RAYLASE
focus on laser

Commands and Functions

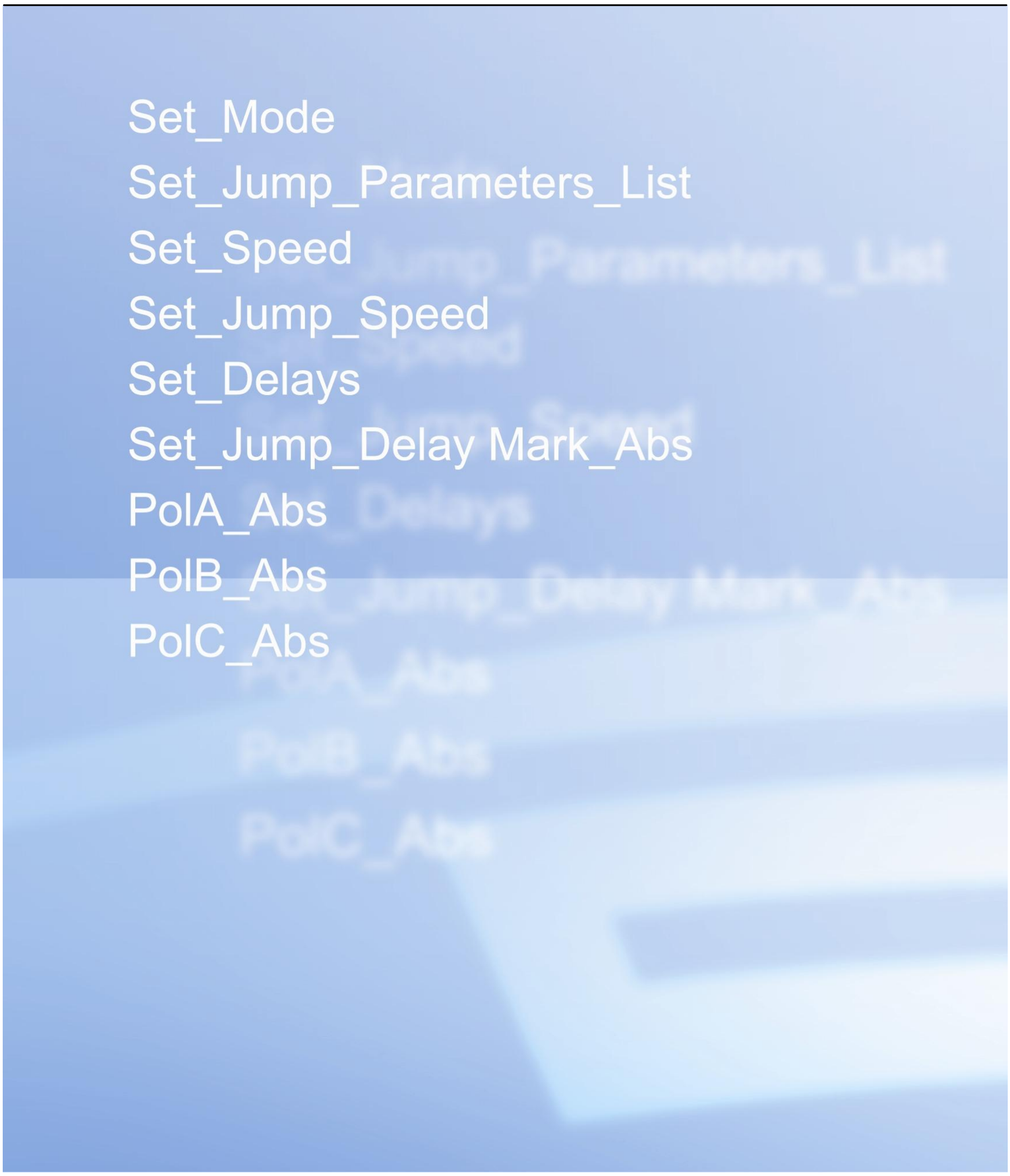

TABLE OF CONTENTS

1 INTRODUCTION

The purpose of this manual is to describe the software commands for those customers who wish to write their own software at DLL level.

It must be emphasized that this is a complex task needing many man hours of work for anything but the simpliest of applications. RAYLASE has graphic based software packages for immediate use – furthermore, a Marker Library is available which can be used to create industrial control programs or GUIs for customers wanting their own look and feel, but who do not want to program at the DLL level.

Note: The RAYLASE DLL drivers can be opened by only one application at a time.

1.1 Overview

There are three basic types of commands: Control commands, list commands and error commands:

Control Commands are mainly used to set up the board’s main functions and start major actions immediately. They operate asynchronously and are usually sent when the lists are not being filled or executed. Exceptions are commands such as Stop_Execution, Read_Status.

List Commands are stored in the so called lists for later execution. Once execution is initiated, they are processed and output synchronously to allow accurate control of the galvanometer scanners fully synchronized to the laser control.

Error Commands are used for error handling purposes, checking if errors occurred, reading error codes and messages. Similar to control commands, error commands are also asynchronous and can be sent any time during the application.

1.2 Visual Basic Compatibility

The Visual Basic Boolean data type is represented in a fundamentally different way from the Visual C++ bool.

This makes it difficult for VB programs to correctly interpret the return values from many of the command functions which are declared “As Boolean” in the following sections.

Consequently, programmers should use the following general scheme for checking the return values from the command functions:

```
Dim Result As Integer
Result = Set_Start_List_1        ’for example
If Result <> 0
Then                             ’call succeeded
Else                             ’call failed
End If
```

Note in particular that the following test will incorrectly appear to fail, even when the function itself succeeds!

```
If True = Set_Start_List_1       ’for example
Then                             ’call succeeded
Else                             ’call failed
End If
```

1.3 User Application Program

The structure of a User Application Program will typically be as follows:

- Initialization
- Filling of lists
- Execution of lists
- Closing the Application

Initialization

Init_Scan_Card	// Bringing the card into an initial state
Set_Mode	// Defines the scanner mode
Load_Cor	// Loads a correction file for the scan head
Set_Gain	// Sets up the fine adjustment of the field size

These control commands should be put at the start of the application program. They will initialize the card and set up the mode of operation to suit a particular laser type, field orientation and optical characteristics of the lens.

Send the *Set_Mode* command at the beginning, since many other commands are interpreted depending on this.

Filling of lists

Next step would then be to prepare list(s) for execution. First select and open the list for storing list commands with:

- *Set_Start_List_1* or *Set_Start_List_2*

Then fill the list with list commands in the following order:

- *Set_Delays*, *Set_Jump_Parameters_List*, *Set_Mark_Parameters_List* are typically issued first to set up the parameters for the following marking objects.
- Marking vectors for the object; for instance, Jump, Mark, PolA, PolB, PolC, etc.
- Other list commands …
- *Set_End_Of_List* as the last command will close the list.

Execution of lists

In this way the list is made ready for execution, which can then be started by sending control command *Execute_List_1* or *Execute_List_2*. It can also be done by *Start_Loop.*

Closing the Application

Execution of the list commands continues automatically until all commands have been executed. Or, if started by *Start_Loop*, until *Quit_Loop* is issued.

Before closing the application it is strongly advised to send control command *Remove_Scan_Card.* It will put the card in a stand-by state and assure its proper functioning in the next application run.

1.4 Lists

List commands, sent from the application software, which define the marking contour are first saved in list buffers on the PC. There are two list buffers, list 1 and list 2 provided for this purpose. Each list has an initial size to accommodate 500,000 list commands and will expand if more list commands are sent.

Load List

Storing data in the lists and processing/execution of the lists can be controlled by a set of control and list commands. Before sending any list commands to the lists, one of the two lists must be opened. It is done by the control commands *Set_Start_List_1* or *Set_Start_List_2*. Only one list can be opened at a time. Opening a list will discard any list commands sent to it previously and it will disable execution of that list until it is closed.

Once a list is opened, any number of list commands can be sent to it in any order.

Close List

After sending all the commands, a list must be closed in order to allow it to be executed. It is done by the list command *Set_End_Of_List*, which is stored as the last command in the list. After this command no more list commands can be stored in it.

Execute List

Once the list is loaded and closed, it can be executed by sending one of the control commands *Execute_List_1* or *Execute_List_2*.

After starting the execution, the list is defined to be in a “busy” state. The real time system executes each command until the list command *Set_End_Of_List* is reached. The “busy” state exists, therefore, until all list commands have been executed. The state of the lists can be checked using the command *Read_Status*.
After that, a list can be restarted with another *Execute_List* command.

While one list is being executed, the second list can be loaded with list commands.
The newly loaded and closed list can be started only after the other list has finished. This can also be automated using the command *Aut_Change*. The *Aut_Change* command allows continuous consecutive execution and filling of lists – commonly called “pipelining”. This method allows a rapid start of execution without having to wait for a large list to be filled.

Stop List

Issuing a *Stop_Execution* control command during execution immediately stops the execution and turns off the laser, if necessary in the middle of a vector. Both lists are deleted and must be loaded again before restarting execution.

Another control command, a *Stop_Execution_No_Clear* can be used instead, causing the same effect except that the lists are not cleared after stopping and can be restarted without refilling.

Only start from the beginning of a list is possible.

Loop Lists

Continuous output is useful when working with a pointer or during testing. A pair of control commands, *Start_Loop* and *Quit_Loop,* can be used for continuous execution of the two lists. Before sending a *Start_Loop* command both lists have to be loaded and closed. A *Quit_Loop* command will stop the execution after the last command of the active list. The loop can be restarted by another *Start_Loop* command. It always starts with list 1, regardless of which list was executed last. The number of list starts in a loop can be defined with *Set_Max_Counts* control command. If no value is set, a default value of 0 is assumed, causing the loop to run indefinitely.

Continuous output of lists can also be achieved with *Loop_To_Start_List_1* or *Loop_To_Start_List_2* list commands. They can be placed anywhere in any of the two lists, causing execution of the list stated in the command to start after the last list command in the current list. So, various combinations can be achieved, looping only one list or executing one list once and then looping only through the second, etc. These commands are also affected by the *Set_Max_Counts* command.

External Synchronization

There are two external TTL control inputs (see Hardware Manual), which can be used for external synchronization of list execution: START_MARK and STOP_MARK.

The START_MARK signal can be polled using *Read_Port*. An *Execute_List_1* or *Execute_List_2* can be issued as soon as the signal goes true. On some controllers, this process can be automated using the list command *Wait_For_External_Start*.

The external signal STOP_MARK has a direct effect on list execution, stopping it immediately it is asserted, even if in the middle of the vector. Lists are not deleted after the STOP_MARK signal has been asserted and can be restarted afterwards. The STOP_MARK signal is level sensitive. It must be de-asserted in order to be able to proceed with list execution.

2 LIST COMMANDS

The list commands described below are listed in alphabetical order.

