*FieldSize.

Parameters

Nr=1	to set the resolver that offsets the X-axis of the deflection control system
Nr=2	to set the resolver that offsets the Y-axis of the deflection control system
StepSize	incremental size for the integration, negative values to change direction, in mm
Range	range of the integration in mm.

9.1.91 rtSetResolverRange

long rtSetResolverRange(long Nr, double Range)

Sets the maximum range of the resolver. In other words, the maximum possible position which can be indicated by the resolver system.

Parameters

Nr	Resolver identifier
Range	Maximum position returned by the resolver (set to 0 for infinite range)

9.1.92 rtSetSpeed

long rtSetSpeed(double Speed)

Call this method to set the marking speed. The speed value remains in force until a next call of rtSetSpeed. All vector interpolation commands (“rtLineTo”, “rtArcTo”, “rtMoveTo”,…) are processed using the marking speed. Jump commands (“rtJumpTo”) are executed using the jump speed.

Parameters

Speed	marking speed in mm/sec, Speed must be set lower than or equal to maximum speed

9.1.93 rtSetVarBlock

long rtSetVarBlock(long Index, char Data)

Sets the data contained in the var block at the specified index position. Using the var block it is possible to define sequences of recurring characters such as text. The following example illustrates how the text "TEST" is written into the first four var block locations:

```
rtSetVarBlock(0,84);
rtSetVarBlock(1,69);
rtSetVarBlock(2,83);
rtSetVarBlock(3,84);
```

Note that the numbers in the second argument are ASCII codes representing each of the letters in the string.

Parameters

Index	index of the terget location in the var block (0-255)
Data	data to be written to the var block

9.1.94 rtSetWobble

long rtSetWobble(double Diameter, long Freq)

The method allows to set the wobble around any marking. Wobble is a time-varying offset with a specified frequency around a circle of a given diameter in the XY-plane. The control system superimposes this offset onto the current co-ordinates. To disable the wobble, set the diameter to zero. Note that wobble is only active when the gate is active.

Parameters

Diameter	diameter in mm of the circular wobble
Freq	freaquency in Hertz (revolutions per second)

9.1.95 rtSleep

long rtSleep(long Time)

This method suspends the execution of the current command list for a specified interval. The motors are stopped and the gate signal is switched off.

Parameters

Time	idle time in µsec

9.1.96 rtStoreCalibration

long rtStoreCalibration()

This method stores the calibration settings to the internal flash memory of the deflector control system.

9.1.97 rtStoreCalibrationFile

long rtStoreCalibrationFile(const char* FileName)

This method stores the current calibration in a calibration file. This method should be called at the end of a calibration. At power up the calibration on the deflection control system is cleared. The file is used to restore it.

Parameters

FileName	0 terminated string containing the full file name and path.

9.1.98 rtStoreCalibrationFileZ

long rtStoreCalibrationFileZ(const char* FileName)

Same as "rtStoreCalibrationFile" but for Z-axis calibration data.

Parameters

FileName	0 terminated string containing the full file name and path.

9.1.99 rtSuspend

long rtSuspend()

This command puts the controller DSP card into sleep mode and halts all executions. Wake-up can be achieved by issuing an ethernet command, but requires a control system of the type "CUA-ETH".

9.1.100 rtVarBlockFetch

long rtVarBlockFetch(long Start, long Size, char* FontName)

Gets information stored in a var block for jobs using job files. See "rtFileOpen" for more information.

Parameters

Start	start location in the var block (0-255)
Size	length of the segmant in the var block in bytes
FontName	pointer to a string containing the font name

9.1.101 rtWaitCanLink

long rtWaitCanLink(long Address, long Value, long Mask)

For systems using a CAN interface this function can be used to send the deflection system into an idle state until a specified message is received. A mask can be used to set "don't care" bits to listen for a range of different values.

Parameters

Address	internal CAN controller address (0-7, 8-15)
Value	value to listen for
Mask	mask for "don't care bits"

9.1.102 rtWaitIO

long rtWaitIO(long Value, long Mask)

Pins on the X7 digital I/O connector that are configured PC DIGITAL IN can be used for waiting. With rtWaitIO the rhothor™ deflection control system goes idle till the IO state defined by Value and mask is reached.

Parameters

Mask	defines bitwise which Pins are to be checked.
Value	defines bitwise the condition of the input pins defined by Mask.

9.1.103 rtWhileIO

long rtIfIO(long Value, long Mask)

rtWhile function let you repeat a block of functions while a specified IO state is True.

Parameters

Mask	defines bitwise which Pins are to be checked.
Value	defines bitwise the condition of the input pins defined by Mask.

9.1.104 rtWaitPosition

long rtWaitPosition(double Window)

Halts execution until the control system reaches a target position specified in an earlier move or jump command. The tolerance around this target position is defined by a rectangle.

Parameters

Window	Size of the rectangular tolerance region around the target position

9.1.105 rtWaitResolver

long rtWaitResolver(long Nr, double TriggerPos, long TriggerMode)

Waits until the resolver returns a specified position.

Parameters

Nr	resolver identifier (1 or 2)
TriggerPos	position to wait for in mm

TriggerMode	(1 or 2) Trigger immediately before or after the specified target position, respectively.

10 Appendix B: File Format Calibration Data File

```
// general:
// "//" line comment
// "[]" data between square braquets is optional
// ";" separator between values
//
// X and Y descriptor:
// defines calibration size and the number of samples (number must be odd)
// default values: calibration size = Fieldsize, samples = 33
// calibrationsize<Fieldsize
//
// Z descriptor:
// defines the Z levels used during the calibration
// default values: Z1=0, Z2=0
// -Fieldsize<=Z1,Z2<=FieldsizeZ/2
//
// array data:
// starting at left bottom up to right top, X direction first
// dx = actual x position - calibrated x position, default value 0
// dy = actual y position - calibrated y position, default value 0
// dz = actual z position - calibrated z position, default value 0
//
[X:x_calibration_size;x_samples] // descriptor for X axis
[Y:y_calibration_size;y_samples] // descriptor for Y axis
[Z:Z1,Z2] // descriptor for Z axis
//
// Z1 array data, size must be equal to xsamples*ysamples
//
(dx;dy[;dz]);...
...
...;(dx;dy[;dz])
//
// Z2 array data, size must be equal to xsamples*ysamples
// if omitted, Z2 array data equals Z1 array data
// Z2 array data is ignored when Z2 equals Z1
//
[(dx;dy[;dz]]),...
...
...;(dx;dy[;dz])]
```