www.newson.be

MANUAL

APPLICATION INTERFACING WITH CUA32 CONTROL UNITS

NEWSON NV

Table of Contents

APPLICATION INTERFACING WITH CUA32 CONTROL UNITS 5
HELLO WORLD, MARKING A SQUARE 5
HELLO WORLD, CREATING A FLASH FILE 5
HELLO WORLD, RUNNING A FLASH FILE 5
SIX AXIS NUMERICAL CONTROL SYSTEM 6
SEPARATE 6
HYBRID 6
CONTROLLING MULTIPLE DEFLECTION SYSTEMS 7
LIBRARY FUNCTIONS (ALPHABETICALLY ORDERED) 8
RTABORT(); 8
RTADDCALIBRATIONDATA(CONST CHAR* FILENAME); 8
RTARCMOVETO(DOUBLE X, DOUBLE Y, DOUBLE BF); 8
RTARCTO(DOUBLE X, DOUBLE Y, DOUBLE BF); 8
RTBURST(LONG TIME); 8
RTCHARDEF(LONG ASCII); 9
RTCIRCLE(DOUBLE X, DOUBLE Y, DOUBLE ANGLE); 9
RTCIRCLEMOVE(DOUBLE X, DOUBLE Y, DOUBLE ANGLE); 9
RTDOLOOP(); 9
RTDOWHILE(); 9
RTELSE(); 9
RTELSEIFIO(LONG VALUE, LONG MASK); 10
RTENDIF(); 10
RTERASEFROMFLASH(CONST CHAR* FILENAME); 10
RTFILECLOSE(); 10
RTFILECLOSEATHOST(); 10
RTFILECLOSEATINDEX(LONG INDEX); 10
RTFILEDOWNLOAD(CONST CHAR* FILENAME, CONST CHAR* DESTFILE); 11
RTFILEFETCH(CONST CHAR* FILENAME); 11
RTFILEOPEN(CONST CHAR* FILENAME); 11
RTFILEUPLOAD(CONST CHAR* SRCFILE, CONST CHAR* FILENAME); 11
RTFILEUPLOADATINDEX(CONST CHAR* SRCFILE, CONST CHAR* FILENAME, LONG INDEX); 11
RTFONTDEF(CONST CHAR* NAME); 12
RTFONTDEFEND(); 12
RTFORMATFLASH(); 12
RTGETANALOG(LONG NR, LONG* VALUE); 12
RTGETCFGIO(LONG NR, LONG *VALUE); 12
RTGETCOUNTER(LONG* VALUE); 12
RTGETDEFLREPLIES(LONG* CH1, LONG* CH2, LONG* CH3); 12
RTGETFIELDSIZE(DOUBLE* SIZE); 13
RTGETFIELDSIZEZ(DOUBLE* SIZE); 13
RTGETFILEINDEX(CONST CHAR* FILENAME, LONG* INDEX); 13
RTGETFIRSTFREEUSBDEVICE(CHAR* NAME); 13
RTGETFLASHFIRSTFILEENTRY(CHAR* NAME, LONG* SIZE); 14
RTGETFLASHMEMORYSIZES(LONG* TOTAL, LONG* ALLOCATED); 14
RTGETFLASHNEXTFILEENTRY(CHAR* NAME, LONG* SIZE); 14
RTGETID(CHAR* NAME); 14
RTGETIO(LONG* VALUE); 14
RTGETIP(CHAR* MAC, CHAR* IP); 15
RTGETLASERLINK(LONG ADDRESS, LONG* VALUE); 15

RTGETMAXSPEED(DOUBLE* SPEED); ... 15
RTGETNEXTFREEUSBDEVICE(CHAR* NAME); ... 15
RTGETQUERYTARGET(LONG* INDEX); ... 15
RTGETRESOLVERS(DOUBLE* X, DOUBLE* Y); ... 15
RTGETSCANNERDELAY(LONG* DELAY); ... 16
RTGETSERIAL(LONG* SERIAL); ... 16
RTGETSETPOINTFILTER(LONG* TIMECONST); ... 16
RTGETSTATUS(LONG* MEMORY); ... 16
RTGETTABLEPOSITIONS(DOUBLE* X, DOUBLE* Y, DOUBLE* Z); ... 16
RTGETTARGET(LONG* MASK); ... 17
RTGETVERSION(CHAR* VERSION); ... 17
RTINCREMENTCOUNTER(); ... 17
RTINDEXFETCH(LONG INDEX); ... 17
RTIFIO(LONG VALUE, LONG MASK); ... 17
RTJUMPTO(DOUBLE X, DOUBLE Y); RTJUMPTO3D(DOUBLE X, DOUBLE Y, DOUBLE Z); ... 17
RTLINETO(DOUBLE X, DOUBLE Y); RTLINETO3D(DOUBLE X, DOUBLE Y, DOUBLE Z); ... 18
RTLISTCLOSE(); ... 18
RTLISTOPEN(LONG MODE); ... 18
RTLOADCALIBRATIONFILE(CONST CHAR* FILENAME); ... 18
RTMOVETO(DOUBLE X, DOUBLE Y); RTMOVETO3D(DOUBLE X, DOUBLE Y, DOUBLE Z); ... 18
RTPARSE(CONST CHAR* CMD); ... 19
RTPULSE(DOUBLE X, DOUBLE Y); RTPULSE3D(DOUBLE X, DOUBLE Y, DOUBLE Z); ... 19
RTRESET(); ... 19
RTRESETCALIBRATION(); ... 19
RTRESETCOUNTER(); ... 19
RTRESETEVENTCOUNTER(); ... 19
RTRESETRESOLVER(LONG NR); ... 19
RTRUNSERVER(LONG ID, VOID* PARAMS1, VOID* PARAMS2); ... 20
RTSELECTDEVICE(CONST CHAR* IP); ... 20
RTSETANALOG(LONG VALUE, LONG MASK); ... 20
RTSETCFGIO(LONG NR, LONG VALUE); ... 20
RTSETCOUNTER(LONG VALUE); ... 21
RTSETFIELDSIZE(DOUBLE SIZE); ... 21
RTSETIMAGEMATRIX(DOUBLE A11, DOUBLE A12, DOUBLE A21, DOUBLE A22);
RTSETIMAGEMATRIX3D(DOUBLE A11, DOUBLE A12, DOUBLE A21, DOUBLE A22, DOUBLE A31, DOUBLE A32); ... 21
RTSETIMAGEOFFSRELXY(DOUBLE X, DOUBLE Y); ... 21
RTSETIMAGEOFFSXY(DOUBLE X, DOUBLE Y); ... 21
RTSETIMAGEOFFSZ(DOUBLE Z); ... 22
RTSETIMAGEROTATION(DOUBLE ANGLE); ... 22
RTSETIO(LONG VALUE, LONG MASK); ... 22
RTSETJUMPSPEED(DOUBLE SPEED); ... 22
RTSETLASER(BOOL ONOFF); ... 22
RTSETLASERLINK(LONG ADDRESS, LONG VALUE); ... 23
RTSETLASERTIMES(LONG GATEONDELAY, LONG GATEOFFDELAY); ... 23
RTSETLOOP(LONG LOOPCTR); ... 23
RTSETMATRIX(DOUBLE A11, DOUBLE A12, DOUBLE A21, DOUBLE A22); ... 23
RTSETMINGATEPERIOD(LONG TIME); ... 24
RTSETOFFSINDEX(LONG INDEX); ... 24
RTSETOFFSXY(DOUBLE X, DOUBLE Y); ... 24
RTSETOFFSZ(DOUBLE Z); ... 24
RTSETOSCILLATOR(LONG NR, DOUBLE PERIOD, DOUBLE PULSEWIDTH); ... 24
RTSETOTF(LONG NR, BOOL ON); ... 25
RTSETPULSEBULGE(DOUBLE FACTOR); ... 25
RTSETQUERYTARGET(LONG INDEX); ... 25

