

FISW Steuerungstechnik GmbH

Master 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.uni-stuttgart.de/sercos>

Master 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
Master Conformizer V 2.01	1
Reference Guide.....	1
Contents	3
About Master Conformizer	4
Technical Support.....	5
Master Conformizer Web Site	5
Master Conformizer Script Language	6
Variables	6
Numbers	6
Operators	6
Flow Control	6
Master Conformizer Script Functions	7
Slave Control Functions	7
Communication Phase Functions	11
Service Channel Functions	13
Name Functions.....	13
Attribute Functions	15
Unit Functions.....	17
Minimum Functions.....	18
Maximum Functions	19
Data Functions	20
List Functions.....	25
Command Functions.....	35
Conformance Test Functions	36
Identification Functions.....	36
Error Handling Functions	41
Extended Service Channel Functions	45
Time Slot Functions.....	48
Telegram Functions.....	56
Physical Test Functions	59
Test Logic Functions	60
Protocol Functions.....	66
Phase Protocol Functions	66
Service Channel Protocol Functions	68
Utility Functions	74

About Master Conformizer

The SERCOS Master 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 Master Devices a rich and powerful script language allowing detailed tests to be easily described. This document describes the Master Conformizer script functions and syntax.

Technical Support

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

Master Conformizer Web Site

The Master 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.

Master 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.

Master Conformizer Script Functions

Slave Control Functions

stop_slave

Stops the real-time slave driver.

```
error_code stop_slave();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the real-time slave was successfully stopped. A non-zero value indicates that the real-time slave 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 slave driver.

get_slave_status

Return the status of the slave.

```
master_status get_slave_status();
```

Parameters

Return Values

This function returns the slave_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 status of the real-time slave driver (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 baud rate.

Communication Phase Functions

read_phase

Return the phase of the active device.

```
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 returns the phase of the active device. If a device does not take part in the communication, phase 0 is returned.

read_mst_phase0_counter

Return the number of MST received in communication phase 0.

```
num_of_mst_phase0 read_mst_phase0_counter();
```

Parameters

Return Values

This function returns the num_of_mst_phase0. 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 MSTs received in communication phase 0. After 10 received MSTs the counter is stopped, because the master has to send 10 MST to close the ring.

Service Channel Functions

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 from the active device 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 of the active device 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).

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 of the active device.

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 of the active device this function is used. The bit can be assigned to a variable and can be used for further tests.

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 of the active device and displays it in the protocol window.

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 of the active device.

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 of the active device.

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 of the active device. 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 of the active device. 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 of the active device. 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 of the active device. 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 of the active device. 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 of the active device 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 is_list(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. `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 is_list(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. 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 of the active device 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 of the active device 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 active device 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.

element_num

Index of the element to 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 specified element of the list of the active device.

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 writes the specified element of the list of the active device with a new value.

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 of the active device. 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 of the active device. If the IDN is not of a list type, the function returns an error.

Command Functions

is_command

Checks if an IDN is a list.

```
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.

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 Master Conformizer (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.

set_active_device

Return the active device.

```
error_code set_active_device(device_address);
```

Parameters

device_address

Address of the device to be the active device.

Return Values

This function returns an `error_code`. A zero value indicates that the active device was successfully set. A non-zero value indicates that the active device 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 active device for which all the script functions are done.

test_device_address

Test if a device address is configured.

```
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 the address is configured in the Master Conformizer. A non-zero value indicates that the address is not configured. 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 address is configured in the Master Conformizer.

scan_for_devices

Returns the configured addresses.

```
num_of_devices scan_for_devices();
```

Parameters

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 returns the number of devices configured in the Master Conformizer. The addresses of this devices are displayed in the protocol window.

get_max_address

Returns the maximum configured address.

```
max_address get_max_address();
```

Parameters

Return Values

This function returns the max_address. 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 configured device address in the Master Conformizer.

Error Handling Functions

simulate_at_failure

Simulates the failure of ATs.

```
error_code simulate_at_failure(phase, num_of_fails);
```

Parameters

phase

Communication phase in which the telegram failures are generated.

num_of_fails

Number of AT failures.

Return Values

This function returns an `error_code`. A zero value indicates that the failure of ATs was possible. A non-zero value indicates that the failure of ATs 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 ATs of the active device in the specified communication phase for the specified number of cycles. After this cycles the ATs are sent again. The function can be called independent of the current phase. The telegram failure generation starts, if the specified phase is achieved.

simulate_at_jitter

Simulates a jitter in the sending of the AT.

```
error_code simulate_at_failure(phase, jitter);
```

Parameters

phase

Communication phase in which the jitter simulation is generated.

jitter

Jitter to be simulated.