Jump_Abs

☑ SP-ICE
☑ RLC

Function	Fast movement, jump of the beam to the specified coordinate.	
Parameter	Coordinates represent end of the vector, as 16 bits signed integer numbers.	
Result	Function Jump_Abs ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Jump_Abs (xval, yval: smallint): bool;
	C	bool Jump_Abs (short xval, short yval);
	Basic	function Jump_Abs (byval xval%, byval yval%) as boolean
Hints	Laser is kept switched off during the execution of command.	
	A jump delay is issued at the end of the jump command.	
	Jump speed and delay should be specified by *Set_Jump_Speed* and *Set_Jump_Delay*, prior to issuing a jump command. If not, default values are used.	
References	*Set_Jump_Parameters_List, Set_Speed, Set_Jump_Speed, Set_Delays, Set_Jump_Delay Mark_Abs, PolA_Abs, PolB_Abs, PolCAbs, Jump_Rel*	

Jump_Rel

☑ SP-ICE
☑ RLC

Function	The same functionality as *Jump_Abs* except that the end position is given relative to the current position.	
Parameter	Coordinates of the end of a vector, as 16 bits signed integer offset from the current position.	
Result	Function Jump_Rel ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Jump_Rel (xval, yval: smallint): bool;
	C	bool Jump_ Rel (short xval, short yval);
	Basic	function Jump_ Rel (byval xval%, byval yval%) as boolean
Hints	The same hints as for *Jump_Abs* apply.	
References	*Jump_Abs*	

Laser_Off

☑ SP-ICE
☑ RLC

Function	Laser is switched off for the defined period of time.	
Parameter	Laser off duration, period of time t in [µs], as 16 bits unsigned integer. 1 ≤ t ≤ 65535 A special case is t = 0; the laser is switched off indefinitely, i.e. the laser will stay off until switched on by another command.	
Result	Function Laser_Off ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Laser_Off (t: word): bool;
	C	bool Laser_Off (unsigned short t);
	Basic	function Laser_Off (byval t%) as boolean
Hints	It might be helpful if you read the description of *Laser_On* list command, first.	
References	*Laser_On, Long_Delay*	

Laser_On

☑ SP-ICE
☑ RLC

Function	Laser is switched on and then, after a defined period of time, switched off again.	
Parameter	Laser on duration, period of time t in [µs], as 16 bits unsigned integer. 1 ≤ t ≤ 65534 A special case is t = 0; the laser is switched on indefinitely, i.e. the laser will stay on until switched off by another command.	
Result	Function Laser_On ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Laser_On (t: word): bool;
	C	bool Laser_On (unsigned short t);
	Basic	function Laser_On (byval t%) as boolean
Hints	Beam is normally not moved during the execution of this command. This command can be used for point-and-shoot applications in drilling and grey scale applications.	
	For very long time periods, use a combination of *Laser_On*, *Long_Delay*, *Long_Delay*, … and *Laser_Off* commands. Note, however, that the time resolution differs in these commands.	
	Actual laser parameters will be used.	
References	*Laser_Off, Long_Delay*	

Long_Delay

☑ SP-ICE
☑ RLC

Function	Execution of list will be paused for a defined period of time (t).	
Parameter	Time delay in [10µs] units, as 16 bits unsigned integer. 1 ≤ t ≤ 65535 corresponding to 10µs to 655,350µs (0.655350 seconds)	
Result	Function Long_Delay ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	Pascal: function Long_Delay (t: word): bool;
	C	bool Long_Delay (unsigned short t);
	Basic	function Long_Delay (byval t%) as boolean
Hints	This command can be used after changing of diode or lamp current with YAG-lasers in order to get a constant laser power as well as for very long drill periods. Several *Long_Delay* commands may be inserted in a list to create even longer delays.	
References	*Laser_On, Laser_Off*	

Loop_To_Start_List

☑ SP-ICE
☑ RLC

Function	Execution of list commands will continue at the start of the specified list. This can be the same list or another one allowing continuous output of one or two lists.	
Parameter	List number (n) either 1 or 2 as a 16 bits unsigned integer. 1 ≤ n ≤ 2	
Result	Function Loop_To_Start_List ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Loop_To_Start_List (n: word): bool;
	C	bool Loop_To_Start_List (unsigned short n);
	Basic	function Loop_To_Start_List (byval n%) as boolean.
Hints	This list command sets the stated list as the next list to be executed.	
	This command can be issued anywhere in the list, but execution will proceed with the current list until all the commands are done and then proceed with the next list.	
	This command is affected by the *Set_Max_Counts* value defining the maximum number of list starts in the same way as *Start_Loop* command.	
References	*Wait_For_External_Start, Set_Max_Counts, Start_Loop*	

Mark_Abs

☑ SP-ICE
☑ RLC

Function		Marking of a straight line from actual position to defined coordinate.
Parameter		Coordinates representing the end of a vector, as 16 bits signed integer.
Result		Function Mark_Abs ok (TRUE) or not ok (FALSE) as boolean.
Calling conventions	Pascal	function Mark_Abs (xval, yval: smallint): bool;
	C	bool Mark_Abs (short xval, short yval);
	Basic	function Mark_Abs (byval xval%, byval yval%) as boolean
Hints		Marking speed should be defined with *Set_Speed* or *Set_Mark_Speed* prior to issuing this command. If not, default values are used.
		Before marking, laser is switched on after a laser on delay and then, when the end of the vector is reached, switched off after a laser off delay.
		Also, at the end of command, a mark delay is inserted.
References		*Set_Speed, Set_Mark_Parameters_List, Set_Mark_Speed, Set_Delays, Set_Mark_Delay, Mark_Rel, Jump_Abs, PolA_Abs, PolB_Abs, PolC_Abs*

Mark_Immediately

SP-ICE with
☑ MOTF-Option
☐ RLC

Function		Restarts Mark-on-the-Fly sequence.
Result		Function Mark_Immediately ok (TRUE) or not ok (FALSE) as Boolean.
Calling conventions	Pascal	function Mark_Immediately (): bool;
	C	bool Mark_Immediately ();
	Basic	function Mark_Immediately () as boolean
Hints		This command resets the encoder counter to 0, enabling proper synchronisation, and proceeds to next list command without any delay.
		Can be used even if Mark-on-the-Fly is not used, in which case the command will just fall through.

Mark_Rel

☑ SP-ICE
☑ RLC

Function		The same functionality as *Mark_Abs* except that the end position is given relative to the current position.
Parameter		Coordinates of the end of a vector, as 16 bits signed integer offset from the current position.
Result		Function Mark_Rel ok (TRUE) or not ok (FALSE) as boolean.
Calling conventions	Pascal	function Mark_Rel (xval, yval: smallint): bool;
	C	bool Mark_ Rel (short xval, short yval);
	Basic	function Mark_ Rel (byval xval%, byval yval%) as boolean
Hints		The same hints as for *Mark_Abs* apply.
References		*Mark_Abs*

PolA_Abs

☑ SP-ICE
☑ RLC

Function	Marking of a straight line from actual position to defined coordinate.	
Parameter	Coordinates representing the end of a vector, as 16 bits signed integer.	
Result	Function PolA_Abs ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function PolA_Abs (xval, yval: smallint): bool;
	C	bool PolA_Abs (short xval, short yval);
	Basic	function PolA_Abs (byval xval%, byval yval%) as boolean
Hints	*PolA_Abs* represents the first vector of a polygon stroke *PolA* - *PolB* ... *PolB* - *PolC*.	
	Marking speed should be defined with *Set_Speed* or *Set_Mark_Parameters_List* prior to executing a command. If not, default or previously defined values are used.	
	Laser will be switched on after laser on delay and remains switched on after reaching the end of the vector.	
	After a *PolA* command a polygon delay will be inserted.	
References	*Set_Speed, Set_Mark_Parameters_List, Set_Delays, Jump_Abs, PolA_Rel, Mark_Abs, PolB_Abs, PolC_Abs*	

PolA_Rel

☑ SP-ICE
☑ RLC

Function	The same functionality as *PolA_Abs* except that the end position is given relative to the current position.	
Parameter	Coordinates of the end of a vector, as 16 bits signed integer offset from the current position.	
Result	Function PolA_Rel ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function PolA_Rel (xval, yval: smallint): bool;
	C	bool PolA_ Rel (short xval, short yval);
	Basic	function PolA_ Rel (byval xval%, byval yval%) as boolean
Hints	The same hints as for *PolA_Abs* apply.	
References	*PolA_Abs*	

PolB_Abs

☑ SP-ICE
☑ RLC

Function	Marking of a straight line from actual position to defined coordinate.	
Parameter	Coordinates representing the end of a vector, as 16 bits signed integer.	
Result	Function PolB_Abs ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function PolB_Abs (xval, yval: smallint): bool;
	C	bool PolB_Abs (short xval, short yval);
	Basic	function PolB_Abs (byval xval%, byval yval%) as boolean
Hints	Marking speed should be defined with *Set_Speed* or *Set_Mark_Parameters_List* prior to executing a command. If not, default or previously defined values are used.	
	Laser remains switched on after reaching the end of the vector.	
	After a *PolB* command a polygon delay will be inserted.	
References	*Set_Speed, Set_Mark_Parameters_List, Set_Delays, Jump_Abs, PolB_Rel, Mark_Abs, PolA_Abs, PolC_Abs*	

PolB_Rel

☑ SP-ICE
☑ RLC

Function	The same functionality as *PolB_Abs* except that the end position is given relative to the current position.	
Parameter	Coordinates of the end of a vector, as 16 bits signed integer offset from the current position.	
Result	Function PolB_Rel ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function PolB_Rel (xval, yval: smallint): bool;
	C	bool PolB_ Rel (short xval, short yval);
	Basic	function PolB_ Rel (byval xval%, byval yval%) as boolean
Hints	The same hints as for *PolB_Abs* apply.	
References	*PolB_Abs*	

PolC_Abs

☑ SP-ICE
☑ RLC

Function	Marking of a straight line from actual position to defined coordinate.	
Parameter	Coordinates representing the end of a vector, as 16 bits signed integer.	
Result	Function PolC_Abs ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function PolC_Abs (xval, yval: smallint): bool;
	C	bool PolC_Abs (short xval, short yval);
	Basic	function PolC_Abs (byval xval%, byval yval%) as boolean
Hints	*PolC_Abs* represents the last vector of a polygon stroke *PolA* - *PolB*...*PolB* - *PolC*.	
	Marking speed should be defined with *Set_Speed* or *Set_Mark_Parameters_List* prior to executing a command. If not, default or previously defined values are used.	
	Laser will be switched off with laser off delay after reaching the end of the vector.	
	After a *PolC* command a mark delay will be inserted.	
References	*Set_Speed, Set_Mark_Parameters_List, Set_Delays, Jump_Abs, PolC_Rel, Mark_Abs, PolA_Abs, PolB_Abs*	

PolC_Rel

☑ SP-ICE
☑ RLC

Function	The same functionality as *PolC_Abs* except that the end position is given relative to the current position.	
Parameter	Coordinates of the end of a vector, as 16 bits signed integer offset from the current position.	
Result	Function PolC_Rel ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function PolC_Rel (xval, yval: smallint): bool;
	C	bool PolC_ Rel (short xval, short yval);
	Basic	function PolC_ Rel (byval xval%, byval yval%) as boolean
Hints	The same hints as for *PolC_Abs* apply.	
References	*PolC_Abs*	

Put_Bitmapline_List

☑ SP-ICE
☑ RLC

Function	This command is used to mark one line of a complete bitmap. The command contains all the information required to position and execute a line of grey values along a line.	
Parameter	Coordinates representing the start and end points of the line, the grey values as an array of 16 bit integers and the length of the array.	
Result	Function Put_Bitmapline_List ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Put_Bitmapline_List (xstartval, ystartval, xendval, ysendval: smallint, pArrayGreyValues [0]:IntArray, ArrayLen: smallint): bool;
	C	bool Put_Bitmapline_List (short xstartval, short ystartval, short xendval, short yendval,&arrayGreyValues[0],short num_of_values);
	Basic	function Put_Bitmapline_List (byval xbeginval%, byval beginyval%, byval xendval%, byval endyval%, IntArray, byval arraysize%) as boolean
Hints	This command is intended for use with CO2 lasers. For CW YAG lasers refer to *Put_Bitmapline_List_Ex.*	
	Put_Bitmapline_List should be called repetitively to create a complete bitmap.	
	Jump speed will be defined with *Set_Speed* or *Set_Jump_Parameters_List.*	
	Laser will be switched on after small jumps for a time period equal to the grey value in µs.	
	A jump will be created between the end of one bitmap-line and the beginning of the next.	
	Time can be saved by outputting the grey scale lines in opposite directions.	
	The length of the array (ArrayLen) must correspond to the number of grey values presented.	
References	*Set_Speed, Set_Jump_Parameters_List, Set_Delays, Put_Bitmapline_List_Ex, Set_Auto_Jump_Delay_List*	

Put_Bitmapline_List_Ex

☑ SP-ICE
☑ RLC

Function	This command has a similar functionality to *Put_Bitmapline_List* but it is designed for CW YAG laser.	
Result	Function Put_Bitmapline_List_Ex ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Put_Bitmapline_List_Ex (xstartval, ystartval, xendval, ysendval: smallint, pArrayGreyValues [0]:IntArray, ArrayLen: smallint): bool;
	C	bool Put_Bitmapline_List_Ex (short xstartval, short ystartval, short xendval, short yendval,&arrayGreyValues[0],short num_of_values);
	Basic	function Put_Bitmapline_List_Ex (byval xbeginval%, byval beginyval%, byval xendval%, byval endyval%, IntArray, byval arraysize%) as boolean
References	*Put_Bitmapline_List*	