RTSETRESOLVER(LONG NR, DOUBLE STEPSIZE, DOUBLE RANGE); 25
RTSETRESOLVERPOSITION(LONG NR, DOUBLE POSITION); 25
RTSETRESOLVERRANGE(LONG NR, DOUBLE RANGE); 26
RTSETRESOLVERTRIGGER(LONG NR, DOUBLE POSITION, LONG IO); 26
RTSETROTATION(DOUBLE ANGLE); 26
RTSETSPEED(DOUBLE SPEED); 26
RTSETTABLE(LONG NR, DOUBLE POSITION); 26
RTSETTABLEDELAY(LONG NR, LONG DELAY); 27
RTSETTABLESNAPSIZE(LONG NR, DOUBLE SNAPSIZE); 27
RTSETTABLESTEPSIZE(LONG NR, DOUBLE STEPSIZE); 27
RTSETTABLEWHILEIO(LONG VALUE, LONG MASK); 28
RTSETTARGET(LONG MASK); 28
RTSETVARBLOCK(LONG I, CHAR DATA); 28
RTSETWOBBLE(DOUBLE DIAM, LONG FREQ); 28
RTSETWOBBLEEX(LONG NTYPE, DOUBLE NAMPL, LONG NFREQ, LONG TTYPE, DOUBLE TAMPL, LONG THARM, LONG TPHASE); 29
RTSLEEP(LONG TIME); 29
RTSTORECALIBRATIONFILE(CONST CHAR* FILENAME); 29
RTSUSPEND(); 29
RTSYSTEMRESUME(); 30
RTSYSTEMSETIO(LONG VALUE, LONG MASK); 30
RTSYSTEMSUSPEND(); 30
RTSYSTEMUARTOPEN(LONG BAUDRATE, CHAR PARITY, CHAR STOPBITS); 30
RTSYSTEMUARTWRITE(LONG BYTES, CHAR* DATA); 30
RTSYSTEMUDPSEND(CHAR* IP, SHORT PORT, CHAR* DATA); 30
RTTABLEARCTO(DOUBLE X, DOUBLE Y, DOUBLE BF); 31
RTTABLEJOG(LONG NR, DOUBLE SPEED, LONG WHILEIO); 31
RTTABLELINETO(DOUBLE X, DOUBLE Y); 31
RTTABLEMOVE(LONG NR, DOUBLE TARGET); 32
RTTABLEMOVETO(DOUBLE X, DOUBLE Y); 32
RTUARTREAD(LONG* BYTES, CHAR* DATA); 32
RTVARBLOCKFETCH(LONG START, LONG SIZE, CONST CHAR* FONTNAME); 32
RTWAITIO(LONG VALUE, LONG MASK); 32
RTWAITPOSITION(DOUBLE WINDOW); 33
RTWAITRESOLVER(LONG NR, DOUBLE TRIGGERPOS, LONG TRIGGERMODE); 33
RTWHILEIO(LONG VALUE, LONG MASK); 33
NOT SUPPORTED FUNCTIONS, KEPT FOR COMPATIBILITY. 33

RETURN CODES LIBRARY FUNCTIONS 35

ERR_OK (-1) 35
ERR_BUSY (2) 35
ERR_JOB (3) 35
ERR_HARDWARE (5) 35
ERR_DATA (13) 35
ERR_IMPLEMENTATION (23) 35

CALIBRATION 36

2D-CALIBRATION 37
3D-CALIBRATION 38

application interfacing with CUA32 control units

This document describes how a target application can control CUA32 devices using the supplied dynamic link library. For more information about the CUA32 devices we refer to the manual "CUA32_Cfg: Installation and Configuration of CUA32 control units".

Hello world, marking a square

All begins with installing the CUA32-MST device (ref. CUA32_Cfg chapter 2.1). Use a USB cable to make the connection with the host. To create and compile a hello world application, the rhothor.lib and rhothorDll.h must be included in the project. When run, the application should first connect with the CUA32 device by calling *rtSelectDevice("USB")*. Afterwards a square can be marked by calling following functions.

```
rtListOpen(1); // open command queue for streaming
rtSetJumpSpeed(5000); // set jump speed
rtSetSpeed(1000); // set marking speed
rtJumpTo(-40,40); // jump to start point
rtLineTo(40,40); // marking top line
rtLineTo(40,-40); // marking right edge
rtLineTo(-40,-40); // marking bottom line
rtLineTo(-40,40); // marking left edge
rtListClose(); // close queue, CUA32 starts marking the square
```

Hello world, creating a flash file

```
rtFileOpen("myfile"); // open file
rtSetJumpSpeed(5000); // add set jump speed
rtSetSpeed(1000); // add set marking speed
rtSetLoop(10); // repeat loop
rtJumpTo(-40,40); // add jump to start point
rtLineTo(40,40); // add marking top line
rtLineTo(40,-40); // add marking right edge
rtLineTo(-40,-40); // add marking bottom line
rtLineTo(-40,40); // add marking left edge
rtDoLoop(); // close loop
rtFileClose(); // close file
```

Hello world, running a flash file

```
rtListOpen(1); // open command queue for streaming
rtFileFetch("myfile"); // fetch file
rtListClose(); // close queue
```

© Copyright Newson NV, 2000-2017

six axis numerical control system

The CUA32 is a six-axis numerical control system. Up to three deflectors and steppers can be connected to the controller. Laser beam deflection systems are very fast but have a limited field size. When larger areas are needed, a mechanical table is used to change the position of the deflection system in relation with the workpiece. A machine comprising a XY table and a deflection system can process parts as large as the travel range of its table at a speed not limited by it. To control such a machine, the application needs to handle four axes'.

separate

Table and deflector are controlled by their own functions, each having their own coordinate system. Image commands (*rtLineTo, rtArcTo...*) will invoke a deflector movement while table commands (*rtTableMove, rtTableLineTo...*) will position the table. When controlled separately, the application needs to make sure that table is put in position before starting the deflectors. To avoid clipping, the graphical content processed by the deflection head must also fit within its limited working area. The latter is straight forward when only small isolated images are needed. Processing images larger than the field size of the deflection head imposes challenges.

Image coordinates pass through several data transformation steps before being transferred to the deflectors. Their values are shifted and rotated as requested by preceding commands (*rtSetImageMatrix, rtSetOffsXY...*). Coordinates used in table commands are send to the stepper controllers as is, without any offsetting nor rotation. Calling *"rtSetOffsXY"* will offset the coordinates of a *"rtMoveTo"* command but will have no effect on a *"rtTableMoveTo"* command.

Image coordinate conversion math

```
|ImgOffsX|   |ImgOffsX|   |ImgA11 ImgA12   0|   |ImgOffsRelX|
|ImgOffsY| = |ImgOffsY| + |ImgA21 ImgA22   0| * |ImgOffsRelY|
|ImgOffsZ|   |ImgOffsZ|   |ImgA31 ImgA32   1|   |          0|

|x|   |ImgA11 ImgA12   0|   | Xcmd|   |ImgOffsX|
|y| = |ImgA21 ImgA22   0| * | Ycmd| + |ImgOffsY|
|z|   |ImgA31 ImgA32   1|   | Zcmd|   |ImgOffsZ|

|X|   |A11 A12   0|   |x|   |OffsX|
|Y| = |A21 A22   0| * |y| + |OffsY|
|Z|   |  0   0   1|   |z|   |OffsZ|
```

hybrid

To reduce application overhead, the CUA32 device supports a hybrid control mode. In this mode, the table and deflectors share the image coordinate system. A *rtLineTo* will mark using both deflectors and steppers at the same time. The long straights will be done using the table while the smaller graphical data will be handled by the deflectors. A hysteresis scheme avoids unnecessary table movement. Because the image coordinate system is used, the positions are rotated and shifted before being send to the hybrid axis's.
In normal hybrid mode, calculated table positions are used to control the deflector's setpoints. When a position feedback mechanism is available, the CUA32 on-the-fly feature can be used to increase overall accuracy.

controlling multiple deflection systems

Several parameters determine the time needed to execute a job. When the available cycle time is shorter, multiple deflection systems can be used (increasing the speed by dividing the work). A master card (CUA32-MST) can connect seven additional slave devices (CUA32-SLV) to the host computer. Every device comprises a six-axis numerical system which can be controlled independently or commonly. Functions *rtSetTarget* and *rtSetQueryTarget* are provided for this purpose and can to be used to address the devices prior to invoking commands. Single device applications can omit those functions when the target controller is mapped to index 1.

