

FISW Steuerungstechnik GmbH

Slave Conformizer V 2.01

Reference Guide

FISW Steuerungstechnik GmbH
Rosenbergstrasse 28
70174 Stuttgart
Germany

Tel. +49 711 121-2443

Fax. +49 711 121-2413

Email: sercos@isw.uni-stuttgart.de

Web: <http://www.isw.unistuttgart.de/sercos>

Slave Conformizer Reference Guide

No part of this document may be reproduced or transmitted in any form or by any means, graphic, electronic, or mechanical, including photocopying, and recording or by any information storage or retrieval system without the prior written permission of the FISW GmbH, unless such copying is expressly permitted by copyright law.

© 2003 FISW Steuerungstechnik GmbH All rights reserved.

While every effort has been made to ensure the accuracy and completeness of all information in this document, FISW GmbH assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors, omissions, or statements result from negligence, accident, or any other cause. FISW GmbH further assumes no liability arising out of the application or use of any product or system described herein; nor any liability for incidental or consequential damages arising from the use of this document. FISW GmbH disclaims all warranties regarding the information contained herein, whether expressed, implied or statutory, including implied warranties of merchantability or fitness for a particular purpose.

FISW Steuerungstechnik GmbH reserves the right to make changes to this document or to the products described herein without further notice.

Microsoft, MS, and Win32 are registered trademarks and Windows and Windows NT are trademarks of Microsoft Corporation.

All other companies and product names may be trademarks or registered trademarks of their respective holders.

Contents

FISW Steuerungstechnik GmbH	1
<i>Slave Conformizer V 2.01</i>	1
Reference Guide V 1.0.....	1
Contents	3
About Slave Conformizer.....	4
Technical Support.....	5
Slave Conformizer Web Site	5
Slave Conformizer Script Language.....	6
Variables	6
Numbers	6
Operators	6
Flow Control	6
Slave Conformizer Script Functions	7
Master Control Functions	7
Communication Phase Switch Functions	12
Service Channel Functions	15
Data Status Functions	15
Name Functions.....	18
Attribute Functions	21
Unit Functions.....	24
Minimum Functions.....	26
Maximum Functions.....	28
Data Functions	30
List Functions.....	35
Command Functions.....	45
Conformance Test Functions	55
Identification Functions.....	55
Error Handling Functions	58
Extended Service Channel Functions	63
Time Slot Functions.....	69
Telegram Functions.....	75
Test Logic Functions	81
Physical Test Functiuons	83
Utility Functions	86
Utility Functions	90
Cyclic Data Functions	94

About Slave Conformizer

The SERCOS Slave Conformizer is a powerful development environment for SERCOS Devices and is capable of testing Slave implementations according to the international standard IEC/EN 61491. It offers developers of SERCOS Slave Devices a rich and powerful script language allowing detailed tests to be easily described. This document describes the Slave Conformizer script functions and syntax.

Technical Support

Technical support questions related to installing and using the Slave Conformizer should be made via Email to sercos@isw.uni-stuttgart.de.

Slave Conformizer Web Site

The Slave Conformizer Customer Web page is located at:

<http://www.sercos.de>

This page provides electronic access to the latest product releases, conformance test script files, documentation and release notes. The documents can be found in the Download Area, Documents for Certification.

Slave Conformizer Script Language

Variables

SSL does not require any type declarations. When SSL-interpreter encounters a new variable name, it automatically creates the variable. If the variable already exists, the SSL-interpreter changes its contents.

Variable names consist of a letter, followed by any number of letters, digits, or underscores. SSL is case sensitive; it distinguishes between uppercase and lowercase letters. `A` and `a` are *not* the same variable.

Numbers

SSL uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. *Scientific notation* uses the letter `e` to specify a power-of-ten scale factor.

All numbers are stored internally using the *double* format specified by the IEEE floating-point standard. Floating-point numbers have a finite *precision* of roughly 16 significant decimal digits and a finite *range* of roughly $-2.23e^{-308}$ to $1.79e^{+308}$.

Operators

Expressions use familiar arithmetic operators and precedence rules. For example:

- + Addition
- Subtraction
- * Multiplication
- / Division
- ^ Power
- () Specify evaluation order

Flow Control

SSL currently only supports simple **if** statement flow control constructs. The **if** statement evaluates a logical expression and executes a single statement when the expression is *true*. Keywords such as **elseif** or **else** which are normally provided for the execution of alternate statements are not yet supported. An **end** keyword, which matches the **if**, terminates the last statement. The statements are delineated by these keywords – no braces or brackets are involved.

Slave Conformizer Script Functions

Master Control Functions

start_master

Starts the real-time master driver.

```
error_code start_master(cycle_time_0, cycle_time_1, baud_rate);
```

Parameters

cycle_time_0

Cycle time of communication phase 0, 1 and 2.

cycle_time_1

Cycle time of communication phase 3 and 4.

baud_rate

Baudrate for the optical transmission.

Return Values

This function returns an `error_code`. A zero value indicates that the real-time master was successfully started. A non-zero value indicates that the real-time master could not be started. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function starts the real-time master driver.

stop_master

Stops the real-time master driver.

```
error_code stop_master();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the real-time master was successfully stopped. A non-zero value indicates that the real-time master could not be stopped. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function stops the real-time master driver.

get_master_status

Return the status of the master.

```
master_status get_master_status();
```

Parameters

Return Values

This function returns the `master_status`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function returns the master status (running or not running).

get_cycle_time

Return the cycle time.

```
cycle_time get_cycle_time();
```

Parameters

Return Values

This function returns the cycle_time in μs for the communication phase 3 and 4. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the current cycle time of the SERCOS interface ring.

get_baud_rate

Return the baudrate.

