SDHLibrary-CPP

0.0.1.3

Generated by Doxygen 1.5.5

Sun Nov 16 15:54:26 2008

Contents

Chapter 1

The C++ libray for controlling the SDH (SCHUNK Dexterous Hand) from a PC.

1.1 General project information

Author:

Dirk Osswald

Project releases:

- Current release name: see PROJECT_RELEASE
- Current release date: see PROJECT_DATE

1.2 Purpose

This documentation describes the **sdh** library for the C++ language. It allows to control the SDH (SCHUNK Dexterous Hand) with a user program from a PC.

1.3 Links

- The SCHUNK homepage: http://www.schunk.com/
- Additionally used libraries:
  - The C++ STL (Standard Template Library), especially the vector<T> type. See http://gcc.gnu.org/onlinedocs/libstdc++/documentation.html
- Used tools:
  - **gcc** (GNU Compiler Collection), namely g++, the C++ compiler. See http://gcc.gnu.org/onlinedocs/gcc/
  - **make** (GNU make), see <a href="http://www.gnu.org/software/make/manual/make.html" http://www.gnu.org/software/make/manual/make.html

– **Doxygen** for generating pdf and html documentation directly from the sources. The documentation can be found: On the web: http://www.stack.nl/~dimitri/doxygen/

Chapter 2

Bug List

Member SDH::cSDH::EmergencyStop(void) For now this will **NOT** work while a "Grasp" command is executing, even if that was initiated non-sequentially!

Member SDH::cSDH::Stop(void) For now this will **NOT** work while a "Grasp" command is executing, even if that was initiated non-sequentially!

Member SDH::cSDH::GripHand(eGraspId grip, double close, double velocity, bool sequ=true) !!! Currently the performing of a skill or grip can **NOT** be interrupted!!! Even if the command is sent with *sequ=false* it **cannot** be stoped or emergency stopped.

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 5

Class Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Chapter 8

Module Documentation

8.1 Settings

8.1.1 Detailed Description

Primary settings to be adjusted by the user.

Defines

- #define SDH_USE_NAMESPACE 1

  Flag, if 1 then all classes are put into a namespace called SDH. If 0 then the classes are left outside any namespace.

8.1.2 Define Documentation

8.1.2.1 #define SDH_USE_NAMESPACE 1

Flag, if 1 then all classes are put into a namespace called SDH. If 0 then the classes are left outside any namespace.

8.2 Derived settings

8.2.1 Detailed Description

Derived settings that users should not have to adjust. Adjustments should be done in Settings

Defines

- #define NAMESPACE_SDH_START namespace SDH {
- #define NAMESPACE_SDH_END }
- #define USING_NAMESPACE_SDH using namespace SDH;

8.2.2 Define Documentation

8.2.2.1 #define NAMESPACE_SDH_END }

8.2.2.2 #define NAMESPACE_SDH_START namespace SDH {

8.2.2.3 #define USING_NAMESPACE_SDH using namespace SDH;

Chapter 9

Namespace Documentation

9.1 SDH Namespace Reference

9.1.1 Detailed Description

A namespace for all classes and functions in the SDHLibrary.

The use of the namespace can be disabled at compile time of the library by setting SDH_USE_-NAMESPACE to 0.

Classes

- class cCANSerial_ESDException

  Derived exception class for low-level CAN ESD related exceptions.

- class cCANSerial_ESD

  Low-level communication class to access a CAN port.

- class cCRC
- class cCRC_DSACON32m

  A derived CRC class that uses a CRC table and initial value suitable for the Weiss Robotics DSACON32m controller.

- class cDBG

  A class to print colored debug messages.

- class cDSAException

  Derived exception class for low-level DSA related exceptions.

- class cDSA
- class cRS232Exception

  Derived exception class for low-level RS232 related exceptions.

- class cRS232

  Low-level communication class to access a serial port on Cygwin and Linux.

- class cSDH

  SDH::cSDH is the end user interface class to control a SDH (SCHUNK Dexterous Hand).

- class cSDHErrorInvalidParameter

  Derived exception class for exceptions related to invalid parameters.

- class cSDHBase

  The base class to control the SCHUNK Dexterous Hand.

- class cMsg

  Class for short, fixed maximum length text messages.

- class cSDHLibraryException

  Base class for exceptions in the SDHLibrary-CPP.

- class cSDHErrorCommunication

  Derived exception class for exceptions related to communication between the SDHLibrary and the SDH.

- class cSDHSerial

  The class to communicate with a SDH via RS232.

- class cSerialBaseException

  Derived exception class for low-level serial communication related exceptions.

- class cSerialBase

  Low-level communication class to access a serial port.

- class cSimpleStringList

  A simple string list. (Fixed maximum number of strings of fixed maximum length).

- class cSimpleTime

  Very simple class to measure elapsed time.

- class cSimpleVectorException

  Derived exception class for low-level simple vector related exceptions.

- class cSimpleVector

  A simple vector implementation.

- class cUnitConverter

  Unit conversion class to convert values between physical unit systems.

Typedefs

- typedef char Int8

  signed integer, size 1 Byte (8 Bit)

- typedef unsigned char UInt8

unsigned integer, size 1 Byte (8 Bit)

- typedef short Int16

  signed integer, size 2 Byte (16 Bit)

- typedef unsigned short UInt16

  unsigned integer, size 2 Byte (16 Bit)

- typedef long Int32

  signed integer, size 4 Byte (32 Bit)

- typedef unsigned long UInt32

  unsigned integer, size 4 Byte (32 Bit)

- typedef **UInt16** tCRCValue

  the data type used to calculate and exchange CRC values with DSACON32m (16 bit integer)

- typedef void * NTCAN_HANDLE

  dummy definition in case ntcan.h is not available

- typedef cSimpleVector(cSDHSerial::* pSetFunction)(int, double *)

  Type of a pointer to a "set-axis-values" function like cSDHSerial::p, cSDHSerial::pos, ..., cSDHSerial::igrip, cSDHSerial::ihold or cSDHSerial::ilim.

- typedef cSimpleVector(cSDHSerial::* pGetFunction)(int, double *)

  Type of a pointer to a "get-axis-values" function like cSDHSerial::p, cSDHSerial::pos, ..., cSDHSerial::igrip, cSDHSerial::ihold or cSDHSerial::ilim.

- typedef double(cUnitConverter::* pDoubleUnitConverterFunction)(double) const

  Type of a pointer to a function like 'double cUnitConverter::ToExternal(double)' or 'cUnitConverterToInternal(double)'.

Functions

- std::ostream & operator<< (std::ostream &stream, cDSA::sControllerInfo const &controller_info)
- std::ostream & operator<< (std::ostream &stream, cDSA::sSensorInfo const &sensor_info)
- std::ostream & operator<< (std::ostream &stream, cDSA::sMatrixInfo const &matrix_info)
- std::ostream & operator<< (std::ostream &stream, cDSA::sResponse const &response)
- std::ostream & operator<< (std::ostream &stream, cDSA const &dsa)
- std::ostream & operator<< (std::ostream &stream, cMsg const &msg)
- std::ostream & operator<< (std::ostream &stream, cSDHLibraryException const &e)
- std::ostream & operator<< (std::ostream &stream, cSimpleStringList &ssl)

  Output of cSimpleStringList objects in 'normal' output streams.

Auxiliary functions

- bool InIndex (int v, int max)
- bool InRange (double v, double min, double max)
- bool InRange (int n, double const *v, double const *min, double const *max)

- double ToRange (double v, double min, double max)
- void ToRange (int n, double *v, double const *min, double const *max)
- void ToRange (std::vector< double > &v, std::vector< double > const &min, std::vector< double > const &max)
- double Approx (double a, double b, double eps)
- bool Approx (int n, double *a, double *b, double *eps)
- double DegToRad (double d)
- double RadToDeg (double r)
- void SleepSec (double t)
- template<typename Function, typename Tp>
  void apply (Function f, Tp &sequence)
- template<typename Function, typename InputIterator>
  Function apply (Function f, InputIterator first, InputIterator last)
- template<typename Function, typename Tp>
  Tp map (Function f, Tp sequence)
- template<typename T>
  std::ostream & operator<< (std::ostream &stream, std::vector< T > v)

Variables

- std::ostream * g_sdh_debug_log = &std::cerr
- cUnitConverter const uc_identity ("any","any","?", 1.0, 0.0, 4)

  Identity converter (internal = external).

- static double M_PI = 4.0*atan(1.0)

9.1.2 Typedef Documentation

9.1.2.1 typedef short SDH::Int16

signed integer, size 2 Byte (16 Bit)

9.1.2.2 typedef long SDH::Int32

signed integer, size 4 Byte (32 Bit)

9.1.2.3 typedef char SDH::Int8

signed integer, size 1 Byte (8 Bit)

9.1.2.4 typedef void* SDH::NTCAN_HANDLE

dummy definition in case ntcan.h is not available

9.1.2.5 typedef double(cUnitConverter::* SDH::pDoubleUnitConverterFunction)(double) const

Type of a pointer to a function like 'double cUnitConverter::ToExternal(double)' or 'cUnitConverterToInternal(double)'.

9.1.2.6 typedef cSimpleVector(cSDHSerial::* SDH::pGetFunction)(int, double *)

Type of a pointer to a "get-axis-values" function like cSDHSerial::p, cSDHSerial::pos, ..., cSDHSerial::igrip, cSDHSerial::ihold or cSDHSerial::ilim.

9.1.2.7 typedef cSimpleVector(cSDHSerial::* SDH::pSetFunction)(int, double *)

Type of a pointer to a "set-axis-values" function like cSDHSerial::p, cSDHSerial::pos, ..., cSDHSerial::igrip, cSDHSerial::ihold or cSDHSerial::ilim.

9.1.2.8 typedef UInt16 SDH::tCRCValue

the data type used to calculate and exchange CRC values with DSACON32m (16 bit integer)

9.1.2.9 typedef unsigned short SDH::UInt16

unsigned integer, size 2 Byte (16 Bit)

9.1.2.10 typedef unsigned long SDH::UInt32

unsigned integer, size 4 Byte (32 Bit)

9.1.2.11 typedef unsigned char SDH::UInt8

unsigned integer, size 1 Byte (8 Bit)

9.1.3 Function Documentation

9.1.3.1 template<typename Function, typename InputIterator> Function SDH::apply (Function *f*, InputIterator *first*, InputIterator *last*) [inline]

Apply a function to every element of a sequence.

Parameters:

first An input iterator.

last An input iterator.

f A unary function object.

Returns:

`f`.

Applies the function object `f` to each element in the range [first,last). `f` must not modify the order of the sequence. If `f` has a return value it is ignored.

9.1.3.2 template<typename Function, typename Tp> void SDH::apply (Function *f*, Tp & *sequence*) [inline]

Apply a function to every element of a sequence, the elements of the sequence are modified by *f*

Parameters:

f A unary function object.

sequence The iterable sequence to modify.

Applies the function object `f` to each element of the *sequence*.

9.1.3.3 bool SDH::Approx (int *n*, double * *a*, double * *b*, double * *eps*)

Return True if list/tuple/array a=(a1,a2,...) is approximately the same as b=(b1,b2,...). I.E. |a_i-b_i| < eps[i]

9.1.3.4 double SDH::Approx (double *a*, double *b*, double *eps*)

Return True if a is approximately the same as b. I.E. |a-b| < eps

9.1.3.5 double SDH::DegToRad (double *d*)

Return d in deg converted to rad

9.1.3.6 bool SDH::InIndex (int *v*, int *max*)

Return True if v is in range [0 .. max[

9.1.3.7 bool SDH::InRange (int *n*, double const * *v*, double const * *min*, double const * *max*)

Return True if in list/tuple/array v=(v1,v2,...) each v_i is in range [min_i..max_i] with min = (min1, min2,...) max = (max1, max2, ..)

9.1.3.8 bool SDH::InRange (double *v*, double *min*, double *max*)

Return True if v is in range [min .. max]

9.1.3.9 template<typename Function, typename Tp> Tp SDH::map (Function *f*, Tp *sequence*) [inline]

map a function to every element of a sequence, returning a copy with the mapped elements

Parameters:

f - A unary function object.

sequence - An iterable object.

Returns:

copy of *sequence* with the mapped elements

9.1.3.10 template<typename T> std::ostream& SDH::operator<< (std::ostream & *stream*, std::vector< T > *v*) [inline]

Overloaded insertion operator for vectors: a comma and space separated list of the vector elements of *v* is inserted into *stream*

Parameters:

stream - the output stream to insert into

v - the vector of objects to insert into *stream*

Returns:

the stream with the inserted objects

Attention:

If you use the SDH namespace then you should be aware that using this overloaded insertion operator can get tricky:

- If you use a "using namespace SDH" directive then things are easy and intuitive:

```
#include <sdh/util.h>
using namespace SDH;
std::vector<int> v;
std::cout << "this is a std::vector: " << v << "\n";
```

- But without the `using namespace SDH` accessing the operator is tricky, you have to use the 'functional' access `operator<<(s,v)` in order to be able to apply the scope resolution operator :: correctly:

```
#include <sdh/util.h>
std::vector<int> v;
SDH::operator<<( std::cout << "this is a std::vector: ", v ) << "\n";
```

- The more intuitive approaches do not work:

```
std::cout << faa ;                // obviously fails with "no match for >>operator<<<< in >>s
std::cout SDH::<< faa ;           // is a syntax error
std::cout SDH::operator<< faa ;   // is a syntax error
std::cout operator SDH::<< faa ;  // is a syntax error
```

9.1.3.11 std::ostream & SDH::operator<< (std::ostream & *stream*, cSimpleStringList & *ssl*)

Output of cSimpleStringList objects in 'normal' output streams.

9.1.3.12 std::ostream & SDH::operator<< (std::ostream & *stream*, cSDHLibraryException const & *e*)

9.1.3.13 std::ostream & SDH::operator<< (std::ostream & *stream*, cMsg const & *msg*)

9.1.3.14 std::ostream & SDH::operator<< (std::ostream & *stream*, cDSA const & *dsa*)

9.1.3.15 std::ostream & SDH::operator<< (std::ostream & *stream*, cDSA::sResponse const & *response*)

9.1.3.16 std::ostream & SDH::operator<< (std::ostream & *stream*, cDSA::sMatrixInfo const & *matrix_info*)

9.1.3.17 std::ostream & SDH::operator<< (std::ostream & *stream*, cDSA::sSensorInfo const & *sensor_info*)

9.1.3.18 std::ostream & SDH::operator<< (std::ostream & *stream*, cDSA::sControllerInfo const & *controller_info*)

9.1.3.19 double SDH::RadToDeg (double *r*)

Return r in rad converted to deg

9.1.3.20 void SDH::SleepSec (double *t*)

Sleep for t seconds. (t is a double!)

9.1.3.21 void SDH::ToRange (std::vector< double > & *v*, std::vector< double > const & *min*, std::vector< double > const & *max*)

Limit each v_i in v to range [min_i..max_i] with min = (min1, min2,...) max = (max1, max2, ..) This modifies *v!*

9.1.3.22 void SDH::ToRange (int *n*, double * *v*, double const * *min*, double const * *max*)

Limit each v_i in v to range [min_i..max_i] with min = (min1, min2,...) max = (max1, max2, ..) This modifies *v!

9.1.3.23 double SDH::ToRange (double *v*, double *min*, double *max*)

Return v limited to range [min .. max]. I.e. if v is < min then min is returned, or if v > max then max is returned, else v is returned

9.1.4 Variable Documentation

9.1.4.1 std::ostream * SDH::g_sdh_debug_log = &std::cerr

9.1.4.2 double SDH::M_PI = 4.0*atan(1.0) [static]

9.1.4.3 cUnitConverter const SDH::uc_identity

Identity converter (internal = external).

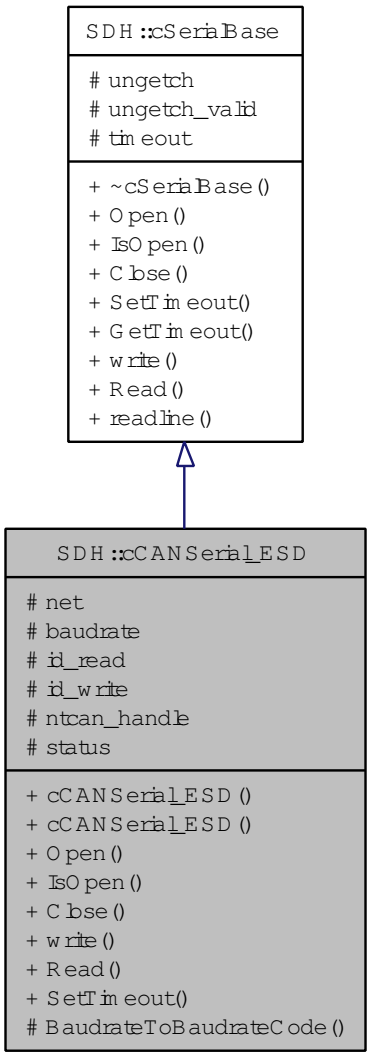
Chapter 10

Class Documentation

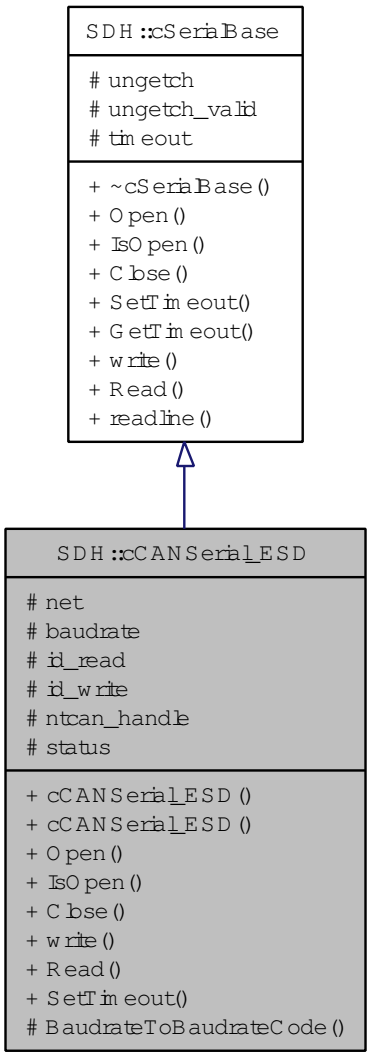
10.1 SDH::cCANSerial_ESD Class Reference

`#include <canserial-esd.h>`

Inheritance diagram for SDH::cCANSerial_ESD:



Collaboration diagram for SDH::cCANSerial_ESD:



10.1.1 Detailed Description

Low-level communication class to access a CAN port.

Public Member Functions

- cCANSerial_ESD (int _net, unsigned long _baudrate, double _timeout, int32_t _id_read, int32_t _id_write) throw (cCANSerial_ESDException*)
- cCANSerial_ESD (NTCAN_HANDLE _ntcan_handle, double _timeout, int32_t _id_read, int32_t _id_write) throw (cCANSerial_ESDException*)
- void Open (void) throw (cCANSerial_ESDException*)
- bool IsOpen (void) throw ()

  Return true if interface to CAN ESD is open.

- void Close (void) throw (cCANSerial_ESDException*)

  Close the previously opened CAN ESD interface port.

- int write (char const *ptr, int len=0) throw (cCANSerial_ESDException*)

  Write data to a previously opened port.

- ssize_t Read (void *data, ssize_t size, long timeout_us, bool return_on_less_data) throw (cCANSerial_ESDException*)
- void SetTimeout (double _timeout) throw (cSerialBaseException*)

  set the timeout for next readline() calls (negative value means: no timeout, wait for ever)

Protected Member Functions

- uint32_t BaudrateToBaudrateCode (unsigned long baudrate) throw (cCANSerial_ESDException*)

  Translate a baudrate given as unsigned long into a baudrate code for struct termios.