Reset_Jump_List

SP-ICE with
☑ MOTF-Option
☐ RLC

Function		Absolute jump to the beginning of the next Mark-on-the-Fly operation. The purpose of this command is to position the spot at the start of the next object removing any accumulated target movement. In the case of *Wait_For_Counter_Value_Ex*, the increments of the encoder, however, will still be counted in the background so that the reference position is not lost. Call has to be issued before list commands *Wait_For_External_Start*, *Wait_For_Counter_Value_Ex* or *Mark_Immediately*.
Parameter		Coordinates, 16 bits signed integers, representing end of the vector.
Result		Function Reset_Jump_List ok (TRUE) or not ok (FALSE) as boolean.
Calling conventions	Pascal	function Reset_Jump_List (xval, yval: smallint): bool;
	C	bool Reset_Jump_List (short xval, short yval);
	Basic	function Reset_Jump_List (byval xval%, byval yval%) as Boolean
Hints		Jump speed will be defined with *Set_Speed* or *Set_Jump_Parameters_List*.
		Laser is switched off.
		A jump delay will be inserted after a jump command.
References		*Put_Bitmapline_List*

Set_Auto_Jump_Delay_List

☑ SP-ICE
☑ RLC

Function		Produces a variable jump delay which depends on the jump length.
Parameter		Function Set_Delays ok (TRUE) or not ok (FALSE) as boolean.
Result		Function Put_Bitmapline_List ok (TRUE) or not ok (FALSE) as boolean.
Calling conventions	Pascal	function Set_Delays (jump_auto_delay, jump_length: word): bool;
	C	bool Set_Delays (unsigned short jump_auto_delay, unsigned short jump_lenght);
	Basic	function Set_Delays (byval jump_auto_delay %, byval jump_lenght%) as boolean
Hints		This function allows the jump delay to increase linearly from jump_delay to jump_auto_delay as the jump distance increases from 0 to variable jump_length.
		Thus, particularly for marking jobs with lots of small jumps like text, it is possible to do the small jumps with little delay and the long jumps (say at the end of the line of text) with a longer delay.
		If jump_auto_delay ≤ jump_delay or jump_length = 0 this command has no effect on jump delay.
References		*Set_Delays, Put_Bitmap_Line_List*

Set_Delays

☑ SP-ICE
☑ RLC

Function	Sets delays for scan head and laser control.	
Parameter	All delays have to be defined as 16 bits unsigned integers.	
	50 ≤ step_period ≤ 65535	Step period of micro-vectors in [µs]; [60µs]
	50 ≤ jump_del ≤ 65535	Delay after a jump command in [µs]; [200µs]
	50 ≤ mark_del ≤ 65535	Delay after a *Mark* or *PolC* command in [µs]; [100µs]
	50 ≤ poly_del ≤ 65535 and poly_del = 0	Delay after a *PolA* or *PolB* command in [µs]; [50µs]
	50 ≤ laser_off_del ≤ 65535	Laser off delay after a *Mark* or *PolC* command in [µs]; [100µs]
	50 ≤ laser_on_del ≤ 65535	Laser on delay after a *Mark* or *PolA* command in [µs]; [200µs]
	Nd:YAG 0 ≤ t1 ≤ 65535 Q-Switch-cycle period in [µs]; [320µs] 1 ≤ t2 ≤ 65535 Q-Switch-pulse width in [µs]; [200µs] 0 ≤ t3 ≤ 65535 FPS length in [µs]; [0µs]	
	CO2 0 ≤ t1 ≤ 65535 Output period of laser pulses in [µs]; [320µs] 1 ≤ t2 ≤ 65535 Width of laser pulse in [µs] (laser is processing); [200µs] 0 ≤ t3 ≤ 65535 Width of laser stand-by pulse in [µs] (laser is in stand-by); [0µs]	
	Note: The standby (tickle) frequency is fixed at 5kHz. Typically the laser standby pulse width is set to 1µs. Refer to manual of your laser supplier.	
Result	Function Set_Delays ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Delays (step_period, jump_del, mark_del, poly_del, laser_off_del, laser_on_del, t1, t2, t3: word): bool;
	C	bool Set_Delays (unsigned short step_period, unsigned short jump_del, unsigned short mark_del, unsigned short poly_del, unsigned short laser_off_del, unsigned short laser_on_del, unsigned short t1, unsigned short t2, unsigned short t3);
	Basic	function Set_Delays (byval step_period%, byval jump_del%, byval mark_del%, byval poly_del%, byval laser_off_del%, byval laser_on_del%, byval t1%, byval t2%, byval t3%) as boolean
Hints	This command should be set before any vector commands in a list. The default values are shown in square brackets.	
	The range of values shown are guaranteed to work with all RAYLASE controllers. For further information about the possible use of shorter values, contact RAYLASE.	
	A typical value for step_period is 60µs. The longer step_period is the greater is the time available for loading list commands.	
	This command sets the t4 parameter (set by *Set_Delays_9_10*) to 0.	
References	*Set_Delays_1_2, Set_Delays_3_4, Set_Delays_5_6, Set_Delays_7_8, Set_Delays_9_10*	

Set_Delays_1_2

☑ SP-ICE
☑ RLC

Function	Sets step period and jump delay.	
Parameter	All values have to be defined as 16 bits unsigned integers. For a valid range and default value of the parameters look at *Set_Delays* list command.	
Result	Function Set_Delays_1_2 ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Delays_1_2 (step_period, jump_del: word): bool;
	C	bool Set_Delays (unsigned short step_period, unsigned short jump_del);
	Basic	function Set_Delays (byval step_period%, byval jump_del%) as boolean
Hints	The same apply as for *Set_Delays*	
References	*Set_Delays*	

Set_Delays_3_4

☑ SP-ICE
☑ RLC

Function	Sets mark and polygon delays.	
Parameter	All values have to be defined as 16 bits unsigned integers. For a valid range and default value of the parameters look at *Set_Delays* list command.	
Result	Function Set_Delays_3_4 ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Delays_3_4 (mark_del, poly_del: word): bool;
	C	bool Set_Delays_3_4 (unsigned short mark_del, unsigned short poly_del);
	Basic	function Set_Delays_3_4 (byval mark_del %, byval poly_del%) as boolean
Hints	The same apply as for *Set_Delays*	
References	*Set_Delays*	

Set_Delays_5_6

☑ SP-ICE
☑ RLC

Function	Sets laser on and laser off delays.	
Parameter	All values have to be defined as 16 bits unsigned integers. For a valid range and default value of the parameters look at *Set_Delays* list command.	
Result	Function Set_Delays_5_6 ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Delays_5_6 (laser_off_del, laser_on_del: word): bool;
	C	bool Set_Delays_5_6 (unsigned short laser_off_del, unsigned short laser_on_del);
	Basic	function Set_Delays_5_6 (byval laser_off_del %, byval laser_on_del %) as boolean
Hints	The same apply as for *Set_Delays*	
References	*Set_Delays*	

Set_Delays_7_8

☑ SP-ICE
☑ RLC

Function		Sets t1 and t2 parameters for laser. The actual meaning depending on the laser type.
Parameter		All values have to be defined as 16 bits unsigned integers. For a description, valid range and default values of the parameters see the *Set_Delays* list command.
Result		Function Set_Delays_7_8 ok (TRUE) or not ok (FALSE) as boolean.
Calling conventions	Pascal	function Set_Delays_7_8 (t1, t2: word): bool;
	C	bool Set_Delays_7_8 (unsigned short t1, unsigned short t2);
	Basic	function Set_Delays_7_8 (byval t1 %, byval t2 %) as boolean
Hints		The same apply as for *Set_Delays.*
References		*Set_Delays*

Set_Delays_9_10

☑ SP-ICE
☑ RLC

Function		Sets t3 and t4 parameters for laser. The actual meaning depending on the laser type.
Parameter		All values have to be defined as 16 bits unsigned integers. For a description, valid range and default value of t3 parameter look at *Set_Delays* list command.
		Parameter t4 is used only for Nd:YAG Mode2 lasers. The valid range is: 0 ≤ t4 ≤ 65535 FPS signal in [µs]; [0µs] If t4 = 0, then Nd:YAG Mode2 version is used. In this case t3 is used for FPS length and the FPS signal = 10µs. If t4 > 0, then Nd:YAG Mode3 version applies. See applications manual.
Result		Function Set_Delays_9_10 ok (TRUE) or not ok (FALSE) as boolean.
Calling conventions	Pascal	function Set_Delays_9_10 (t3, t4: word): bool;
	C	bool Set_Delays_9_10 (unsigned short t3, unsigned short t4);
	Basic	function Set_Delays_9_10 (byval t3 %, byval 4 %) as boolean
Hints		The same apply as for *Set_Delays*.
		The *Set_Delays* command sets t4 parameter to 0.
References		*Set_Delays*

Set_End_Of_List

☑ SP-ICE
☑ RLC

Function	Closes the list that was opened with the last *Set_Start_List_n* command. No more commands can be stored in the list. It is ready to be executed (see *Execute_List_n*).	
Result	Function Set_End_Of_List ok (TRUE) or not ok (FALSE) as boolean	
Calling conventions	Pascal	function Set_End_Of_List: bool;
	C	bool Set_End_Of_List (void);
	Basic	function Set_End_Of_List () as boolean
Hints	This is a list command which also works as a control command, changing the list status to "closed". During list execution the command is seen as the last in the list. At this time, it acts as a trigger for the evaluation of the *Aut_Change* flag, *Loop* command etc.	
	If there are no open lists, the command has no effects.	
	The command will close even an empty list with no commands in it, storing itself as the only command in it and allowing it to be executed as it is. It will not cause any errors or problems during execution.	
	Trying to use *Execute_List* on a list which is not closed by this command, will cause an error.	
References	*Set_Start_List_1, Set_Start_List_2, Execute_List_n, Aut_Change*	

Set_Jump_Parameters_List

☑ SP-ICE
☑ RLC

Function	Sets the jump speed of the galvanometer scanners through step_period and jump_step_size.	
Parameter	Both parameters have to be defined as 16 bits unsigned integer.	
	20 ≤ step_period ≤ 65535	Step period of microvectors in [µs]
	1 ≤ jump_step_size ≤ 65535	Step size of jump microvectors in [bits]
Result	Function Set_Jump_Parameters_List ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Jump_Parameters_List (step_period, jump_size: word);
	C	bool Set_Jump_Parameters_List (unsigned short step_period, unsigned short jump_size);
	Basic	function Set_Jump_Parameters_List (byval step_period %, byval jump_size) as boolean
Hints	With this command the actual jump speed of the galvanometer scanners is changed.	
	Note that the value for step_period is unique in the system, meaning it is the same for all commands.	
	Jump speed = (jump_size / step_period) * 1000 [bits per ms]	
	If not set by this or any other command, default values are assumed: jump_size = 50[bits]. See also *Set_Delays*.	
References	*Set_Delays, Set_Mark_Parameters_List, Set_Jump_Speed, Set_Speed*	

Set_Mark_Parameters_List

☑ SP-ICE
☑ RLC

Function	Sets the mark speed of the galvanometer scanners through step period and step size.	
Parameter	Both parameters have to be defined as 16 bits unsigned integer.	
	20 ≤ step_period ≤ 65535	Step period of microvectors in [µs]
	1 ≤ step_size ≤ 65535	Step size of marking microvectors in [bits]
Result	Function Set_Mark_Parameters_List ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Mark_Parameters_List (step_period, step_size: word);
	C	bool Set_Mark_Parameters_List (unsigned short step_period, unsigned short step_size);
	Basic	function Set_Jump_Parameters_List (byval step_period %, byval step_size) as boolean
Hints	With this command the actual mark speed of the galvanometer scanners is changed.	
	Note that the value for step_period is unique in the system, meaning it is same for all commands.	
	Mark speed = (step_size/step_period) * 1000 [bits per msec]	
	If not set by this or any other command, default values are assumed: mark_size = 50[bits] See also Set_Delays	
References	*Set_Delays, Set_Jump_Parameters_List, Set_Speed, Set_Mark_Speed,*	

Set_Wobble_List

☑ SP-ICE
☑ RLC

Function	Defines the width and period for the wobble function.	
Parameter	Width and period as 16 bit unsigned integers.	
Result	Function Set_Wobble_List ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Wobble_List (usWidth, usPeriod: word): bool;
	C	bool Set_Wobble_List(unsigned short usWidth, unsigned short usPeriod);
	Basic	function Set_Wobble_List (byval usWidth, byval usPeriod%) as boolean
Hints	While marking the beam rotates around the requested vector path by the defined width and the period.	
	The function stays active until the next *Set_Wobble_List* is programmed either to change the parameters or to disable the wobble function by setting the wobble width to 0.	
	The wobble function depends on the mark step_size. Be sure to program first the mark step_size and then the wobble function.	