library functions (alphabetically ordered)

rtAbort();

This function closes the current command list, purges all commands already stored in hardware and forces the system and laser in idle state.
* applies to all targets
* applicable when connected

rtAddCalibrationData(const char FileName);*

This function call adds offset data to the target's calibration stored in flash.
* applies to least significant masked target (*rtSetTarget*)
* applicable when system is idle
* parameters:
FileName : name of the text file holding the offset data

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

This function adds an arc wise movement of the setpoint to the command list. The arc starts from the current position and goes to coordinate (*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.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
X,Y : target position, range -8388.608...8388.607 mm
BF : bulge factor

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

This function adds an arc wise marking to the command list. The arc starts at the current position and goes to coordinate (*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.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
X,Y : target position, range -8388.608...8388.607 mm
BF : bulge factor

rtBurst(long Time);

This function adds a burst command to the list. During execution of a burst, the gate signal is activated while the motors are standing still. The rtBurst function can be used to increase corner sharpness. Hovering the setpoint at the corner buys the deflectors more time to reach the corners position.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode

© Copyright Newson NV, 2000-2017

* parameters:
Time : hover time, range 0...2147483647 µsec

rtCharDef(long Ascii);

rtCircle(double X, double Y, double Angle);

This command adds a circle marking to the list. The parameters *X* and *Y* define the center point of the circle. The Radius is defined as the distance between the center point and the current position. The angle is positive for counter clockwise and negative for clockwise marking.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
X,Y : center position, range -8388.608...8388.607 mm
Angle : radians

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

This command adds a circular movement to the list. Its execution is like the *rtCircle* command, but the gate signal is switched off.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
X,Y : center position, range -8388.608...8388.607 mm
Angle : radians

rtDoLoop();

This command marks the end of a counter loop. When processed, the loop counter is decremented. When not zero, command sequencing restarts after the *rtSetLoop* command.
* applies to all targets
* applicable in compile mode

rtDoWhile();

This function marks the end of a conditional loop. When processed, the command fetcher goes back to the *rtWhileIO*.
* applies to all targets
* applicable in compile mode

rtElse();

This function adds a control flow command to the command list. When the previous IO testing returned false, the command sequencer will fetch all instructions following the rtElse command.
* applies to all targets
* applicable in compile mode

rtElseIfIO(long Value, long Mask);

This function adds a control flow command to the command list. When this command is processed, the IO state of the least significant target is tested against the declared value. The comparison only considers those IO with their corresponding mask bit set to one. When the result returns equal, the instruction sequencing is continued. When the comparison returns false, the instruction sequencer continues at the next *rtElse*, *rtElseIfIO* or *rtEndIf* command. This function needs calculated instruction locations and therefore cannot be used in streaming mode.
* applies to all targets
* applicable in compile mode
* parameters:
Value : 0...65535
Mask : 0...65535

rtEndIf();

This function marks the end of the conditional command sequencing.
* applies to all targets
* applicable in compile mode

rtEraseFromFlash(const char FileName);*

Function erases a file from flash. All file names used on the CUA32 are case sensitive. When file is not found, ERR_OK is returned.
* applies to all targets
* applicable when system is idle

rtFileClose();

Function compiles and saves the command list to flash memory.
* applies to all targets
* applicable when system is idle

rtFileCloseAtHost();

Function compiles and saves command list as a binary file on the host computer. The file can be uploaded afterwards using *rtFileUpload* or *rtFileUploadAtIndex*.
* applies to all targets
* applicable when system is idle

rtFileCloseAtIndex(long Index);

Function closes the command list and stores the complete list to flash memory at the designated sector. The CUA32 flash is formatted to 250 sectors. 249 of them are available for saving files. The first sector, with index 0, is reserved to hold boot start code. Each sector is 256 Kbyte in size. When a command list doesn't fit in a single sector, it continues in the next. The application should verify that the complete command list is stored in consecutive free flash memory.
* applies to all targets
* applicable when system is idle
* parameters:
Index : sector number

© Copyright Newson NV, 2000-2017

rtFileDownload(const char FileName, const char* DestFile);*

This function copies a file stored on flash to the host.
* applies to all targets
* applicable when system is idle
* parameters:
FileName : flash file name (zero terminated string)
DestFile : destination file name flash file name (zero terminated string)

rtFileFetch(const char FileName);*

This command adds all commands comprised by the file on the system's flash to the command list.
* applies to all targets
* applicable in streaming and compile mode
* parameters:
FileName : zero terminated string

rtFileOpen(const char FileName);*

rtFileOpen allocates memory on the host computer and opens the list for **compilation**. The host maintains the complete command chain during its construction. No list commands are send to the device. In this mode the instruction set becomes extended with control flow commands. When *rtFileClose* is called, the list is compiled and saved on the CUA32 flash. When *rtFileCloseAtHost* is called, host memory is used.
* applies to all targets
* applicable when system is idle
* parameters:
FileName: file name (zero terminated string, length < 244 bytes)

rtFileUpload(const char SrcFile, const char* FileName);*

Function copies file from host to flash.
* applies to all targets
* applicable when system is idle
* parameters:
SrcFile : source file name (zero terminated string)
FileName : flash file name (zero terminated string, length < 244 bytes)

rtFileUploadAtIndex(const char SrcFile, const char* FileName, long Index);*

This function copies a file from host to flash at specified index. The CUA32 flash is formatted to 250 sectors. 249 of them are available for saving files. The first sector, with index 0, is reserved to hold boot start code. Each sector is 256 Kbyte in size. When a file doesn't fit in a sector, it continues in the next. The application should verify that the all the data from the source file is stored in consecutive free flash memory.
* applies to all targets
* applicable when system is idle
* parameters:
SrcFile : source file name (zero terminated string)
FileName : flash file name (zero terminated string, length < 244 bytes)

rtFontDef(const char Name);*

rtFontDefEnd();

rtFormatFlash();

This function deletes all files, including the boot start file, from the flash
* applies to all targets
* applicable when system is idle

rtGetAnalog(long Nr, long Value);*

Function samples the analog voltage on the selected IO pin
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
Nr=5: query voltage on IO5
Nr=6: query voltage on IO6
Nr=7: query voltage on IO7
Nr=8: query voltage on IO8
Value: place holder to store the sample (mV)

*rtGetCfgIO(long Nr, long *Value);*

The function queries the configuration setting of the selected IO pin. IO functionality and their respective configuration setting can be depicted from the rhothor.exe configuration page. The returned value equals the IO pin's drop-down list index.
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
Nr: 1,2...,17
Value: placeholder to store configuration setting

rtGetCounter(long Value);*

The function queries the target's counter value. This counter is altered when *rtIncrementCounter* or *rtSetCounter* is executed. Querying this counter allows the host to determine which commands are executed and which commands are still in the queue.
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
Value: place holder to store the target's counter.

rtGetDeflReplies(long CH1, long* CH2, long* CH3);*

The function returns the last replies of the connected deflectors. The CUA32 controller uses 20-bit absolute setpoint control to steer the deflectors. For a complete description of the data flow between control card and deflectors see manual A3G_RTA chapter 4.2.1. In short, the deflectors reply by sending three bytes. An error byte is added to be used for validation. When zero, the first