Protected Attributes

- int net

  the ESD CAN net to use

- unsigned long baudrate

  the baudrate to use in bit/s

- int32_t id_read

  the CAN ID used for reading

- int32_t id_write

  the CAN ID used for writing

- NTCAN_HANDLE ntcan_handle

  the handle to the driver

- int status

10.1.2 Constructor & Destructor Documentation

10.1.2.1 cCANSerial_ESD::cCANSerial_ESD (int *_net*, unsigned long *_baudrate*, double *_timeout*, int32_t *_id_read*, int32_t *_id_write*) throw (cCANSerial_ESDException*)

Constructor: constructs an object to communicate with an SDH via CAN bus using an ESD CAN card.

Parameters:

_net - the ESD CAN net to use

_baudrate - the baudrate in bit/s. Only some bitrates are valid: (1000000,800000,500000,250000,125000,100000,50000,20000,10000)

_timeout - the timeout in seconds (0 for no timeout = wait for ever)

_id_read - the CAN ID to use for reading (The SDH sends data on this ID)

_id_write - the CAN ID to use for writing (The SDH receives data on this ID)

10.1.2.2 cCANSerial_ESD::cCANSerial_ESD (NTCAN_HANDLE *_ntcan_handle*, double *_timeout*, int32_t *_id_read*, int32_t *_id_write*) throw (cCANSerial_ESDException*)

Constructor: constructs an object to communicate with an SDH via CAN bus using an ESD CAN card by reusing an already existing handle.

Parameters:

_ntcan_handle - the ESD CAN handle to reuse

_timeout - the timeout in seconds (0 for no timeout = wait for ever)

_id_read - the CAN ID to use for reading (The SDH sends data on this ID)

_id_write - the CAN ID to use for writing (The SDH receives data on this ID)

10.1.3 Member Function Documentation

10.1.3.1 uint32_t cCANSerial_ESD::BaudrateToBaudrateCode (unsigned long *baudrate*) throw (cCANSerial_ESDException*) [protected]

Translate a baudrate given as unsigned long into a baudrate code for struct termios.

10.1.3.2 void cCANSerial_ESD::Open (void) throw (cCANSerial_ESDException*) [virtual]

Open the device as configured by the parameters given to the constructor

Implements SDH::cSerialBase.

10.1.3.3 bool cCANSerial_ESD::IsOpen (void) throw () [virtual]

Return true if interface to CAN ESD is open.

Implements SDH::cSerialBase.

10.1.3.4 void cCANSerial_ESD::Close (void) throw (cCANSerial_ESDException*) [virtual]

Close the previously opened CAN ESD interface port.

Implements SDH::cSerialBase.

10.1.3.5 int cCANSerial_ESD::write (char const * *ptr*, int *len* = 0) throw (cCANSerial_ESDException*) [virtual]

Write data to a previously opened port.

Write *len* bytes from **ptr* to the CAN device

Parameters:

ptr - pointer the byte array to send in memory

len - number of bytes to send

Returns:

the number of bytes actually written

Implements SDH::cSerialBase.

10.1.3.6 ssize_t cCANSerial_ESD::Read (void * *data*, ssize_t *size*, long *timeout_us*, bool *return_on_less_data*) throw (cCANSerial_ESDException*) [virtual]

Read data from device. This function waits until *max_time_us* us passed or the expected number of bytes are received via serial line. if (*return_on_less_data* is true (default value), the number of bytes that have been received are returned and the data is stored in *data* If the *return_on_less_data* is false, data is only read from serial line, if at least *size* bytes are available.

Implements SDH::cSerialBase.

10.1.3.7 void cCANSerial_ESD::SetTimeout (double *_timeout*) throw (cSerialBaseException*) [virtual]

set the timeout for next readline() calls (negative value means: no timeout, wait for ever)

Reimplemented from SDH::cSerialBase.

10.1.4 Member Data Documentation

10.1.4.1 int SDH::cCANSerial_ESD::net [protected]

the ESD CAN net to use

10.1.4.2 unsigned long SDH::cCANSerial_ESD::baudrate [protected]

the baudrate to use in bit/s

10.1.4.3 int32_t SDH::cCANSerial_ESD::id_read [protected]

the CAN ID used for reading

10.1.4.4 int32_t SDH::cCANSerial_ESD::id_write [protected]

the CAN ID used for writing

10.1.4.5 NTCAN_HANDLE SDH::cCANSerial_ESD::ntcan_handle [protected]

the handle to the driver

10.1.4.6 int SDH::cCANSerial_ESD::status [protected]

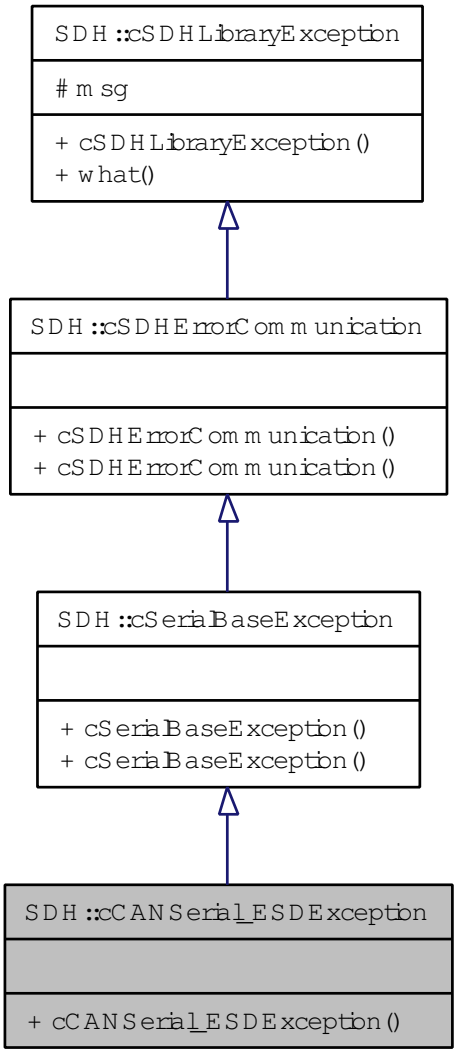
The documentation for this class was generated from the following files:

- sdh/canserial-esd.h
- sdh/canserial-esd.cpp

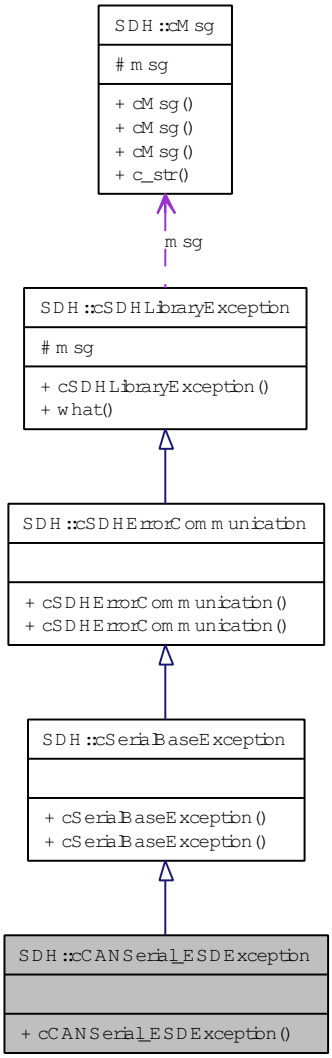
10.2 SDH::cCANSerial_ESDException Class Reference

`#include <canserial-esd.h>`

Inheritance diagram for SDH::cCANSerial_ESDException:



Collaboration diagram for SDH::cCANSerial_ESDException:



10.2.1 Detailed Description

Derived exception class for low-level CAN ESD related exceptions.

Public Member Functions

- cCANSerial_ESDException (cMsg const &_msg)

10.2.2 Constructor & Destructor Documentation

10.2.2.1 SDH::cCANSerial_ESDException::cCANSerial_ESDException (cMsg const & *_msg*) [inline]

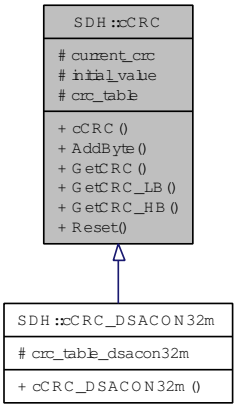
The documentation for this class was generated from the following file:

- sdh/canserial-esd.h

10.3 SDH::cCRC Class Reference

`#include <crc.h>`

Inheritance diagram for SDH::cCRC:



10.3.1 Detailed Description

Generic class to calculate a CRC using a given, precalculated table.

Use derived classes like cCRC_DSACON32m with a specifically set CRC table.

Public Member Functions

- cCRC (tCRCValue const *_crc_table, tCRCValue _initial_value)

  constructor: create a new cCRC object and initialize the current value of the CRC checksum. crc_table *is the CRC table to use.*

- tCRCValue AddByte (unsigned char byte)

  insert byte into CRC calculation and return the new current CRC checksum

- tCRCValue GetCRC ()

  return the current CRC value

- **UInt8** GetCRC_LB ()

  return the low byte of the current CRC value

- **UInt8** GetCRC_HB ()

  return the high byte of the current CRC value

- tCRCValue Reset ()

  reset the current CRC value to its initial value and return it;

Protected Attributes

- tCRCValue current_crc

  current value of the CRC checksum

- tCRCValue initial_value

  initial value of the CRC checksum

- tCRCValue const * crc_table

  table with precalculated CRC values

10.3.2 Constructor & Destructor Documentation

10.3.2.1 SDH::cCRC::cCRC (tCRCValue const * *_crc_table*, tCRCValue *_initial_value*) [inline]

constructor: create a new cCRC object and initialize the current value of the CRC checksum. *crc_table* is the CRC table to use.

10.3.3 Member Function Documentation

10.3.3.1 tCRCValue SDH::cCRC::AddByte (unsigned char *byte*) [inline]

insert byte into CRC calculation and return the new current CRC checksum

10.3.3.2 tCRCValue SDH::cCRC::GetCRC () [inline]

return the current CRC value

10.3.3.3 UInt8 SDH::cCRC::GetCRC_LB () [inline]

return the low byte of the current CRC value

10.3.3.4 UInt8 SDH::cCRC::GetCRC_HB () [inline]

return the high byte of the current CRC value

10.3.3.5 tCRCValue SDH::cCRC::Reset () [inline]

reset the current CRC value to its initial value and return it;

10.3.4 Member Data Documentation

10.3.4.1 tCRCValue SDH::cCRC::current_crc [protected]

current value of the CRC checksum

10.3.4.2 tCRCValue SDH::cCRC::initial_value [protected]

initial value of the CRC checksum

10.3.4.3 tCRCValue const* SDH::cCRC::crc_table [protected]

table with precalculated CRC values

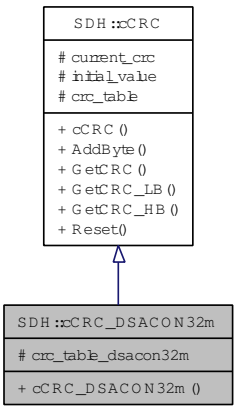
The documentation for this class was generated from the following file:

- sdh/crc.h

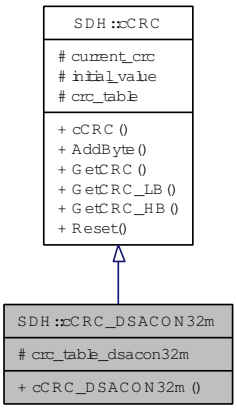
10.4 SDH::cCRC_DSACON32m Class Reference

`#include <crc.h>`

Inheritance diagram for SDH::cCRC_DSACON32m:



Collaboration diagram for SDH::cCRC_DSACON32m:



10.4.1 Detailed Description

A derived CRC class that uses a CRC table and initial value suitable for the Weiss Robotics DSACON32m controller.

Public Member Functions

- cCRC_DSACON32m (void)

  constructor to create a cCRC object suitable for checksumming the communication with a DSACON32m tactile sensor controller

Static Protected Attributes

- static tCRCValue const crc_table_dsacon32m [256]

  the CRC table used by the DSACON32m controller

10.4.2 Constructor & Destructor Documentation

10.4.2.1 SDH::cCRC_DSACON32m::cCRC_DSACON32m (void) [inline]

constructor to create a cCRC object suitable for checksumming the communication with a DSACON32m tactile sensor controller

10.4.3 Member Data Documentation

10.4.3.1 tCRCValue const cCRC_DSACON32m::crc_table_dsacon32m [static, protected]

the CRC table used by the DSACON32m controller

The documentation for this class was generated from the following files:

- sdh/crc.h
- sdh/crc.cpp

10.5 SDH::cDBG Class Reference

`#include <dbg.h>`

10.5.1 Detailed Description

A class to print colored debug messages.

- The printing can be switched on or off so the debug code can remain in the code. (default is off)
- The messages can be colorized (default is red).
- The output can be redirected. (default is sys.stderr)
- Debug messages can be printed in a functional way or in C++ stream like way

If the environment variable "SDH_NO_COLOR" is defined then the messages are printed without coloring (usefull for logging or if your terminal does not support colors.

Example:

```
#include "sdh/dbg.h"
d = cDBG( true );
g = cDBG( true, "green" );

d.PDM( "This message is printed in default color red" );
g << "and this one in a nice green ";

g << "of course you can debug print any objects that have a string representation: " << 08 << 15 << tru

g << "Messages can be turned of and on, e.g. selected by command line options";
g.SetFlag(false);
g << "This messages is not printed";
```

Public Member Functions

- cDBG (bool flag=false, char *color="red", std::ostream *fd=&std::cerr)
- ~cDBG ()
- void SetFlag (bool flag)
- bool GetFlag (void) const
- void SetColor (char *color)
- void SetOutput (std::ostream *fd)
- void PDM (char const *fmt,...) SDH__attribute__((format(printf

Protected Attributes

- char * debug_color
- char * normal_color
- std::ostream * output
- bool debug_flag

10.5.2 Constructor & Destructor Documentation

10.5.2.1 SDH::cDBG::cDBG (bool *flag* = `false`, char * *color* = `"red"`, std::ostream * *fd* = `&std::cerr`) `[inline]`

constructor: construct a cDBG object

Parameters:

flag - the initial state of the flag, if true then messages sent to the object are printed. Default is false. Can be changed with SetFlag()

color - the name of the color to use, default is "red". Can be changed with SetColor()

fd - the ostream to use for output, default is stderr. Can be changed with SetOutput()

10.5.2.2 SDH::cDBG::~cDBG () `[inline]`

10.5.3 Member Function Documentation

10.5.3.1 void SDH::cDBG::SetFlag (bool *flag*) `[inline]`

Set debug_flag of this cDBG object to flag. After setting the flag to true debug messages are printed, else not.

10.5.3.2 bool SDH::cDBG::GetFlag (void) const `[inline]`

Get debug_flag of this cDBG object.

10.5.3.3 void SDH::cDBG::SetColor (char * *color*) `[inline]`

Set debug_color of this cDBG object to color. color is a string like "red", see util.py for valid names.

Attention:

The string is **NOT** copied, just a pointer is stored

10.5.3.4 void SDH::cDBG::SetOutput (std::ostream * *fd*) `[inline]`

Set output of this cDBG object to fd, which must be a file like object like sys.stderr

10.5.3.5 void SDH::cDBG::PDM (char const * *fmt*, ...)

Print debug messages of printf like fmt, ... in the color set with SetColor, but only if debug_flag is true.

10.5.4 Member Data Documentation

10.5.4.1 char* SDH::cDBG::debug_color [protected]

10.5.4.2 char* SDH::cDBG::normal_color [protected]

10.5.4.3 std::ostream* SDH::cDBG::output [protected]

10.5.4.4 bool SDH::cDBG::debug_flag [protected]

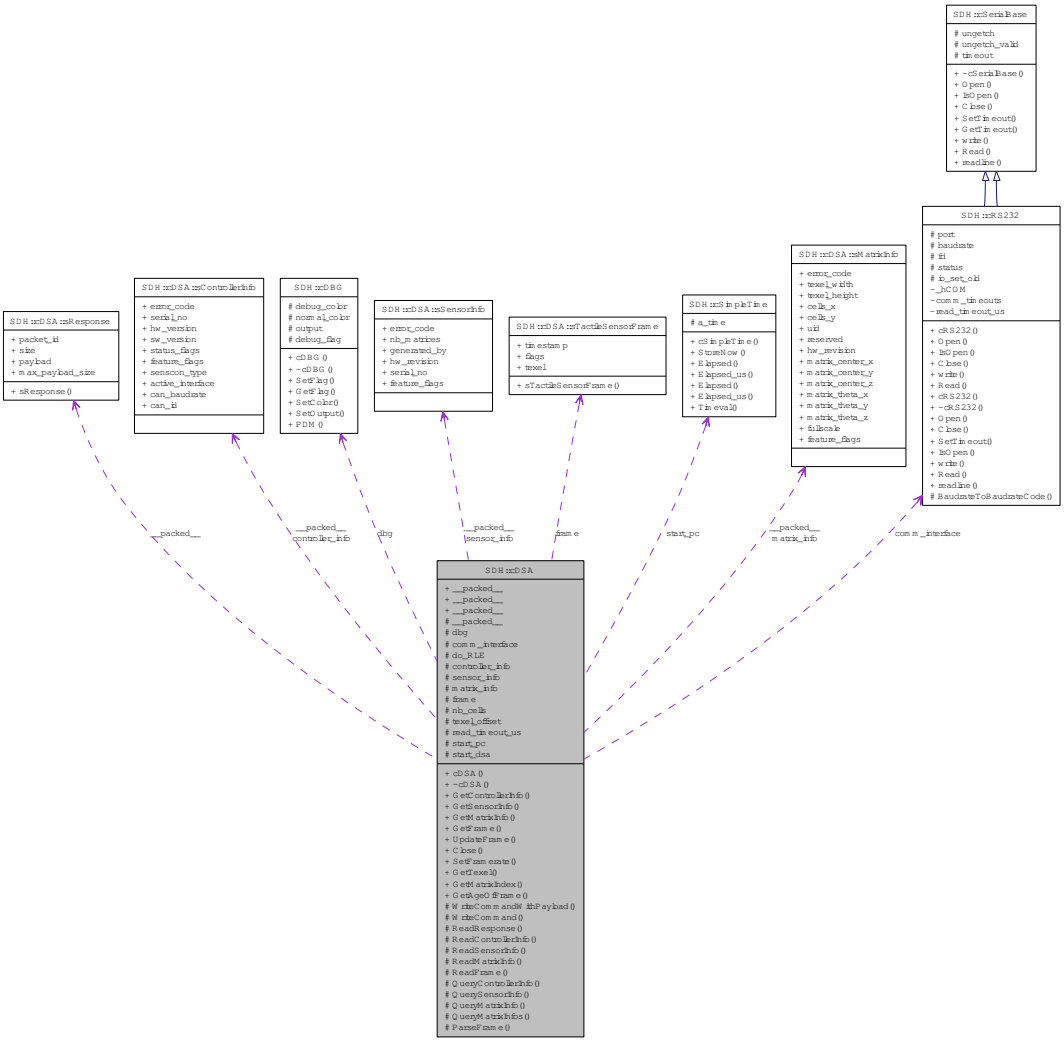
The documentation for this class was generated from the following file:

- sdh/dbg.h

10.6 SDH::cDSA Class Reference

`#include <dsa.h>`

Collaboration diagram for SDH::cDSA:



10.6.1 Detailed Description

Class to communicate with the tactile sensor controller DSACON32m of the SDH

Public Types

- typedef **UInt16** tTexel

  data type for a single 'texel' (tactile sensor element)

Public Member Functions

- cDSA (int debug_level=0, int port=1)
- ~cDSA ()

  Destructur: clean up and delete dynamically allocated memory.