Wait_For_Counter_Value_Ex

SP-ICE with
☑ MOTF-Option
☐ RLC

Function	Proceeds to the next list command after a number of encoder counts.	
Parameter	Encoder counts as 32 bit integer.	
Result	Function Wait_For_Counter_Value_Ex ok (TRUE) or not ok (FALSE) as boolean	
Calling conventions	Pascal	function Wait_For_Counter_Value_Ex (s: longint): bool;
	C	bool Wait_For_Counter_Value_Ex (long s);
	Basic	function Wait_For_Counter_Value_Ex (byval s&) as Boolean
Hints	To be used for triggering a mark after a number of encoder counts referenced to: - The position of the encoder at the instant the SP-ICE card was switched on OR - The position of the encoder at the instant of the execution of the previous *Mark_Immediately* list command OR - The position of the encoder at the time of the previous *Wait_For_Counter_Value_Ex* list command.	
	If an *Execute_List* command occurs and the *Wait_For_Counter_Value_Ex* command has a value equal or lower than the current Encoder value, then the mark will start immediately and this will be the reference point for the next *Wait_For_Counter_Value_Ex*.	
	If the command *Wait_For_Counter_Value_Ex* is encountered in a list and the mode for Marking on the Fly has not been set, this command will act the same as *Mark_Immediately* command.	
	The command *Wait_For_Counter_Value_Ex* replaces *Wait_For_Counter_Value* command.	
References	*Set_Dig_Gain_Ex*	

Wait_For_External_Start

☑ SP-ICE
☐ RLC

Function	Causes execution of a list to halt until the hardware input signal \START_MARK goes true.	
Result	Function Wait_For_External_Start ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Wait_For_External_Start: bool;
	C	bool Wait_For_External_Start (void);
	Basic	function Wait_For_External_Start () as boolean
Hints	\START_MARK must be there for at least the length of step_period.	
	This signal is edge triggered so that \START_MARK signal must be reset and then set again before the next start can be made.	
	Normally, the \START_MARK signal can be reset with the output signal \MIP (Mark_In_Progress), creating a handshake. Please, see the description of the command *Write_Port_List*.	

Write_DA_List

☑ SP-ICE
☑ RLC

Function	Outputs an 8 bit value through D/A converter to the interface signal ANA_OUT.	
Parameter	Output value, as a 16 bit unsigned integer, value 0 to 255.	
Result	Function Write_DA_List ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Write_DA_List (value: word);
	C	bool Write_DA_List (unsigned short value);
	Basic	function Write_DA_List (byval value%) as boolean
Hints	With this command normally the lamp current of YAG lasers will be set.	
	Output can be used as optional digital interface for setting the lamp current. For more information contact RAYLASE.	
	Only the 8 least significant bits define the D/A converter output.	
References	*Write_DA, Write_Port_List, Write_Port*	

Write_Port_List

☑ SP-ICE
☑ RLC

Function	Output to the interfaces. Since this is a list command, it can be used for synchronisation purposes, for instance, setting a hardware output between two vectors.			
Parameter	Output of 16 bits unsigned integer.			
	Valid port adresses	26H	Z-Channel	Z-DAC CANNEL[1]
		28H	O-Channel	P-DAC CANNEL[1]
		0CH	Port C	Bit 4 = \Mark In Progress[2] [3] [4] Bit 5 = \Remote_EXE_1[3] Bit 7 = \Remote_EXE_2[3]
		0AH	Port B	16 bits[6]
				8 bits (PB0 - PB7) [5]
		0EH	Port D	Option[7]
Result	Function Write_Port_List ok (TRUE) or not ok (FALSE) as boolean			
Calling conventions	Pascal	function Write_Port_List (port, value: word)		
	C	bool Write_Port_List (unsigned short port, unsigned short value)		
	Basic	function Write_Port_List (byval port%, byval value%) as boolean		
Hints	Other port addresses than specified above will be ignored.			
	The value is output to the port with the next list command which requires either laser or galvanometer scanner output. Therefore, if two or more consecutive *Write_Post_List* commands are inserted in the list, only the last one will be output.			
	Also, note that there are no "bit setting"/"bit clearing" commands; a whole word is output to port.			
	Output to Z-Channel will be overwritten, if 3rd axis correction is not disabled. (see control command *Set Mode*).			
References	*Write_DA_List, Write_DA, Write_Port, Read_Port, Set_Mode*			

	SP-ICE	RLC-USB	RLC-PCI
1) Scan Head Interface	☑	☑	☑
2) Restricted Laser / I/O Interface	☐	☐	☑
3) Laser / I/O Interface	☑	☑	☑
4) Extended Laser / I/O Interface	☑	☐	☐
5) Lee compatible Interface	☐	☑	☑
6) Port B	☑	☐	☐
7) Port D	☑	☐	☐

3 CONTROL COMMANDS

The control commands described below are listed in alphabetical order.

Aut_Change

☑ SP-ICE
☑ RLC

Function	Activates automatic switching from one list to the other. It is thus possible for the controller to work continuously, executing one list whilst the other is being filled. When the execution of the first list is complete, execution of the second list begins with virtually no delay.	
Result	Function Aut_Change ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Aut_Change: bool;
	C	bool Aut_Change (void);
	Basic	function Aut_Change () as boolean
Hints	The correct sequence of steps for continuous working are as follows: 1. Fill list 1 and close the list 2. Send *Execute_List_1* 3. Fill list 2 and close the list 4. Send *Aut_Change* 5. Read_Status and wait for list 1 not busy 6. Fill list 1 and close the list 7. Send *Aut_Change* 8. Read_Status and wait for list 2 not busy 9. Repeat steps 3 to 8 10. Exit the sequence if all vectors are done	
	This procedure assures proper order of list execution and avoids starting a list which has not been filled or trying to fill list that is being executed.	
	For critical applications, it is advisable to have, as the last vector in the list, one which switches off the laser – such as a *PolC*, *Mark* or a similar command.	
	The number of list starts is limited by the maximum allowed number of list starts set by the *Set_Max_Counts* command. If not set, the default value is 0, which will allow the lists to switch indefinitely.	
References	*Read_Status, Set_Max_Counts*	

Corr_File_Name

☑ SP-ICE
☑ RLC

Function	Reads back the correction file name that has been sent to the card.	
Result	Correction file name as string of characters	
Calling conventions	Pascal	function Corr_File_Name: char;
	C	char* Corr_File_Name (void);
	Basic	function Corr_File_Name () as string
References	*Load_Corr_N*	

Disable_Laser

☑ SP-ICE
☑ RLC

Function	Disables laser output. The job will run as normal but the LM_GATE and LM signals will stay false. This function is provided for systems using a pointer during which the laser modulation should be inactive but deflection of mirrors should occur. Only the laser pointer will then be visible but no marking will occur.	
Result	Function Disable_Laser ok (TRUE) or not ok (FALSE) as boolean	
Calling conventions	Pascal	function Disable_Laser: bool;
	C	bool Disable_Laser (void);
	Basic	function Disable_Laser () as boolean
Hints	After executing the *Disable_Laser* command, the laser modulation can be reactivated with *Enable_Laser*.	
	The command will not stop laser modulation immediately but rather affect the next list or control command which starts laser modulation.	
References	*Enable_Laser*	

Enable_Laser

☑ SP-ICE
☑ RLC

Function	Enables laser modulation during marking. That is, LM and LM_GATE will be re-activated.	
Result	Function Enable_Laser ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Enable_Laser: bool;
	C	bool Enable_Laser (void);
	Basic	function Enable_Laser () as boolean
Hints	The default state is laser modulation enabled.	
	The command will not start laser modulation immediately but rather enable laser modulation in the next list/control command which starts laser modulation.	
References	*Disable_Laser*	

Execute_List_1

☑ SP-ICE
☑ RLC

Function		Starts output of data of list 1.
Result		Function Execute_List_1 ok (TRUE) or not ok (FALSE) as boolean.
Calling conventions	Pascal	function Execute_List_1: bool;
	C	bool Execute_List_1 (void);
	Basic	function Execute_List_1 () as boolean
Hints		Execution (Real-time output) of data from list 1 starts. It is assumed that commands have been put into the list that it has been closed with *Set_End_Of_List* command.
		This command will return an error if either list is currently busy.
		The status of the list – i. e., whether it is busy – can be checked by polling using the *Read_Status* command.
		During of execution of list 1, commands can be downloaded to list 2.
References		*Execute_List_2, Set_End_Of_List, Read_Status, Aut_Change*

Execute_List_2

☑ SP-ICE
☑ RLC

Function		Starts output of data of list 2.
Result		Function Execute_List_2 ok (TRUE) or not ok (FALSE) as boolean.
Calling conventions	Pascal	function Execute_List_2: bool;
	C	bool Execute_List_2 (void);
	Basic	function Execute_List_2() as boolean
Hints		Execution (Real-time output) of data from list 2 starts. It is assumed that commands have been put into the list that it has been closed with Set_End_Of_List command.
		This command will return an error if either list is currently busy.
		The status of the list – i. e., whether it is busy – can be checked by polling using the *Read_Status* command.
		During execution of list 2, commands can be downloaded to list 1.
References		*Execute_List_1*

Get_Active_Card

☑ SP-ICE
☐ RLC

Function	Reads the active SP-ICE card number in a master-master configuration. Valid card numbers are starting from 1, 2, … corresponding to the order of inserting cards in the system.	
Result	Returns the active SP-ICE card as an unsigned 16 bit value.	
Calling conventions	Pascal	function Get_Active_Card: word;
	C	unsigned int Get_Active_Card (void);
	Basic	function Get_Active_Card ()%
References	*Set_Active_Card*	

Get_Counts

☑ SP-ICE
☑ RLC

Function	Reads the counter of start of lists, in automatic operation i. e. when using the *Aut_Change* function or *Start_Loop*. It can be used for test purposes.	
Result	Returns the value of the counter of start of lists, as a 32 bits signed integer.	
Calling conventions	Pascal	function Get_Counts: longint;
	C	Long Get_Counts (void);
	Basic	function Get_Counts ()&
Hints	Counter is reset whenever a new *Start_Loop* or *Execute_List* command is issued.	
	The counter is incremented when a list is started and not when it finishes. Lists, which are terminated during execution, are counted as well.	
References	*Set_Max_Counts, Start_Loop, Quit_Loop, Aut_Change*	

Get_CPU_Type

☑ SP-ICE
☐ RLC

Function	Returns the CPU type on the SP-ICE card.	
Result	CPU type as unsigned short.	
	The following values are valid: 0 => CPU without a floating point unit 1 => CPU with floating point unit 65535 => unknown type of CPU	
Calling conventions	Pascal	function Get_CPU_Type (): word;
	C	unsigned short Get_CPU_Type ();
	Basic	function Get_CPU_Type () as word
Hints	Some options like 3D are recommended to be used only with CPUs with floating point unit. This command can be used to check the suitability of the card for 3D before attempting to download 3D support software.	