Return Values

This function returns an `error_code`. A zero value indicates that the jitter simulation of ATs was possible. A non-zero value indicates that the jitter simulation of ATs 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 send two following ATs of the active device in the specified communication phase (phase 3 or 4) with the specified jitter. The ATs are sent at the sending time plus the jitter. After this two cycles the ATs are sent at the correct time again. The function can be called independent of the current phase. The jitter simulation starts, if the specified phase is achieved.

simulate_wrong_at_length

Simulates the sending of ATs with a wrong length.

error_code simulate_at_failure(phase, num_of_fails);

Parameters

phase

Communication phase in which the telegram with the wrong length are sent.

num_of_fails

Number of ATs with wrong length.

Return Values

This function returns an `error_code`. A zero value indicates that the sending of ATs with wrong length was possible. A non-zero value indicates that the sending of ATs with wrong length 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 ATs with a wrong length of the active device in the specified communication phase for the specified number of cycles. After this cycles the ATs are sent with the correct length again. The function can be called independent of the current phase. The sending of ATs with the wrong length starts, if the specified phase is achieved.

simulate_wrong_device_address

Simulates the sending of ATs with a wrong address.

```
error_code simulate_wrong_device_address(phase, new_address, num_of_fails);
```

Parameters

phase

Communication phase in which the telegram with the wrong address are sent.

new_address

Wrong Address of the ATs.

num_of_fails

Number of ATs with wrong address.

Return Values

This function returns an `error_code`. A zero value indicates that the sending of ATs with wrong address was possible. A non-zero value indicates that the sending of ATs with wrong address 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 ATs with the specified new address instead of ATs with the address of the active device in the specified communication phase for the specified number of cycles. After this cycles the ATs are sent with the correct address. The function can be called independent of the current phase. The sending of ATs with the wrong address starts, if the specified phase is achieved.

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 of the active device in bytes. With this function also the length of a list in bytes can be appointed.

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 in the IDN data base of the active device. So if special IDNs that should be used for further tests 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 in the IDN base of the active device which can be also written. So if special IDNs that can be written should be used for further test can be searched automatically.

Time Slot Functions

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 active device (S-0-0003, S-0-0004, S-0-0005, S-0-0088, S-0-0090, S-0-0096 and S-0-0087).

read_time_slots

Reads the time slots.

```
error_code read_time_slots();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the time slots were successfully read. A non-zero value indicates that the time slots 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 IDNs for the time slots from the active device (S-0-0001, S-0-0002, S-0-0006, S-0-0007, S-0-0008 and S-0-0089).

test_t1

Test the time slot calculation for the AT sending times.

```
error_code test_t1();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the time slot for `t1` was correctly calculated. A non-zero value indicates that the time slot for `t1` was not correctly calculated. 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 tests, if the time slot calculation for the AT sending time (`t1`) is done correctly for all configured devices. The results are displayed in the protocol window.

test_t2

Test the time slot calculation for the MDT sending time.

```
error_code test_t2();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the time slot for `t2` was correctly calculated. A non-zero value indicates that the time slot for `t2` was not correctly calculated. 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 tests, if the time slot calculation for the MDT sending time (`t2`) is done correctly. The results are displayed in the protocol window.

test_t3

Test the time slot calculation for the Command value valid time (t3).

```
error_code test_t3();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the time slot for t3 was correctly calculated. A non-zero value indicates that the time slot for t3 was not correctly calculated. 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 tests, if the time slot calculation for the Command value valid time (t3) is done correctly. The results are displayed in the protocol window.

test_t4

Test the time slot calculation for the Feedback acquisition capture point (t4).

```
error_code test_t4();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the time slot for t4 was correctly calculated. A non-zero value indicates that the time slot for t4 was not correctly calculated. 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 tests, if the time slot calculation for the Feedback acquisition capture point (t4) is done correctly. The results are displayed in the protocol window.

set_t1

Set the time t1.

```
error_code set_t1(t1);
```

Parameters

phase

Phase in which t1 should be set.

t1

Sending time of the AT.

num_of_fail

Number of ATs sent with the new t1.

Return Values

This function returns an `error_code`. A zero value indicates that the sending time of the AT was successfully set. A non-zero value indicates that the sending time of the AT 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 ATs of the active device in the specified communication phase (phase 1 and 2) for the specified number of cycles. After this cycles the ATs are sent at the normal sending time again. The function can be called independent of the current phase. The telegram failure generation starts, if the specified phase is achieved.

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.

Telegram Functions

get_telegram_type

Reads the telegram type.