- sControllerInfo const & GetControllerInfo (void) const

  Return a reference to the sControllerInfo read from the remote DSACON32m controller.

- sSensorInfo const & GetSensorInfo (void) const

  Return a reference to the sSensorInfo read from the remote DSACON32m controller.

- sMatrixInfo const & GetMatrixInfo (int m) const

  Return a reference to the sMatrixInfo of matrix m *read from the remote DSACON32m controller.*

- sTactileSensorFrame const & GetFrame () const

  return a reference to the latest tactile sensor frame without reading from remote DSACON32m

- sTactileSensorFrame const & UpdateFrame ()

  read the tactile sensor frame from remote DSACON32m and return a reference to it. A command to query the frame (periodically) must have been sent before.

- void Close (void)

  Set the framerate of the remote DSACON32m controller to 0 and close connection to it.

- void SetFramerate (**UInt16** framerate, bool do_RLE=false, bool do_data_acquisition=true)
- tTexel GetTexel (int m, int x, int y) const
- int GetMatrixIndex (int fi, int part)
- unsigned long GetAgeOfFrame (sTactileSensorFrame *frame_p)

Public Attributes

- struct SDH::cDSA::sControllerInfo __packed__
- struct SDH::cDSA::sSensorInfo __packed__
- struct SDH::cDSA::sMatrixInfo __packed__

Protected Member Functions

- void WriteCommandWithPayload (**UInt8** command, **UInt8** *payload, **UInt16** payload_len)
- void WriteCommand (**UInt8** command)
- void ReadResponse (sResponse *response)
- void ReadControllerInfo (sControllerInfo *_controller_info)
- void ReadSensorInfo (sSensorInfo *_sensor_info)
- void ReadMatrixInfo (sMatrixInfo *_matrix_info)
- void ReadFrame (sTactileSensorFrame *frame_p)
- void QueryControllerInfo (sControllerInfo *_controller_info)
- void QuerySensorInfo (sSensorInfo *_sensor_info)
- void QueryMatrixInfo (sMatrixInfo *_matrix_info, int matrix_no)
- void QueryMatrixInfos (void)
- void ParseFrame (sResponse *response, sTactileSensorFrame *frame_p)

Protected Attributes

- struct SDH::cDSA::sResponse __packed__
- cDBG dbg

  A stream object to print coloured debug messages.

- cRS232 comm_interface

  the serial RS232 communication interface to use

- bool do_RLE

  flag, true if data should be sent Run Length Encoded by the remote DSACON32m controller

- sControllerInfo controller_info

  the controller info read from the remote DSACON32m controller

- sSensorInfo sensor_info

  the sensor info read from the remote DSACON32m controller

- sMatrixInfo * matrix_info

  the matrix infos read from the remote DSACON32m controller

- sTactileSensorFrame frame

  the latest frame received from the remote DSACON32m controller

- int nb_cells

  The total number of sensor cells.

- int * texel_offset

  an array with the offsets of the first texel of all matrices into the whole frame

- long read_timeout_us

  default timeout used for reading in us

- cSimpleTime start_pc
- **UInt32** start_dsa

Friends

- std::ostream & operator<< (std::ostream &stream, cDSA::sResponse const &response)

Classes

- struct sControllerInfo

  A data structure describing the controller info about the remote DSACON32m controller.

- struct sMatrixInfo

  A data structure describing a single sensor matrix connected to the remote DSACON32m controller.

- struct sResponse

data structure for storing responses from the remote DSACON32m controller

- struct sSensorInfo
  A data structure describing the sensor info about the remote DSACON32m controller.
- struct sTactileSensorFrame

10.6.2 Member Typedef Documentation

10.6.2.1 typedef UInt16 SDH::cDSA::tTexel

data type for a single 'texel' (tactile sensor element)

10.6.3 Constructor & Destructor Documentation

10.6.3.1 cDSA::cDSA (int *debug_level* = `0`, int *port* = `1`)

Constructor for cDSA

10.6.3.2 cDSA::~cDSA ()

Destructur: clean up and delete dynamically allocated memory.

10.6.4 Member Function Documentation

10.6.4.1 void cDSA::WriteCommandWithPayload (UInt8 *command*, UInt8 * *payload*, UInt16 *payload_len*) `[protected]`

10.6.4.2 void SDH::cDSA::WriteCommand (UInt8 *command*) `[inline, protected]`

10.6.4.3 void cDSA::ReadResponse (sResponse * *response*) `[protected]`

Read any response from the remote DSACON32m

- tries to find the preamble within the next at most DSA_MAX_PREAMBLE_SEARCH bytes from the device
- reads the packet id and size
- reads all data indicated by the size read
- if there is enough space in the payload of the response then the received data is stored there if there is not enough space in the payload of the response then the data is received but forgotten (to keep the communication line clear)
- if sent, the CRC checksum is read and the data is checked. For invalid data an exception is thrown

10.6.4.4 void cDSA::ReadControllerInfo (sControllerInfo * *_controller_info*) `[protected]`

Read and parse a controller info response from the remote DSA

10.6.4.5 void cDSA::ReadSensorInfo (sSensorInfo * *_sensor_info*) [protected]

Read and parse a sensor info response from the remote DSA

10.6.4.6 void cDSA::ReadMatrixInfo (sMatrixInfo * *_matrix_info*) [protected]

Read and parse a matrix info response from the remote DSA

10.6.4.7 void cDSA::ReadFrame (sTactileSensorFrame * *frame_p*) [protected]

read and parse a full frame response from remote DSA

10.6.4.8 void cDSA::QueryControllerInfo (sControllerInfo * *_controller_info*) [protected]

Send command to resquest controller info from remote DSACON32m. Read and parse the response from the remote DSACON32m.

10.6.4.9 void cDSA::QuerySensorInfo (sSensorInfo * *_sensor_info*) [protected]

Send command to request sensor info from remote DSACON32m. Read and parse the response from the remote DSACON32m.

10.6.4.10 void cDSA::QueryMatrixInfo (sMatrixInfo * *_matrix_info*, int *matrix_no*) [protected]

Send command to request matrix info from remote DSACON32m. Read and parse the response from the remote DSACON32m.

10.6.4.11 void cDSA::QueryMatrixInfos (void) [protected]

Query all matrix infos

10.6.4.12 void cDSA::ParseFrame (sResponse * *response*, sTactileSensorFrame * *frame_p*) [protected]

Parse a full frame response from remote DSA

10.6.4.13 sControllerInfo const& SDH::cDSA::GetControllerInfo (void) const [inline]

Return a reference to the sControllerInfo read from the remote DSACON32m controller.

10.6.4.14 sSensorInfo const& SDH::cDSA::GetSensorInfo (void) const [inline]

Return a reference to the sSensorInfo read from the remote DSACON32m controller.

10.6.4.15 sMatrixInfo const& SDH::cDSA::GetMatrixInfo (int *m*) const [inline]

Return a reference to the sMatrixInfo of matrix *m* read from the remote DSACON32m controller.

10.6.4.16 sTactileSensorFrame const& SDH::cDSA::GetFrame () const [inline]

return a reference to the latest tactile sensor frame without reading from remote DSACON32m

10.6.4.17 sTactileSensorFrame const& SDH::cDSA::UpdateFrame () [inline]

read the tactile sensor frame from remote DSACON32m and return a reference to it. A command to query the frame (periodically) must have been sent before.

10.6.4.18 void cDSA::Close (void)

Set the framerate of the remote DSACON32m controller to 0 and close connection to it.

10.6.4.19 void cDSA::SetFramerate (UInt16 *framerate*, bool *do_RLE* = false, bool *do_data_acquisition* = true)

10.6.4.20 cDSA::tTexel cDSA::GetTexel (int *m*, int *x*, int *y*) const

Return texel value at column *x* row *y* of matrix *m* of the last frame

10.6.4.21 int SDH::cDSA::GetMatrixIndex (int *fi*, int *part*) [inline]

return the matrix index of the sensor matrix attached to finger with index *fi* [1..3] and *part* [0,1] = [proximal,distal]

10.6.4.22 unsigned long SDH::cDSA::GetAgeOfFrame (sTactileSensorFrame * *frame_p*) [inline]

return age of frame in ms (time in ms from frame sampling until now)

10.6.5 Friends And Related Function Documentation

10.6.5.1 std::ostream& operator<< (std::ostream & *stream*, cDSA::sResponse const & *response*) [friend]

10.6.6 Member Data Documentation

10.6.6.1 struct SDH::cDSA::sControllerInfo SDH::cDSA::__packed__

10.6.6.2 struct SDH::cDSA::sSensorInfo SDH::cDSA::__packed__

10.6.6.3 struct SDH::cDSA::sMatrixInfo SDH::cDSA::__packed__

10.6.6.4 struct SDH::cDSA::sResponse SDH::cDSA::__packed__ [protected]

10.6.6.5 cDBG SDH::cDSA::dbg [protected]

A stream object to print coloured debug messages.

10.6.6.6 cRS232 SDH::cDSA::comm_interface [protected]

the serial RS232 communication interface to use

10.6.6.7 bool SDH::cDSA::do_RLE [protected]

flag, true if data should be sent Run Length Encoded by the remote DSACON32m controller

10.6.6.8 sControllerInfo SDH::cDSA::controller_info [protected]

the controller info read from the remote DSACON32m controller

10.6.6.9 sSensorInfo SDH::cDSA::sensor_info [protected]

the sensor info read from the remote DSACON32m controller

10.6.6.10 sMatrixInfo* SDH::cDSA::matrix_info [protected]

the matrix infos read from the remote DSACON32m controller

10.6.6.11 sTactileSensorFrame SDH::cDSA::frame [protected]

the latest frame received from the remote DSACON32m controller

10.6.6.12 int SDH::cDSA::nb_cells [protected]

The total number of sensor cells.

10.6.6.13 int* SDH::cDSA::texel_offset [protected]

an array with the offsets of the first texel of all matrices into the whole frame

10.6.6.14 long SDH::cDSA::read_timeout_us [protected]

default timeout used for reading in us

10.6.6.15 cSimpleTime SDH::cDSA::start_pc [protected]

10.6.6.16 UInt32 SDH::cDSA::start_dsa [protected]

The documentation for this class was generated from the following files:

- sdh/dsa.h
- sdh/dsa.cpp

10.7 SDH::cDSA::sControllerInfo Struct Reference

`#include <dsa.h>`

10.7.1 Detailed Description

A data structure describing the controller info about the remote DSACON32m controller.

Public Attributes

- **UInt16** error_code
- **UInt32** serial_no
- **UInt8** hw_version
- **UInt16** sw_version
- **UInt8** status_flags
- **UInt8** feature_flags
- **UInt8** senscon_type
- **UInt8** active_interface
- **UInt32** can_baudrate
- **UInt16** can_id

10.7.2 Member Data Documentation

10.7.2.1 UInt16 SDH::cDSA::sControllerInfo::error_code

10.7.2.2 UInt32 SDH::cDSA::sControllerInfo::serial_no

10.7.2.3 UInt8 SDH::cDSA::sControllerInfo::hw_version

10.7.2.4 UInt16 SDH::cDSA::sControllerInfo::sw_version

10.7.2.5 UInt8 SDH::cDSA::sControllerInfo::status_flags

10.7.2.6 UInt8 SDH::cDSA::sControllerInfo::feature_flags

10.7.2.7 UInt8 SDH::cDSA::sControllerInfo::senscon_type

10.7.2.8 UInt8 SDH::cDSA::sControllerInfo::active_interface

10.7.2.9 UInt32 SDH::cDSA::sControllerInfo::can_baudrate

10.7.2.10 UInt16 SDH::cDSA::sControllerInfo::can_id

The documentation for this struct was generated from the following file:

- sdh/dsa.h

10.8 SDH::cDSA::sMatrixInfo Struct Reference

`#include <dsa.h>`

10.8.1 Detailed Description

A data structure describing a single sensor matrix connected to the remote DSACON32m controller.

Public Attributes

- **UInt16** error_code
- float texel_width
- float texel_height
- **UInt16** cells_x
- **UInt16** cells_y
- **UInt8** uid [6]
- **UInt8** reserved [2]
- **UInt8** hw_revision
- float matrix_center_x
- float matrix_center_y
- float matrix_center_z
- float matrix_theta_x
- float matrix_theta_y
- float matrix_theta_z
- float fullscale
- **UInt8** feature_flags

10.8.2 Member Data Documentation

10.8.2.1 UInt16 SDH::cDSA::sMatrixInfo::error_code

10.8.2.2 float SDH::cDSA::sMatrixInfo::texel_width

10.8.2.3 float SDH::cDSA::sMatrixInfo::texel_height

10.8.2.4 UInt16 SDH::cDSA::sMatrixInfo::cells_x

10.8.2.5 UInt16 SDH::cDSA::sMatrixInfo::cells_y

10.8.2.6 UInt8 SDH::cDSA::sMatrixInfo::uid[6]

10.8.2.7 UInt8 SDH::cDSA::sMatrixInfo::reserved[2]

10.8.2.8 UInt8 SDH::cDSA::sMatrixInfo::hw_revision

10.8.2.9 float SDH::cDSA::sMatrixInfo::matrix_center_x

10.8.2.10 float SDH::cDSA::sMatrixInfo::matrix_center_y

10.8.2.11 float SDH::cDSA::sMatrixInfo::matrix_center_z

10.8.2.12 float SDH::cDSA::sMatrixInfo::matrix_theta_x

10.8.2.13 float SDH::cDSA::sMatrixInfo::matrix_theta_y

10.8.2.14 float SDH::cDSA::sMatrixInfo::matrix_theta_z

10.8.2.15 float SDH::cDSA::sMatrixInfo::fullscale

10.8.2.16 UInt8 SDH::cDSA::sMatrixInfo::feature_flags

The documentation for this struct was generated from the following file:

- sdh/dsa.h

10.9 SDH::cDSA::sResponse Struct Reference

`#include <dsa.h>`

10.9.1 Detailed Description

data structure for storing responses from the remote DSACON32m controller

Public Member Functions

- sResponse (**UInt8** *_payload, int _max_payload_size)
  constructor to init pointer and max size

Public Attributes

- **UInt8** packet_id
- **UInt16** size
- **UInt8** * payload
- int max_payload_size

10.9.2 Constructor & Destructor Documentation

10.9.2.1 SDH::cDSA::sResponse::sResponse (UInt8 * *_payload*, int *_max_payload_size*) [inline]

constructor to init pointer and max size

10.9.3 Member Data Documentation

10.9.3.1 UInt8 SDH::cDSA::sResponse::packet_id

10.9.3.2 UInt16 SDH::cDSA::sResponse::size

10.9.3.3 UInt8* SDH::cDSA::sResponse::payload

10.9.3.4 int SDH::cDSA::sResponse::max_payload_size

The documentation for this struct was generated from the following file:

- sdh/dsa.h

10.10 SDH::cDSA::sSensorInfo Struct Reference

`#include <dsa.h>`

10.10.1 Detailed Description

A data structure describing the sensor info about the remote DSACON32m controller.

Public Attributes

- **UInt16** error_code
- **UInt16** nb_matrices
- **UInt16** generated_by
- **UInt8** hw_revision
- **UInt32** serial_no
- **UInt8** feature_flags

10.10.2 Member Data Documentation

10.10.2.1 UInt16 SDH::cDSA::sSensorInfo::error_code

10.10.2.2 UInt16 SDH::cDSA::sSensorInfo::nb_matrices

10.10.2.3 UInt16 SDH::cDSA::sSensorInfo::generated_by

10.10.2.4 UInt8 SDH::cDSA::sSensorInfo::hw_revision

10.10.2.5 UInt32 SDH::cDSA::sSensorInfo::serial_no

10.10.2.6 UInt8 SDH::cDSA::sSensorInfo::feature_flags

The documentation for this struct was generated from the following file:

- sdh/dsa.h

10.11 SDH::cDSA::sTactileSensorFrame Struct Reference

`#include <dsa.h>`

10.11.1 Detailed Description

A data structure describing a full tactile sensor frame read from the remote DSACON32m controller

Remarks:

- An object of this type is stored within the cDSA object
- You can get a reference to the sTactileSensorFrame object with the GetFrame() function.
- The object must be updated manually (for now)
  - by setting an appropriat framerate with SetFramerate() once
  - by calling UpdateFrame() periodically
-

Public Member Functions

- sTactileSensorFrame (void)

  constructor

Public Attributes

- **UInt32** timestamp

  the timestamp of the frame. Use GetAgeOfFrame() to set this into relation with the time of the PC.

- **UInt8** flags

  internal data

- tTexel * texel

  an 2D array of tTexel elements. Use GetTexel() for easy access to specific individuall elements.

10.11.2 Constructor & Destructor Documentation

10.11.2.1 SDH::cDSA::sTactileSensorFrame::sTactileSensorFrame (void) `[inline]`

constructor

10.11.3 Member Data Documentation

10.11.3.1 UInt32 SDH::cDSA::sTactileSensorFrame::timestamp

the timestamp of the frame. Use GetAgeOfFrame() to set this into relation with the time of the PC.

10.11.3.2 UInt8 SDH::cDSA::sTactileSensorFrame::flags

internal data

10.11.3.3 tTexel* SDH::cDSA::sTactileSensorFrame::texel

an 2D array of tTexel elements. Use GetTexel() for easy access to specific individuall elements.

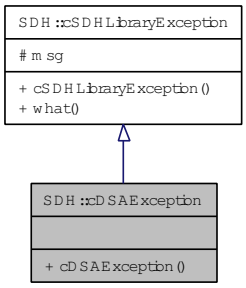
The documentation for this struct was generated from the following file:

- sdh/dsa.h

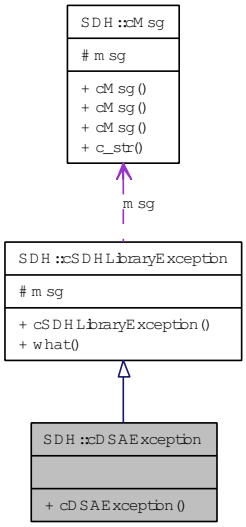
10.12 SDH::cDSAException Class Reference

`#include <dsa.h>`

Inheritance diagram for SDH::cDSAException:



Collaboration diagram for SDH::cDSAException:



10.12.1 Detailed Description

Derived exception class for low-level DSA related exceptions.