© Copyright Newson NV, 2000-2017

three bytes contains the reply. When one, the deflector didn't reply and connection with CUA32 device should be verified. The reply is formatted as follows:
Bit 0 : error setpoint changed too fast
Bit 1 : error deflector safety fuse tripped
Bit 2 : error deflector overload
Bit 3 : 0 bit
Bit 4...Bit 23 : 20-bit two's complement actual position 0x80000...0x7FFFF
Bit 24 : error reply
Bit 25…31 : 0
Communication between target controller and connected deflectors is binary. The relation between deflector coordinates and system coordinates is defined as follows:
CH1,CH2 : deflector (bits) = system coordinate (mm) * 16640000 / field size
CH3 : deflector (bits) = system coordinate (mm) * 16640000 / field size Z
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
CH1 : place holder for reply X deflector
CH2 : place holder for reply Y deflector
CH3 : place holder for reply Z deflector

rtGetFieldSize(double Size);*

Function returns the field size of the XY deflection system.
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
Size : place holder to store the target's field size (mm)

rtGetFieldSizeZ(double Size);*

Function returns the field size of Z deflection system.
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
Size : place holder to store the target's Z field size (mm)

rtGetFileIndex(const char FileName, long* Index);*

This function searches the flash for a file with the declared *FileName*. When found, its index is returned. When not found, index is set to -1.
* applies to all targets
* applicable when system is idle
* parameters
FileName : file name to be searched (zero terminated string)
Index : place holder to receive file index

rtGetFirstFreeUSBDevice(char Name);*

This function searches the host USB hubs for available CUA32 devices. When found, it's USB ID string is returned. By concussively calling *rtGetNextFreeUSBDevice* a complete list of available

CUA32 devices can be obtained. A USB ID string is a zero-terminated string with a total length limited to 64 bytes. When no free device is find a zero string is returned.
* applicable always
* parameters
Name : 64-byte place holder to receive the USB ID string (zero terminated string)

rtGetFlashFirstFileEntry(char Name, long* Size);*

Function searches flash for a file header. The function starts at sector with index 0 and returns the index of the first valid sector. This function should be called prior to calling *rtGetNextFileEntry*.
* applies to all targets
* applicable when system is idle
* parameters
Name : 244-byte place holder to receive file name (zero terminated string)
Size : place holder to receive file size

rtGetFlashMemorySizes(long Total, long* Allocated);*

This function returns both total and allocated flash memory sizes. The CUA32 master device is fitted with a flash disk comprising 250 sectors each 256 Kbyte in size.
* applies to all targets
* applicable when system is idle
* parameters
Total, Allocated : placeholders to store the memory sizes (bytes)

rtGetFlashNextFileEntry(char Name, long* Size);*

This function searches flash for the next file header. Search is started at current index (result of *rtGetFlashFirstFileEntry* or previous call).
* applies to all targets
* applicable when system is idle
* parameters
Name : 244-byte place holder to receive file name (zero terminated string)
Size : place holder to receive file size

rtGetID(char Name);*

This function returns the "USB ID" string as specified in the rhothor configuration program.
* applicable when connected
* parameters
Name : 64-byte place holder for the USB ID string (zero terminated string)

rtGetIO(long Value);*

Function returns the current digital levels sampled on the IO's of the selected target. Bit 0 holds value of IO1, bit 1 of IO2.... The bit values related with the analog IO's are calculated. On those IO’s the analog voltage is sampled and compared with midscale value (2.5V). When higher a one bit will be returned.
* applies to query target (*rtSetQueryTarget*)
* applicable always
* parameters:

Value : place holder to store the sample

rtGetIP(char Mac, char* IP);*

When host connection is done over USB, the ethernet connector can be used to send UDP commands over the internet. The devices can be directly connected or connected over a switch. To map IP with MAC addresses, the CUA32 master controller maintains a ARP-cache containing four entries. The MAC addresses are device specific and known. The *rtGetIP* function allows the application to obtain their IP address needed for internet communication.
* applicable always
* parameters:
Mac : “dd.dd.dd.dd.dd.dd” (zero terminated string, dd: “0”…”255”)
IP : 16-byte place holder to receive IP address (IPV4)

rtGetLaserLink(long Address, long Value);*

Function queries the laser link connected to the target controller.
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
Address : 0x80...0xFF
Value : place holder to store the reply.

rtGetMaxSpeed(double Speed);*

The function returns the maximal speed of the target controller. The maximal speed is set by to 100 times the field size / sec.
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
Speed : placeholder to store the speed (mm/sec)

rtGetNextFreeUSBDevice(char Name);*

This function should be called repeatedly after calling *rtGetFirstFreeUSBDevice* to obtain the USB ID strings of available CUA32 devices. When no remaining free devices are found a zero string is returned. A USB ID string is a zero-terminated string with a total length limited to 64 bytes.
* applicable always
* parameters
Name : place holder for the USB ID string (zero terminated string)

rtGetQueryTarget(long Index);*

This function returns the index, 1 to 8, of the current query target (*rtSetQueryTarget*).
* applicable when connected
* parameters
Index : place holder for current query target

rtGetResolvers(double X, double* Y);*

System returns the on the fly positions for both X and Y axis in mm. (resolution 1 µm)

* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
X,Y : placeholders to store the on-the-fly-offset counters (mm)

rtGetScannerDelay(long Delay);*

Rhothor deflectors can be customer tuned to any delay between 60 and 350 µsec. Shorter delays will increase system bandwidth while fast operational speeds require a longer delay setting for the deflection system. The setting and tuning is done through the rhothor executable. The result is queried using this function. When combined with *rtGetSetpointFilter* the application can calculate the theoretical value for the laser delay (*rtSetLaserTimes*):
laser on delay = setpoint filter + scanner delay – laser rise time
laser off delay = setpoint filter + scanner delay – laser fall time
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
Delay : place holder for the deflector delay (µsec).

rtGetSerial(long Serial);*

rtGetSetpointFilter(long TimeConst);*

This function returns the time constant of the setpoint filter. The setpoint filter can be set using the rhothor executable. This function combined with *rtGetScannerDelay* allows the application to calculate the theoretical values for the laser delay (*rtSetLaserTimes*).
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
TimeConst : place holder for the setpoint filter time constant (µsec)

rtGetStatus(long Memory);*

This function returns ERR_BUSY (2) when there are still commands waiting for execution. When the command queue is empty, the command returns ERR_OK (-1) . When the parameter Memory is not NULL, the queue size is returned. In some cases, the application could use this size when adding commands. When the queue becomes too large, the application should suspend command generation to avoid running out of memory.
* applies to all targets
* applicable when connected
* parameters:
Memory : placeholder to receive the estimated size of the command list (bytes)

rtGetTablePositions(double X, double* Y, double* Z);*

Function returns the actual table setpoint positions.
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
X,Y,Z : place holders to store the stepper current setpoints (mm)

© Copyright Newson NV, 2000-2017

rtGetTarget(long Mask);*

This function returns the current target mask (*rtSetTarget*).
* applicable when connected
* parameters
Mask : place holder for current mask

rtGetVersion(char Version);*

rtIncrementCounter();

This function appends an increment counter event to the command list. Every target has a counter that can be controlled and queried through commands. The counter can be used by the application to determine which commands have been processed and which commands are still in the queue.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode

rtIndexFetch(long Index);

This command adds all commands stored in flash at the selected index to the command list.
* applies to all targets
* applicable in streaming and compile mode
* parameters:
Index : 1...249

rtIfIO(long Value, long Mask);

This function adds a control flow command to the command list. Processing a *rtIfIO* command starts by waiting until all connected targets are idle. Afterwards the IO state of the least significant target is tested against the declared value. The comparison only considers those IO with their corresponding mask bit set to one. When the result returns equal, the instruction sequencing is continued until *rtElse* or *rtElseIfIO* is encountered. When the comparison returns false, the instruction sequencer continues at the next *rtElse, rtElseIfIO* or *rtEndIf*.
* applies to all targets
* applicable in compile mode
* parameters:
Value : 0...65535
Mask : 0...65535

rtJumpTo(double X, double Y); rtJumpTo3D(double X, double Y, double Z);

This command adds a jump to the command list. Jumps are executed by the target processors as a linear ramp at jump speed. Laser is switched to idle during command execution.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
X, Y, Z : target position, range -8388.608...8388.607 mm

rtLineTo(double X, double Y); rtLineTo3D(double X, double Y, double Z);

This command adds a line marking to the command list. The line starts at current position and extends towards the target position. Marking speed (*rtSetSpeed*) is used during execution.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
X,Y,Z : target position, range -8388.608...8388.607 mm

rtListClose();

This function closes the command list.
* applies to all targets
* applicable in open list mode (*rtListOpen*)

rtListOpen(long Mode);