```
telegram_type get_telegram_type();
```

Parameters

Return Values

This function returns the telegram_type. 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 configured telegram type.

`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 part of the MDT for the active device 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 for the active device.

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).

Test Logic Functions

get_last_error

Returns the last occurred error.

```
last_error_code get_last_error();
```

Parameters

Return Values

The function returns a 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.

```
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 the 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.

fail_test_on_yes

Signals a test failure in a user interaction.

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

Parameters

[text]

Text to be displayed in the dialog.

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 makes the script file waiting for a user interaction. The specified text is displayed in a dialog window. If the user selects the yes answer to the dialog the test is marked as failed. So this dialog can be used for questions answered with yes means a test failure.

fail_test_on_no

Signals a test failure in a user interaction.

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

Parameters

[text]

Text to be displayed in the dialog.

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 makes the script file waiting for a user interaction. The specified text is displayed in a dialog window. If the user selects the no answer to the dialog the test is marked as failed. So this dialog can be used for questions answered with no means a test failure.

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.

Protocol Functions

Phase Protocol Functions

set_phase_protocol_level

Sets the protocol level for the phase protocol.

```
error_code set_trans_mode(level);
```

Parameters

mode

Level of the phase protocol.

0 no phase protocol

1 phase protocol

Return Values

This function returns an `error_code`. A zero value indicates that the protocol level was successfully set. A non-zero value indicates that the protocol level 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 level of the phase protocol.

get_phase_protocol_level

Reads the protocol level for the phase protocol.

```
level get_phase_protocol_level();
```

Parameters

Return Values

This function returns an `error_code`. A zero value indicates that the protocol level was successfully read. A non-zero value indicates that the protocol level 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 level of the phase protocol.

Service Channel Protocol Functions

set_sc_protocol_level

Sets the protocol level for the service channel protocol.

```
error_code set_sc_protocol_level(level);
```

Parameters

level

Level of the service channel protocol.

- 0 no service channel protocol
- 1 service channel protocol for read and write access to the data element and errors
- 2 service channel protocol for read and write access to all elements and errors
- 3 service channel protocol for all operations

Return Values

This function returns an `error_code`. A zero value indicates that the protocol level was successfully set. A non-zero value indicates that the protocol level 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 level of the service channel protocol.

get_sc_protocol_level

Reads the protocol level for the service channel protocol.

```
level get_sc_protocol_level();
```

Parameters

Return Values

This function returns the level. 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 level of the service channel protocol.

get_last_ident

Reads the last ident written or read on the service channel.

```
ident get_last_ident();
```

Parameters

Return Values

This function returns the ident. 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 last IDN which was written or read over the service channel.

get_last_element

Reads the last element written or read over the service channel.

```
element get_last_element();
```

Parameters

Return Values

This function returns the 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 index of the last element of an IDN which was written or read over the service channel.

get_last_operation

Reads the last operation done over the service channel.

```
operation get_last_operation();
```

Parameters

Return Values

This function returns the operation. 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 last operation (read or write) done over the service channel.

get_last_written_ident

Reads the written IDN in a phase.

```
ident get_last_written_ident();
```

Parameters

phase

Phase of which the last written IDN should be read.

Return Values

This function returns the ident. 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 last written IDN in the specified phase. With this functions it is for example possible to test if a master writes IDN 127 or IDN 128 as last IDN in communication phase 2 or 3.

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.

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 for the active device.

read_device_status_bit

Reads one bit of the device status word.

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

Parameter

bit_num

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 one bit of the device status bit. This function have no real-time behavior.

write_device_status_bit

Writes one bit of the device status word.

```
error_code write_device_status_bit(bit_num, device_status_bit);
```

Parameter

bit_num

Bit of the device status word to be written.

device_status_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 device status word. It should only be used to write the bits 14 and 15 in communication phase 4, because the other bits of the device status word are handled by the real-time driver.

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.

get_operation_mode

Returns the operation mode.

```
op_mode get_operation_mode();
```

Parameter

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 returns the actual configured operation mode.

main_power_on

Switches the main power on.

```
error_code main_power_on();
```

Parameter

Return Values

This function returns an `error_code`. A zero value indicates that the main power was successfully switched on. A non-zero value indicates that the main power could not be switched on. 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 main power on. Bit 15 of the device status word is set to 1 and bit 14 is set to 0.

main_power_off

Switches the main power off.

```
error_code main_power_off();
```

Parameter

Return Values

This function returns an `error_code`. A zero value indicates that the main power was successfully switched off. A non-zero value indicates that the main power could not be switched off. 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 main power off. Bit 15 of the device status word is set to 0 and bit 14 is set to 1.

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.