Public Member Functions

- cDSAException (cMsg const &_msg)

10.12.2 Constructor & Destructor Documentation

10.12.2.1 SDH::cDSAException::cDSAException (cMsg const & *_msg*) [inline]

The documentation for this class was generated from the following file:

- sdh/dsa.h

10.13 cDSAOptions Class Reference

`#include <dsaoptions.h>`

Public Member Functions

- cDSAOptions (void)

  constructor: init members to their default values

- void Parse (int argc, char **argv, char const *helptext, char const *progname, char const *version, char const *libname, char const *librelease)

Public Attributes

- char * usage
- int dsaport
- int debug_level
- std::ostream * debuglog
- bool fullframe
- int framerate
- bool resulting
- bool sensorinfo
- bool controllerinfo
- int matrixinfo
- bool do_RLE

10.13.1 Constructor & Destructor Documentation

10.13.1.1 cDSAOptions::cDSAOptions (void)

constructor: init members to their default values

10.13.2 Member Function Documentation

10.13.2.1 void cDSAOptions::Parse (int *argc*, char ** *argv*, char const * *helptext*, char const * *progname*, char const * *version*, char const * *libname*, char const * *librelease*)

parse the command line parameters *argc*, *argv* into members. *helptext*, *progname*, *version*, *libname* and *librelease* are used when printing online help. start parsing at option with index *p_option_index parse all options if parse_all is true, else only one option is parsed

10.13.3 Member Data Documentation

10.13.3.1 char* cDSAOptions::usage

10.13.3.2 int cDSAOptions::dsaport

10.13.3.3 int cDSAOptions::debug_level

10.13.3.4 std::ostream* cDSAOptions::debuglog

10.13.3.5 bool cDSAOptions::fullframe

10.13.3.6 int cDSAOptions::framerate

10.13.3.7 bool cDSAOptions::resulting

10.13.3.8 bool cDSAOptions::sensorinfo

10.13.3.9 bool cDSAOptions::controllerinfo

10.13.3.10 int cDSAOptions::matrixinfo

10.13.3.11 bool cDSAOptions::do_RLE

The documentation for this class was generated from the following files:

- demo/dsaoptions.h
- demo/dsaoptions.cpp

10.14 SDH::cMsg Class Reference

`#include <sdhexception.h>`

10.14.1 Detailed Description

Class for short, fixed maximum length text messages.

Simple message objects for short, fixed maximum length text messages, but with printf like initialization.

An object of type SDH::cMsg contains an ASCII-Z string of maximum length eMAX_MSG. It can be initialized with a 'printf-style' format string by the constructor. It is used in the SDHLibrary to further specify the cause of an exception in human readable form.

See SDH::cSDHLibraryException::cSDHLibraryException() for exemplary use.

Public Member Functions

- cMsg ()
  Default constructor, init message to empty string.
- cMsg (cMsg const &other)
  Copy constructor, copy message content of other object to this object.
- cMsg (char const *fmt,...) SDH__attribute__((format(printf
  Constructor with printf like format, argument parameters.
- char const * c_str () const
  Return the C-string representation of the messag in this object.

Protected Types

- enum { eMAX_MSG = 256 }
  anonymous enum instead of define macros

Protected Attributes

- char msg [eMAX_MSG]

10.14.2 Member Enumeration Documentation

10.14.2.1 anonymous enum [protected]

anonymous enum instead of define macros

Enumerator:

eMAX_MSG maximum length in bytes of a message to store

10.14.3 Constructor & Destructor Documentation

10.14.3.1 cMsg::cMsg ()

Default constructor, init message to empty string.

10.14.3.2 cMsg::cMsg (cMsg const & *other*)

Copy constructor, copy message content of other object to this object.

10.14.3.3 cMsg::cMsg (char const * *fmt*, ...)

Constructor with printf like format, argument parameters.

10.14.4 Member Function Documentation

10.14.4.1 char const * cMsg::c_str () const

Return the C-string representation of the messag in this object.

10.14.5 Member Data Documentation

10.14.5.1 char SDH::cMsg::msg[eMAX_MSG] [protected]

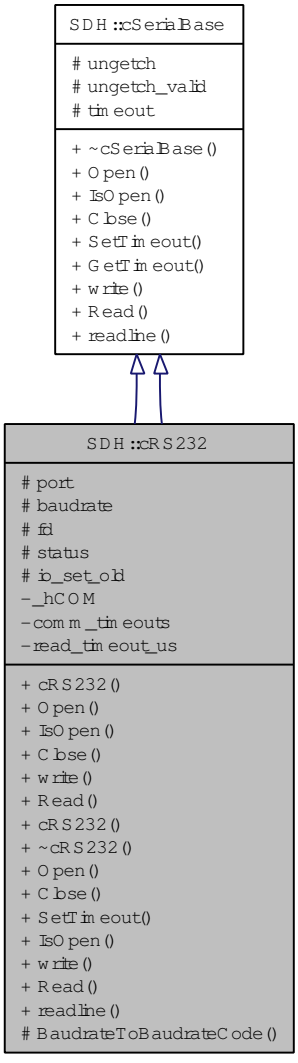
The documentation for this class was generated from the following files:

- sdh/sdhexception.h
- sdh/sdhexception.cpp

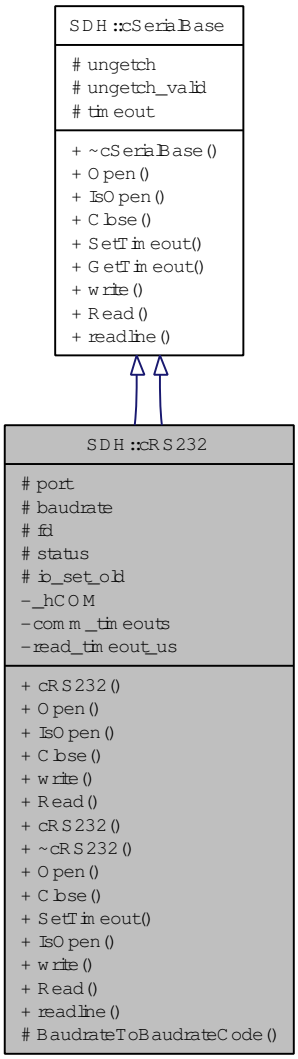
10.15 SDH::cRS232 Class Reference

`#include <rs232-vcc.h>`

Inheritance diagram for SDH::cRS232:



Collaboration diagram for SDH::cRS232:



10.15.1 Detailed Description

Low-level communication class to access a serial port on Cygwin and Linux.

Low-level communication class to access a serial port on VCC Windows.

Public Member Functions

- cRS232 (int _port, unsigned long _baudrate, double _timeout)
- void Open (void) throw (cRS232Exception*)
- bool IsOpen (void) throw ()

  Return true if port to RS232 is open.

- void Close (void) throw (cRS232Exception*)

  Close the previously opened rs232 port.

- int write (char const *ptr, int len=0) throw (cRS232Exception*)

  Write data to a previously opened port.

- ssize_t Read (void *data, ssize_t size, long timeout_us, bool return_on_less_data) throw (cRS232Exception*)
- cRS232 (int _port, unsigned long _baudrate, double _timeout)
- ~cRS232 (void)
- void Open (void) throw (cRS232Exception*)

  Open rs232 port port.

- void Close (void) throw (cRS232Exception*)

  Close the previously opened communication channel.

- virtual void SetTimeout (double _timeout) throw (cSerialBaseException*)

  set the timeout for next readline() calls (negative value means: no timeout, wait for ever)

- bool IsOpen () throw ()

  Return true if communication channel is open.

- int write (char const *ptr, int len=0) throw (cRS232Exception*)

  Write data to a previously opened port.

- ssize_t Read (void *data, ssize_t size, long timeout_us, bool return_on_less_data) throw (cRS232Exception*)
- char * readline (char *line, int size, char *eol, bool return_on_less_data) throw (cRS232Exception*)

  Read a line from the device.

Protected Member Functions

- tcflag_t BaudrateToBaudrateCode (unsigned long baudrate) throw (cRS232Exception*)

  Translate a baudrate given as unsigned long into a baudrate code for struct termios.

Protected Attributes

- int port

  the RS232 portnumber to use

- unsigned long baudrate

  the baudrate in bit/s

- int fd

  the file descriptor of the RS232 port

- int status
- termios io_set_old

10.15.2 Constructor & Destructor Documentation

10.15.2.1 cRS232::cRS232 (int *_port*, unsigned long *_baudrate*, double *_timeout*)

Constructor: constructs an object to communicate with an SDH via RS232

Parameters:

_port - rs232 device number: 0='COM1'='/dev/ttyS0', 1='COM2'='/dev/ttyS1', ...

_baudrate - the baudrate in bit/s

_timeout - the timeout in seconds

10.15.2.2 SDH::cRS232::cRS232 (int *_port*, unsigned long *_baudrate*, double *_timeout*)

Constructor: constructs an object to communicate with an SDH via RS232

Parameters:

_port - rs232 device number: 0='COM1'='/dev/ttyS0', 1='COM2'='/dev/ttyS1', ...

_baudrate - the baudrate in bit/s

_timeout - the timeout in seconds

10.15.2.3 cRS232::~cRS232 (void)

10.15.3 Member Function Documentation

10.15.3.1 tcflag_t cRS232::BaudrateToBaudrateCode (unsigned long *baudrate*) throw (cRS232Exception*) [protected]

Translate a baudrate given as unsigned long into a baudrate code for struct termios.

10.15.3.2 void cRS232::Open (void) throw (cRS232Exception*) [virtual]

Open the device as configured by the parameters given to the constructor

Implements SDH::cSerialBase.

10.15.3.3 bool cRS232::IsOpen (void) throw () [virtual]

Return true if port to RS232 is open.

Implements SDH::cSerialBase.

10.15.3.4 void cRS232::Close (void) throw (cRS232Exception*) [virtual]

Close the previously opened rs232 port.

Implements SDH::cSerialBase.

10.15.3.5 int cRS232::write (char const * *ptr*, int *len* = 0) throw (cRS232Exception*) [virtual]

Write data to a previously opened port.

Write *len* bytes from *ptr* to the rs232 device

Parameters:

ptr - pointer the byte array to send in memory

len - number of bytes to send

Returns:

the number of bytes actually written

!! dwWritten is always 0! Damn bloody windows

Implements SDH::cSerialBase.

10.15.3.6 ssize_t cRS232::Read (void * *data*, ssize_t *size*, long *timeout_us*, bool *return_on_less_data*) throw (cRS232Exception*) [virtual]

Read data from device. This function waits until *max_time_us* us passed or the expected number of bytes are received via serial line. if (*return_on_less_data* is true (default value), the number of bytes that have been received are returned and the data is stored in *data* If the *return_on_less_data* is false, data is only read from serial line, if at least *size* bytes are available.

Implements SDH::cSerialBase.

10.15.3.7 void SDH::cRS232::Open (void) throw (cRS232Exception*) [virtual]

Open rs232 port *port*.

Open the device with the parameters provided in the constructor

Implements SDH::cSerialBase.

10.15.3.8 void SDH::cRS232::Close (void) throw (cRS232Exception*) [virtual]

Close the previously opened communication channel.

Implements SDH::cSerialBase.

10.15.3.9 void cRS232::SetTimeout (double *_timeout*) throw (cSerialBaseException*) [virtual]

set the timeout for next readline() calls (negative value means: no timeout, wait for ever)

Reimplemented from SDH::cSerialBase.

10.15.3.10 bool SDH::cRS232::IsOpen () throw () [inline, virtual]

Return true if communication channel is open.

Implements SDH::cSerialBase.

10.15.3.11 int SDH::cRS232::write (char const * *ptr*, int *len* = 0) throw (cRS232Exception*) [virtual]

Write data to a previously opened port.

Write *len* bytes from *ptr* to the rs232 device

Parameters:

ptr - pointer the byte array to send in memory

len - number of bytes to send

Returns:

the number of bytes actually written

Implements SDH::cSerialBase.

10.15.3.12 ssize_t SDH::cRS232::Read (void * *data*, ssize_t *size*, long *timeout_us*, bool *return_on_less_data*) throw (cRS232Exception*) [virtual]

Read data from device. This function waits until *max_time_us* us passed or the expected number of bytes are received via serial line. if (*return_on_less_data* is true (default value), the number of bytes that have been received are returned and the data is stored in *data* If the *return_on_less_data* is false, data is only read from serial line, if at least *size* bytes are available.

Implements SDH::cSerialBase.

10.15.3.13 char * cRS232::readline (char * *line*, int *size*, char * *eol*, bool *return_on_less_data*) throw (cRS232Exception*) [virtual]

Read a line from the device.

A line is terminated with one of the end-of-line (eol) characters (’

’ by default) or until timeout

Parameters:

line - ptr to where to store the read line

size - space available in line (bytes)

eol - a string containing all the chars that mark an end of line

return_on_less_data - if (*return_on_less_data* is true (default value), the number of bytes that have been received are returned and the data is stored in *data* If the *return_on_less_data* is false, data is only read from serial line, if at least *size* bytes are available.

A pointer to the line read is returned.

Reimplemented from SDH::cSerialBase.

10.15.4 Member Data Documentation

10.15.4.1 int SDH::cRS232::port [protected]

the RS232 portnumber to use

the RS232 port number to use (port 0 is COM1)

10.15.4.2 unsigned long SDH::cRS232::baudrate [protected]

the baudrate in bit/s

10.15.4.3 int SDH::cRS232::fd [protected]

the file descriptor of the RS232 port

10.15.4.4 int SDH::cRS232::status [protected]

10.15.4.5 termios SDH::cRS232::io_set_old [protected]

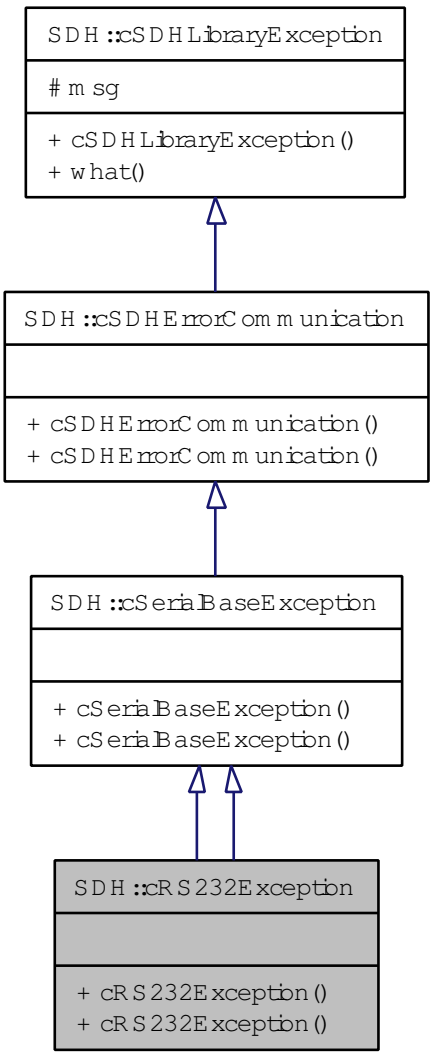
The documentation for this class was generated from the following files:

- sdh/rs232-cygwin.h
- sdh/rs232-vcc.h
- sdh/rs232-cygwin.cpp
- sdh/rs232-vcc.cpp

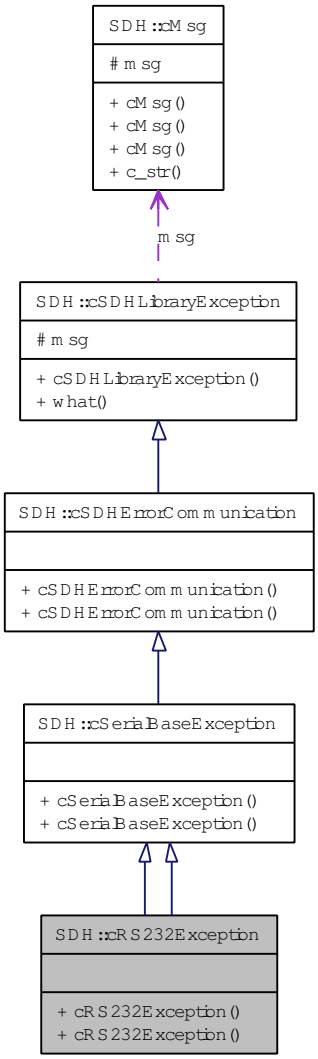
10.16 SDH::cRS232Exception Class Reference

`#include <rs232-vcc.h>`

Inheritance diagram for SDH::cRS232Exception:



Collaboration diagram for SDH::cRS232Exception:



10.16.1 Detailed Description

Derived exception class for low-level RS232 related exceptions.

Public Member Functions

- cRS232Exception (cMsg const &_msg)
- cRS232Exception (cMsg const &_msg)

10.16.2 Constructor & Destructor Documentation

10.16.2.1 SDH::cRS232Exception::cRS232Exception (cMsg const & *_msg*) [inline]

10.16.2.2 SDH::cRS232Exception::cRS232Exception (cMsg const & *_msg*) [inline]

The documentation for this class was generated from the following files:

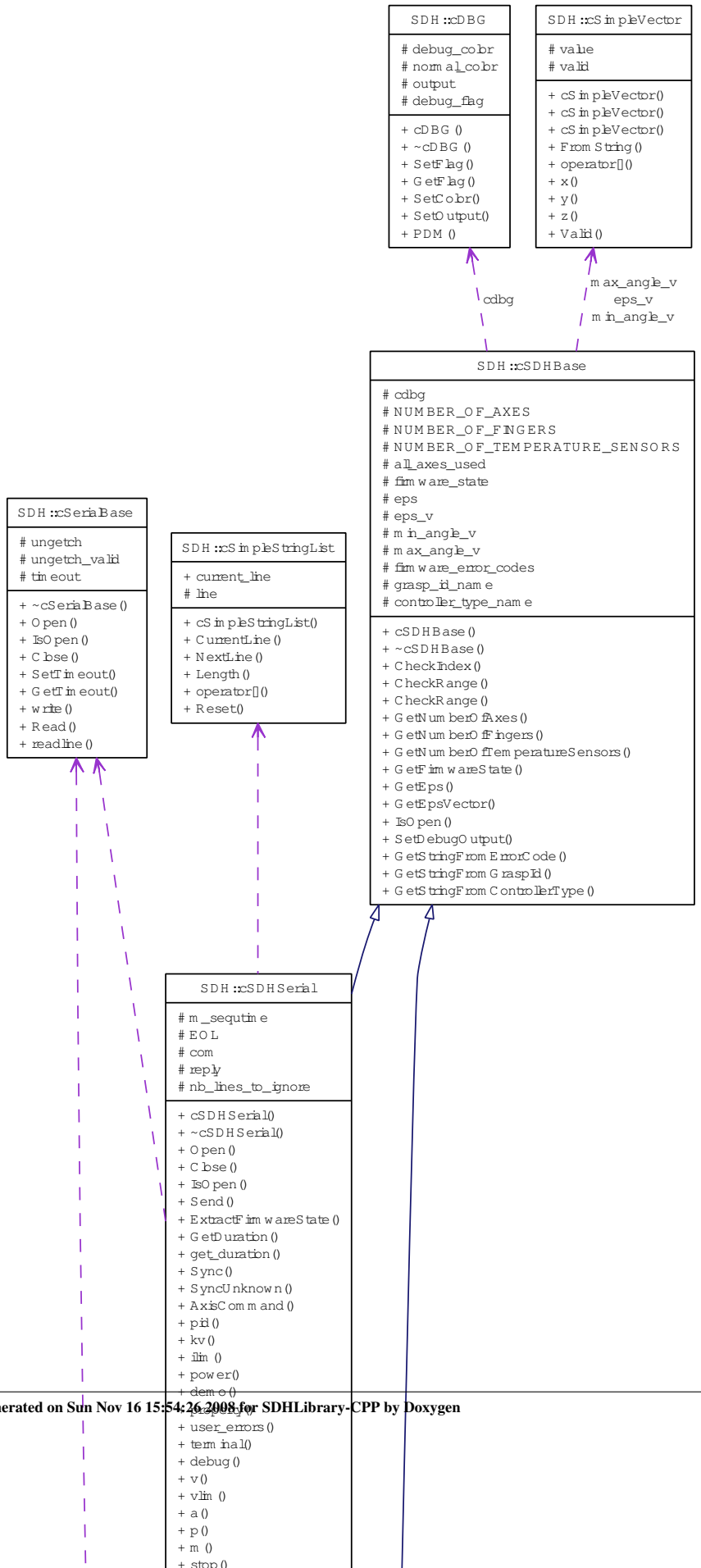
- sdh/rs232-cygwin.h
- sdh/rs232-vcc.h

10.17 SDH::cSDH Class Reference

`#include <sdh.h>`

Inheritance diagram for SDH::cSDH:

Collaboration diagram for SDH::cSDH:



10.17.1 Detailed Description

SDH::cSDH is the end user interface class to control a SDH (SCHUNK Dexterous Hand).