```
baud_rate get_baud_rate();
```

Parameters

Return Values

This function returns the `baud_rate` of the optical transmission in Mbit/s. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the actually configured baudrate.

Communication Phase Switch Functions

read_phase

Return the phase of the SERCOS interface ring.

```
phase read_phase();
```

Parameters

Return Values

This function returns the phase. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the actual communication phase.

write_phase

Write the phase.

```
error_code write_phase(com_phase);
```

Parameters

com_phase

Communication phase which should be written.

Return Values

This function returns an `error_code`. A zero value indicates that the phase was successfully written. A non-zero value indicates that the phase could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes the phase for the SERCOS ring by sending MSTs with the selected phase. A check if this phase switch is allowed is not be done. Also no IDNs are written within this function. This function is almost used to simulate errors during a phase upshift or downshift.

change_phase

Change the phase of the SERCOS interface ring.

```
error_code change_phase(com_phase);
```

Parameters

com_phase

Communication phase to which a phase shift should be done.

Return Values

This function returns an `error_code`. A zero value indicates that the phase was successfully changed. A non-zero value indicates that the phase could not be changed. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function change the phase for the SERCOS ring. The phase shift is always done in correct order and with the reading and writing of all necessary IDNs for the phase shift.

Service Channel Functions

Data Status Functions

`close_service_channel`

Closes the service channel.

```
error_code close_service_channel();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the service channel was successfully closed. A non-zero value indicates that the service channel could not be changed. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function close the service channel by using the master control word. The service channel can be opened again by calling the function `read_data_status`.

read_data_status

Reads the data status of an IDN.

```
data_status read_data_status(ident_num);
```

Parameters

ident_num

IDN of which the data status should be read.

Return Values

This function returns the `data_status`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function opens the service channel by reading the data status of the specified `ident_number`.

read_data_status_bit

Reads one bit of the data status of an IDN.

```
data_status_bit read_data_status_bit(ident_num, bit_num);
```

Parameters

ident_num

IDN of which the data status should be read.

bit_num

Bit of the data status to be read.

Return Values

This function returns the selected `data_status_bit`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

To read one bit of the data status this function is used. The bit can be assigned to a variable and can be used for further tests.

Name Functions

read_name

Reads the name of an IDN.

```
error_code read_name(ident_num);
```

Parameters

ident_num

IDN of which the name should be read.

Return Values

This function returns an `error_code`. A zero value indicates that the name was successfully read. A non-zero value indicates that the name could not be read. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the name of an IDN and displays it in the protocol window.

read_p_name

Reads the name of a manufacturer specific IDN.

```
error_code read_p_name(ident_num);
```

Parameters

ident_num

Manufacturere specific IDN of which the name should be read.

Return Values

This function returns an `error_code`. A zero value indicates that the name was successfully read. A non-zero value indicates that the name could not be read. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the name of a manufacturer specific IDN and displays it in the protocol window. The IDN must be stated as a number without regarding of the "P" (e.g. P-0-0123 as 123).

write_name

Writes the name of an IDN.

```
error_code write_name(ident_num);
```

Parameters

ident_num

IDN of which the name should be written.

Return Values

This function returns an `error_code`. A zero value indicates that the name was successfully written. A non-zero value indicates that the name could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the name of an IDN and writes it back with the same text to check if the name can be written or not.

Attribute Functions

read_attribute

Reads the attribute of an IDN.

```
attribute read_attribute(ident_num);
```

Parameters

ident_num

IDN of which the attribute should be read.

Return Values

This function returns the attribute. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the attribute of the specified IDN.

read_attribute_bit

Reads one bit of the attribute of an IDN.

```
attribute_bit read_attribute_bit(ident_num, bit_num);
```

Parameters

ident_num

IDN of which the attribute should be read.

bit_num

Bit of the attribute to be read.

Return Values

This function returns the selected attribute_bit. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

To read one bit of the attribute this function is used. The bit can be assigned to a variable and can be used for further tests.

write_attribute

Writes the attribute of an IDN.

```
error_code write_attribute(ident_num, attribute);
```

Parameters

ident_num

IDN of which the attribute should be written.

attribute

Value of the attribute to be written.

Return Values

This function returns an `error_code`. A zero value indicates that the attribute was successfully written. A non-zero value indicates that the attribute could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes a new value to the attribute of an IDN.

Unit Functions

read_unit

Reads the unit of an IDN.

```
error_code read_unit(ident_num);
```

Parameters

ident_num

IDN of which the unit should be read.

Return Values

This function returns an `error_code`. A zero value indicates that the unit was successfully read. A non-zero value indicates that the unit could not be read. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the unit of an IDN and displays it in the protocol window.

write_unit

Writes the unit of an IDN.

```
error_code write_unit(ident_num);
```

Parameters

ident_num

IDN of which the unit should be written.

Return Values

This function returns an `error_code`. A zero value indicates that the unit was successfully written. A non-zero value indicates that the unit could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the unit of an IDN and writes it back with the same text to check if the unit can be written or not.

Minimum Functions

read_min

Reads the minimum value of an IDN.

```
min_value read_min(ident_num);
```

Parameters

ident_num

IDN of which the minimum should be read.

Return Values

This function returns the `min_value`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the minimum value of the specified IDN.

`write_min`

Writes the minimum value of an IDN.

```
error_code write_min(ident_num, min_value);
```

Parameters

ident_num

IDN of which the minimum value should be written.

min_value

Value of the minimum value to be written.

Return Values

This function returns an `error_code`. A zero value indicates that the minimum value was successfully written. A non-zero value indicates that the minimum value could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes a new value to the minimum value of an IDN.