Get_DLL_Version

Supported for legacy applications only

☑ SP-ICE
☑ RLC

Function	Reads back the version number of the DLL Program Library running under Windows	
Result	DLL-versions number, as 16 bits unsigned integer	
Calling conventions	Pascal	function Get_DLL_Version: word;
	C	unsigned short Get_DLL_Version (void);
	Basic	function Get_DLL_Version ()%
Hints	New applications should use *Get_Library_Version*	
References	*Get_SPC1_Version, Get_Version, Get_SPC1_Mode*	

Get_Device_Description_String

☐ SP-ICE
☑ RLC

Function	Returns a pointer to the descriptor provided by the RLC device.	
Result	Pointer as 32 bit integer	
Calling conventions	Pascal	function Get_Device_Description_String: int;
	C	int Get_ Device_Description_String;
	Basic	function Get_ Device_Description_String () int
Hints	This allows the application to differentiate between RLC-USB and RLC-PCI.	
	Valid strings:	“RLC-USB Device"
		"RLC-PCI Device"
	Hint for Basic users: *Get_Device_Description_String* returns the address of the string, which must be retrieved using: Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (pdst As Any, pSrc As Any, ByVal nBytes As Long)	
References	*Get_Driver_Version, Get_Hardware_Version, Get_Firmware_Version*	

Get_Driver_Version

☐ SP-ICE
☑ RLC

Function	Reads back the version number of the hardware driver.	
Result	Version number as 32 bit integer	
Calling conventions	Pascal	function Get_Driver_Version: int;
	C	int Get_Driver_Version;
	Basic	function Get_Driver_Version () int
Hints	The value can be easily converted for display in the standard version number format (mm.nn.rr.bb).	
References	*Get_Device_Description_String, Get_Hardware_Version, Get_Firmware_Version*	

Get_Firmware_Version

☐ SP-ICE
☑ RLC

Function	Reads back the version number of the firmware.	
Result	Version number as 32 bit integer.	
Calling conventions	Pascal	function Get_Firmware_Version: int;
	C	int Get_ Firmware _Version;
	Basic	function Get_ Firmware _Version () int
Hints	The value can be easily converted for display in the standard version number format (mm.nn.rr.bb).	
References	*Get_Device_Description_String, Get_Hardware_Version, Get_Driver_Version*	

Get_Hardware_Version

☐ SP-ICE
☑ RLC

Function	Reads back the version number of the card.	
Result	Version number as 32 bit integer.	
Calling conventions	Pascal	function Get_Hardware_Version: int;
	C	int Get_ Hardware _Version;
	Basic	function Get_ Hardware _Version () int
Hints	The value can be easily converted for display in the standard version number format (mm.nn.rr.bb).	
References	*Get_Device_Description_String, Get_Driver_Version, Get_Firmware_Version*	

Get_Ident_Ex

☑ SP-ICE
☐ RLC

Function	Returns the SP-ICE card serial number.	
Result	The SP-ICE card serial number as 16 bit integer.	
Calling conventions	Pascal	function Get_Ident_Ex (): word;
	C	unsigned short Get_Ident_Ex ();
	Basic	function Get_Ident_Ex () as word

Get_Jump_Speed

☑ SP-ICE
☑ RLC

Function	Reads back jump speed.	
Result	Jump speed as 64 bit IEEE-floating double.	
Calling conventions	Pascal	function Get_Jump_Speed: double;
	C	double Get_Jump_Speed;
	Basic	function Get_Jump_Speed () as double
Hints	Parameter jump_speed can be set with control commands *Set_Speed*, *Set_Jump_Speed*, or *Set_Jump_Parameters_List*. Jump speed = (jump_step_size / step_period) * 1000 [bits per msec]	
References	*Get_Mark_Speed, Set_Speed, Set_Jump_Speed, Set_Mark_Speed, Set_Jump_Parameters_List, Set_Mark_Parameters_List*	

Get_Library_Version

☐ SP-ICE
☑ RLC

Function	Reads back the version number of the dynamic link library.	
Result	Version number as 32 bit integer.	
Calling conventions	Pascal	function Get_Library_Version: int;
	C	int Get_Library_Version;
	Basic	function Get_Library_Version () as integer
Hints	The value can be easily converted for display in the standard version number format (mm.nn.rr.bb).	

Get_Mark_Speed

☑ SP-ICE
☑ RLC

Function	Reads back marking speed.	
Result	Mark speed as 64 bit IEEE-floating double.	
Calling conventions	Pascal	function Get_Mark_Speed: double;
	C	double Get_Mark_Speed;
	Basic	function Get_Mark_Speed () as double
Hints	Mark speed = (step_size / step_period) * 1000 [bits per msec] Parameter mark_speed can be set with control commands *Set_Speed*, *Set_Mark_Speed*, or *Set_Mark_Parameters_List*. Also some commands (*Set_Delays*, *Set_Mark_Parameters_List*) that change the step period and step size affect the resulting mark speed. Check each of these commands for precise explanation.	
References	*Set_Mark_Speed, Set_Speed, Set_Jump_Speed, Get_Jump_Speed, Set_Jump_Parameters_List, Set_Mark_Parameters_List, Set_Delays, Set_Mark_Parameters_List, Set_Jump_Parameters_List*	

Get_Mode_Mask

☑ SP-ICE
☑ RLC

Function	Returns the system mode mask.	
Parameters	Pointer to a 16 bit unsigned short system mask.	
	The meaning of each bit in the mask is as follows: D0 – SWAP_XY – swapping X and Y coordinates D1 – MASTER_SLAVE mode D2 – INVERT_X – inverting X coordinate D3 – INVERT_Y – inverting Y coordinate D4 – CO2 type of lasers D5 – YAG type of lasers D6 – WELDING mode D7 – SKIP_CORRECTION – overriding correction calculation D8 – DIRECT_Z – allows direct access to Z-Axis D9 – POWER_CONTROL mode D10 – LM_GATED D11 – LM_GATE_SENSE D12 – MOTF mode D13 – 3D mode D14 – not used D15 – not used	
Result	An error code as 32 bit integer. The value result = 0 means no error.	
Calling conventions	Pascal	function Get_Library_Version(var mode: smallint): int;
	C	int Get_Mode_Mask (signed short *mode);
	Basic	Function Get_Mode_Mask (ByRef mode As Integer) As Long
Hints	The mode mask indicates the setting of specific options and modes. If a bit is set the mode is enabled and if not set then it is disabled.	
	These modes are always available on the card: INVERT_Y, INVERT_X, SWAP_XY, SKIP_CORRECTION, YAG, CO2, LMGATE_SENSE, LM_GATED and POWER_CONTROL.	
	These modes have to be enabled on the card: MASTER_SLAVE, WELDING, MOTF and 3D. Contact RAYLASE for further details.	
	Read the explanation for *Set_Mode* command for further details on each mode.	
References	*Set_Mode*	

Get_SPC1_Version

☑ SP-ICE
☐ RLC

Function	Reads back the version number of the FPGA on the SP-ICE Card.	
Result	FPGA version as 16 bits unsigned integer.	
Calling conventions	Pascal	function Get_SPC1_Version: word;
	C	unsigned short Get_SPC1_Version (void);
	Basic	function Get_SPC1_Version ()%
References	*Get_Version, Get_DLL_Version*	

Get_System_Status

☑ SP-ICE
☐ RLC

Function	Returns the system status.	
Result	System status as unsigned short. The following values are valid: D0 DLL_TYPE 0 - SPC, 1 - SPI-CE D1 - D3 non zero-based index of the active card D4 SSF_ADDR not used D5 SSF_INTR not used D6 - D8 active serial port number of the SP-ICE card D9 SSF_PAR 1 - Indicates that the parallel port is used 0 - Indicates that the parallel port is not used	
Calling conventions	Pascal	Get_System_Status (): word;
	C	unsigned short Get_System_Status ();
	Basic	function Get_System_Status () as word

Get_Version

☑ SP-ICE
☐ RLC

Function	Reads back the version number of the SPICE.RTB software.	
Result	SP-ICE card software version number, as 16 bits unsigned integer.	
Calling conventions	Pascal	function Get_Version: word;
	C	unsigned short Get_Version (void);
	Basic	function Get_Version ()%
References	*Get_SPC1_Version, Get_DLL_Version*	

Get_XY_Pos

☑ SP-ICE
☑ RLC

Function	Reads back the last commanded beam position for the currently active head.	
Parameters	Last commanded beam coordinates as 16 bits signed integer.	
Result	Function Get_XY_Pos ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Get_XY_Pos (var xpos, ypos: smallint): bool;
	C	bool Get_XY_Pos (signed short *xpos, signed short *ypos);
	Basic	function Get_XY_Pos (xpos%, ypos%) as boolean
Hint	Note that this command returns the commanded target positions. There is no feedback from the hardware.	

Init_Scan_Card

☑ SP-ICE
☑ RLC

Function	Initialize the card bringing it into an initial state.	
Parameters	Card number, as 16 bit unsigned integer. Only value 1 is supported for the RLC unit.	
Result	TRUE on success, otherwise FALSE.	
Calling conventions	Pascal	function Init_Scan_Card: bool;
	C	bool Init_Scan_Card_Ex(unsigned short N);
	Basic	function Init_Scan_Card_Ex(N%) as boolean
Hints	This command is supported for backwards-compatibility with existing applications only. New applications should use *Init_Scan_Card_Ex*	
	This command should be called first in an application program.	
	If called any time later on, it stops execution of lists, turns off laser, discards list commands sent to card and the correction files.	
References	*Init_Scan_Card_Ex, Remove_Scan_Card*	

Init_Scan_Card_Ex

☑ SP-ICE
☑ RLC

Function	Initialize the card bringing it into an initial state.	
Parameters	Card number, as 16 bit unsigned integer. Only value 1 is supported for the RLC unit.	
Result	ERR_OK (0) on success, otherwise a non-zero ERROR_CODE.	
Calling conventions	Pascal	function Init_Scan_Card_Ex: integer;
	C	int Init_Scan_Card_Ex(unsigned short N);
	Basic	function Init_Scan_Card_Ex(N%) as long
Hints	This command should be called first in an application program.	
	If called any time later on, it stops execution of lists, turns off laser, discards list commands sent to card and the correction files.	
References	*Remove_Scan_Card_Ex*	

Load_Cor

☑ SP-ICE
☑ RLC

<table>
<tr><td>Function</td><td colspan="2">Loads a correction file for the default scan head, to suit the optical characteristics of the deflection system and lens.</td></tr>
<tr><td>Parameters</td><td colspan="2">Pointer to the name of the correction file.</td></tr>
<tr><td>Result</td><td colspan="2">Function Load_Cor ok (TRUE) or not ok (FALSE) as boolean.</td></tr>
<tr><td rowspan="3">Calling conventions</td><td>Pascal</td><td>function Load_Corr_N (var lpstrFileName: pchar): bool;</td></tr>
<tr><td>C</td><td>bool Load_Corr_N (const char* lpstrFileName);</td></tr>
<tr><td>Basic</td><td>function Load_Corr_N (lpstrFileName$) as boolean</td></tr>
<tr><td rowspan="2">Hints</td><td colspan="2">Correction file should be loaded after <i>Init_Scan_Card</i> command. If no file is loaded, default (zero) values are used.</td></tr>
<tr><td colspan="2">New correction file replaces the previous one.
Small changes in the positioning can be done with <i>Set_Gain</i> and <i>Set_Offset</i>.</td></tr>
<tr><td>References</td><td colspan="2"><i>Load_Corr_N, Init_Scan_Card, Corr_File_Name, Set_Gain, Set_Offset</i></td></tr>
</table>

Load_Corr_N

☑ SP-ICE
☑ RLC

<table>
<tr><td>Function</td><td colspan="2">Loads a correction file for the specified scan head, to suit for optical characteristics of the deflection system and lens.</td></tr>
<tr><td>Parameters</td><td colspan="2">Pointer to the name of the correction file and the number of the scan head.</td></tr>
<tr><td>Result</td><td colspan="2">Function Load_Cor ok (TRUE) or not ok (FALSE) as boolean.</td></tr>
<tr><td rowspan="3">Calling conventions</td><td>Pascal</td><td>function Load_Corr_N (var lpstrFileName: pchar, N: word): bool;</td></tr>
<tr><td>C</td><td>bool Load_Corr_N (const char* lpstrFileName, int N);</td></tr>
<tr><td>Basic</td><td>function Load_Corr_N (lpstrFileName$, byval N%) as boolean</td></tr>
<tr><td rowspan="2">Hints</td><td colspan="2">Correction files should be loaded after <i>Init_Scan_Card</i> command. If no files are loaded, default (zero) values are used.</td></tr>
<tr><td colspan="2">New correction file replace the previous one.</td></tr>
<tr><td>References</td><td colspan="2"><i>Load_Cor, Init_Scan_Card, Corr_File_Name, Set_Gain, Set_Offset</i></td></tr>
</table>

Quit_Loop

☑ SP-ICE
☑ RLC

Function	At the end of the active list, quits the continuous output of lists, started with the *Start_Loop* Command.	
Result	Function Quit_Loop ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Quit_Loop: bool;
	C	bool Quit_Loop (void);
	Basic	function Quit_Loop () as boolean;
Hints	Execution of the active list is continued until the last command in the list is executed.	
	Quit_Loop command does not affect contents of the lists. After a *Quit_Loop* command, a new *Start_Loop* command can be issued; execution will proceed with list 1.	
References	*Start_Loop*	