A general overview of the structure and architecture used is given here.

Remarks:

- The cSDH class provides methods to access the 7 axes of the SDH individually as well as on a finger level.
  - When accessing the axes individually then the following axis indices must be used to address an axis / some axes:
    - 0 : common base axis of finger 0 and 2
    - 1 : proximal axis of finger 0
    - 2 : distal axis of finger 0
    - 3 : proximal axis of finger 1
    - 4 : distal axis of finger 1
    - 5 : proximal axis of finger 2
    - 6 : distal axis of finger 2
  - When accessing the axes on finger level then every finger has 3 axes for a uniform interface of the access methods. Her the following finger axis indices must be used:
    - 0 : base axis of finger (for finger 1 this is a "virtual" axis with min angle = max angle = 0.0)
    - 1 : proximal axis of finger
    - 2 : distal axis of finger
- Vector-like parmeters: The interface functions defined here make full use of the flexibility provided by the STL vector<T> type. I.E. for parameters of functions like axis indices or axis angles not only single numerical values can be given, but also vectors of `int` or `double` values. This way the same (overloaded) interface function can address a single axis individually or multiple axes in a call, as required by the application. Such parameters are herein refered to as "vectors".
- Parameters for methods are checked for validity. In case an invalid parameter is given the method throws a cSDHErrorInvalidParameter exception.
- The underlying physical unit system of parameters that do have a unit (like angles, velocities or temperatures) can be adapted to the users or the applications need. See also unit conversion objects". The default converter objects are set as the uc_* member variables (uc_angle, uc_angular_velocity, uc_angular_acceleration, uc_time, uc_temperature, uc_position). The units are changed in the communication between user application and cSDH object instance only (USERAPP and SDHLibrary-CPP in the overview figure"). For now the firmware knows only about its internal unit system.

Kinematic parameters of the Hand

- double l1

  length of limb 1 (proximal joint to distal joint) in mm

- double l2

  length of limb 2 (distal joint to fingertip) in mm

- double d

- double h
- std::vector< std::vector< double > > offset
- cSerialBase * com
- cSDHSerial comm_interface

  The object to interface with the SDH attached via serial RS232 or CAN.

- virtual void SetDebugOutput (std::ostream *debuglog)

  change the stream to use for debug messages

Miscellaneous methods

- bool IsVirtualAxis (int iAxis) throw (cSDHLibraryException*)

  Return `true` *if index* iAxis *refers to a virtual axis.*

- void UseRadians (void)
- void UseDegrees (void)
- int GetFingerNumberOfAxes (int iFinger) throw (cSDHLibraryException*)
- int GetFingerAxisIndex (int iFinger, int iFingerAxis) throw (cSDHLibraryException*)
- char const * GetFirmwareRelease (void) throw (cSDHLibraryException*)
- char const * GetInfo (char const *what) throw (cSDHLibraryException*)
- std::vector< double > GetTemperature (std::vector< int > const &sensors) throw (cSDHLibraryException*)
- double GetTemperature (int iSensor) throw (cSDHLibraryException*)
- static char const * GetLibraryRelease (void)
- static char const * GetLibraryName (void)

Public Types

- enum eMotorCurrentMode { eMCM_MOVE = 0, eMCM_GRIP = 1, eMCM_HOLD = 2, eMCM_-DIMENSION }

  the motor current can be set specifically for these modes:

- enum eAxisState {
  eAS_IDLE = 0, eAS_POSITIONING, eAS_SPEED_MODE, eAS_NOT_INITIALIZED,
  eAS_CW_BLOCKED, eAS_CCW_BLOCKED, eAS_DISABLED, eAS_LIMITS_REACHED,
  eAS_DIMENSION }

  The state of an axis (see TPOSCON_STATE in global.h of the firmware).

Public Member Functions

- cSDH (bool _use_radians=false, bool _use_fahrenheit=false, int _debug_level=0)

  Constructor of cSDH class.

- virtual ~cSDH ()

Communication methods

- void OpenRS232 (int _port=0, unsigned long _baudrate=115200, double _timeout=-1) throw (cSDHLibraryException*)
- void OpenCAN_ESD (int _net=0, unsigned long _baudrate=1000000, double _timeout=0.0, int32_t _id_read=0x43, int32_t _id_write=0x42) throw (cSDHLibraryException*)
- void OpenCAN_ESD (NTCAN_HANDLE _ntcan_handle, double _timeout=0.0, int32_t _id_read=0x43, int32_t _id_write=0x42) throw (cSDHLibraryException*)
- void Close (bool leave_enabled=false) throw (cSDHLibraryException*)
- virtual bool IsOpen (void) throw ()

Auxiliary movement methods

- void EmergencyStop (void) throw (cSDHLibraryException*)
- void Stop (void) throw (cSDHLibraryException*)
- void SetController (eControllerType controller) throw (cSDHLibraryException*)
- eControllerType GetController (void)
- void SetVelocityProfile (eVelocityProfile velocity_profile) throw (cSDHLibraryException*)
- eVelocityProfile GetVelocityProfile (void) throw (cSDHLibraryException*)

Methods to access SDH on axis-level

- void SetAxisMotorCurrent (std::vector< int > const &axes, std::vector< double > const &motor_currents, eMotorCurrentMode mode=eMCM_MOVE) throw (cSDHLibraryException*)
- void SetAxisMotorCurrent (int iAxis, double motor_current, eMotorCurrentMode mode=eMCM_MOVE) throw (cSDHLibraryException*)
- std::vector< double > GetAxisMotorCurrent (std::vector< int > const &axes, eMotorCurrentMode mode=eMCM_MOVE) throw (cSDHLibraryException*)
- double GetAxisMotorCurrent (int iAxis, eMotorCurrentMode mode=eMCM_MOVE) throw (cSDHLibraryException*)
- void SetAxisEnable (std::vector< int > const &axes, std::vector< double > const &states) throw (cSDHLibraryException*)
- void SetAxisEnable (int iAxis=All, double state=1.0) throw (cSDHLibraryException*)
- void SetAxisEnable (std::vector< int > const &axes, std::vector< bool > const &states) throw (cSDHLibraryException*)
- void SetAxisEnable (int iAxis=All, bool state=true) throw (cSDHLibraryException*)
- std::vector< double > GetAxisEnable (std::vector< int > const &axes) throw (cSDHLibraryException*)
- double GetAxisEnable (int iAxis) throw (cSDHLibraryException*)
- std::vector< eAxisState > GetAxisActualState (std::vector< int > const &axes) throw (cSDHLibraryException*)
- eAxisState GetAxisActualState (int iAxis) throw (cSDHLibraryException*)
- void WaitAxis (std::vector< int > const &axes, double timeout=-1.0) throw (cSDHLibraryException*)
- void WaitAxis (int iAxis, double timeout=-1.0) throw (cSDHLibraryException*)
- void SetAxisTargetAngle (std::vector< int > const &axes, std::vector< double > const &angles) throw (cSDHLibraryException*)
- void SetAxisTargetAngle (int iAxis, double angle) throw (cSDHLibraryException*)
- std::vector< double > GetAxisTargetAngle (std::vector< int > const &axes) throw (cSDHLibraryException*)
- double GetAxisTargetAngle (int iAxis) throw (cSDHLibraryException*)
- std::vector< double > GetAxisActualAngle (std::vector< int > const &axes) throw (cSDHLibraryException*)
- double GetAxisActualAngle (int iAxis) throw (cSDHLibraryException*)
- void SetAxisTargetVelocity (std::vector< int > const &axes, std::vector< double > const &velocities) throw (cSDHLibraryException*)
- void SetAxisTargetVelocity (int iAxis, double velocity) throw (cSDHLibraryException*)
- std::vector< double > GetAxisTargetVelocity (std::vector< int > const &axes) throw (cSDHLibraryException*)

- double GetAxisTargetVelocity (int iAxis) throw (cSDHLibraryException*)
- std::vector< double > GetAxisLimitVelocity (std::vector< int > const &axes) throw (cSDHLibraryException*)
- double GetAxisLimitVelocity (int iAxis) throw (cSDHLibraryException*)
- std::vector< double > GetAxisActualVelocity (std::vector< int >const &axes) throw (cSDHLibraryException*)
- double GetAxisActualVelocity (int iAxis) throw (cSDHLibraryException*)
- void SetAxisTargetAcceleration (std::vector< int >const &axes, std::vector< double >const &accelerations) throw (cSDHLibraryException*)
- void SetAxisTargetAcceleration (int iAxis, double acceleration) throw (cSDHLibraryException*)
- std::vector< double > GetAxisTargetAcceleration (std::vector< int >const &axes) throw (cSDHLibraryException*)
- double GetAxisTargetAcceleration (int iAxis) throw (cSDHLibraryException*)
- std::vector< double > GetAxisMinAngle (std::vector< int > const &axes) throw (cSDHLibraryException*)
- double GetAxisMinAngle (int iAxis) throw (cSDHLibraryException*)
- std::vector< double > GetAxisMaxAngle (std::vector< int > const &axes) throw (cSDHLibraryException*)
- double GetAxisMaxAngle (int iAxis) throw (cSDHLibraryException*)
- std::vector< double > GetAxisMaxVelocity (std::vector< int > const &axes) throw (cSDHLibraryException*)
- double GetAxisMaxVelocity (int iAxis) throw (cSDHLibraryException*)
- std::vector< double > GetAxisMaxAcceleration (std::vector< int > const &axes) throw (cSDHLibraryException*)
- double GetAxisMaxAcceleration (int iAxis) throw (cSDHLibraryException*)
- double MoveAxis (std::vector< int >const &axes, bool sequ) throw (cSDHLibraryException*)
- double MoveAxis (int iAxis, bool sequ) throw (cSDHLibraryException*)

Methods to access SDH on finger-level

- void SetFingerEnable (std::vector< int > const &fingers, std::vector< double > const &states) throw (cSDHLibraryException*)
- void SetFingerEnable (int iFinger, double state=1.0) throw (cSDHLibraryException*)
- void SetFingerEnable (std::vector< int > const &fingers, std::vector< bool > const &states) throw (cSDHLibraryException*)
- void SetFingerEnable (int iFinger, bool state) throw (cSDHLibraryException*)
- std::vector< double > GetFingerEnable (std::vector< int > const &fingers) throw (cSDHLibraryException*)
- double GetFingerEnable (int iFinger) throw (cSDHLibraryException*)
- void SetFingerTargetAngle (int iFinger, std::vector< double > const &angles) throw (cSDHLibraryException*)
- void SetFingerTargetAngle (int iFinger, double a0, double a1, double a2) throw (cSDHLibraryException*)
- std::vector< double > GetFingerTargetAngle (int iFinger) throw (cSDHLibraryException*)
- void GetFingerTargetAngle (int iFinger, double &a0, double &a1, double &a2) throw (cSDHLibraryException*)
- std::vector< double > GetFingerActualAngle (int iFinger) throw (cSDHLibraryException*)
- void GetFingerActualAngle (int iFinger, double &a0, double &a1, double &a2) throw (cSDHLibraryException*)
- std::vector< double > GetFingerMinAngle (int iFinger) throw (cSDHLibraryException*)
- void GetFingerMinAngle (int iFinger, double &a0, double &a1, double &a2) throw (cSDHLibraryException*)
- std::vector< double > GetFingerMaxAngle (int iFinger) throw (cSDHLibraryException*)
- void GetFingerMaxAngle (int iFinger, double &a0, double &a1, double &a2) throw (cSDHLibraryException*)
- std::vector< double > GetFingerXYZ (int iFinger, std::vector< double > const &angles) throw (cSDHLibraryException*)

- std::vector< double > GetFingerXYZ (int iFinger, double a0, double a1, double a2) throw (cSDHLibraryException*)
- double MoveFinger (std::vector< int >const &fingers, bool sequ=true) throw (cSDHLibraryException*)
- double MoveFinger (int iFinger, bool sequ=true) throw (cSDHLibraryException*)
- double MoveHand (bool sequ=true) throw (cSDHLibraryException*)

Methods to access SDH grip skills

- double GetGripMaxVelocity (void)
- double GripHand (eGraspId grip, double close, double velocity, bool sequ=true) throw (cSDHLibraryException*)

Public Attributes

Predefined index vector objects

- std::vector< int > all_axes
  A vector with indices of all axes (in natural order), including the virtual axis.
- std::vector< int > all_fingers
  A vector with indices of all fingers (in natural order).
- std::vector< int > all_temperature_sensors
  A vector with indices of all temperature sensors.

Predefined unit conversion objects

Pointers to the unit converter objects used by this cSDH object.

The refered objects convert values between different unit systems. Example: convert angle values between degrees *and* radians*, temperatures between* degrees celsius *and* degrees fahrenheit *or the like.*

A cSDH object uses these converter objects to convert between external (user) and internal (SDH) units. The user can easily change the converter object that is used for a certain kind of unit. This way a cSDH object can easily report and accept parameters in the user or application specific unit system.

Additionally, users can easily add conversion objects for their own, even more user- or application-specific unit systems.

- const cUnitConverter * uc_angle
  unit convert for (axis) angles: default = SDH::cSDH::uc_angle_degrees
- const cUnitConverter * uc_angular_velocity
  unit convert for (axis) angular velocities: default = SDH::cSDH::uc_angular_velocity_degrees_per_-second
- const cUnitConverter * uc_angular_acceleration
  unit convert for (axis) angular accelerations: default = SDH::cSDH::uc_angular_acceleration_-degrees_per_second_squared
- const cUnitConverter * uc_time
  unit convert for times: default = uc_time_seconds
- const cUnitConverter * uc_temperature
  unit convert for temperatures: default = SDH::cSDH::uc_temperature_celsius

- const cUnitConverter * uc_motor_current
  unit converter for motor curent: default = SDH::cSDH::uc_motor_current_ampere

- const cUnitConverter * uc_position
  unit converter for position: default = SDH::cSDH::uc_position_millimeter

Static Public Attributes

Predefined unit conversion objecs

Some predefined cUnitConverter unit conversion objects to convert values between different unit systems. These are static members since the converter objects do not depend on the individiual cSDH object.

For every physical unit used in the cSDH class there is at least one (most of the time more than one) predefined unit converter. For example for angles there are radians *and* degrees.

- static cUnitConverter const uc_angle_degrees
  Default converter for angles (internal unit == external unit): degrees.

- static cUnitConverter const uc_angle_radians
  Converter for angles: external unit = radians.

- static cUnitConverter const uc_time_seconds
  Default converter for times (internal unit == external unit): seconds.

- static cUnitConverter const uc_time_milliseconds
  Converter for times: external unit = milliseconds.

- static cUnitConverter const uc_temperature_celsius
  Default converter for temparatures (internal unit == external unit): degrees celsius.

- static cUnitConverter const uc_temperature_fahrenheit
  Converter for temperatures: external unit = degrees fahrenheit.

- static cUnitConverter const uc_angular_velocity_degrees_per_second
  Default converter for angular velocities (internal unit == external unit): degrees / second.

- static cUnitConverter const uc_angular_velocity_radians_per_second
  Converter for angular velocieties: external unit = radians/second.

- static cUnitConverter const uc_angular_acceleration_degrees_per_second_squared
  Default converter for angular accelerations (internal unit == external unit): degrees / second.

- static cUnitConverter const uc_angular_acceleration_radians_per_second_squared
  Converter for angular velocieties: external unit = radians/second.

- static cUnitConverter const uc_motor_current_ampere
  Default converter for motor current (internal unit == external unit): Ampere.

- static cUnitConverter const uc_motor_current_milliampere
  Converter for motor current: external unit = milli Ampere.

- static cUnitConverter const uc_position_millimeter

Default converter for position (internal unit == external unit): millimeter.

- static cUnitConverter const uc_position_meter
  Converter for position: external unit = meter.

Protected Member Functions

Internal helper methods

- void SetAxisValueVector (std::vector< int > const &axes, std::vector< double > const &values, pSetFunction ll_set, pGetFunction ll_get, cUnitConverter const *uc, std::vector< double > const &min_values, std::vector< double > const &max_values, char const *name) throw (cSDHLibraryException*)
- std::vector< double > GetAxisValueVector (std::vector< int > const &axes, pGetFunction ll_get, cUnitConverter const *uc, char const *name) throw (cSDHLibraryException*)
- std::vector< int > ToIndexVector (int index, std::vector< int > &all_replacement, int maxindex, char const *name) throw (cSDHLibraryException*)
- pSetFunction GetMotorCurrentModeFunction (eMotorCurrentMode mode) throw (cSDHLibraryException*)
- std::vector< double > _GetFingerXYZ (int fi, std::vector< double > r_angles) throw (cSDHLibraryException*)

Protected Attributes

- int NUMBER_OF_AXES_PER_FINGER
  The number of axis per finger (for finger 1 this includes the "virtual" base axis).
- int NUMBER_OF_VIRTUAL_AXES
  The number of virtual axes.
- int nb_all_axes
  The number of all axes including virtual axes.
- std::vector< int > finger_number_of_axes
  Mapping of finger index to number of real axes of fingers:.
- std::vector< std::vector< int > > finger_axis_index
  Mapping of finger index, finger axis index to axis index:.
- std::vector< double > f_zeros_v
  Vector of 3 epsilon values.
- std::vector< double > f_ones_v
  Vector of 3 1.0 values.
- std::vector< double > zeros_v
  Vector of nb_all_axes 0.0 values.
- std::vector< double > ones_v
  Vector of nb_all_axes 1.0 values.

- std::vector< double > f_min_motor_current_v

  Minimum allowed motor currents (in internal units (Ampere)), including the virtual axis.

- std::vector< double > f_max_motor_current_v

  Maximum allowed motor currents (in internal units (Ampere)), including the virtual axis.

- std::vector< double > f_min_angle_v

  Minimum allowed axis angles (in internal units (degrees)), including the virtual axis.

- std::vector< double > f_max_angle_v

  Maximum allowed axis angles (in internal units (degrees)), including the virtual axis.

- std::vector< double > f_min_velocity_v

  Minimum allowed axis velocity (in internal units (degrees/second)), including the virtual axis.

- std::vector< double > f_max_velocity_v

  Maximum allowed axis velocity (in internal units (degrees/second)), including the virtual axis.

- std::vector< double > f_min_acceleration_v

  *Minimum allowed axis acceleration (in internal units (degrees/(second * second))), including the virtual axis.*

- std::vector< double > f_max_acceleration_v

  *Maximum allowed axis acceleration (in internal units (degrees/(second * second))), including the virtual axis.*

- double grip_max_velocity

  Maximum allowed grip velocity (in internal units (degrees/second)).

10.17.2 Member Enumeration Documentation

10.17.2.1 enum SDH::cSDH::eMotorCurrentMode

the motor current can be set specifically for these modes:

Enumerator:

eMCM_MOVE The motor currents used while "moving" with a MoveHand() or MoveFinger() command.

eMCM_GRIP The motor currents used while "gripping" with a GripHand() command.

eMCM_HOLD The motor currents used after "gripping" with a GripHand() command (i.e. "holding").

eMCM_DIMENSION Endmarker and Dimension.

10.17.2.2 enum SDH::cSDH::eAxisState

The state of an axis (see TPOSCON_STATE in global.h of the firmware).