Maximum Functions

read_max

Reads the maximum value of an IDN.

```
max_value read_max(ident_num);
```

Parameters

ident_num

IDN of which the maximum value should be read.

Return Values

This function returns the `max_value`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the maximum value of the specified IDN.

`write_max`

Writes the maximum value of an IDN.

```
error_code write_max(ident_num, max_value);
```

Parameters

ident_num

IDN of which the maximum value should be written.

max_value

Value of the maximum value to be written.

Return Values

This function returns an `error_code`. A zero value indicates that the maximum value was successfully written. A non-zero value indicates that the maximum value could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes a new value to the maximum value of an IDN.

Data Functions

read_data

Reads the data of an IDN.

```
data read_data(ident_num);
```

Parameters

ident_num

IDN of which the data should be read.

Return Values

This function returns the data of an IDN. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the data of the specified IDN. The function returns an error, if the IDN is a list. Then the list functions should be used instead.

read_data_bit

Reads one bit of the data of an IDN.

```
data_bit read_data_bit(ident_num, bit_num);
```

Parameters

ident_num

IDN of which the data should be read.

bit_num

Bit of the data to be read.

Return Values

This function returns the `data_bit`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the specified bit of the data of the specified IDN. The function returns an error, if the IDN is not an integer value. Then the function can not be used for this IDN.

write_data

Writes the data value of an IDN.

```
error_code write_data(ident_num, data);
```

Parameters

ident_num

IDN of which the data value should be written.

data

Value of the data to be written.

Return Values

This function returns an `error_code`. A zero value indicates that the data value was successfully written. A non-zero value indicates that the data value could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes a new value to the data value of an IDN. The function returns an error, if the IDN is a list. Then the list functions should be used instead.

read_p_data

Reads the data of a manufacturer specific IDN.

```
data read_p_data(ident_num);
```

Parameters

ident_num

Manufacturer specific IDN of which the data should be read.

Return Values

This function returns the data of a manufacturer specific IDN. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the data of the specified manufacturer specific IDN. The parameter `ident_num` is used without regarding the "P" (e.g. for P-0-0123 use 123). The function returns an error, if the IDN is a list. Then the list functions should be used instead.

write_p_data

Writes the data value of a manufacturer specific IDN.

```
error_code write_p_data(ident_num, data);
```

Parameters

ident_num

Manufacturer specific IDN of which the data value should be written.

data

Value of the data to be written.

Return Values

This function returns an `error_code`. A zero value indicates that the data value was successfully written. A non-zero value indicates that the data value could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes a new value to the data value of a manufacturer specific IDN. The function returns an error, if the IDN is a list. Then the list functions should be used instead.

List Functions

is_list

Checks if an IDN is a list.

```
list_flag is_list(ident_num);
```

Parameters

ident_num

IDN to be checked.

Return Values

This function returns a `list_flag`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function checks if the specified IDN is of a list type (`list_flag = 1`) or not (`list_flag = 0`) by reading and interpreting the attribute.

read_list_length

Reads the actual length of a list.

```
list_length read_list_length(ident_num);
```

Parameters

ident_num

IDN for the list of which the length should be read.

Return Values

This function returns the list_length. A function's error code can also be determined after a function has been called by calling the function get_last_error directly afterwards.

Comments

This function returns the actual length of a list by reading the list. List_length means the number of elements within the list (not the length in bytes). So the list_length is the quotient of the list length in bytes and the length of every element. If the IDN is not a list type the function can not be used.

read_list_max_length

Reads the maximum length of a list.

```
list_max_length read_list_max_length(ident_num);
```

Parameters

ident_num

IDN for the list of which the maximum length should be read.

Return Values

This function returns the `list_max_length`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function returns the maximum length of a list by reading the list. `List_max_length` means the maximum number of elements within the list (not the maximum length in bytes). So the `list_max_length` is the quotient of the maximum list length in bytes and the length of every element. If the IDN is not a list type the function can not be used.

read_list

Reads the list.

```
error_code read_list(ident_num);
```

Parameters

ident_num

IDN for the list to be read.

Return Values

This function returns an `error_code`. A zero value indicates that the list was successfully read. A non-zero value indicates that the list could not be read. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the list specified by the IDN and displays all list elements in the protocol window. If the IDN is not a list type the function also returns an error.

delete_list

Deletes the list.

error_code delete_list(ident_num);

Parameters

ident_num

IDN for the list to be deleted.

Return Values

This function returns an `error_code`. A zero value indicates that the list was successfully deleted. A non-zero value indicates that the list could not be deleted. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function deletes the list specified by the IDN by setting the actual length of the list to zero. If the IDN is not a list type the function also returns an error.

read_text

Reads the data of an IDN of ASCII type.

```
error_code read_text(ident_num);
```

Parameters

ident_num

IDN for the text to be read.

Return Values

This function returns an `error_code`. A zero value indicates that the text was successfully read. A non-zero value indicates that the text could not be read. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the data of an IDN of the type ASCII string and displays the string in the protocol window. If the IDN is not of the type ASCII string the function also returns an error.

read_list_element

Reads one element of a list.

list_element read_list_element(ident_num, element_num);

Parameters

ident_num

IDN of the list which should be read.

Return Values

This function returns the list_element. A function's error code can also be determined after a function has been called by calling the function get_last_error directly afterwards.

Comments

This function reads the list specified by the IDN, selects the specified element and returns it. If the IDN is not of a list type, the function returns an error.

write_list_element

Writes one element of a list.

```
error_code write_list_element(ident_num, element_num, data);
```

Parameters

ident_num

IDN of the list which should be written.

element_num

Element of the list to be written.

data

Value of the element of the list to be written.