This command opens a command list.
* applies to all targets
* applicable when system is idle
* parameters:
Mode =1: *rtListOpen(1)* will open a list for **streaming**. The list operates like a FIFO memory. The application adds commands to the queue, while the CUA32 device retrieves and processes them. To maximize data bandwidth commands are gathered and send in batches to the CUA32 device. Each batch is 1024 bytes in size. When rtListClose is called, the last batch is padded with NOP commands before being send. The CUA device has 32-Kbyte dedicated dual port memory to hold the queue data. When the command list becomes larger, the library will extend the queue capacity using host memory. To avoid running out of memory, the application should regularly query the total size of the command queue (*rtGetStatus*).
Mode =3: *rtListOpen(3)* allocates 256KB memory on the host computer and open the list for **compilation**. The host maintains the complete command chain during its construction. No list commands are send to the device. In this mode the instruction set becomes extended with control flow commands. When rtListClose is called, the complete list is compiled and saved on the CUA32 flash as a boot start file. This file will be started whenever the CUA32 device is powered on. In general, a single command occupies 8 bytes, so the size of the command list will be limited to 32000 commands.
Mode =4: *rtListOpen(4)* operates much like *rtListOpen(3).* Instead of creating a boot start file, *rtListClose* compiles, transfers, and starts the list for immediate execution. This mode allows using control flow commands without the need to create files. As in mode 3, the size of the command list is limited to 32000 commands.

rtLoadCalibrationFile(const char FileName);*

This function copies the file content, containing the calibration, in the systems flash memory.
* applies to least significant masked target (*rtSetTarget*)
* applicable when system is idle

rtMoveTo(double X, double Y); rtMoveTo3D(double X, double Y, double Z);

This command adds a linear movement to the command list. The movement starts at current position and extends towards the target position. During the movement laser is set to idle and

marking speed is used. This function can be used to improve quality at sharp corners. The actual laser lines can be extended by shadow lines to allow deflectors to ramp.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
X,Y,Z : target position, range -8388.608...8388.607 mm

rtParse(const char Cmd);*

reserved rhothor function

rtPulse(double X, double Y); rtPulse3D(double X, double Y, double Z);

This command adds a pulse command to the list. Pulse commands are executed by starting a linear ramp with idling laser towards the target position. The last portion of the movement is done with activated laser signal. Duration of this pulse is determined by the setting of oscillator 3 and the selected laser mode (rhothor.exe). The speed used for ramping is calculated based upon the distance, the selected marking speed (*rtSetSpeed*) and minimal gate period time. (*rtSetMinGatePeriod*). When the distance towards target position is smaller than the minimal gate period, the ramping speed is reduced.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
X,Y,Z : target position, range -8388.608...8388.607 mm

rtReset();

rtResetCalibration();

This function resets the target's calibration.
* applies to least significant masked target (*rtSetTarget*)
* applicable when system is idle

rtResetCounter();

This function appends a reset counter event to the command list.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode

rtResetEventCounter();

rtResetResolver(long Nr);

This function adds a reset resolver command to the list. When the command is executed, the target controller will reset the selected on-the-fly counter.
* applies to all masked targets (rtSetTarget)
* applicable in streaming and compile mode
* parameters:
Nr =1: on-the-fly counter X axis

Nr =2: on-the-fly counter Y axis

rtRunServer(long Id, void Params1, void* Params2);*

reserved rhothor function

rtSelectDevice(const char IP);*

This method is used to setup a physical connection with a CUA32 device. Once connected all library functionality becomes available to the application. The device and how the connection is made is declared by the content of the IP string.
* applicable when not selected
* parameters
* example *IP* : *USB*
The connection with the CUA32 device will be made over USB. The USB tree will be searched for the first available CUA32 device.
* example *IP* : *USB "newson"*
The connection with the CUA32 device will be made over USB. The USB tree will be searched for a device with USB ID set to "newson".
* example *IP* : *TCP "172.16.224.20"*
The connection with the CUA32 device will be made over the ethernet connection using TCP-IP protocol and targeting IP address 172.16.224.20.
* example *IP* : *UDP "172.16.224.20"*
The connection with the CUA32 device will be made over the ethernet connection using UDP-IP protocol and targeting IP address 172.16.224.20.

rtSetAnalog(long Value, long Mask);

This function adds an analog alteration command to the command list. When executed, the target's analog outputs will be set to the desired value. To change, the analog output must be enabled (rhothor.exe) and the mask bit must be set. The target's DA convertors have a 5V full scale and a resolution of 12 bit.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Value : voltage, range 0...5000 mV
Mask : IO bit 5,6,7 and or 8 must be set to change the analog output

rtSetCfgIO(long Nr, long Value);

This function adds a IO configuration change in the command list. IO functionality and their respective selection values can be depicted from the rhothor.exe configuration page. It's list index is the value needed to select the function as declared in the drop-down list.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Nr : 1:IO1, 2:IO2... 17:IO17
Value : function number

© Copyright Newson NV, 2000-2017

rtSetCounter(long Value);

This function appends a counter preload event to the command list. When executed the target's sequence counter is loaded with the desired value.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Value : 0...65535

rtSetFieldSize(double Size);

This function adds a field size change in the command list. When this command is processed the scaling of the coordinates send to Ch1 and Ch2 deflectors is altered. The relation between system and deflector units is defined as follows:
deflector setpoint (bit) = (16640000/field size) * system setpoint (mm)
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Size : deflectors field size, range -8388.608...8388.607 mm

rtSetImageMatrix(double a11, double a12, double a21, double a22);
rtSetImageMatrix3D(double a11, double a12, double a21, double a22, double a31, double a32);

Those functions add an image transformation matrix change event to the command list. When executed the target's image transformation (*ImgAij*) matrix is altered. An application can rotate and stretch images in the XY- plane using the transformation matrix. When the 2D function is used, the parameters *ImgA31* and *ImgA32* are set to zero.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
a11,a12,a21,a22,a31,a32 : factor

rtSetImageOffsRelXY(double X, double Y);

This function adds an offset change event in the command list. When executed the declared offset values are transformed (*ImgAij*) and added to the image offset vector. This relative offset can be used to create fonts. A font is a set of images (characters). Each image contains line commands specifying the typeface and a *rtSetImageOffsRelXY* command. The latter defines the character width and can be used to shift the cursor to the next position.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
X,Y : relative image offset position, range -8388.608...8388.607 mm

rtSetImageOffsXY(double X, double Y);

This function adds an offset change event in the command list. When executed the image offset vector (*ImgOffsX, ImgOffsY*) will be loaded with the declared values.
* applies to all masked targets (rtSetTarget)
* applicable in streaming and compile mode
* parameters:

X,Y : image offset position, range -8388.608...8388.607 mm
description:
This method sets the absolute XY-image offset vector.

rtSetImageOffsZ(double Z);

This function adds an image Z-offset change event in the command list. When executed the image Z-offset vector (*ImgOffsZ*) will be loaded with the declared value.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Z : image offset position, range -8388.608...8388.607 mm

rtSetImageRotation(double Angle);

This function adds an image transformation matrix change event to the command list. When executed the target's image transformation matrix (*ImgAij*) will be loaded with a rotation function. The function is the same as:
rtSetImageMatrix(cos(Angle),-sin(Angle),sin(Angle),cos(Angle))
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Angle : rotation angle in radians

rtSetIO(long Value, long Mask);

This function adds an output change event to the command list. When executed by the target, the output values will be overwritten as follows:
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….
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Value, Mask : range 0...65535

rtSetJumpSpeed(double Speed);

* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Speed : speed in mm/s

rtSetLaser(bool OnOff);

This function adds a laser control event to the command list. During command processing, the laser is controlled together with the deflection motors. When *rtSetLaser(1)* is executed, the laser is activated continuously until explicitly switched off by calling *rtSetLaser(0)*. This functionality is useful for laser power calibration.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:

OnOff =0: gate signal will be active during marking and switched off on idle.
OnOff =1: gate signal will always be on

rtSetLaserLink(long Address, long Value);

This function adds a laser link data upload event. When executed, the target will load the laser link's addressed register with *Value*. Execution takes 130 µsec during which the laser is idled.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
Address : range 0...127
Value : range 0...255

rtSetLaserTimes(long GateOnDelay, long GateOffDelay);