Enumerator:

eAS_IDLE axis is idle

eAS_POSITIONING the goal position has not been reached yet

eAS_SPEED_MODE axis is in speed mode

eAS_NOT_INITIALIZED axis is not initialized or doesn't exist

eAS_CW_BLOCKED axis is blocked in counterwise direction

eAS_CCW_BLOCKED axis is blocked is blocked in against counterwise direction

eAS_DISABLED axis is disabled

eAS_LIMITS_REACHED position limits reached and axis stopped

eAS_DIMENSION Endmarker and Dimension.

10.17.3 Constructor & Destructor Documentation

10.17.3.1 cSDH::cSDH (bool *_use_radians* = `false`, bool *_use_fahrenheit* = `false`, int *_debug_level* = `0`)

Constructor of cSDH class.

Creates an new object of type cSDH. One such object is needed for each SDH that you want to control. The constructor initializes internal data structures. A connection the SDH is **not** yet established, see OpenRS232() on how to do that.

After an object is created the user can adjust the unit systems used to set/report parameters to/from SDH. This is shown in the example code below. The default units used (if not overwritten by constructor parameters) are:

- `degrees` [°] for (axis) angles
- `degrees per second` [°/s] for (axis) angular velocities
- `seconds` [s] for times
- `degrees celsius` [°C] for temperatures

Parameters:

_use_radians : Flag, if true then use radians and radians/second to set/report (axis) angles and angular velocities instead of default degrees and degrees/s.

_use_fahrenheit : Flag, if true then use degrees fahrenheit to report temperatures instead of default degrees celsius.

_debug_level : The level of debug messages to print

- 0: (default) no messages
- 1: messages of this cSDH instance
- 2: like 1 plus messages of the inner cSDHSerial instance

Examples:

Common use:

```
// Include the cSDH interface
#include <sdh.h>

// Create a cSDH object 'hand'.
cSDH hand();
```

The mentioned change of a unit system can be done like this:

```
// Assuming 'hand' is a cSDH object ...

// override default unit converter for (axis) angles:
hand.uc_angle = &cSDH::uc_angle_radians;

// override default unit converter for (axis) angular velocities:
hand.uc_angular_velocity = &cSDH::uc_angular_velocity_radians_per_second;

// override default unit converter for (axis) angular accelerations:
hand.uc_angular_acceleration = &cSDH::uc_angular_acceleration_radians_per_second_squared;

// instead of the last 3 calls the following shortcut could be used:
hand.UseRadians();

// override default unit converter for times:
hand.uc_time  = &cSDH::uc_time_milliseconds;

// override default unit converter for temperatures:
hand.uc_temperature = &cSDH::uc_temperature_fahrenheit;

// override default unit converter for positions:
hand.uc_position = &cSDH::uc_position_meter;
```

For convenience the most common settings can be specified as bool parameters for the constructor, like in:

```
// Include the cSDH interface
#include <sdh.h>

// Create a cSDH object 'hand' that uses
// - the non default radians and radians/s units,
// - the default temperature in degrees celsius,
// - A debug level of 2
cSDH hand( true, false, 2 );
```

unit convert for times: default = uc_time_seconds

unit convert for temperatures: default = uc_temperature_celsius

unit converter for motor curent: default = uc_motor_current_ampere

unit converter for position: default = uc_position_millimeter

The number of axis per finger (for finger 1 this includes the "virtual" base axis)

The number of virtual axes

Mapping of finger index to number of real axes of fingers:

Mapping of finger index, finger axis index to axis index:

Maximum allowed grip velocity (in internal units (degrees/second))

10.17.3.2 cSDH::~cSDH () [virtual]

Virtual destructor to make compiler happy

If the connection to the SDH hardware/firmware is still open then the connection is closed, which will stop the axis controllers (and thus prevent overheating).

10.17.4 Member Function Documentation

10.17.4.1 virtual void SDH::cSDH::SetDebugOutput (std::ostream * *debuglog*) [inline, virtual]

change the stream to use for debug messages

Reimplemented from SDH::cSDHBase.

10.17.4.2 void cSDH::SetAxisValueVector (std::vector< int > const & *axes*, std::vector< double > const & *values*, pSetFunction *ll_set*, pGetFunction *ll_get*, cUnitConverter const * *uc*, std::vector< double > const & *min_values*, std::vector< double > const & *max_values*, char const * *name*) throw (cSDHLibraryException*) [protected]

Generic set function: set some given axes to given values

Parameters:

axes - a vector of axis indices

values - a vector of values

ll_set - a pointer to the low level set function to use

ll_get - a pointer to the low level get function to use (for those axes where the given value is NaN)

uc - a pointer to the unit converter object to use before sending values to *ll_set*

min_values - a vector with the minimum allowed values

max_values - a vector with the maximum allowed values

name - a string with the name of the values (for constructing error message)

Remarks:

- The length of the *axis* and *values* vector must match.
- The indices can be given in any order, but the order of the elements of *axes* and *values* must match too. I.e. `values`[i] will be applied to axis `axes`[i] (not axis `i`)
- The indices are checked if they are valid axis indices.
- The values are checked if they are in the allowed range [*min_values* .. *f_max_values*], i.e. it is checked that `value`[i], converted to the internal unit system by *uc->ToInternal()*, is in [*min_values*[axes[i]] .. *max_values*[axes[i]]].
- If **any** index or value is invalid then **none** of the specified values is sent to the SDH, instead a SDH::cSDHErrorInvalidParameter* exception is thrown.

10.17.4.3 std::vector< double > cSDH::GetAxisValueVector (std::vector< int > const & *axes*, pGetFunction *ll_get*, cUnitConverter const * *uc*, char const * *name*) throw (cSDHLibraryException*) [protected]

Generic get function: get some given axes values

Parameters:

axes - a vector of axis indices

ll_get - a pointer to the low level get function to use

uc - a pointer to the unit converter object to apply before returning values

name - a string with the name of the values (for constructing error message)

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the addressed values for the selected axes.
- The values are converted to external unit system using the *uc* unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

10.17.4.4 std::vector< int > cSDH::ToIndexVector (int *index*, std::vector< int > & *all_replacement*, int *maxindex*, char const * *name*) throw (cSDHLibraryException*) [protected]

Internal helper function: return a vector of checked indices according to index.

Parameters:

index - The index to vectorize or All

all_replacement - a vector to return if *index* is All

maxindex - the *index* is checked if in [0.. maxindex[(i.e. not including *maxindex*)

name - A name for the things index, used to report out of bounds errors

Returns:

- If *index* is All then *all_replacement* is returned.
- If *index* is a single number >= 0 then it is checked if in [0.. maxindex[and a vector of length 1 is returned containing only *index*.
- In case *index* exceeds *maxindex* a (cSDHErrorInvalidParameter*) exception is thrown.

10.17.4.5 pSetFunction cSDH::GetMotorCurrentModeFunction (eMotorCurrentMode *mode*) throw (cSDHLibraryException*) [protected]

Internal helper function: return the get/set function of the comm_interface object that is responsible for setting/getting motor currents in *mode*.

10.17.4.6 std::vector< double > cSDH::_GetFingerXYZ (int *fi*, std::vector< double > *r_angles*) throw (cSDHLibraryException*) [protected]

return cartesian [x,y,z] position in mm of fingertip for finger fi at angles r_angles (rad)

10.17.4.7 bool cSDH::IsVirtualAxis (int *iAxis*) throw (cSDHLibraryException*)

Return `true` if index *iAxis* refers to a virtual axis.

10.17.4.8 void cSDH::UseRadians (void)

Shortcut to set the unit system to radians.

After calling this axis angles are set/reported in radians and angular velocities are set/reported in radians/second

Examples:

```
// Assuming 'hand' is a cSDH object ...

// make hand object use radians and radians/second for angles and angular velocities
hand.UseRadians();
```

10.17.4.9 void cSDH::UseDegrees (void)

Shortcut to set the unit system to degrees.

After calling this (axis) angles are set/reported in degrees and angular velocities are set/reported in degrees/second

Examples:

```
// Assuming 'hand' is a cSDH object ...

// make hand object use degrees and degrees/second for angles and angular velocities
hand.UseDegrees();
// as degrees, degrees/second are the default this is needed only if the
// unit system was changed before
```

10.17.4.10 int cSDH::GetFingerNumberOfAxes (int *iFinger*) throw (cSDHLibraryException*)

Return the number of real axes of finger with index *iFinger*.

Parameters:

iFinger - index of finger in range [0..NUMBER_OF_FINGERS-1]

Returns:

- Number of real axes of finger with index *iFinger*
- If *iFinger* is invalid a (cSDHErrorInvalidParameter*) exception is thrown.

Examples:

```
// Assuming 'hand' is a cSDH object ...

cout << "The finger 0 has " << hand.GetFingerNumberOfAxes( 0 ) << " real axes\n";
```

10.17.4.11 int cSDH::GetFingerAxisIndex (int *iFinger*, int *iFingerAxis*) throw (cSDHLibraryException*)

Return axis index of *iFingerAxis* axis of finger with index iFinger

For *iFinger=2*, iFingerAxis=0 this will return the index of the virtual base axis of the finger

Parameters:

iFinger - index of finger in range [0..NUMBER_OF_FINGERS-1]

iFingerAxis - index of finger axis in range [0..NUMBER_OF_AXES_PER_FINGER-1]

Returns:

- Axis index of *iFingerAxis-th* axis of finger with index *iFinger*
- If *iFinger* or *iFingerAxis* is invalid a (cSDHErrorInvalidParameter*) exception is thrown.

Examples:

```
// Assuming 'hand' is a cSDH object ...

cout << "The 1st axis of finger 2 has real axis index " << hand.GetFingerNumberOfAxes( 2, 0 )
```

10.17.4.12 char const * cSDH::GetLibraryRelease (void) [static]

Return the release name of the library (not the firmware of the SDH) as string.

Examples:

```
// static member functon, so no cSDH object is needed for access:

cout << "The SDHLibrary reports release name " << cSDH::GetReleaseLibrary() << "\n";
```

10.17.4.13 char const * cSDH::GetLibraryName (void) [static]

Return the name of the library as string.

Examples:

```
// static member functon, so no cSDH object is needed for access:

cout << "The SDHLibrary reports name " << cSDH::GetLibraryName() << "\n";
```

10.17.4.14 char const * cSDH::GetFirmwareRelease (void) throw (cSDHLibraryException*)

Return the release name of the firmware of the SDH (not the library) as string.

This will throw a (cSDHErrorCommunication*) exception if the connection to the SDH is not yet opened.

Examples:

```
// Assuming 'hand' is a cSDH object ...

cout << "The SDH firmware reports release " << hand.GetFirmwareRelease() << "\n";
```

10.17.4.15 char const * cSDH::GetInfo (char const * *what*) throw (cSDHLibraryException*)

Return info according to *what* # # The following values are valid for *what:* # - "date-library" : date of the SDHLibrary-python release # - "release-library" : release name of the sdh.py python module # - "release-firmware" : release name of the SDH firmware (requires # an opened communication to the SDH) # - "date-firmware" : date of the SDH firmware (requires # an opened communication to the SDH) # - "release-soc" : release name of the SDH SoC (requires # an opened communication to the SDH) # - "date-soc" : date of the SDH SoC (requires # an opened communication to the SDH) # - "id-sdh" : ID of SDH # - "sn-sdh" : Serial number of SDH # #

Examples:

```
#

    #    # Assuming 'hand' is a sdh.cSDH object ...
    #
    #    print "The SDH firmware reports release %s" % ( hand.GetInfo( "release-firmware" ) )
    #
    #

# #
```

10.17.4.16 std::vector< double > cSDH::GetTemperature (std::vector< int > const & *sensors*) throw (cSDHLibraryException*)

Return temperature(s) measured within the SDH.

Parameters:

sensors - A vector of indices of temperature sensors to access.

- index 0 is controller temperature
- index 1 is driver temperature

Remarks:

- The indices in *sensors* are checked if they are valid sensor indices.
- If **any** sensor index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

To access a single temperature sensor use GetTemperature(int), see there.

Returns:

The temperatures of the selected sensors are returned as std::vector<double> in the configured temperature unit system uc_temperature.

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Get measured values of all sensors
std::vector<double> temps = hand.GetTemperature( hand.all_temperature_sensors );
// Now temps is something like { 40.5, 35.5 }

// Get controller temperature only:
double temp_controller = hand.GetTemperature( 0 );
// Now temp_controller is something like 40.5
```

```
// If we - for some obscure islandish reason - would want
// temperatures reported in degrees fahrenheit, the unit
// converter can be changed:
hand.uc_temperature = &cSDH::uc_temperature_fahrenheit;

// Get all temperaturs again:
temps = hand.GetTemperature( hand.all_temperature_sensors );
// Now temps is something like {104.9, 95.9}
```

10.17.4.17 double cSDH::GetTemperature (int *iSensor*) throw (cSDHLibraryException*)

Like GetTemperature(std::vector<int>const&), just for one sensor *iSensor* and returning a single temperature as double.

10.17.4.18 void cSDH::OpenRS232 (int *_port* = 0, unsigned long *_baudrate* = 115200, double *_timeout* = -1) throw (cSDHLibraryException*)

Open connection to SDH via RS232.

Parameters:

_port : The number of the serial port to use. The default value port=0 refers to 'COM1' in Windows and to the corresponding '/dev/ttyS0' in Linux.

_baudrate,: the baudrate in bit/s, the default is 115200 which happens to be the default for the SDH too

_timeout : The timeout to use:

- -1 : wait forever
- T : wait for T seconds

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Open connection to SDH via default port:
hand.OpenRS232();

// Use a different port 2 == COM3 == /dev/ttyS2 for a second hand "hand2":
cSDH hand2();
hand2.OpenRS232( 2 );
```

10.17.4.19 void cSDH::OpenCAN_ESD (int *_net* = 0, unsigned long *_baudrate* = 1000000, double *_timeout* = 0.0, int32_t *_id_read* = 0x43, int32_t *_id_write* = 0x42) throw (cSDHLibraryException*)

Open connection to SDH via CAN using an ESD CAN card. If the library was compiled without ESD CAN support then this will just throw an exception. See setting for WITH_ESD_CAN in the top level makefile.

Parameters:

_net : The ESD CAN net number of the CAN port to use. (default: 0)

_baudrate : the CAN baudrate in bit/s. Only some bitrates are valid: (1000000 (default),800000,500000,250000,125000,100000,50000,20000,10000)

_timeout : The timeout to use:

- <= 0 : wait forever (default)
- T : wait for T seconds

_id_read - the CAN ID to use for reading (The SDH sends data on this ID, default=0x43)

_id_write - the CAN ID to use for writing (The SDH receives data on this ID, default=0x42)

Examples:

```
// Assuming 'hand' is a cSDH object ...

// use default parameters for net, baudrate, timeout and IDs
hand.OpenCAN_ESD( );

// use non default settings:
// net=1, baudrate=500000, timeout=1.0, id_read=0x143, id_write=0x142
hand.OpenCAN_ESD( 1, 500000, 1.0, 0x143, 0x142 );
```

10.17.4.20 void cSDH::OpenCAN_ESD (NTCAN_HANDLE *_ntcan_handle*, double *_timeout* = 0.0, int32_t *_id_read* = 0x43, int32_t *_id_write* = 0x42) throw (cSDHLibraryException*)

Open connection to SDH via CAN using an ESD CAN card using an existing handle. If the library was compiled without ESD CAN support then this will just throw an exception. See setting for WITH_ESD_-CAN in the top level makefile.

Parameters:

_ntcan_handle : The ESD CAN handle to reuse to connect to the ESD CAN driver

_timeout : The timeout to use:

- <= 0 : wait forever (default)
- T : wait for T seconds

_id_read - the CAN ID to use for reading (The SDH sends data on this ID, default=0x43)

_id_write - the CAN ID to use for writing (The SDH receives data on this ID, default=0x42)

Examples:

```
// Assuming 'hand' is a cSDH object ...
// and 'handle' is a valid ESD NTCAN_HANDLE

// use default parameters for timeout and IDs
hand.OpenCAN_ESD( handle );

// or use non default settings:
// timeout=1.0, id_read=0x143, id_write=0x142
hand.OpenCAN_ESD( handle, 1.0, 0x143, 0x142 );
```

10.17.4.21 void cSDH::Close (bool *leave_enabled* = false) throw (cSDHLibraryException*)

Close connection to SDH.

The default behaviour is to **not** leave the controllers of the SDH enabled (to prevent overheating). To keep the controllers enabled (e.g. to keep the finger axes actively in position) set *leave_enabled* to true. Only already enabled axes will be left enabled.

Parameters:

leave_enabled - Flag: true to leave the controllers on, false (default) to disable the controllers (switch powerless)

This throws a (cSDHErrorCommunication*) exception if the connection was not opened before.

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Close connection to SDH, power off controllers:
hand.Close();

// To leave the already enabled controllers enabled:
hand.Close( true );
```

10.17.4.22 bool cSDH::IsOpen (void) throw () [virtual]

Return true if connection to SDH firmware/hardware is open.

Implements SDH::cSDHBase.

10.17.4.23 void cSDH::EmergencyStop (void) throw (cSDHLibraryException*)

Stop movement of all axes of the SDH and switch off the controllers

This command will always be executed sequentially: it will return only after the SDH has confirmed the emergency stop.

Bug

For now this will **NOT** work while a "Grasp" command is executing, even if that was initiated non-sequentially!

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Perform an emergency stop:
hand.EmergencyStop();
```

10.17.4.24 void cSDH::Stop (void) throw (cSDHLibraryException*)

Stop movement of all axes but keep controllers on

This command will always be executed sequentially: it will return only after the SDH has confirmed the stop

Bug

For now this will **NOT** work while a "Grasp" command is executing, even if that was initiated non-sequentially!

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Perform a stop:
hand.Stop();
```

10.17.4.25 void cSDH::SetController (eControllerType *controller*) throw (cSDHLibraryException*)

Set the type of axis controller to be used in the SDH

(This is currently not very usefull as only one controller type is defined.)

Parameters:

controller - identifier of controller to set. Valid values are defined in eControllerType

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Set the pose controller in the SDH:
hand.SetController( hand.eCT_POSE );
```

10.17.4.26 cSDHBase::eControllerType cSDH::GetController (void)

Get the type of axis controller used in the SDH

This is currently unimplemented in the SDH. eCT_POSE will be returned always for now.

10.17.4.27 void cSDH::SetVelocityProfile (eVelocityProfile *velocity_profile*) throw (cSDHLibraryException*)

Set the type of velocity profile to be used in the SDH

Parameters:

velocity_profile - Name or number of velocity profile to set. Valid values are defined in eVelocityProfileType

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Set the sin square velocity profile in the SDH:
hand.SetVelocityProfile( hand.eVP_SIN_SQUARE );

// Or else set the ramp velocity profile in the SDH:
hand.SetVelocityProfile( hand.eVP_RAMP )
```

10.17.4.28 cSDHBase::eVelocityProfile cSDH::GetVelocityProfile (void) throw (cSDHLibraryException*)

Get the type of velocity profile used in the SDH

Returns:

the currently set velocity profile as integer, see eVelocityProfileType

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Get the velocity profile from the SDH:
velocity_profile = hand.GetVelocityProfile();
// now velocity_profile is something like eVP_SIN_SQUARE or eVP_RAMP
```

10.17.4.29 void cSDH::SetAxisMotorCurrent (std::vector< int > const & *axes*, std::vector< double > const & *motor_currents*, eMotorCurrentMode *mode* = eMCM_MOVE) throw (cSDHLibraryException*)

Set the maximum allowed motor current(s) for axes.