Return Values

This function returns an `error_code`. A zero value indicates that the element was successfully changed. A non-zero value indicates that the element could not be changed. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function changes the value of the specified list element by reading the complete list, replacing the specified element and writing back the whole new list. If the IDN is not of a list type, the function returns an error.

insert_list_element

Inserts one element in a list.

```
error_code insert_list_element(ident_num, element_num, data);
```

Parameters

ident_num

IDN of the list which should be written.

element_num

Element of the list to be inserted.

data

Value of the element of the list to be inserted.

Return Values

This function returns an `error_code`. A zero value indicates that the element was successfully inserted. A non-zero value indicates that the element could not be inserted. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function inserts one additional element at the specified area in the list by reading the complete list, inserting the new element and writing back the whole new list. If the IDN is not of a list type, the function returns an error.

delete_list_element

Removes one element from a list.

error_code delete_list_element(ident_num, element_num);

Parameters

ident_num

IDN of the list which should be written.

element_num

Element of the list to be removed.

Return Values

This function returns an `error_code`. A zero value indicates that the element was successfully removed. A non-zero value indicates that the element could not be removed. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function removes one element at the specified area in the list by reading the complete list, deleting this element and writing back the hole modified list. If the IDN is not of a list type, the function returns an error.

Command Functions

is_command

Checks if an IDN is a command.

```
command_flag is_command(ident_num);
```

Parameters

ident_num

IDN to be checked.

Return Values

This function returns the `command_flag`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function checks if the specified IDN is a command (`command_flag = 1`) or not (`command_flag = 0`) by reading and interpreting the attribute.

write_command

Does the command handling of an IDN.

```
error_code write_command(ident_num);
```

Parameters

ident_num

IDN of which the command handling should be done.

Return Values

This function returns an `error_code`. A zero value indicates that the command handling was successfully executed. A non-zero value indicates that the command handling could not be done. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function does the whole handling of a command process. It sets, enables the command, waits for the command modification bit of the drive status word and clears the command. If the IDN is not a command, the function returns an error.

clear_device_error

Clears the error in a device.

```
error_code clear_device_error();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the error clearing was successfully executed. A non-zero value indicates that the error clearing could not be done. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function resets the device by executing the command IDN 99.

set_command

Sets the command.

```
error_code set_command(ident_num);
```

Parameters

ident_num

IDN of the command to be set.

Return Values

This function returns an `error_code`. A zero value indicates that the command was successfully set. A non-zero value indicates that the command could not be set. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sets a command by setting the bit 0 of the data to 1. If the IDN is a not a command, the functions returns an error.

enable_command

Enables the command.

```
error_code enable_command(ident_num);
```

Parameters

ident_num

IDN of the command to be enabled.

Return Values

This function returns an `error_code`. A zero value indicates that the command was successfully enabled. A non-zero value indicates that the command could not be enabled. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function enables a command by setting the bit 1 of the data to 1. If the IDN is a not a command, the functions returns an error.

start_command

Starts the command.

```
error_code start_command(ident_num);
```

Parameters

ident_num

IDN of the command to be started.

Return Values

This function returns an `error_code`. A zero value indicates that the command was successfully started. A non-zero value indicates that the command could not be started. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function starts (sets and enables) a command by setting the bit 1 and the bit 0 of the data to 1. If the IDN is not a command, the function returns an error.

suspend_command

Suspends the command.

```
error_code suspend_command(ident_num);
```

Parameters

ident_num

IDN of the command to be suspended.

Return Values

This function returns an `error_code`. A zero value indicates that the command was successfully suspended. A non-zero value indicates that the command could not be suspended. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function suspends a command by resetting the bit 1 of the data to 0. If the IDN is a not a command, the functions returns an error.

abort_command

Aborts the command.

```
error_code abort_command(ident_num);
```

Parameters

ident_num

IDN of the command to be aborted.

Return Values

This function returns an `error_code`. A zero value indicates that the command was successfully aborted. A non-zero value indicates that the command could not be aborted. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function aborts a command by resetting the bit 0 of the data to 0. If the IDN is a not a command, the functions returns an error.

await_command

Waits for the end of a procedure command handling.

```
error_code await_command();
```

Parameters

Return Values

This function returns a `error_code`. A zero value indicates that a previously started command has finished executing. A non-zero value indicates that the command has not yet finished executing. A function's error code can also be determined after a function has been called by calling the function `get_last_error` afterwards.

Comments

`await_command` waits 1000 communication cycles for the command modification bit (bit 5) of the device's status word to change from 0 to 1. After a command has completed execution, the function `read_data_status` can be used to determine whether the command was executed successfully or not.

clear_command

Clears a command.

```
error_code clear_command(ident_num);
```

Parameters

ident_num

IDN of the command to be cleared.

Return Values

This function returns an `error_code`. A zero value indicates that the command was successfully cleared. A non-zero value indicates that the command could not be cleared. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function clears (abort and suspend) a command by resetting the bit 0 and the bit1 of the data to 0. If the IDN is a not a command, the functions returns an error.

Conformance Test Functions

Identification Functions

get_active_device

Return the active device.

```
active_device get_active_device();
```

Parameters

Return Values

This function returns the `active_device`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

All the test and the script functions are done for one selected device in the ring (the active device), which can be selected by the function `select_active_device` or by the GUI. This function returns the address of the active device.

test_device_address

Test if a device answers on an address.