Read_Port

☑ SP-ICE
☑ RLC

Function	Reads values from the interfaces.			
Parameter	Read in a 16 bits unsigned integer from the port.			
	Valid port adresses	0CH	Port C	Bit 0 = \START_MARK[1) 2) 3)] Bit 1 = \general Purpose Input Bit 2 = \general Purpose Input Bit 3 = \STOP_MARK[1) 2) 3)]
		08H	Port A	8 bits buffered, 16 bits unbuffered[5)]
				7 bits (PA0 – PA6)[4)]
		10H	Port E	used for MOTF-Option [6)]
		2EH	Status Channel 1	
		32H	Status Channel 2	
Result	Function Read_Port ok (TRUE) or not ok (FALSE) as boolean.			
Calling conventions	Pascal	function Read_Port (port: word, var value: word): bool;		
	C	Bool Read_Port (unsigned short port, unsigned short *pvalue);		
	Basic	function Read_Port (byval port%, value%) as boolean		
Hints	Other port addresses than specified above will be ignored.			
References	*Write_Port_List, Write_Port*			

	SP-ICE	RLC-USB	RLC-PCI
1) Restricted Laser / I/O Interface	☐	☐	☑
2) Laser / I/O Interface	☑	☑	☑
3) Extended Laser / I/O Interface	☑	☐	☐
4) Lee compatible Interface	☐	☑	☑
5) Port A	☑	☐	☐
6) Port E	☑	☐	☐

Read_Status

☑ SP-ICE
☑ RLC

Function	Reads back the card status.	
Result	16 bits unsigned integer.	
	Bit 0 Load1	Indicates that list 1 is open for data input and all following list commands will be stored in it Is set with *Set_Start_List_1* and reset with *Set_End_Of_List*.
	Bit 1 Load2	Indicates that list 2 is open for data input and all following list commands will be stored in it. Is set with *Set_Start_List_2* and reset with *Set_End_Of_List*.
	Bit 2 Ready1	Indicates that list 1 has been filled and closed. It is set with *Set_End_Of_List*.
	Bit 3 Ready2	Indicates that list 2 has been filled and closed. It is set with *Set_End_Of_List*.
	Bit 4 Busy1	Indicates that list 1 is being executed.
	Bit 5 Busy2	Indicates that list 2 is being executed.
	Bit 6 Busy	Indicates that one of the lists is being executed.
	Bit 7 LaserOn	Indicates that laser is on.
	Bit 8 Scan Complete	Indicates that scanning was finished either regularly at the end of the list or interrupted during execution.
	Bit 9	Previously used for indication that manual operation is switched on.
	Bit 10	Previously used for manual movement indicating that scanners are moved with control command.
	Bit 11 Marking Busy	Indicates that marking is not yet finished – this occurs when there are still commands in the output buffer to be processed even though all list commands have been interpreted.
	Bit 12	not used
	Bit 13	not used
	Bit 15 STOP Marking C).	The hardware signal STOP_MARK was received (through port Laser will be switched off list execution stopped. Clear this bit with *Stop_Execution*
Calling conventions	Pascal	function Read_Status: word;
	C	unsigned short Read_Status (void);
	Basic	function Read_Status ()%
References	*Execute_List_n, Set_End_Of_List, Stop_Execution*	

Remove_Scan_Card

☑ SP-ICE
☑ RLC

Function		Shuts down the card.
Parameters		Card number, as 16 bit unsigned integer. Only value 1 is supported for the RLC card.
Result		TRUE on success, otherwise FALSE.
Calling conventions	Pascal	function Remove_Scan_Card: bool;
	C	bool Remove_Scan_Card (void);
	Basic	function Remove_Scan_Card () as Boolean
Hints		This command is supported for backwards-compatibility with existing applications only. New applications should use *Remove_Scan_Card_Ex*.
		This command should be called before closing application program.
		It also causes a *Stop_Execution* command, which stops execution of list commands and resets command lists. All serial and parallel ports are cleared.
		Use *Init_Scan_Card* to reconnect to the card.
References		*Init_Scan_Card, Remove_Scan_Card_Ex*

Remove_Scan_Card_Ex

☑ SP-ICE
☑ RLC

Function		Shuts down the card.
Parameters		Card number, as 16 bit unsigned integer. Only value 1 is supported for the RLC card.
Result		ERR_OK (0) on success, otherwise a non-zero ERROR_CODE.
Calling conventions	Pascal	function Remove_Scan_Card: integer;
	C	int Remove_Scan_Card (void);
	Basic	function Remove_Scan_Card () as long
Hints		Command Remove_Scan_Card should be called before closing application program.
		It also causes a *Stop_Execution* command, which stops execution of list commands and resets command lists. All serial and parallel ports are cleared.
		Use *Init_Scan_Card_Ex* to reconnect to the card.
References		*Init_Scan_Card_Ex*

Restart_List_1

☑ SP-ICE
☑ RLC

Function	Starts execution of list1.	
Result	Function Restart_List_1 ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Restart_List_1: bool;
	C	bool Restart_List_1 (void);
	Basic	function Restart_List_1 () as boolean
Hints	Counter for start of lists is reset to 0.	
	This command is identical to *Execute_List_1* command.	
References	*Restart_List_2, Execute_List_1, Execute_List_2*	

Restart_List_2

☑ SP-ICE
☑ RLC

Function	Starts execution of list2.	
Result	Function Restart_List_2 ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Restart_List_2: bool;
	C	bool Restart_List_2 (void);
	Basic	function Restart_List_2 () as boolean
Hints	Counter for start of lists is reset to 0.	
	This command is identical to *Execute_List_2* command.	
References	*Restart_List_1, Execute_List_1, Execute_List_2*	

Set_Active_Card

☑ SP-ICE
☑ RLC

Function	Defines the active card.	
Parameters	Card number, as 16 bits unsigned integer. Only value 1 is supported for the RLC card.	
Result	Function Set_Active_Card ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Active_Card (base: word): bool;
	C	bool Set_Active_Card (unsigned short base);
	Basic	function Set_Active_Card (byval base%) as boolean
Hints	The system checks if the card is installed and if ok sets it as the active card in a master-master application. All the following control or list commands will be sent to the specified card.	
References	*Get_Active_Card*	

Set_Auto_Delay

☑ SP-ICE
☑ RLC

Function	Sets step period of microvectors for marking and jump commands.	
Parameters	Step period in [µs], as 16 bits unsigned integer. For valid value range of step period, see *Set_Delays* list command.	
Result	Function Set_Auto_Delay ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Auto_Delay (usStepPeriod: word): bool;
	C	bool Set_Auto_Delay (unsigned short usStepPeriod);
	Basic	function Set_Auto_Delay (byval usStepPeriod%) as boolean;
Hints	For the specified step_period in the command and the current mark_speed and jump_speed values, the system calculates mark step_size and jump_step_size. If the re-calculated values are out of the valid range (see *Set_Delays* list command), the command returns a false flag and retains the previous value for step_period.	
	Parameter step_period can also be set with *Set_Delays* list command during list execution.	
	The same value for step_period is used in *Set_Mark_Parameters_List*, *Set_Jump_Parameters_List* and *Set_Delays* commands.	
References	*Set_Jump_Delay, Set_Mark_Delay, Set_Poly_Delay, Set_Laser_Off_Delay, Set_Laser_On_Delay, Set_T1, Set_T2, Set_T3, Set_Delays*	

Set_Dig_Gain_Ex

SP-ICE with
☑ MOTF-Option
☐ RLC

Function	Sets resolution of encoder in counts.	
Parameters	Encoder resolution as a 64 bit IEEE- floating double.	
Result	Function Set_Dig_Gain_Ex ok (TRUE) or not ok (FALSE) as boolean	
Calling conventions	Pascal	function Set_Dig_Gain_Ex (DigGain double): bool;
	C	bool Set_Dig_Gain_Ex (double DigGain);
	Basic	function Set_Dig_Gain_Ex (byval DigGain#) as Boolean
Hints	*Set_Dig_Gain_Ex* has to be implemented for the Mark-on-the-Fly application. If not set a default value of 1count/bit is used.	
	Set_Dig_Gain_Ex replaces *Set_Dig_Gain* which used to have a resolution of counts per 100bits and accept an integer value.	
	This command resets the current value for the number of encoder counts.	
References	*Set_Rot_Grad*	

Set_Gain

☑ SP-ICE
☑ RLC

Function	Defines the gain and offset for X and Y.	
Parameters	Gain values for X und Y axis, in the range (0.01-100), as 64 bits IEEE-floating double.	
	Offset values for X and Y axis, with no range limit, as 16 bits signed integer.	
Result	Function Set_Gain ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Gain (gain_x, gain_y: double; offset_x, offset_y: smallint): bool;
	C	bool Set_Gain (double gain_x, double gain_y, short offset_x, short offset_y);
	Basic	function Set_Gain (byval gainx#, byval gainy#, byval offset_x%, byval offset_y%) as boolean
Hints	The gain factors allow small corrections to be made to the calibration; i. e. if a square is marked as a rectangle it can be corrected with the gain factors.	
	Gain and offset values are used to modify all X and Y coordinates in list commands, according to the following: actual_coordinate = programmed_coordinate * gain + offset It is programmer‘s responsibility, to assure that new coordinates are still within the valid field. No checking or error reporting is done by the system. Instead, maximum allowed values will be issued during execution if the coordinates lie outside the valid field.	
References	*Set_Gain_X, Set_Gain_Y, Set_Offset_X, Set_Offset_Y*	

Set_Gain_X

☑ SP-ICE
☑ RLC

Function	Defines the gain factor for X axis.	
Parameters	Gain value for X axis, in the range (0.01-100), as 64 bits IEEE-floating double.	
Result	Function Set_Gain_X ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Gain_X (gain_x: double): bool
	C	bool Set_Gain_X (double gain_x);
	Basic	function Set_Gain_X (byval gainx#) as boolean
Hints	See the hints section for *Set_Gain* command	
References	*Set_Gain, Set_Gain_Y, Set_Offset_X, Set_Offset_Y*	

Set_Gain_Y

☑ SP-ICE
☑ RLC

Function	Defines the gain factor for Y axis.	
Parameters	Gain value for Y axis, in the range (0.01-100), as 64 bits IEEE-floating double.	
Result	Function Set_Gain_Y ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Gain_Y (gain_y: double): bool;
	C	bool Set_Gain_Y (double gain_y);
	Basic	function Set_Gain_Y (byval gainy#) as boolean
Hints	See the hints section for *Set_Gain* command	
References	*Set_Gain, Set_Gain_X, Set_Offset_X, Set_Offset_Y*	

Set_Jump_Delay

☑ SP-ICE
☑ RLC

Function	Sets jump delay.	
Parameters	Jump delay in the range 20-65535 [µs], as 16 bits unsigned integer.	
Result	Function Set_Jump_Delay ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Jump_Delay (usJumpDelay: word): bool;
	C	bool Set_Jump_Delay (unsigned short usJumpDelay);
	Basic	function Set_Jump_Delay (byval usJumpDelay%) as boolean
Hints	Parameter jump delay (jump_del) can also be set with *Set_Delays* list command during list execution.	
	If not set by any command, default value of 200µs is assumed.	
References	*Set_Auto_Delay, Set_Mark_Delay, Set_Poly_Delay, Set_Laser_Off_Delay, Set_Laser_On_Delay, Set_T1, Set_T2, Set_T3, Set_Delays*	

Set_Jump_Speed

☑ SP-ICE
☑ RLC

Function	Sets jump speed and the corresponding jump size.	
Parameters	Jump speed as 64 bit IEEE-floating double in [bits/ms].	
Result	Function Set_Jump_Speed ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Jump_Speed (dJumpSpeed: double): bool;
	C	bool Set_Jump_Speed (double dJumpSpeed);
	Basic	function Set_Jump_Speed (byval dJumpSpeed#) as boolean
Hints	Parameter jump_speed can also be set with *Set_Speed* control command or *Set_Jump_Parameters_List* list command.	
	jump_step_size = (step_period [µs] * jump_speed [bits/ms]) / 1000 If the value for jump_step_size is out of valid range (see *Set_Jump_Parameters_List* list command), old value for jump_step_size and jump_speed are kept and an error flag (ERR_OUT_OF_LIMIT) is set.	
References	*Set_Speed, Set_Mark_Speed, Set_Jump_Parameters_List, Set_Mark_Parameters_List*	