The maximum allowed motor currents are sent to the SDH. The motor currents can be stored:

- axis specific
- mode specific (see eMotorCurrentMode)

Parameters:

axes - A vector of axis indices to access.

motor_currents - A vector of motor currents to set. If any of the numbers in the vector is NaN (Not a Number) then the currently set axis motor current will be kept for the corresponding axis. The value(s) are expected in the configured motor current unit system uc_motor_current.

mode - the mode to set the maximum motor current for. One of the eMotorCurrentMode modes.

Remarks:

- The lengths of the *axes* and *motor_currents* vector must match.
- The indices can be given in any order, but the order of their elements must match, i.e. motor_currents[i] will be applied to axis axes[i] (not axis i).
- The indices are checked if they are valid axis indices.
- The motor currents are checked if they are in the allowed range [0 .. f_max_motor_current_v], i.e. it is checked that motor_currents[i], converted to internal units, is in [0 .. f_max_motor_currents_v[axes[i]]].
- If **any** index or value is invalid then **none** of the specified values is sent to the SDH, instead a SDH::cSDHErrorInvalidParameter* exception is thrown.

See also SetAxisMotorCurrent(int,double,eMotorCurrentMode) for an overloaded variant to set a single axis motor current or to set the same motor current for all axes.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Set maximum allowed motor current of all axes to the given values in mode "eMCM_MOVE"::
std::vector<double> all_motor_currents;
all_motor_currents.push_back( 0.0 );
all_motor_currents.push_back( 0.1 );
```

```
all_motor_currents.push_back( 0.2 );
all_motor_currents.push_back( 0.3 );
all_motor_currents.push_back( 0.4 );
all_motor_currents.push_back( 0.5 );
all_motor_currents.push_back( 0.6 );

hand.SetAxisMotorCurrent( hand.all_axes, all_motor_currents );


// Set maximum allowed motor current of all axes to 0.1 A in mode "eMCM_HOLD":
hand.SetAxisMotorCurrent( hand.All, 1.0, eMCM_HOLD );

// Set maximum allowed motor current of axis 3 to 0.75 A in mode "eMCM_MOVE":
hand.SetAxisMotorCurrent( 3, 0.75, eMCM_MOVE );

// Set maximum allowed motor current of for axis 0, 4 and 2 to 0.0 A,
// 0.4 A and 0.2 A respectively in mode "eMCM_GRIP"
std::vector<int> axes042;
axes042.push_back( 0 );
axes042.push_back( 4 );
axes042.push_back( 2 );
std::vector<double> motor_currents042;
motor_currents042.push_back( 0.0 );
motor_currents042.push_back( 0.4 );
motor_currents042.push_back( 0.2 );

hand.SetAxisMotorCurrent( axes042, states042, eMCM_GRIP );
```

10.17.4.30 void cSDH::SetAxisMotorCurrent (int *iAxis*, double *motor_current*, eMotorCurrentMode *mode* = eMCM_MOVE) throw (cSDHLibraryException*)

Like SetAxisMotorCurrent(std::vector<int>const&,std::vector<double>const&,eMotorCurrentMode), just for a single axis *iAxis* and a single motor current *motor_current*, see there.

If *iAxis* is All then *motor_current* is set for all axes.

10.17.4.31 std::vector< double > cSDH::GetAxisMotorCurrent (std::vector< int > const & *axes*, eMotorCurrentMode *mode* = eMCM_MOVE) throw (cSDHLibraryException*)

Get the maximum allowed motor current(s) of axis(axes).

The maximum allowed motor currents are read from the SDH. The motor currents are stored:

- axis specific
- mode specific (see eMotorCurrentMode)

Parameters:

axes - A vector of axis indices to access.

mode - the mode to set the maximum motor current for. One of the eMotorCurrentMode modes.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the motor currents of the selected axes.

- The values are converted to the selected external unit system using the configured uc_motor_-current unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisMotorCurrent(int,eMotorCurrentMode) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get maximum allowed motor currents of all axes
std::vector<double> v = hand.GetAxisMotorCurrent( hand.all_axes );
// now v is something like {0.1, 0.2, 0.3, 0.4, 0.5, 0,6, 0.7}

// Get maximum allowed motor current of axis 3 in mode "eMCM_MOVE"
double mc3 = hand.GetAxisMotorCurrent( 3, eMCM_MOVE );
// mc3 is now something like 0.75

// Get maximum allowed motor current of axis 3 and 5 in mode "eMCM_GRIP"
std::vector<int> axes35;
axes35.push_back( 3 );
axes35.push_back( 5 );

v = hand.GetAxisMotorCurrent( axes35, eMCM_GRIP );
// now L is something like {0.5,0.5};
```

10.17.4.32 double cSDH::GetAxisMotorCurrent (int *iAxis*, eMotorCurrentMode *mode* = `eMCM_MOVE`) throw (cSDHLibraryException*)

Like GetAxisMotorCurrent(std::vector<int>const&,eMotorCurrentMode), just for a single axis, see there for details and examples.

10.17.4.33 void cSDH::SetAxisEnable (std::vector< int > const & *axes*, std::vector< double > const & *states*) throw (cSDHLibraryException*)

Set enabled/disabled state of axis controller(s).

The controllers of the selected axes are enabled/disabled in the SDH. Disabled axes are not powered and thus might not remain in their current pose due to gravity, inertia or other external influences. But to prevent overheating the axis controllers should be switched of when not needed.

Parameters:

axes - A vector of axis indices to access.

states - A vector of enabled states (0 = disabled, !=0 = enabled) to set. If any of the numbers in the vector is `NaN` (Not a Number) then the currently set enabled state will be kept for the corresponding axis.

Remarks:

- The lengths of the *axes* and *states* vector must match.
- The indices can be given in any order, but the order of their elements must match, i.e. `state`[i] will be applied to axis `axes`[i] (not axis `i`).
- The indices are checked if they are valid axis indices.

- If **any** index is invalid then **none** of the specified values is sent to the SDH, instead a SDH::cSDHErrorInvalidParameter* exception is thrown.

See also SetAxisEnable(int,double), SetAxisEnable(int,bool) for overloaded variants to set a single axis enabled/disabled or to set the same state for all axes. See further SetAxisEnable(std::vector<int>const&,std::vector<bool>const&) for a variant that accepts a `bool` vector for the states to set.

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Enable all axes:
hand.SetAxisEnable( hand.all_axes, hand.ones_v );

// Disable all axes:
hand.SetAxisEnable( All, 0 );

// Enable axis 0 and 2 while disabling axis 4:
std::vector<int> axes042;
axes042.push_back( 0 );
axes042.push_back( 4 );
axes042.push_back( 2 );

std::vector<double> states042;
states042.push_back( 1.0 );
states042.push_back( 0.0 );
states042.push_back( 1.0 );

hand.SetAxisEnable( axes042, states042 );


// Disable axis 2
hand.SetAxisEnable( 2, false );
```

10.17.4.34 void cSDH::SetAxisEnable (int *iAxis* = `All`, double *state* = `1.0`) throw (cSDHLibraryException*)

Like SetAxisEnable(std::vector<int>const&,std::vector<double>const&), just for a single axis *iAxis* and a single axis state *state*, see there.

If *iAxis* is All then *state* is applied to all axes.

10.17.4.35 void cSDH::SetAxisEnable (std::vector< int > const & *axes*, std::vector< bool > const & *states*) throw (cSDHLibraryException*)

Like SetAxisEnable(std::vector<int>const&,std::vector<double>const&), just accepting a vector of `bool` values as states, see there.

10.17.4.36 void cSDH::SetAxisEnable (int *iAxis* = `All`, bool *state* = `true`) throw (cSDHLibraryException*)

Like SetAxisEnable(std::vector<int>const&,std::vector<double>const&), just for a single axis *iAxis* and a single axis state *state*, see there.

If *iAxis* is All then *state* is applied to all axes.

10.17.4.37 std::vector< double > cSDH::GetAxisEnable (std::vector< int > const & *axes*) throw (cSDHLibraryException*)

Get enabled/disabled state of axis controller(s).

The enabled/disabled state of the controllers of the selected axes is read from the SDH.

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of enabled/disabled states as doubles (0=disabled, 1.0=enabled) of the selected axes.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisEnable(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Get enabled state of all axes:
std::vector<double> v = hand.GetAxisEnable( hand.all_axes );
// now v is something like {0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0}

// Get enabled state of axis 3 and 5
std::vector<int> axes35;
axes35.push_back( 3 );
axes35.push_back( 5 );

v = hand.GetAxisEnable( axes35 );
// now v is something like {1.0, 0.0}


// Get enabled state of axis 3
double v3 = hand.GetAxisEnable( 3 );
// now v3 is something like 1.0
```

10.17.4.38 double cSDH::GetAxisEnable (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisEnable(std::vector<int>const&), just for a single axis *iAxis*, see there for details and examples.

10.17.4.39 std::vector< cSDH::eAxisState > cSDH::GetAxisActualState (std::vector< int > const & *axes*) throw (cSDHLibraryException*)

Get the current actual state(s) of axis(axes).

The actual axis states are read from the SDH.

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the actual states of the selected axes.
- The values are given as eAxisState enum values
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisActualState(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get actual axis state of all axes
std::vector<eAxisState> v = hand.GetAxisActualState( hand.all_axes )
// now v is something like {eAS_IDLE, eAS_POSITIONING, eAS_IDLE, eAS_IDLE, eAS_IDLE, eAS_DIS

// Get actual axis state of axis 3
eAxisState v3 = hand.GetAxisActualState( 3 );
// v3 is now something like eAS_IDLE


// Get actual state of axis 2 and 5
std::vector<int> axes25;
axes25.push_back( 2 );
axes25.push_back( 5 );

v = hand.GetAxisActualState( axes25 );
// now v is something like {eAS_IDLE, eAS_DISABLED}
```

10.17.4.40 cSDH::eAxisState cSDH::GetAxisActualState (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisActualState(std::vector<int>const&), just for a single axis *iAxis*, see there for details and examples.

10.17.4.41 void cSDH::WaitAxis (std::vector< int > const & *axes*, double *timeout* = `-1.0`) throw (cSDHLibraryException*)

Wait until the movement(s) of of axis(axes) has finished

The state of the given axis(axes) is(are) queried until all axes are no longer moving.

Parameters:

axes - A vector of axis indices to access.

timeout - a timeout in seconds or -1.0 (default) to wait indefinetly.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

- If *timeout* < 0 then this function will wait arbitrarily long
- If a *timeout* is given then this function will throw a cSDHErrorCommunication exception if the given axes are still moving after *timeout* many seconds

See also WaitAxis(int,double) for an overloaded variant to wait for a single axis or all axes.

Examples:

See also the demo program demo-simple3

```
// Assuming "hand" is a cSDH object ...

// Set a new target pose for axis 1,2 and 3
std::vector<int> axes123;
axes123.push_back( 1 );
axes123.push_back( 2 );
axes123.push_back( 3 );

std::vector<double> angles123;
angles123.push_back( -20.0 );
angles123.push_back( -30.0 );
angles123.push_back( -40.0 );

hand.SetAxisTargetAngle( axes123, angles123 );

// Move axes there non sequentially:
hand.MoveAxis( axes123, false );

// The last call returned immediately so we now have time to
// do something else while the hand is moving:

// ... insert any calculation here ...

// Before doing something else with the hand make shure the
// selected axes have finished the last movement:
hand.WaitAxis( axes123 );


// go back home (all angles to 0.0):
hand.SetAxisTargetAngle( hand.All, 0.0 );

// Move all axes there non sequentially:
hand.MoveAxis( hand.All, False );

// ... insert any other calculation here ...

// Wait until all axes are there, with a timeout of 10s:
hand.WaitAxis( hand.All, 10.0 );
```

10.17.4.42 void cSDH::WaitAxis (int *iAxis*, double *timeout* = -1.0) throw (cSDHLibraryException*)

Like WaitAxis(std::vector<int>const&,double), just for a single axis *iAxis*, see there for details and examples.

If *iAxis* is All then wait for all axes axes.

10.17.4.43 void cSDH::SetAxisTargetAngle (std::vector< int > const & *axes*, std::vector< double > const & *angles*) throw (cSDHLibraryException*)

Set the target angle(s) for axis(axes).

The target angles are stored in the SDH, the movement is not executed until an additional move command is sent.

Parameters:

axes - A vector of axis indices to access.

angles - A vector of axis target angles to set. If any of the numbers in the vector is `NaN` (Not a Number) then the currently set axis target angle will be kept for the corresponding axis. The value(s) are expected in the configured angle unit system uc_angle.

Remarks:

- Setting the target angle will **not** make the axis/axes move.
- The lengths of the *axes* and *angles* vector must match.
- The indices can be given in any order, but the order of their elements must match, i.e. `angles`[i] will be applied to axis `axes`[i] (not axis `i`).
- The indices are checked if they are valid axis indices.
- The angles are checked if they are in the allowed range [0 .. f_max_angle_v], i.e. it is checked that `angles`[i], converted to internal units, is in [0 .. `f_max_angle_v`[axes[i]]].
- If **any** index or value is invalid then **none** of the specified values is sent to the SDH, instead a SDH::cSDHErrorInvalidParameter* exception is thrown.

See also SetAxisTargetAngle(int,double) for an overloaded variant to set a single axis target angle or to set the same target angle for all axes.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Set target axis angle of all axes to the given values:
std::vector<double> all_angles;
all_angles.push_back( 0.0 );
all_angles.push_back( -11.0 );
all_angles.push_back( -22.0 );
all_angles.push_back( -33.0 );
all_angles.push_back( -44.0 );
all_angles.push_back( -55.0 );
all_angles.push_back( -66.0 );

hand.SetAxisTargetAngle( hand.all_axes, all_angles );


// Set target axis angle of axis 3 to -42°:
hand.SetAxisTargetAngle( 3, -42.0 );

// Set target angle of for axis 0, 4 and 2 to 0.0°, -44.4° and -2.22° respectively:
std::vector<int> axes042;
axes042.push_back( 0 );
axes042.push_back( 4 );
axes042.push_back( 2 );
std::vector<double> angles042;
angles042.push_back( 0.0 );
angles042.push_back( -44.4 );
angles042.push_back( -2.22 );

hand.SetAxisTargetAngle( axes042, angles042 );


// Set target axis angle of all axes to 0° (home-position)
hand.SetAxisTargetAngle( hand.All, 0.0 );
```

10.17.4.44 void cSDH::SetAxisTargetAngle (int *iAxis*, double *angle*) throw (cSDHLibraryException*)

Like SetAxisTargetAngle(std::vector<int>const&,std::vector<double>const&), just for a single axis *iAxis* and a single angle *angle*, see there for details and examples.

If *iAxis* is All then *motor_current* is set for all axes.

10.17.4.45 std::vector< double > cSDH::GetAxisTargetAngle (std::vector< int > const & *axes*) throw (cSDHLibraryException*)

Get the target angle(s) of axis(axes).

The currently set target angles are read from the SDH.

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the target angles of the selected axes.
- The values are converted to the selected external unit system using the configured uc_angle unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisTargetAngle(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get target axis angle of all axes
std::vector<double> v = hand.GetAxisTargetAngle( hand.all_axes );
// now v is something like {0.0, 0.0, 42.0, 0.0, 47.11, 0,0, 0.0}

// Get target axis angle of axis 2
double v2 = hand.GetAxisTargetAngle( 2 );
// v2 is now something like 42.0


// Get target axis angle of axis 2 and 4
std::vector<int> axes24;
axes24.push_back( 2 );
axes24.push_back( 4 );

v = hand.GetAxisTargetAngle( axes24 );
// now v is something like {42.0, 47.11}
```

10.17.4.46 double cSDH::GetAxisTargetAngle (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisTargetAngle(std::vector<int>const&), just for a single axis *iAxis* and returning a single angle, see there for details and examples.

10.17.4.47 std::vector< double > cSDH::GetAxisActualAngle (std::vector< int > const & *axes*) throw (cSDHLibraryException*)

Get the current actual angle(s) of axis(axes).

The actual angles are read from the SDH.

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the actual angles of the selected axes.
- The values are converted to the selected external unit system using the configured uc_angle unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisActualAngle(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get actual axis angle of all axes
std::vector<double> v = hand.GetAxisActualAngle( hand.all_axes );
// now v is something like {0.0, 0.0, 42.0, 0.0, 47.11, 0,0, 0.0}

// Get actual axis angle of axis 2
double v2 = hand.GetAxisActualAngle( 2 );
// 2 is now something like 42.0


// Get actual axis angle of axis 2 and 4
std::vector<int> axes24;
axes24.push_back( 2 );
axes24.push_back( 4 );

v = hand.GetAxisActualAngle( axes24 );
// now v is something like {42.0, 47.11}
```

10.17.4.48 double cSDH::GetAxisActualAngle (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisActualAngle(std::vector<int>const&), just for a single axis *iAxis* and returning a single angle, see there for details and examples.

10.17.4.49 void cSDH::SetAxisTargetVelocity (std::vector< int > const & *axes*, std::vector< double > const & *velocities*) throw (cSDHLibraryException*)

Set the target velocity(s) for axis(axes).

The target velocities are stored in the SDH. A movement is not executed until an additional move command is sent.

Parameters:

axes - A vector of axis indices to access.

velocities - A vector of axis target angles to set. If any of the numbers in the vector is `NaN` (Not a Number) then the currently set axis target velocity will be kept for the corresponding axis. The value(s) are expected in the configured angular velocity unit system uc_angular_velocity.

Remarks:

- Setting the target velocity will **not** make the axis/axes move.
- The lengths of the *axes* and *velocities* vector must match.
- The indices can be given in any order, but the order of their elements must match, i.e. `velocities`[i] will be applied to axis `axes`[i] (not axis `i`).
- The indices are checked if they are valid axis indices.
- The velocities are checked if they are in the allowed range [0 .. f_max_velocity_v], i.e. it is checked that `velocities`[i], converted to internal units, is in [0 .. `f_max_velocity_v`[axes[i]]].
- If **any** index or value is invalid then **none** of the specified values is sent to the SDH, instead a SDH::cSDHErrorInvalidParameter* exception is thrown.

See also SetAxisTargetVelocity(int,double) for an overloaded variant to set a single axis target velocity or to set the same target velocity for all axes.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Set target axis velocity of all axes to the given values:
std::vector<double> all_velocities;
all_velocities.push_back( 0.0 );
all_velocities.push_back( 11.0 );
all_velocities.push_back( 22.0 );
all_velocities.push_back( 33.0 );
all_velocities.push_back( 44.0 );
all_velocities.push_back( 55.0 );
all_velocities.push_back( 66.0 );

hand.SetAxisTargetVelocity( hand.all_axes, all_velocities );


// Set target axis velocity of axis 3 to 42°/s:
hand.SetAxisTargetVelocity( 3, 42.0 );

// Set target velocity of for axis 0,4 and 2 to 0.0°/s, 44.4°/s and 2.22°/s respectively:
std::vector<int> axes042;
axes042.push_back( 0 );
axes042.push_back( 4 );
axes042.push_back( 2 );
std::vector<double> velocities042;
velocities042.push_back( 0.0 );
velocities042.push_back( 44.4 );
velocities042.push_back( 2.22 );

hand.SetAxisTargetVelocity( axes042, velocities042 );


// Set target axis velocity of all axes to 47.11°/s
hand.SetAxisTargetVelocity( hand.All, 47.11 );
```

10.17.4.50 void cSDH::SetAxisTargetVelocity (int *iAxis*, double *velocity*) throw (cSDHLibraryException*)

Like SetAxisTargetVelocity(std::vector<int>const&,std::vector<double>const&), just for a single axis *iAxis* and a single velocity *velocity*, see there for details and examples.

10.17.4.51 std::vector< double > cSDH::GetAxisTargetVelocity (std::vector< int > const & *axes*) throw (cSDHLibraryException*)

Get the target velocity(s) of axis(axes).

The currently set target velocities are read from the SDH.

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the target velocities of the selected axes.
- The values are converted to the selected external unit system using the configured uc_angular_velocity unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisTargetVelocity(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get target axis velocity of all axes
std::vector<double> v = hand.GetAxisTargetVelocity( hand.all_axes );
// now v is something like {0.0, 0.0, 42.0, 0.0, 47.11, 0,0, 0.0}

// Get target axis velocity of axis 2
double v2 = hand.GetAxisTargetVelocity( 2 );
// v2 is now something like 42.0


// Get target axis velocity of axis 2 and 4
std::vector<int> axes24;
axes24.push_back( 2 );
axes24.push_back( 4 );

v = hand.GetAxisTargetVelocity( axes24 );
// now v is something like {42.0, 47.11}
```

10.17.4.52 double cSDH::GetAxisTargetVelocity (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisTargetVelocity(std::vector<int>const&), just for a single axis *iAxis* and returning a single velocity, see there for details and examples.