```
error_code test_device_address(address);
```

Parameters

address

Address to be checked.

Return Values

This function returns an `error_code`. A zero value indicates that ATs with the specified address were received. A non-zero value indicates that no ATs with the specified address were received. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sends MDTs with the specified address in communication phase 1 and 2 and waits up to 10 cycles for receiving ATs with the same address. The number of wait cycles is displayed in the protocol window.

scan_for_devices

Scan the ring for the available devices.

```
num_of_devices scan_for_devices();
```

Parameters

ident_num

IDN of the command to be cleared.

Return Values

This function returns the `num_of_devices`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sends MDTs with addresses from 1 up to 254 address in communication phase 1 and 2 and waits up to 10 cycles for receiving ATs with the same address as in the MDT. The number of detected devices is returned and the addresses of this devices are displayed in the protocol window.

Error Handling Functions

read_device_status

Reads the device status word.

```
device_status read_device_status();
```

Parameters

Return Values

This function returns the `device_status`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the device status word of the active device.

read_device_status_bit

Reads one bit of the device status word.

```
device_status_bit read_device_status_bit(bit_num);
```

Parameters

bit_num

Number of the bit of the device status word to be read.

Return Values

This function returns the `device_status_bit`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the specified bit of device status word of the active device. This bit can be used for further tests.

simulate_mdt_failure

Simulates the failure of MDTs.

```
error_code simulate_mdt_failure(num_of_fails);
```

Parameters

num_of_fails

Number of MDT failures.

Return Values

This function returns an `error_code`. A zero value indicates that the failure of MDTs was possible. A non-zero value indicates that the failure of MDTs was not possible. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function stops the sending of MDTs for the specified number of cycles. After this cycles the MDTs are sent again. So it is possible to simulate a defined number of MDT failures to test the error handling of slave devices.

simulate_mst_failure

Simulates the failure of MSTs.

error_code simulate_mst_failure(num_of_fails);

Parameters

num_of_fails

Number of MST failures.

Return Values

This function returns an **error_code**. A zero value indicates that the failure of MSTs was possible. A non-zero value indicates that the failure of MSTs was not possible. A function's error code can also be determined after a function has been called by calling the function **get_last_error** directly afterwards.

Comments

This function stops the sending of MSTs for the specified number of cycles. After this cycles the MSTs are sent again. So it is possible to simulate a defined number of MST failures to test the error handling of slave devices.

simulate_mdt_jitter

Simulates jitter in the sending of MDTs.

```
error_code simulate_mst_failure(mdt_jitter);
```

Parameters

mdt_jitter

MDT jitter to be simulated.

Return Values

This function returns an `error_code`. A zero value indicates that the simulation of jitter was possible. A non-zero value indicates that the simulation of jitter was not possible. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sends two MDTs in following cycles at the sending time plus the value of the jitter. So the sending time of the MDT can be varied to test the error handling of slave devices.

Extended Service Channel Functions

read_length

Reads the length of an element of an IDN.

```
element_length read_length(ident_num, element_num);
```

Parameters

ident_num

IDN of to be read.

element_num

Index of the Element of which the length should be read.

- | | |
|---|---------------|
| 1 | data status |
| 2 | Name |
| 3 | Attribute |
| 4 | Unit |
| 5 | Minimum Value |
| 6 | Maximum Value |
| 7 | Data |

Return Values

This function returns the `element_length`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the length of an element of an IDN in bytes. With this function also the length of a list in bytes can be appointed.

`write_min_ex`

Writes the minimum value of an IDN with the specified length.

```
error_code write_min_ex(ident_num, min_value, length);
```

Parameters

ident_num

IDN of which the minimum value should be written.

min_value

Value to be written.

length

Length of the written value.

Return Values

This function returns an `error_code`. A zero value indicates that the minimum value was successfully written. A non-zero value indicates that the minimum value could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes the minimum value of the specified IDN with the specified value and the specified length. A check if the length is correct is not be done. So a write access with a wrong length can be simulated using this function to test the error handling of a slave device.

`write_max_ex`

Writes the maximum value of an IDN with the specified length.

error_code `write_max_ex(ident_num, min_value, length);`

Parameters

ident_num

IDN of which the maximum value should be written.

min_value

Value to be written.

length

Length of the written value.

Return Values

This function returns an `error_code`. A zero value indicates that the maximum value was successfully written. A non-zero value indicates that the maximum value could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes the maximum value of the specified IDN with the specified value and the specified length. A check if the length is correct is not be done. So a write access with a wrong length can be simulated using this function to test the error handling of a slave device.

write_data_ex

Writes the data of an IDN with the specified length.

error_code write_data_ex(ident_num, min_value, length);

Parameters

ident_num

IDN of which the data should be written.

min_value

Value to be written.

length

Length of the written value.

Return Values

This function returns an `error_code`. A zero value indicates that the data was successfully written. A non-zero value indicates that the data could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes the data of the specified IDN with the specified value and the specified length. A check if the length is correct is not be done. So a write access with a wrong length can be simulated using this function to test the error handling of a slave device.

get_readable_ident

Searches for an IDN with the specified data type.

readable_ident get_readable_ident(ident_type);

Parameters

ident_type

Type for which an IDN should be searched.

- | | |
|---|-----------------|
| 1 | 2 byte |
| 2 | 4 byte |
| 4 | variable 1 byte |
| 5 | variable 2 byte |
| 6 | variable 4 byte |

Return Values

This function returns an readable_ident. If no IDN could be found, it returns DBL_MAX. A function's error code can also be determined after a function has been called by calling the function get_last_error directly afterwards.

Comments

This function searches for an IDN with the specified data type. So if special IDNs should be used for further test, they can be searched automatically.

get_writeable_ident

Searches for a writeable IDN with the specified data type.