This function adds a configuration event to the command list. Actual deflector positions are delayed in time. By shifting the laser signal, resynchronization with deflector movement is obtained. *rtGetScannerDelay* and *rtGetSetpointFilter* return the selected time constants from both, deflector and setpoint filter. When laser rise and fall times are known, *GateOnDelay* and *GateOffDelay* can be calculated. When not, a trial and error approach should be used. The calculation:
laser on delay = setpoint filter + scanner delay – laser rise time
laser off delay = setpoint filter + scanner delay – laser fall time
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
GateOnDelay, GateOffDelay : 1...2047 µsec

rtSetLoop(long LoopCtr);

This function adds a control flow command to the command list. When the instruction is processed, all commands between rtSetLoop and rtDoLoop are grouped and repeated. Parameter "LoopCtr" declares the number of executions. A zero value will create in an infinite loop. Nesting of control flow commands is limited to 16 levels.
* applies to all targets
* applicable in compile mode
* parameters:
LoopCtr : 0...65535

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

This command adds a transformation matrix change event to the command list. When executed the target's transformation matrix is altered. An application can rotate and stretch all graphical output in the XY- plane using the transformation matrix.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
a11,a12,a21,a22 : factor

rtSetMinGatePeriod(long Time);

This command adds a configuration event to the command list. The minimal gate period is used to limit output frequency when pulse commands (*rtPulse, rtPuls3D*) are processed. After execution, the period between pulse outputs from said functions will not be smaller than the declared time.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Time : 0...65535 µsec

rtSetOffsIndex(long Index);

rtSetOffsXY(double X, double Y);

This function adds an offset change event in the command list. All output coordinates are shifted by an offset vector. When executed, this command will set this shift to the declared values.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
X,Y : image offset position, range -8388.608...8388.607 mm
description:
This method sets the absolute XY-offset vector.

rtSetOffsZ(double Z);

This function adds a Z-offset change event in the command list. All output coordinates are Z-shifted by a Z-offset vector (*OffsZ*). When executed, this command will set this shift to the declared value.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Z : image offset position, range -8388.608...8388.607 mm

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

This function adds a configuration event to the command list. Combined with controlling connected deflectors, the target controller also controls the laser. Several steering modes are available including Gated, Q-switched, PWM-CO2 and speed modulated Q-switching. Configuration is done over three programmable oscillators which can be set up by this function.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Nr =1: Function sets period and pulse width CO2 activated
Nr =2: Function sets pulse width CO2 idle (period must be the same as the Nr 1 setting)
Nr =3: Function sets period and pulse width Burst mode
Period : range 0...819.18 µsec
PulseWidth : range 0...Period

rtSetOTF(long Nr, bool On);

This command adds a configuration event to the command list. Its execution switches the on-the-fly offsetting on or off. When activated, the on-the-fly counter value is substracted from the graphical coordinates. The resulting difference is send to the deflector. The on-the-fly counter is not controlled by this function and remains activated.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Nr = 1: function applies to X axis
Nr = 2: function applies to Y axis
On : 1 for on, 0 for off

rtSetPulseBulge(double Factor);

rtSetQueryTarget(long Index);

This function sets the query target index. A single CUA32 device can comprise up to 8 target controllers, each having 17 IO's and up to three connected rhothor deflectors. When IO's are polled and target status is queried, the query index is used to select the target of interest. At default, the query index is set to 1. This function takes 50 milliseconds to execute.
* applicable when connected
* parameters
Index : 1…8

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

This mode adds an on-the-fly-configuration event to the command list. Using the CUA32 controller, on the fly marking is easily realized. The target controller uses a A/B resolver input scheme connected to an on-the-fly-offset counter. This counter in altered with step size whenever a flank is detected. The direction is determined by phase shift. The minimal time between flank changes is 1 µsec. When this command is executed, the step size is defined, and the offset counter cleared. The range parameter defines the on-the-fly counters count range. When set to zero, the on-the-fly counter will use its entire count range.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Nr = 1: function applies to X axis
Nr = 2: function applies to Y axis
StepSize : mm
Range : 0... 8388.607 mm

rtSetResolverPosition(long Nr, double Position);

This function adds an on-the-fly-data change event to the command list. When executed, the target's on-the-fly-offset counter is loaded with the declared value.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Nr = 1: function applies to X axis
Nr = 2: function applies to Y axis

© Copyright Newson NV, 2000-2017

Position : -8388.607... 8388.607 mm

rtSetResolverRange(long Nr, double Range);

This function adds an on-the-fly-data change event to the command list. When executed, the range of the target's on-the-fly-offset counter is set to the declared value.
Nr = 1: function applies to X axis
Nr = 2: function applies to Y axis
Range : -8388.607... 8388.607 mm

rtSetResolverTrigger(long Nr, double Position, long IO);

rtSetRotation(double Angle);

This function adds a transformation matrix change event to the command list. When executed the target's transformation matrix (*Aij*) will be loaded with a rotation function. The function is the same as *rtSetMatrix(cos(Angle),-sin(Angle),sin(Angle),cos(Angle))*
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Angle : rotation angle in radians

rtSetSpeed(double Speed);

This function adds a speed change event to the command list. During execution of marking functions (rtLineTo, rtMoveTo, rtArcTo...) the deflectors setpoints are moved at marking speed. When executed, this function sets the target's marking speed to the declared value.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Speed : speed in mm/s

rtSetTable(long Nr, double Position);

This function adds a table-data-event to the command list. Besides controlling deflectors and laser, the target is also able to control stepper motors. The target controller comprises three stepper position regulators to provide the needed logic. When this command is executed, the controller doesn't move the connected stepper but loads it's setpoint and actual position with the declared value. This function could be used after calling rtSetTableWhileIO as a part of the axis reference cycle.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Nr =1: stepper X axis
Nr =2: stepper Y axis
Nr =3: stepper Z-axis
Position : range -8388.608...8388.607 mm

rtSetTableDelay(long Nr, long Delay);

The table logic operates under direct (sperate mode) or implicit control (hybrid mode). The stepper's actual position counter is regulated to align with the obtained setpoint. When correctly set, this regulation guarantees that the outgoing pulse signals can be handled by the actual stepper motor. To limit pulse frequency changes, the logic comprises a low pass filter. When executed, this function sets it's time constant. An easy way to determine the value to be set is using a trial and error approach. Choose a delay time which feels right and invoke a table move command at the desired speed. Make sure that the movement is long enough so the full speed could be reached. When executed correctly, decrease the delay time and repeat the trial. When the first trial already faulted increase the delay. The maximal frequency that can be generated by the CUA32 controller is 100 KHz. However, limited pull up torque of the stepper motors will reduce this frequency. The application should make sure that no table movements are issued when the speed (*rtSetSpeed*) is set too high.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Nr =1: stepper X axis
Nr =2: stepper Y axis
Nr =3: stepper Z-axis
Delay : range 0...2147483.647 µsec

rtSetTableSnapSize(long Nr, double SnapSize);

This function adds a configuration event to the command list. When executed, this function activates hybrid axis mode. The stepper channel's setpoint is controlled by graphical coordinates. Graphical functions (*rtArcTo, rtLineTo...*) generate those positions which are normally executed by the deflectors. When called with zero as snap size, hybrid marking is deactivated. When hybrid mode is activated using direct table functions like "rtTableMove" should be avoided. When table movements are needed, switch of the hybrid mode before issue table commands.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Nr =1: stepper X axis
Nr =2: stepper Y axis
Nr =3: stepper Z-axis
SnapSize : range 0...... 8388.607 mm

rtSetTableStepSize(long Nr, double StepSize);

This function adds a table configuration event to the command list. When executed the stepper's position increment for every outgoing step is defined. Steppers are controlled by CUA32 systems using a direction and a step signal. The direction signal specifies the movement's orientation while the frequency and number of outgoing pulses determine speed and distance of the travel. Both signals are controlled by internal logic. This function activates this logic and should be called prior to any other table commands. When called with step size set to zero, the logic is idled.
To calculate step size:
Step size = speed / step count
speed: the spindle's speed in mm/rotation
step count: the stepper motor's step count in #steps/rotation
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode

* parameters:
Nr =1: stepper X axis
Nr =2: stepper Y axis
Nr =3: stepper Z-axis
StepSize : mm/step

rtSetTableWhileIO(long Value, long Mask);

This function adds a table configuration event to the command list. The stepper regulators comprise a safety system to avoid jamming the driven stage against mechanical limitations. Any input can be chosen to provide a stop when a table axis is moving outside his working area. The input is shared between all three regulators.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Value : 0...65535
Mask : 0...65535
example:
Assume a multiple axis system each fitted with normal open limit switches at their travel edges. All limit switches are mounted in parallel and connected between Vio and IO16
rtSetTableWhileIO(0x0000,0x8000); // input must be low to be save
example:
Assume a multiple axis system each fitted with normal closed limit switches at their travel edges. All limit switches are mounted in series and connected between Vio and IO14
rtSetTableWhileIO(0x2000,0x2000); // input must be high to be save

rtSetTarget(long Mask);

A complete CUA32 system can comprise up to 8 target controllers. Commands can be send to one or several target controllers at the same time to provide synchronized or independent control of all connected deflection systems. Activation of a target controller is done by simply setting its mask bit. When single target functions are called, the least significant activated bit is used to define the function's target.
* parameters:
Mask : 0...255
* example
rtSetTarget(0b01000001); single target functions will apply to target 1 while masked target functions will apply to targets 1 and 7

rtSetVarBlock(long i, char data);

rtSetWobble(double Diam, long Freq);

This function adds a wobble configuration to the command list. Increasing the laser processing width can be achieved by adding a circular movement. This process is called wobble. Period and frequency are set when this function is executed. To minimize deflector movements, the wobble function is only activated when the gate signal is on.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:

© Copyright Newson NV, 2000-2017

Diam : mm
Freq : 100...4000 Hz
* example:
rtSetWobble(0.1,1000); // diameter 0.1 mm, spinning at 1KHz

rtSetWobbleEx(long nType, double nAmpl, long nFreq, long tType, double tAmpl, long tHarm, long tPhase);

This function adds a wobble configuration to the command list. Instead of simple circular wobbling, this function has separate configuration of the normal and the tangential wobble component. The signal added orthogonal to the core movement is designated normal. The wobble part that runs in the same direction is designated the tangent part.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
nType =0: normal wobble disabled
nType =1: normal sine wave
nAmpl : normal amplitude, mm
nFreq : normal frequency 100...4000 Hz
tType =0: tangent component disabled
tType =1: tangent cosine wave
tAmpl : tangent amplitude, mm
tHarm : tangent frequency = tHarm*normal frequency
tPhase : -180...180 degrees
example:
rtSetWobbleEx(1,0.1,1000,1,0.1,1,0); // same as calling rtSetWobble(0.1,1000);

rtSleep(long Time);

This command adds a sleep event to the command list. When the command is executed, the target controller suspends and starts a timer. The command processing is resumed when the timer elapses. The command rtSleep is often used to implement a wait-after-jump feature.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* parameters:
Time : 0...2147483647 µsec

rtStoreCalibrationFile(const char FileName);*

This function copies the target's calibration in a readable file on the host.
* applies to least significant masked target (*rtSetTarget*)
* applicable when system is idle
* parameters
FileName : file name under which the calibration will be saved.

rtSuspend();

This function adds a suspend command to the command list. When processed, the targets will go in suspended state. *rtSuspend* is a list command, all pending list commands will be executed. A suspended system can be restarted by calling *rtSystemResume*;
* applies to all masked targets (*rtSetTarget*)

* applicable in streaming and compile mode

rtSystemResume();
Restarts command processing in case the system was suspended.
* applies to all targets
* applicable when connected

rtSystemSetIO(long Value, long Mask);
This command sets the digital output values
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….
* applies to query target (*rtSetQueryTarget*)
* applicable when connected
* parameters:
Value, Mask : range 0…65535

rtSystemSuspend();
Immediately suspend all command processing. The target controllers will stop fetching instructions from their FIFO's. The instructions that are being processed during this call will complete normally.
* applies to all targets
* applicable when connected

rtSystemUartOpen(long baudrate, char parity, char stopbits);
This function opens the serial interface.
* applicable when connected
* parameters:
baudrate : 1200...115600 bit/sec
parity : 'n','e','o','s' for none, even, odd, or space
stopbits : 1 or 2

rtSystemUartWrite(long bytes, char data);*
This function copies data on the serial port. Data is copied as is, no zero byte is appended.
* applicable when connected
* parameters:
bytes : number of bytes to be transmitted, range 1 to 511
data : pointer to the data that must be send

rtSystemUDPsend(char IP, short port, char* data);*
This method triggers an internet communication. The content is send to the IP address as is and may contain zero bytes. The first element of *data* holds the number of bytes to be transmitted. IP address can be obtained by calling *rtGetIP*.
* applicable when connected
* parameters
IP : target IP address
port : target port

data[0] : length of data to be send, 0...255 bytes
&data[1]: pointer to data to be send.

rtTableArcTo(double X, double Y, double BF);

This command adds an arc style marking to the command list. The arc starts at the current position and goes to coordinate (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. Marking speed (rtSetSpeed) is used during execution. During execution the XY table is used and not the deflection system. When both systems are used independently to mark lines, the application should make sure that the deflectors are in a known position whenever markings are made using the stepper controls.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* abortable (*rtAbort*)
* parameters:
X,Y : target position, range -8388.608...8388.607 mm

rtTableJog(long Nr, double Speed, long WhileIO);

This function adds a table move event to the command list. When executed, the table setpoint will be moved using the requested speed. The function aborts when the designated input (*WhileIO*) becomes low. This command ignores the safety settings done with rtSetTableWhileIO. Calling *rtAbort* will also force function exit. The *rtTableJog* function can be used to implement referencing or jog features. *WhileIO* input can be used as a referencing switch while the command *rtAbort* could be used as a jog stop.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* abortable (*rtAbort*)
* parameters:
Nr =1: stepper X axis
Nr =2: stepper Y axis
Nr =3: stepper Z-axis
Speed : mm/s
WhileIO : 1 for IO1, 2 for IO2, ... ,16 for IO16

rtTableLineTo(double X, double Y);

This command adds a line marking to the command list. The line starts at current table position and extends towards the target position. Marking speed (*rtSetSpeed*) is used during execution. During execution the XY table is used and not the deflection system. When both systems are needed independently to mark lines, the application should make sure that the deflectors are in a known position whenever markings are made using the stepper controls.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* abortable (*rtAbort*)
* parameters:
X,Y : target position, range -8388.608...8388.607 mm

rtTableMove(long Nr, double Target);

This function adds a table move event to the command list. When executed, the stepper's setpoint will move towards the desired position. Marking speed (*rtSetSpeed*) is used and laser is idled during execution.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* abortable (*rtAbort*)
* parameters:
Nr=1: stepper X axis
Nr=2: stepper Y axis
Nr=3: stepper Z-axis
Target: range -8388.608...8388.607 mm

rtTableMoveTo(double X, double Y);

This command adds a table move to the command list. The move starts at current table position and extends towards the target position. Marking speed (*rtSetSpeed*) is used and laser is idled during execution.
* applies to all masked targets (*rtSetTarget*)
* applicable in streaming and compile mode
* abortable (*rtAbort*)
* parameters:
X,Y: target position, range -8388.608...8388.607 mm

rtUartRead(long bytes, char* data);*

This function reads received data from serial interface. The content is stored in location pointed by data. The number of received bytes is stored in parameter bytes. At most, the function returns 511 bytes, so the parameter data should point to a 512-byte array.
* applicable when connected
* parameters:
bytes: placeholder for the number of received bytes
data: 512-byte placeholder for the received data.

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

rtWaitIO(long Value, long Mask);

This function adds a control flow command to the command list. Processing a *rtWaitIO* command starts by waiting until all connected targets are idle. Afterwards the IO state of the least significant target is tested against the declared value. The comparison only considers those IO with their corresponding mask bit set to one. When the result returns
equal, command execution is resumed.
* applies to all targets
* applicable in streaming and compile mode
* abortable (*rtAbort*)
* parameters:
Value: 0...65535
Mask: 0...65535

rtWaitPosition(double Window);

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

This function adds a local control flow command to the command list. During processing, the target controller suspends command processing until the selected on-the-fly counter has reached its desired value.