☑ SP-ICE
☑ RLC

Set_Laser_Off_Delay

Function	Sets laser off time delay.	
Parameters	Laser off time delay in the range (20-65535)[µs], as 16 bits unsigned integer.	
Result	Function Set_Laser_Off_Delay ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Laser_Off_Delay (usLaserOffDelay: word): bool;
	C	bool Set_Laser_Off_Delay (unsigned short usLaserOffDelay);
	Basic	function Set_Laser_Off_Delay (byval usLaserOffDelay%) as boolean
Hints	Parameter laser off delay can be set with list command *Set_Delays* during execution of list.	
References	*Set_Auto_Delay, Set_Jump_Delay, Set_Poly_Delay, Set_Laser_On_Delay, Set_Mark_Delay, Set_T1, Set_T2, Set_T3, Set_Delays*	

☑ SP-ICE
☑ RLC

Set_Laser_On_Delay

Function	Sets laser on time delay.	
Parameters	Laser on time delay in the range (20 - 65535)[µs], as 16 bits, unsigned integer.	
Result	Function Set_Laser_On_Delay ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Laser_On_Delay (usLaserOnDelay: word): bool;
	C	bool Set_Laser_On_Delay (unsigned short usLaserOnDelay);
	Basic	function Set_Laser_On_Delay (byval usLaserOnDelay%) as boolean
Hints	Parameter laser on delay can be set with list command *Set_Delays* during execution of list.	
References	*Set_Auto_Delay, Set_Jump_Delay, Set_Poly_Delay, Set_Laser_Off_Delay, Set_Mark_Delay, Set_T1, Set_T2, Set_T3, Set_Delays*	

☑ SP-ICE
☑ RLC

Set_Mark_Delay

Function	Sets mark delay.	
Parameters	Mark delay in the range 20 - 65535 [µs], as 16 bits unsigned integer.	
Result	Function Set_Mark_Delay ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Mark_Delay (usMarkDelay: word): bool;
	C	bool Set_Mark_Delay (unsigned short usMarkDelay);
	Basic	function Set_Mark_Delay (byval usMarkDelay%) as boolean
Hints	Parameter mark delay (mark_del) can be set with list command *Set_Delays* during execution of list.	
	If not set by any command, default value of 100µs is assumed.	
References	*Set_Auto_Delay, Set_Jump_Delay, Set_Poly_Delay, Set_Laser_Off_Delay, Set_Laser_On_Delay, Set_T1, Set_T2, Set_T3, Set_Delays*	

Set_Mark_Speed

☑ SP-ICE
☑ RLC

Function	Sets mark speed and the corresponding mark step size.	
Parameters	Mark speed as 64 bit IEEE-floating double in [bits/ms].	
Result	Function Set_Jump_Speed ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Mark_Speed (dMarkSpeed: double): bool;
	C	bool Set_Mark_Speed (double dMarkSpeed);
	Basic	function Set_Mark_Speed (byval dMarkSpeed#) as boolean
Hints	Mark step_size = (step_period [µs] * mark_speed [bits/ms]) / 1000 If the value for mark step_size is out of valid range (refer to *Set_Delays* list command), previous values for mark step_size and mark_speed are kept and an error flag (ERR_OUT_OF_LIMIT) is set.	
	Parameter mark speed can also be set with control command *Set_Speed* or list command *Set_Mark_Parameters_List*. Use *Set_Mark_Parameters_List* for direct control from the application of the parameters.	
References	*Set_Speed, Set_Jump_Speed, Set_Jump_Parameters_List, Set_Mark_Parameters_List*	

Set_Max_Counts

☑ SP-ICE
☑ RLC

Function	Defines the maximum number of list starts in a loop.	
Parameters	Maximum number of starts of list, as 32 bits signed integer. 0 ≤ counts ≤ 2 147 483 647	
Result	Function Set_Max_Counts ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Max_Counts (counts: longint): bool;
	C	bool Set_Max_Counts (long counts);
	Basic	function Set_Max_Counts (byval counts&) as boolean
Hints	If max_counts is set to 0 or > 1.000.000, the number of starts is not limited and the loop runs indefinitely.	
	After reaching the maximum number of external starts, next list in a loop is not executed. The loop can be reinitiated with another *Start_Loop* command. The value for max_counts is withheld and does not have to be set again before *Start_Loop*.	
	If not set with this command, a default value max_counts = 0 is assumed.	
References	*Get_Counts, Start_Loop, Quit_Loop, Aut_Change, Loop_To_Start_List*	

Set_Mode

☑ SP-ICE
☑ RLC

Function	Defines the scanner mode.	
Parameters	Mode as an unsigned 16 bit value.	
	The following modes are active if the corresponding bit is set to 1: Bit 0: **Swap XY** X and Y axis are swapped. Swapping is done first and then inverting of X or Y if the corresponding flag is set. This mode setting can be used with bit 2 and 3 to do rotation and mirroring in 8 possible cases. Bit 1: --- Bit 2: **Invert X** Inversion is done after swapping XY, if required. Bit 3: **Invert Y** Inversion is done after swapping XY, if required. Bits 4, 5: 0 CO$_2$-Mode. 00 YAG-Mode-2 01 YAG-Mode-1 11 Diode-Laser-Mode Bit 6: --- Bit 7: **Skip correction** No correction will be made by the field correction algorithm. Bit 8: **Disable 3rd axis correction** Allows *Write_Port* and *Write_Port_List* commands to use the Z-axis independently, without being overwritten by the correction output. Bit 9: --- Bit 10: **LM signal** Always set to 1. If set to 0: Last pulse of LM signal will be continuing after laser off delay. If set to 1: Last pulse of LM signal will be switched off exactly at laser off. Bit 11: **LM_GATE active LOW/HIGH** If set to 0 (default mode): LM_GATE signal is LOW_ACTIVE If set to 1: LM_GATE signal is HIGH_ACTIVE Bit 12: --- Bit 13: **3D** set mode Bit 14 - 15: Reserved	
Result	Function Set_Mode ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function procedure Set_Mode (mode: word): bool;
	C	bool Set_Mode (unsigned short mode);
	Basic	function Set_Mode (byval mode%) as boolean
Hints	If bits D1, D6, D12 or D13 are set, modes are allowed only if corresponding hardware keys have been set.	
References	*Get_SPC1_Mode, Write_Port, Write_Port_List, Get_Mode_Mask*	

Set_Offset_X

☑ SP-ICE
☑ RLC

Function	Defines the offset for X axis.	
Parameters	Offset value for X axis as 16 bits signed integer.	
Result	Function Set_Offset_X ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Offset_X (ssoffset_x): bool;
	C	bool Set_Offset_X (short ssoffset_x);
	Basic	function Set_Offset_X (byval ssoffset_x%) as boolean
Hints	The offset factor for X axis is dedicated to make small adoption of the calibration. The specified value for offset is added to programmed X position. If the calculated value for X is not inside the allowed range, maximum possible value is output. If not programmed, a default value of offset_X = 0 is used.	
References	*Set_Gain, Set_Gain_X, Set_Gain_Y, Set_Offset_Y*	

Set_Offset_Y

☑ SP-ICE
☑ RLC

Function	Defines the offset for Y axis.	
Parameters	Offset value for Y axis as 16 bits signed integer.	
Result	Function Set_Offset_Y ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Offset_Y (ssoffset_y): bool;
	C	bool Set_Offset_Y (short ssoffset_y);
	Basic	function Set_Offset_Y (byval ssoffset_y%) as boolean
Hints	The offset factor for Y axis is dedicated to make small adoption of the calibration. The specified value for offset is added to programmed Y position. If the calculated value for Y is not inside the allowed range, maximum possible value is output. If not programmed, a default value of offset_Y = 0 is used.	
References	*Set_Gain, Set_Gain_X, Set_Gain_Y, Set_Offset_X*	

Set_Poly_Delay

☑ SP-ICE
☑ RLC

Function	Sets polygon delay.	
Parameters	Poly delay in the range 0 - 65535 [µs], as 16 bits unsigned integer.	
Result	Function Set_Poly_Delay ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Poly_Delay (usPolyDelay: word): bool;
	C	bool Set_Poly_Delay (unsigned short usPolyDelay);
	Basic	function Set_Poly_Delay (byval usPolyDelay%) as boolean
Hints	Parameter polygon delay (poly_del) can be set with *Set_Delays* list command during list execution.	
	If not set by any command, default value of 50µs is assumed.	
References	*Set_Auto_Delay, Set_Jump_Delay, Set_Mark_Delay, Set_Laser_Off_Delay, Set_Laser_On_Delay, Set_T1, Set_T2, Set_T3, Set_Delays*	

Set_Rot_Grad

SP-ICE with
☑ MOTF-Option
☐ RLC

Function	Defines orientation of the moving part relative to the scan head.	
Parameters	Orientation of the moving part in radians (radians = α / 180° * π). Value has to be 64 bit IEEE- floating double, - 2π ≤ Deg ≤ 2π.	
Result	Function Set_Rot_Grad ok (TRUE) or not ok (FALSE) as boolean	
Calling conventions	Pascal	function Set_Rot_Grad (DigGain double): bool;
	C	bool Set_Rot_Grad (double DigGain);
	Basic	function Set_Rot_Grad (byval DigGain#) as Boolean
Hints	- Web is moving along X-axis of scan head, count is increasing: Deg = 0 - Web is moving along Y-axis of scan head, count is increasing: Deg = π/2 - Web is moving along X-axis of scan head, count is decreasing: Deg = +/-π - Web is moving along Y-axis of scan head, count is decreasing: Deg = -π/2 This command resets the current value for the number of encoder counts. The default value for *Set_Rot_Grad* is 0°.	
	This command resets the current value for the number of encoder counts.	
References	*Set_Dig_Gain_Ex*	

Set_Speed

☑ SP-ICE
☑ RLC

Function	Defines jump and marking speed.	
Parameters	Jump speed and marking speed in [bits/ms]. Values have to be 64 bit IEEE-floating double.	
Result	Function Set_Speed ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Speed (jump_speed, mark_speed: double): bool;
	C	bool Set_Speed (double jump_speed, double mark_speed);
	Basic	function Set_Speed (byval jump_speed#, byval mark_speed#) as boolean
Hints	This command sets jump and mark speed in one command, just like *Set_Mark_Speed* and *Set_Jump_Speed* control commands do.	
	See the hints section for *Set_Mark_Speed* commands for important information about this and *Set_Jump_Speed* command.	
	Both speeds should be higher than 100 [bits/ms].	
References	*Set_Jump_Speed, Set_Mark_Speed, Set_Jump_Parameters_List, Set_Mark_Parameters_List.*	

Set_Start_List_1

☑ SP-ICE
☑ RLC

Function	Selects list 1 as the active list for download. All following list commands will be directed to list 1.	
Result	Function Set_Start_List_1 ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Start_List_1: bool;
	C	bool Set_Start_List_1 (void);
	Basic	function Set_Start_List_1 () as boolean
Hints	List 1 is selected as active only if it is not being executed at the same time, if it is, an ERR_SCAN_ACTIVE error is set and the command neglected.	
	Setting a list to be active for download deletes any prior list commands sent to it and resets the Set_End_Of_List flag if it was previously set.	
	If list 2 is being executed, list commands can still be downloaded to list 1.	
References	*Set_Start_List_2*	

Set_Start_List_2

☑ SP-ICE
☑ RLC

Function	Selects list 2 as the active list for download. All following list commands will be directed to list 2.	
Result	Function Set_Start_List_2 ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_Start_List_2: bool;
	C	bool Set_Start_List_2 (void);
	Basic	function Set_Start_List_2 () as boolean
Hints	List 2 is selected as active only if it is not being executed at the same time, otherwise an ERR_SCAN_ACTIVE error is set and the command neglected.	
	Setting a list to be active for download deletes any prior list commands sent to it and resets the Set_End_Of_List flag if previously set.	
	If list 1 is being executed, list commands can still be downloaded to list 2.	
References	*Set_Start_List_1*	