```
writeable_ident get_writeable_ident(ident_type);
```

Parameters

ident_type

Type for which an IDN should be searched.

3	2 byte
4	4 byte
7	variable 1 byte
8	variable 2 byte
9	variable 4 byte

Return Values

This function returns a `writeable_ident`. If no IDN could be found, it returns `DBL_MAX`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function searches for an IDN with the specified data type which can be also written. So if special IDNs that can be written should be used for further test they can be searched automatically.

Time Slot Functions

do_time_slots

Does a complete time slot calculation.

```
error_code do_time_slots();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the time slot calculation was successfully done. A non-zero value indicates that the time slot calculation could not be done. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function does a complete time slot calculation. First all necessary IDNs (S-0-0003, S-0-0004, S-0-0005, S-0-0088, S-0-0090, S-0-0096 and S-0-0087) are read from the drive and the time slots are calculated. The calculated time slots (S-0-001, S-0-0002, S-0-0089, S-0-0006, S-0-0007 and S-0-0008) are then written to the slave devices.

read_time_values

Reads the necessary time values for a time slot calculation.

```
error_code read_time_values();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the time values were successfully read. A non-zero value indicates that the time values could not be read completely. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the necessary IDNs for the time slot calculation from the slave device (S-0-0003, S-0-0004, S-0-0005, S-0-0088, S-0-0090, S-0-0096 and S-0-0087) and save them internally.

calc_time_slots

Calculate the time slots for a slave device.

```
error_code calc_time_slot(idn3, idn4, idn5, idn88, idn90, idn96, idn87);
```

Parameters

idn3

Value of IDN S-0-0003.

idn4

Value of IDN S-0-0004.

idn5

Value of IDN S-0-0005.

idn88

Value of IDN S-0-0088.

idn90

Value of IDN S-0-0090.

idn96

Value of IDN S-0-0096.

idn87

Value of IDN S-0-0087.

Return Values

This function returns an `error_code`. A zero value indicates that the time slot calculation was successfully done. A non-zero value indicates that the time slot calculation could not be done. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function calculate the time slots for the slave devices (t1, t2, t3 and t4) and saves the values internally. The values are also displayed in the protocol window.

write_time_slots

Writes the time slots to the slave device.

```
error_code write_time_slot();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the time slots were successfully written. A non-zero value indicates that the time slots could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes the internally saved NC cycle time , the communication cycle time and the time slots t1, t2, t3 and t4 to the IDNs S-0-001, S-0-0002, S-0-0089, S-0-0006, S-0-0007 and S-0-0008.

get_max_jitter

Returns the maximum jitter.

```
max_jitter get_max_jitter();
```

Parameters

Return Values

This function returns the max_jitter. A function's error code can also be determined after a function has been called by calling the function get_last_error directly afterwards.

Comments

This function returns the maximum allowed jitter for the selected baud rate and communication cycle time in communication phase 3 and 4.

set_t2

Set the time t2.

```
error_code set_t2(t2);
```

Parameters

t2

Sending time of the MDT.

Return Values

This function returns an `error_code`. A zero value indicates that the sending time of the MDT was successfully set. A non-zero value indicates that the sending time of the MDT could not be set. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sets the sending time of the MDT in communication phase 1 or 2. The function can only be called in communication phase 1 or 2.

Telegram Functions

set_telegram_type

Sets the telegram type.

```
error_code set_telegram_type(tel_type);
```

Parameters

tel_type

Telegram type to be set.

Return Values

This function returns an `error_code`. A zero value indicates that the telegram type was successfully set. A non-zero value indicates that the time slots could not be set. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sets the telegram type to the specified telegram type. The telegram type is saved internally and is not written directly to IDN S-0-0015. This is done during the phase upshift.

get_mdt_config

Shows the configuration of the MDT.

```
error_code get_mdt_config();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the MDT configuration was successfully read. A non-zero value indicates that the MDT configuration could not be read. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the telegram configuration of the MDT and shows the telegram type and the cyclically configured IDNs of the MDT in the protocol window.

get_at_config

Shows the configuration of the AT.

```
error_code get_at_config();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the AT configuration was successfully read. A non-zero value indicates that the AT configuration could not be read. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the telegram configuration of the AT and shows the telegram type and the cyclically configured IDNs of the AT in the protocol window.

`get_mdt_length`

Reads the length of the MDT.

```
mdt_length get_mdt_length();
```

Parameters

Return Values

This function returns the `mdt_length`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the length of the MDT in bytes.

get_at_length

Reads the length of the AT.

```
at_length get_at_length();
```

Parameters

Return Values

This function returns the `at_length`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the length of the AT in bytes.

test_telegram_7

Tests configurations for the application telegram.

```
error_code test_telegram_7(at_length, mdt_length);
```

Parameters

at_length

Maximum length for the AT in byte.

mdt_length

Maximum length for the MDT in byte.

Return Values

This function returns an `error_code`. A zero value indicates that the application telegram was successfully set for all configurations. A non-zero value indicates that the application telegram could not be set for all configurations. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sets different configurations for the application telegram by using a special test algorithm. An application telegram with the maximum allowed length is configured and a phase upshift to communication phase 4 is done. Then the first cyclically configured IDN is replaced with a new one and so on until all allowed IDNs were configured at least one time cyclically.

Test Logic Functions

set_operation_mode

Sets the primary operation mode.

```
error_code set_operation_mode(op_mode);
```

Parameters

op_mode

Primary operation mode to be set.

Return Values

This function returns an `error_code`. A zero value indicates that the telegram type was successfully set. A non-zero value indicates that the time slots could not be set. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sets the primary operation mode the specified operation mode. The operation mode is saved internally and is not written directly to IDN S-0-0032. This is done during the phase upshift.

get_operation_mode

Reads the primary operation mode.

```
op_mode get_operation_mode();
```

Parameters

Return Values

This function returns the `op_mode`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the configured primary operation mode that is saved internally, not the actual value of the IDN S-0-0032.