* applies to all targets
* applicable in streaming and compile mode
* parameters:

Nr =1: on-the-fly counter X axis
Nr =2: on-the-fly counter Y axis
TriggerPos : trigger position, range -8388.608...8388.607 mm
TriggerMode = 1: wait until on-the-fly counter is higher than TriggerPos
TriggerMode = 2: wait until on-the-fly counter is lower than TriggerPos

rtWhileIO(long Value, long Mask);

This function adds a control flow command to the command list. Processing a *rtWhileIO* command starts by waiting until all connected targets are idle. Afterwards the IO state of the least significant target is tested against the declared value. The comparison only considers those IO with their corresponding mask bit set to one. When the result returns equal, the instruction sequencing is continued. When the comparison returns false, the instruction sequencer fetches the first command after rtDoWhile. This function needs calculated instruction locations and therefore cannot be used in streaming mode. The front-end controller of the CUA32 device has a 16-level deep stack to provide nesting of control flow commands.

* applies to all targets
* applicable in compile mode
* parameters:

Value : 0...65535
Mask : 0...65535

Not supported functions, kept for compatibility.

bcSamplePoint(double X, double Y, long Row, long Col, double Sweep, double OffsetX, double* OffsetY);*
bcSelectDevice(const char CommPort);*
rtAddCalibrationDataZ(const char FileName);*
rtGetCanLink(long Address, long Value);*
rtLoadCalibration();
rtLoadCalibrationFileZ(const char FileName);*
rtResetCalibrationZ();
rtScanCanLink(long Address, long Node, long Index, long SubIndex);
rtSendUartLink(const char Data);*
rtSetCanLink(long Node, long Index, long SubIndex, const char Data);*
rtSetHover(long Time);
rtSetLead(long Time);
rtSetMaxSpeed(double Speed);
rtSetTableOffsXY(double X, double Y);
rtSetTableSnap(double Distance);

rtStoreCalibration();
rtStoreCalibrationFileZ(const char FileName);*
rtWaitCanLink(long Address, long Value, long Mask);
rtWaitEventCounter(long Count);

return codes library functions

ERR_OK (-1)
The function call completed successfully.

ERR_BUSY (2)
The application tried to invoke a function which requires an idle system on a non-idle system. To solve, make sure that the command list is closed (*rtListClose, rtFileClose or rtAbort*) and try again.

ERR_JOB (3)
The application invoked a command list function without having the queue properly opened. To solve, make sure that the command list is ready to receive commands (*rtListOpen, rtFileOpen*) and try again. Open the command list in compile mode (*rtListOpen(4)*) when control flow commands are needed.

ERR_HARDWARE (5)
Failed to set up or maintain physical connection with the CUA32 hardware. Verify IP-address, USB ID string and electrical connection with host and try again.

ERR_DATA (13)
The application invoked a function with invalid parameters. Verify if the function parameters are within the specification and ranges as described in this manual.

ERR_IMPLEMENTATION (23)
The function is not supported by the library. Function entry is kept for compatibility reasons with previous library versions.

© Copyright Newson NV, 2000-2017

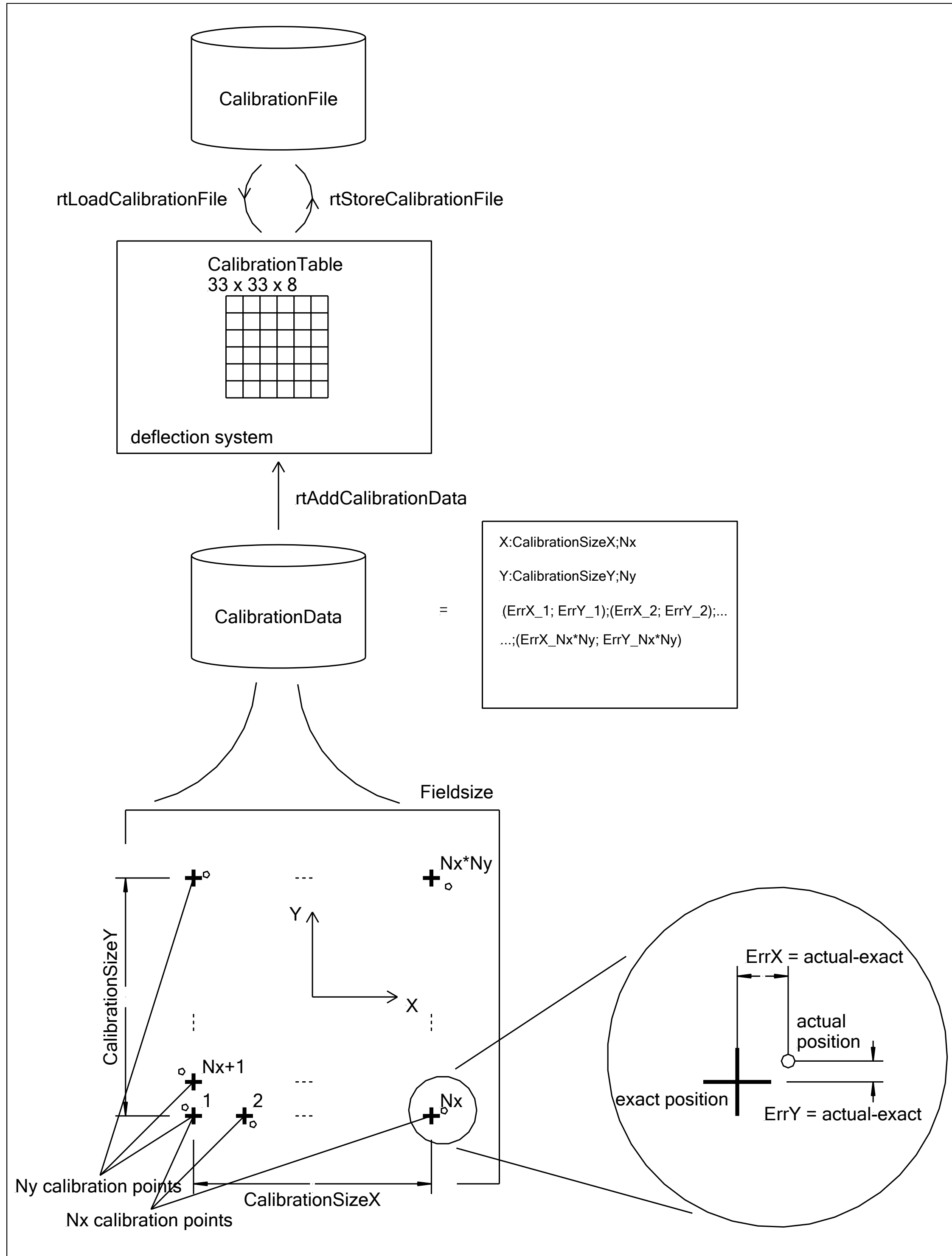
calibration

Deflection systems suffer from positional errors. The CUA32 systems handle these errors by counter steering during the movement of the laser beam.

Marking 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 must 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 and 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.

If a 3D-deflection system is being used without a flat field f-theta lens an obvious additional error source is the shape of the focal area. This area is spherical instead of flat. Furthermore, without the use of tele centric optics, the field size will change with height. A calibration on 2 Z-positions will be needed.

© Copyright Newson NV, 2000-2017


CalibrationFile
rtLoadCalibrationFile
rtStoreCalibrationFile
CalibrationTable
33 x 33 x 8
deflection system
rtAddCalibrationData
CalibrationData
=
X:CalibrationSizeX;Nx
Y:CalibrationSizeY;Ny
(ErrX_1; ErrY_1);(ErrX_2; ErrY_2);...
...;(ErrX_Nx*Ny; ErrY_Nx*Ny)
Fieldsize
Nx*Ny
Y
X
CalibrationSizeY
Nx+1
1
2
Nx
Ny calibration points
Nx calibration points
CalibrationSizeX
ErrX = actual-exact
actual
position
exact position
ErrY = actual-exact

3D-calibration