Set_T1

☑ SP-ICE
☑ RLC

Function	Sets Q-switch cycle time (Nd:YAG) or output period of laser pulses (CO_2).	
Parameters	Time as 16 bits, unsigned integer.	
Result	Function Set_T1 ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Set_T1 (usT1: word): bool;
	C	bool Set_T1 (unsigned short usT1);
	Basic	function Set_T1 (byval usT1%) as boolean
Hints	T1 can be set with list command *Set_Delays* during execution of list.	
	See the hints section for the *Set_Delays* command for important information on valid value range and default value.	
References	*Set_Auto_Delay, Set_Jump_Delay, Set_Mark_Delay, Set_Poly_Delay, Set_Laser_Off_Delay, Set_Laser_On_Delay, Set_T2, Set_T3, Set_Delays*	

Set_T2

☑ SP-ICE
☑ RLC

<table>
<tr><td>Function</td><td colspan="2">Sets Q-switch pulse width (Nd:YAG) or laser pulse width (CO_2).</td></tr>
<tr><td>Parameters</td><td colspan="2">Pulse width as 16 bits unsigned integer.</td></tr>
<tr><td>Result</td><td colspan="2">Function Set_T2 ok (TRUE) or not ok (FALSE) as boolean.</td></tr>
<tr><td rowspan="3">Calling conventions</td><td>Pascal</td><td>function Set_T2 (usT2: word): bool;</td></tr>
<tr><td>C</td><td>bool Set_T2 (unsigned short usT2);</td></tr>
<tr><td>Basic</td><td>function Set_T2 (byval usT2%) as boolean</td></tr>
<tr><td rowspan="2">Hints</td><td colspan="2">T2 can be set with list command <i>Set_Delays</i> during execution of list.</td></tr>
<tr><td colspan="2">See the hints section for the <i>Set_Delays</i> command for important information on valid value range and default value.</td></tr>
<tr><td>References</td><td colspan="2"><i>Set_Auto_Delay, Set_Jump_Delay, Set_Mark_Delay, Set_Poly_Delay, Set_Laser_Off_Delay, Set_Laser_On_Delay, Set_T1, Set_T3, Set_Delays</i></td></tr>
</table>

Set_T3

☑ SP-ICE
☑ RLC

<table>
<tr><td>Function</td><td colspan="2">Sets FPS-length (YAG-1) or width of laser stand-by pulse (CO_2) or for YAG-2 sets the time from the FPS Pulse to the first laser modulation.</td></tr>
<tr><td>Parameters</td><td colspan="2">FPS length (YAG-1) or width of laser stand-by pulse (CO2) or time from FPS Pulse to laser modulation (YAG-2) as 16 bits, unsigned integer.</td></tr>
<tr><td>Result</td><td colspan="2">Function Set_T3 ok (TRUE) or not ok (FALSE) as boolean.</td></tr>
<tr><td rowspan="3">Calling conventions</td><td>Pascal</td><td>function Set_T3 (usT3: word): bool;</td></tr>
<tr><td>C</td><td>bool Set_T3 (unsigned short usT3);</td></tr>
<tr><td>Basic</td><td>function Set_T3 (byval usT3%) as boolean</td></tr>
<tr><td rowspan="2">Hints</td><td colspan="2">T3 can be set with list command <i>Set_Delays</i> during execution of list.</td></tr>
<tr><td colspan="2">See the hints section for the <i>Set_Delays</i> command for important information on valid value range and default value.</td></tr>
<tr><td>References</td><td colspan="2"><i>Set_Auto_Delay, Set_Jump_Delay, Set_Mark_Delay, Set_Poly_Delay, Set_Laser_Off_Delay, Set_Laser_On_Delay, Set_T1, Set_T2, Set_Delays</i></td></tr>
</table>

Start_Laser_Manually

☑ SP-ICE
☑ RLC

Function	Laser is switched on, if no list is being executed. The current active laser parameters will be used.	
Result	Function Start_Laser_Manually ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Start_Laser_Manually: bool;
	C	bool Start_Laser_Manually (void);
	Basic	function Start_Laser_Manually () as boolean
Hints	If a list is being executed the command is ignored and the global error flag is set to ERR_CMD_NOT_ALLOWED.	
References	*Stop_Laser_Manually*	

Start_Loop

☑ SP-ICE
☑ RLC

Function	Starts continuous output of both lists.	
Result	Function Start_Loop ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Start_Loop: bool;
	C	bool Start_Loop (void);
	Basic	function Start_Loop () as boolean
Hints	The command *Start_Loop* can be used only if both lists are filled with commands, ready for execution and are not already being executed.	
	Execution always starts with list 1 and then continues to list 2, then list 1 and so on.	
	Loop can be finished with *Quit_Loop*, after the last list command in the active list currently being executed.	
	Loop is performed a defined number of times. See *Set_Max_Counts* command for further details.	
References	Quit_Loop, Set_Max_Counts	

Stop_Execution

☑ SP-ICE
☑ RLC

Function	Stops execution of a list immediately, switches off the laser and discards both lists.	
Result	Function Stop_Execution ok (TRUE) or not ok (FALSE) as boolean.	
Calling conventions	Pascal	function Stop_Execution: bool;
	C	bool Stop_Execution (void);
	Basic	function Stop_Execution () as boolean
Hints	The beam stops immediately at the current position, the laser is turned off and both lists will be deleted.	
References	*Stop_Execution_NoClear*	

Stop_Execution_NoClear

☑ SP-ICE
☑ RLC

Function		Stops execution of a list immediately and switches off the laser. This command is identical to *Stop_Execution* except that it does not discard commands from the lists.
Result		Function Stop_Execution ok (TRUE) or not ok (FALSE) as boolean.
Calling conventions	Pascal	function Stop_Execution_NoClear: bool;
	C	bool Stop_Execution_NoClear (void);
	Basic	function Stop_Execution_NoClear () as boolean
Hints		The beam stops immediately at the current position and the laser is turned off.
		This command does not delete the lists.
References		*Stop_Execution*

Stop_Laser_Manually

☑ SP-ICE
☑ RLC

Function		Laser is switched off, if no list is being executed.
Result		Function Stop_Laser_Manually ok (TRUE) or not ok (FALSE) as boolean.
Calling conventions	Pascal	function Stop_Laser_Manually: bool;
	C	bool Stop_Laser_Manually (void);
	Basic	function Stop_Laser_Manually () as boolean
Hints		Active laser parameters are used.
References		*Start_Laser_Manually*

Write_DA

☑ SP-ICE
☑ RLC

Function		Outputs an 8 bit value through, D/A converter to the interface signal ANA_OUT.
Parameters		Output of 16 bit unsigned integer. Value 0 to 255.
Result		Function Write_DA ok (TRUE) or not ok (FALSE) as boolean.
Calling conventions	Pascal	function Write_DA (value: word): bool;
	C	bool Write_DA (unsigned short value);
	Basic	function Write_DA (byval value%) as boolean
Hints		With this command normally the lamp current of YAG lasers is set
		Only the 8 least significant bits define the D/A converter output.
References		*Write_DA_List, Write_Port*

Write_Port

☑ SP-ICE
☑ RLC

Function	Output to the interfaces.			
Parameter	Output of 16 bits unsigned integer.			
	Valid port adresses	26H	Z-Channel	Z-DAC CANNEL[1]
		28H	O-Channel	P-DAC CANNEL[1]
		0CH	Port C	Bit 4 = \Mark In Progress[2) 3) 4)] Bit 5 = \Remote_EXE_1[3] Bit 7 = \Remote_EXE_2[3]
		0AH	Port B	16 bits[6]
				8 bits (PB0 - PB7)[5]
		0EH	Port D	Option[7]
Result	Function Write_Port ok (TRUE) or not ok (FALSE) as boolean.			
Calling conventions	Pascal	function Write_Port (port, value: word);		
	C	bool Write_Port(unsigned short port, unsigned short value);		
	Basic	function Write_Port (byval port%, byval value%) as boolean		
Hints	This is an asynchronous operation. Care must be taken to ensure that this command is only used when the controller is not actively marking to avoid conflict with list commands. This is most critical for Port C which is also used by internal timing, but may apply to any port. The use of *Write_Port_List* is recommended to ensure absolute synchronism with other output commands.			
	Other port addresses than specified above will be ignored.			
	Output to Z-Channel will be overwritten with the execution of the next list command, unless 3rd axis correction has been disabled. See *Set_Mode* control command.			
	The whole word is output to the selected port, affecting all the bits. If only some bits need to be set/reset, previous values sent to other bits must be maintained as well.			
References	*Write_Port_List, Read_Port, Write_DA_List, Write_DA , Set_Mode*			

	SP-ICE	RLC-USB	RLC-PCI
1) Scan Head Interface	☑	☑	☑
2) Restricted Laser / I/O Interface	☐	☐	☑
3) Laser / I/O Interface	☑	☑	☑
4) Extended Laser / I/O Interface	☑	☐	☐
5) Lee compatible Interface	☐	☑	☑
6) Port B	☑	☐	☐
7) Port D	☑	☐	☐

4 ERROR HANDLING COMMANDS

The error handling commands described below are listed in alphabetical order.

Get_Error_Message

☑ SP-ICE
☑ RLC

Function	Reads back the error message for the specified error code in the command. Can be used to create a table with error codes and corresponding messages.	
Parameters	Error code as 32 bit integer and a pointer to a string variable.	
Result	Error message as string.	
Calling conventions	Pascal	function Get_Error_Message (iErrorCode:word): char;
	C	char* Get_Error_Message (int iErrorCode);
	Basic	function Get_Error_Message (byval iErrorCode%) as string
References	*Get_Last_Error_Message, Get_Last_Error_Code*	

Get_Last_Error_Code

☑ SP-ICE
☑ RLC

Function	Reads back the last error code that occurred prior to issuing this command.	
Result	Error code as 16 bits signed integer.	
Calling conventions	Pascal	function Get_Last_Error_Code: int;
	C	int Get_Last_Error_Code (void);
	Basic	function Get_Last_Error_Code () as integer
References	*Get_Error_Message, Get_Last_Error_Message*	

Get_Last_Error_Message

☑ SP-ICE
☑ RLC

Function	Reads back the last error message.	
Result	Error message as string of characters.	
Calling conventions	Pascal	function Get_Last_Error_Message: char;
	C	char* Get_Last_Error_Message (void);
	Basic	function Get_Last_Error_Message () as string
References	*Get_Error_Message, Get_Last_Error_Code*	

Error Codes

Error codes with the corresponding messages are given in the following table:

Error Code	Description
0	No error
1	Not enough memory available
4	Scan card not initialized
5	End-of-list command missing
6	No scan in progress
7	Action not possible (scan in progress)
8	Scan already in progress
9	No vectors available
10	Invalid vector list
12	Invalid list index
13	List incomplete
14	Could not add list command
15	Command not allowed now
16	Parameter out of bounds
19	Invalid pointer
20	File not found
21	Invalid file format
22	Command ignored
26	No card active
28	Card not configured for function
29	Unknown error
30	Invalid card number
36	Card initialization command sequence failed
37	Function call failed
40	The card did not execute the command successfully

5 UNDOCUMENTED COMMANDS

Commands which are dedicated to some special applications are not included in this manual.

Master-Slave

Set_Head_Mask, Get_Head_Mask

PCD

Enable_Custlist

Set_Custlist_Parameters, Get_Custlist_Parameters

Stand-alone version

Load_Corr_File_From_Target_Disk

Output_To_File

Copy_File_To_Target_Disk

Delete_File_On_Target_Disk

Welding

Jump_To_Start_List

Set_JobControl_List

Skip_Var_List, Skip_Var_List_Back

Skip_Counter_List, Set_Counter_List

Read_Port_To_Var_List

JobControl_To_Var_List

3D

Set_3DMode, Reset_3DMode

Set_3DParameters, Get_3DParameters

Jump_Abs_3D, Mark_Abs_3D,

PolA_Abs_3D, PolB_Abs_3D, PolC_Abs_3D

6 UNSUPPORTED COMMANDS

Some of the control commands in the previous version of this manual are no longer supported by the SP-ICE software from versions SP-ICE.dll v11231 and SPICERT.RTB v 11424.

These commands are still in the SPIC_Export.h file for compatibility purposes, but, if called from the application software, return a false flag.

The unsupported commands are

Get_DXDY_Manual

Get_Manual_Move

Get_XY_Manual

Goto_XY

Set_Control_Mode

Set_DXDY_Manual

Set_Manual_Delay

Start_Manual_Move

Start_Manual_Operation

Stop_Manual_Move

Stop_Manual_Operation

Get_SPC1_Mode

The following commands are not implemented in the SP-ICE.dll

Set_Base

Select_List

Select_Valid_List

INDEX