Physical Test Functions

set_trans_mode

Sets the transmission mode of the optical transmitter.

```
error_code set_trans_mode(mode);
```

Parameters

mode

Transmission mode to be set.

- 1 normal operation
- 2 zero bit stream
- 3 continuous light

Return Values

This function returns an `error_code`. A zero value indicates that the transmission mode was successfully set. A non-zero value indicates that the transmission mode could not be set. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sets the mode of the optical transmission (e.g. for test purpose).

set_trans_power

Sets the transmission power of the optical transmitter.

error_code set_trans_power(power);

Parameters

power

Transmission mode to be set.

- 1 low power
- 2 medium power
- 3 medium high power
- 4 high power

Return Values

This function returns an `error_code`. A zero value indicates that the transmission power was successfully set. A non-zero value indicates that the transmission power could not be set. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sets the optical output power of the optical transmission.

get_trans_power

Reads the transmission power of the optical transmitter.

```
trans_power get_trans_power();
```

Parameters

Return Values

This function returns the trans_power. A function's error code can also be determined after a function has been called by calling the function get_last_error directly afterwards.

Comments

This function reads the optical output power of the optical transmission.

Utility Functions

get_last_error

Returns the last occurred error.

```
last_error_code abort_test();
```

Parameters

Return Values

The function returns the last_error_code.

Comments

This function returns the last occurred error. This error can be caused by a communication error or by the return value of a function which can not be executed correctly. With this function the result of a read function can also be checked.

fail_test

Signals a test failure in a script file.

```
error_code fail_test();
```

Parameters

Return Values

The function returns a `error_code`. A zero value indicates that the test fail was successfully signaled. A non-zero value indicates that the test fail could not be signaled.

Comments

This function causes the currently executing script to signal an error and to increase the test failure counter. A script file calling this function is considered to have failed, but the script execution is not to be stopped.

abort_test

Aborts a test script file which is being executed.

error_code abort_test();

Parameters

Return Values

The function returns a `error_code`. A zero value indicates that the test was successfully aborted. A non-zero value indicates that the test could not be aborted.

Comments

This function causes the currently executing script be terminated prematurely and should be used in script files instead of `fail_test` if a continuation of the test is either not possible or does not make any sense. A script file calling this function is considered to have failed.

skip_test

Aborts a test script file which is being executed.

```
error_code skip_test();
```

Parameters

Return Values

The function returns a `error_code`. A zero value indicates that the test was successfully skipped. A non-zero value indicates that the test could not be skipped.

Comments

This function causes the currently executing script be terminated prematurely and should be used in script files, if a condition for further tests is not been given, but this behavior is not an error. A script file calling this function is not considered to have failed.

Utility Functions

pause

Interrupts the execution of a test script.

```
error_code pause(time_in_seconds);
```

Parameters

time_in_seconds

Time the execution of the script file should be interrupted in seconds.

Return Values

This function returns an `error_code`. A zero value indicates that the script was successfully interrupted. A non-zero value indicates that the script file could not be interrupted. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function interrupts the execution of the script file for the specified time in seconds.

wait_for_user

Makes the script file waiting for a user interaction.

```
error_code wait_for_user('[text]');
```

Parameters

[text]

Text to be displayed in the dialog.

Return Values

This function returns an `error_code`. A zero value indicates that the dialog was successfully displayed. A non-zero value indicates that the dialog could not be displayed. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function makes the script file waiting for a user interaction. The specified text is displayed in a dialog window. The execution of the script file is stopped until the user acknowledges or cancels the dialog.

round

Round a value to an integer value.

return_value round(value);

Parameters

value

Value to be round.

Return Values

This function returns the return_value. A function's error code can also be determined after a function has been called by calling the function get_last_error directly afterwards.

Comments

This function rounds the specified value to an integer value.

print_value

Prints a value in the protocol window.

```
error_code print_value(value);
```

Parameters

value

Value to be printed.

Return Values

This function returns an `error_code`. A zero value indicates that the value was successfully print. A non-zero value indicates that the value could not be print. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function prints the specified value in the protocol window.

Cyclic Data Functions

read_master_control

Reads the master control word.

```
master_control read_master_control();
```

Parameters

Return Values

This function returns the `master_control`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the master control word.

read_master_control_bit

Reads one bit of the master control word.

```
master_control_bit read_master_control_bit(bit_num);
```

Parameter

bit_num

Bit of the master control word to be read.

Return Values

This function returns the `master_control_bit`. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads one bit of the master control word. This function have no real-time behavior.

write_master_control_bit

Writes one bit of the master control word.

```
error_code write_master_control_bit(bit_num, master_control_bit);
```

Parameter

bit_num

Bit of the master control word to be written.

master_control_bit

Value of the bit to be written.

Return Values

This function returns an `error_code`. A zero value indicates that the master control bit was successfully written. A non-zero value indicates that the master control bit could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes one bit of the master control word. It should only be used to write the bits 13, 14 and 15 in communication phase 4, because the other bits of the master control word are handled by the real-time driver. To switch the operation mode, a separate function (`switch_operation_mode`) is defined.

set_device_status_start_trigger

Sets a start trigger by using one bit of the device status word.

```
error_code set_device_status_start_trigger(bit_num, bit_value);
```

Parameter

bit_num

Bit of the device status word to set the trigger.

bit_value

Value of the bit to set the trigger.

Return Values

This function returns an `error_code`. A zero value indicates that the trigger was successfully set. A non-zero value indicates that the trigger could not be set. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sets a trigger for the specified bit number of the device status word and for the specified value of this bit. If the mentioned bit of the device status word changes to the mentioned value an internal counter is started and counts the cycles until the condition of the stop function (`set_device_status_stop_trigger` or `set_master_control_stop_trigger`) is fulfilled. The value of the counter can be read by `get_counter_value`.

set_device_status_stop_trigger

Sets a stop trigger by using one bit of the device status word.

```
error_code set_device_status_stop_trigger(bit_num, bit_value);
```

Parameter

bit_num

Bit of the device status word to set the stop trigger.

bit_value

Value of the bit to set the trigger.

Return Values

This function returns an `error_code`. A zero value indicates that the trigger was successfully set. A non-zero value indicates that the trigger could not be set. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sets a stop trigger for the specified bit number of the device status word and for the specified value of this bit. If the mentioned bit of the device status word changes to the mentioned value the internal counter which is started is by the trigger start functions (`set_device_status_start_trigger` or `set_master_control_start_trigger`) is stopped. The value of the counter can be read by `get_counter_value`.

set_master_control_start_trigger

Sets a start trigger by using one bit of the master control word.

error_code set_master_control_start_trigger(bit_num, bit_value);

Parameter

bit_num

Bit of the master control word to set the trigger.

bit_value

Value of the bit to set the trigger.

Return Values

This function returns an `error_code`. A zero value indicates that the trigger was successfully set. A non-zero value indicates that the trigger could not be set. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sets a trigger for the specified bit number of the master control word and for the specified value of this bit. If the mentioned bit of the master control word changes to the mentioned value an internal counter is started and counts the cycles until the condition of the stop function (`set_device_status_stop_trigger` or `set_master_control_stop_trigger`) is fulfilled. The value of the counter can be read by `get_counter_value`.

set_master_control_stop_trigger

Sets a stop trigger by using one bit of the master control word.

```
error_code set_master_control_word_stop_trigger(bit_num, bit_value);
```

Parameter

bit_num

Bit of the master control word control to set the trigger.

bit_value

Value of the bit to set the trigger.

Return Values

This function returns an `error_code`. A zero value indicates that the trigger was successfully set. A non-zero value indicates that the trigger could not be set. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function sets a stop trigger for the specified bit number of the master control word and for the specified value of this bit. If the mentioned bit of the master control word changes to the mentioned value the internal counter which is started is by the trigger start functions (`set_device_status_start_trigger` or `set_master_control_start_trigger`) is stopped. The value of the counter can be read by `get_counter_value`.

get_counter_value

Reads the value of the internal counter.

```
error_code get_counter_value();
```

Parameter

Return Values

This function returns the counter_value. A function's error code can also be determined after a function has been called by calling the function get_last_error directly afterwards.

Comments

This function reads the value (in number of cycles) of the internal counter started by the trigger start functions (set_device_status_start_trigger or set_master_control_start_trigger) and stopped by the stop trigger functions (set_device_status_stop_trigger or set_master_control_stop_trigger).

switch_operation_mode

Switches the operation mode.

error_code switch_operation_mode(op_mode);

Parameter

op_mode

New value of the operation mode.

Return Values

This function returns an `error_code`. A zero value indicates that the operation mode was successfully switched. A non-zero value indicates that the operation mode could not be switched. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function switches the operation mode of a drive by setting or resetting the bits 8,9 and 11 of the master control word.

write_cyclic_data

Writes a value to the cyclic data.

```
error_code write_cyclic_data(ident_num, data);
```

Parameter

ident_num

IDN in the cyclic configured data to which the data should be written.

data

Value to be written.

Return Values

This function returns an `error_code`. A zero value indicates that the data was successfully written. A non-zero value indicates that the data could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function writes data to a cyclic configured IDN in the MDT. The data is written every communication cycle with the same value. A check on limits of the data is not be done.

write_cyclic_ramp_data

Generates a ramp for cyclic data.

```
error_code write_cyclic_ramp_data(ident_num, data, add_data, increase_cycles,  
                                constant_cycles, decrease_cycles);
```

Parameter

ident_num

IDN in the cyclic configured data to which the data should be written.

data

Start value of the ramp generation.

add_data

Value of the step of the ramp generation per cycle.

increase_cycles

Number of increase cycles.

constant_cycles

Number of constant cycles.

decrease_cycles

Number of decrease cycles.

Return Values

This function returns an `error_code`. A zero value indicates that the ramp generation parameter were successfully written. A non-zero value indicates that the ramp generation parameter could not be written. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function generates a ramp for the specified cyclically configured IDN. The ramp starts with writing the *data* value to the cyclically configured IDN. A check for this value is not be done, so make sure that this value is equal to the corresponding actual value. For the number of *increase_cycles* the value *add_data* is added to the cyclically configured IDN every cycle. A check for this value is not be done. For the number of *constant_cycles* the value stays the same. For the number of *decrease_cycles* the value *add_data* is subtracted from the cyclically configured IDN every cycle. A check on limits of the data is not be done. The function returns after setting the parameter for the ramp configuration and does not wait for the end of the ramp generation, so the user can stop this movement e.g. in emergency situations.

read_cyclic_data

Reads the value of the cyclic data.

```
cyclic_data read_cyclic_data(ident_num);
```

Parameter

ident_num

IDN in the cyclic configured data of which the data should be read.

Return Values

This function returns the cyclic data. A function's error code can also be determined after a function has been called by calling the function `get_last_error` directly afterwards.

Comments

This function reads the value of a cyclically configured IDN in the AT. The function has no real-time behavior and is not deterministic.