10.17.4.53 std::vector< double > cSDH::GetAxisLimitVelocity (std::vector< int > const & *axes*) throw (cSDHLibraryException*)

Get the velocity limit(s) of axis(axes).

The velocity limit(s) are read from the SDH.

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the velocity limits of the selected axes.
- The values are converted to the selected external unit system using the configured uc_angular_velocity unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisLimitVelocity(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get axis velocity limits of all axes
std::vector<double> v = hand.GetAxisLimitVelocity( hand.all_axes );
// now v is something like {81.0, 140.0, 120.0, 140.0, 120.0, 140.0, 120.0}

// Get axis velocity limit of axis 2
double v2 = hand.GetAxisLimitVelocity( 2 );
// v2 is now something like 120.0


// Get axis velocity limits of axis 2 and 4
std::vector<int> axes24;
axes24.push_back( 2 );
axes24.push_back( 4 );

v = hand.GetAxisLimitVelocity( axes24 );
// now v is something like {120.0,120.0}
```

10.17.4.54 double cSDH::GetAxisLimitVelocity (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisLimitVelocity(std::vector<int>const&), just for a single axis *iAxis* and returning a single velocity limit, see there for details and examples.

10.17.4.55 std::vector< double > cSDH::GetAxisActualVelocity (std::vector< int >const & *axes*) throw (cSDHLibraryException*)

Get the actual velocity(s) of axis(axes).

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the target velocities of the selected axes.
- The values are converted to the selected external unit system using the configured uc_angular_velocity unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisTargetVelocity(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get actual axis velocity of all axes
std::vector<double> v = hand.GetAxisActualVelocity( hand.all_axes );
// now v is something like {0.1, 0.2, 0.3, 13.2, 0.5, 0.0, 0.7}

// Get actual axis velocity of axis 2 and 4
std::vector<int> axes24;
axes24.push_back( 2 );
axes24.push_back( 4 );
v = hand.GetAxisActualVelocity( axes24 );
// now L is something like {13.2, 0.0}

// Get actual axis velocity of axis 2
double v3 = hand.GetAxisActualVelocity( 2 );
// v3 is now something like 13.2
```

10.17.4.56 double cSDH::GetAxisActualVelocity (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisActualVelocity(std::vector<int>const&), just for a single axis *iAxis* and returning a single velocity, see there for details and examples.

10.17.4.57 void cSDH::SetAxisTargetAcceleration (std::vector< int >const & *axes*, std::vector< double >const & *accelerations*) throw (cSDHLibraryException*)

Set the target acceleration(s) for axis(axes).

The target accelerations are stored in the SDH. A movement is not executed until an additional move command is sent.

Parameters:

axes - A vector of axis indices to access.

accelerations - A vector of axis target accelerations to set. If any of the numbers in the vector is `NaN` (Not a Number) then the currently set axis target angle will be kept for the corresponding axis. The value(s) are expected in the configured angular acceleration unit system uc_angular_acceleration.

Remarks:

- The lengths of the *axes* and *accelerations* vector must match.
- The indices can be given in any order, but the order of their elements must match, i.e. `accelerations`[i] will be applied to axis `axes`[i] (not axis `i`).
- The indices are checked if they are valid axis indices.
- The accelerations are checked if they are in the allowed range [0 .. f_max_velocity_v], i.e. it is checked that `accelerations`[i], converted to internal units, is in [0 .. `f_max_velocity_v`[axes[i]]].
- If **any** index or value is invalid then **none** of the specified values is sent to the SDH, instead a SDH::cSDHErrorInvalidParameter* exception is thrown.

See also SetAxisTargetAcceleration(int,double) for an overloaded variant to set a single axis target acceleration or to set the same target acceleration for all axes.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Set target axis acceleration of all axes to the given values:
std::vector<double> all_accelerations;
all_accelerations.push_back( 0.0 );
all_accelerations.push_back( 111.0 );
all_accelerations.push_back( 222.0 );
all_accelerations.push_back( 333.0 );
all_accelerations.push_back( 444.0 );
all_accelerations.push_back( 555.0 );
all_accelerations.push_back( 666.0 );

hand.SetAxisTargetAcceleration( hand.all_axes, all_accelerations );


// Set target axis acceleration of axis 3 to 420°/s²:
hand.SetAxisTargetAcceleration( 3, 420.0 );

// Set target acceleration of for axis 0,4 and 2 to 0.0°/s², 444.0°/s² and 222°/s² respectiv
std::vector<int> axes042;
axes042.push_back( 0 );
axes042.push_back( 4 );
axes042.push_back( 2 );
std::vector<double> accelerations042;
accelerations042.push_back( 0.0 );
accelerations042.push_back( 444.0 );
accelerations042.push_back( 222.0 );

hand.SetAxisTargetAcceleration( axes042, accelerations042 );


// Set target axis acceleration of all axes to 471.1°/s
hand.SetAxisTargetAcceleration( hand.All, 471.1 );
```

10.17.4.58 void cSDH::SetAxisTargetAcceleration (int *iAxis*, double *acceleration*) throw (cSDHLibraryException*)

Like SetAxisTargetAcceleration(std::vector<int>const&,std::vector<double>const&), just for a single axis *iAxis* and a single acceleration *acceleration*, see there for details and examples.

10.17.4.59 std::vector< double > cSDH::GetAxisTargetAcceleration (std::vector< int >const & *axes*) throw (cSDHLibraryException*)

Get the target acceleration(s) of axis(axes).

The currently set target accelerations are read from the SDH.

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the target accelerations of the selected axes.
- The values are converted to the selected external unit system using the configured uc_angular_acceleration unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisTargetAcceleration(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get target axis acceleration of all axes
std::vector<double> v = hand.GetAxisTargetAcceleration( hand.all_axes );
// now v is something like {0.0, 0.0, 42.0, 0.0, 47.11, 0,0, 0.0}

// Get target axis acceleration of axis 2
double v2 = hand.GetAxisTargetAcceleration( 2 );
// v2 is now something like 42.0


// Get target axis acceleration of axis 2 and 4
std::vector<int> axes24;
axes24.push_back( 2 );
axes24.push_back( 4 );

v = hand.GetAxisTargetAcceleration( axes24 );
// now v is something like {42.0, 47.11}
```

10.17.4.60 double cSDH::GetAxisTargetAcceleration (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisTargetAcceleration(std::vector<int>const&), just for a single axis *iAxis* and returning a single acceleration, see there for details and examples.

10.17.4.61 std::vector< double > cSDH::GetAxisMinAngle (std::vector< int > const & *axes*) throw (cSDHLibraryException*)

Get the minimum angle(s) of axis(axes).

The minimum angles are currently not read from the SDH, but are stored in the library.

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the min angles of the selected axes.
- The values are converted to the selected external unit system using the configured uc_angle unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisMinAngle(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get minimum axis angles of all axes
std::vector<double> v = hand.GetAxisMinAngle( hand.all_axes );
// now v is something like {0.0, -90.0, -90.0, -90.0, -90.0, -90.0, -90.0}

// Get minimum axis angle of axis 3
double v3 = hand.GetAxisMinAngle( 3 );
// v3 is now something like -90.0

// Get minimum axis angle of axis 2 and 4
std::vector<int> axes24;
axes24.push_back( 2 );
axes24.push_back( 4 );

v = hand.GetAxisMinAngle( axes24 );
// now v is something like {-90.0, -90.0}


// Or if you change the angle unit system:
hand.UseRadians();

v = hand.GetAxisMinAngle( hand.all_axes );
// now v is something like {0.0, -1.5707963267948966, -1.5707963267948966, -1.57079632679489
```

10.17.4.62 double cSDH::GetAxisMinAngle (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisMinAngle(std::vector<int>const&), just for a single axis *iAxis* and returning a single minimum angle, see there for details and examples.

10.17.4.63 std::vector< double > cSDH::GetAxisMaxAngle (std::vector< int > const & *axes*) throw (cSDHLibraryException*)

Get the maximum angle(s) of axis(axes).

The maximum angles are currently not read from the SDH, but are stored in the library.

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the max angles of the selected axes.
- The values are converted to the selected external unit system using the configured uc_angle unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisMaxAngle(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get maximum axis angles of all axes
std::vector<double> v = hand.GetAxisMaxAngle( hand.all_axes );
// now v is something like {90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0}

// Get maximum axis angle of axis 3
double v3 = hand.GetAxisMaxAngle( 3 );
// v3 is now something like 90.0

// Get maximum axis angle of axis 2 and 4
std::vector<int> axes24;
axes24.push_back( 2 );
axes24.push_back( 4 );

v = hand.GetAxisMaxAngle( axes24 );
// now v is something like {90.0, 90.0}


// Or if you change the angle unit system:
hand.UseRadians();

v = hand.GetAxisMaxAngle( hand.all_axes );
// now v is something like { 1.5707963267948966, 1.5707963267948966, 1.5707963267948966, 1.5
```

10.17.4.64 double cSDH::GetAxisMaxAngle (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisMaxAngle(std::vector<int>const&), just for a single axis *iAxis* and returning a single maximum angle, see there for details and examples.

10.17.4.65 std::vector< double > cSDH::GetAxisMaxVelocity (std::vector< int > const & *axes*) throw (cSDHLibraryException*)

Get the maximum velocity(s) of axis(axes).

The maximum velocities are currently not read from the SDH, but are stored in the library.

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the max angular velocities of the selected axes.
- The values are converted to the selected external unit system using the configured uc_angular_velocity unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisMaxVelocity(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get maximum axis angular velocities of all axes
std::vector<double> v = hand.GetAxisMaxAngle( hand.all_axes );
// now v is something like {28.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0}

// Get maximum axis angular velocity of axis 3
double v3 = hand.GetAxisMaxAngle( 3 );
// v3 is now something like 100.0

// Get maximum axis angular velocity of axis 2 and 4
std::vector<int> axes24;
axes24.push_back( 2 );
axes24.push_back( 4 );

v = hand.GetAxisMaxAngle( axes24 );
// now v is something like {100.0, 100.0}


// Or if you change the angular velocity unit system:
hand.UseRadians();

v = hand.GetAxisMaxAngle( hand.all_axes );
// now v is something like {0.488692190, 1.74532925, 1.74532925, 1.74532925, 1.74532925, 1.7
```

10.17.4.66 double cSDH::GetAxisMaxVelocity (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisMaxVelocity(std::vector<int>const&), just for a single axis *iAxis* and returning a single minimum angle, see there for details and examples.

10.17.4.67 std::vector< double > cSDH::GetAxisMaxAcceleration (std::vector< int > const & *axes*) throw (cSDHLibraryException*)

Get the maximum acceleration(s) of axis(axes).

The maximum accelerations are currently not read from the SDH, but are stored in the library.

Parameters:

axes - A vector of axis indices to access.

- The indices in *axes* are checked if they are valid axis indices.
- If **any** axis index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the max angular accelerations of the selected axes.
- The values are converted to the selected external unit system using the configured uc_angular_acceleration unit converter object.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisMaxAcceleration(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get maximum axis angular accelerations of all axes
std::vector<double> v = hand.GetAxisMaxAngle( hand.all_axes );
// now v is something like {1000.0, 1000.0, 1000.0, 1000.0, 1000.0, 1000.0, 1000.0}

// Get maximum axis angular acceleration of axis 3
double v3 = hand.GetAxisMaxAngle( 3 );
// v3 is now something like 1000.0

// Get maximum axis angular acceleration of axis 2 and 4
std::vector<int> axes24;
axes24.push_back( 2 );
axes24.push_back( 4 );

v = hand.GetAxisMaxAngle( axes24 );
// now v is something like {1000.0, 1000.0}


// Or if you change the angular acceleration unit system:
hand.UseRadians();

v = hand.GetAxisMaxAngle( hand.all_axes );
// now v is something like {0.488692190, 1.74532925, 1.74532925, 1.74532925, 1.74532925, 1.7
```

10.17.4.68 double cSDH::GetAxisMaxAcceleration (int *iAxis*) throw (cSDHLibraryException*)

Like GetAxisMaxAcceleration(std::vector<int>const&), just for a single axis *iAxis* and returning a single minimum angle, see there for details and examples.

10.17.4.69 double cSDH::MoveAxis (std::vector< int >const & *axes*, bool *sequ*) throw (cSDHLibraryException*)

Move selected axis/axes to the previously set target pose with the previously set velocity profile, (maximum) target velocities and target accelerations

Parameters:

axes - A vector of axis indices to access.

sequ - flag: if true (default) then the function executes sequentially and returns not until after the SDH has finished the movement. If false then the function returns immediately after the movement command has been sent to the SDH (the currently set target axis angles for other axes will then be **overwritten** with their current actual axis angles).

- The indices in *axes* are checked if they are valid axis indices.
- If any index is invalid then no movement is performed, instead a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

The expected/elapsed execution time for the movement in the configured time unit system uc_time

Remarks:

- The axes will be enabled automatically.
- Currently the actual movement velocity of an axis is determined by the SDH firmware to make the movements of all involved axes start and end synchronously at the same time. Therefore the axis that needs the longest time for its movement at its given maximum velocity determines the velocities of all the other axes.
- Other axes than those selected by *axes* will **NOT** move, even if target axis angles for the axes have been set. (Remember: as axis 0 is used by finger 0 and 2 these two fingers cannot be moved completely idependent of each other.)
- If *sequ* is true then the currently set target axis angles for other fingers will be restored upon return of the function.
- If *sequ* is false then the currently set target axis angles for other fingers will be **overwritten** with their current actual axis angles

See also MoveAxis(int,bool) for an overloaded variant to move a single axis.

Examples:

```
// Assuming 'hand' is a cSDH object ...

// create an index vector for adressing axes 0, 4 and 2 (in that order)
std::vector<int> axes042;
axes042.push_back( 0 );
axes042.push_back( 4 );
axes042.push_back( 2 );

// Set a new target pose for axes 0, 4 and 2:
std::vector<double> angles042;
angles042.push_back( 0.0 );
angles042.push_back( -44.4 );
angles042.push_back( -22.2 );

hand.SetFingerTargetAngle( axes042, angles042 );


// First move Axis 0 only to its new target position:
hand.MoveAxis( 0 );

// The axis 0 has now reached its target position 0.0°.  The
// target poses for axes 4 and 2 are still set since the
// last MoveAxes() call was sequentially (und thus it could
// restore the previously set target axis angles of not
// selected axes after the movement finished)
```

```
// So move axes 4 and 2 now, this time non-sequentially:
std::vector<int> axes42;
axes42.push_back( 4 );
axes42.push_back( 2 );

double t = hand.MoveAxes( axis42, false );

// The two axes 4 and 2 are now moving to their target position.
// We have to wait until the non-sequential call has finished:
SleepSec( t );

// The axes 4 and 2 have now moved to -44.4 and -22.2.

// The target angles for other axes have by now been
// overwritten since the last MoveAxis() call was
// non-sequentially (und thus it could \b NOT restore the
// previously set target axis angles of not selected axes
// after the movement finished)


// Set new target angles for all axes ("home pose");
hand.SetAxisTargetAngle( hand.All, 0.0 );

// Now move all axes back to home pose:
hand.MoveAxes( hand.All );
```

10.17.4.70 double cSDH::MoveAxis (int *iAxis*, bool *sequ*) throw (cSDHLibraryException*)

Like MoveAxis(std::vector<int>const&,bool), just for a single axis *iAxis* (or all axes if All is given).

10.17.4.71 void cSDH::SetFingerEnable (std::vector< int > const & *fingers*, std::vector< double > const & *states*) throw (cSDHLibraryException*)

Set enabled/disabled state of axis controllers of finger(s).

The controllers of the axes of the selected fingers are enabled/disabled in the SDH. Disabled axes are not powered and thus might not remain in their current pose due to gravity, inertia or other external influences. But to prevent overheating the axis controllers should be switched of when not needed.

Parameters:

fingers - A vector of finger indices to access.

states - A vector of enabled states (0 = disabled, !=0 = enabled) to set. If any of the numbers in the vector is `NaN` (Not a Number) then the currently set enabled state will be kept for the corresponding axis.

Remarks:

- The lengths of the *fingers* and *states* vector must match.
- The indices can be given in any order, but the order of their elements must match, i.e. `state`[i] will be applied to finger `fingers`[i] (not finger `i`).
- The indices are checked if they are valid finger indices.
- If **any** index is invalid then **none** of the specified values is sent to the SDH, instead a SDH::cSDHErrorInvalidParameter* exception is thrown.
- As axis 0 is used for finger 0 and 2, axis 0 is disabled only if both finger 0 and 1 are disabled.

See also SetFingerEnable(int,double), SetFingerEnable(int,bool) for overloaded variants to set a single finger enabled/disabled or to set the same state for all fingers. See further SetFingerEnable(std::vector<int>const&,std::vector<bool>const&) for a variant that accepts a `bool` vector for the states to set.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Enable finger 1 and 2 while disabling finger 0 :
std::vector<double> states012;
states012.push_back( 0.0 );
states012.push_back( 1.0 );
states012.push_back( 1.0 );

hand.SetFingerEnable( hand.all_axes, states012 );
// (this will keep axis 0 (used by the disabled finger 0) enabled,
// since axis 0 is needed by the enabled finger 2 too);

// Enable all fingers:
hand.SetFingerEnable( hand.All,true );

// Disable all fingers:
hand.SetFingerEnable( hand.All, 0.0 );

// Disable finger 2:
hand.SetFingerEnable( 2, false );
```

10.17.4.72 void cSDH::SetFingerEnable (int *iFinger*, double *state* = 1.0) throw (cSDHLibraryException*)

Like SetFingerEnable(std::vector<int>const&,std::vector<double>const&), just for a single finger *iAxis* and a single angle *angle*, see there for details and examples.

10.17.4.73 void cSDH::SetFingerEnable (std::vector< int > const & *fingers*, std::vector< bool > const & *states*) throw (cSDHLibraryException*)

Like SetFingerEnable(std::vector<int>const&,std::vector<double>const&), just with states as vector of `bool` values, see there for details and examples.

10.17.4.74 void cSDH::SetFingerEnable (int *iFinger*, bool *state*) throw (cSDHLibraryException*)

Like SetFingerEnable(std::vector<int>const&,std::vector<double>const&), just for a single finger *iAxis* and a single angle *angle*, see there for details and examples.

10.17.4.75 std::vector< double > cSDH::GetFingerEnable (std::vector< int > const & *fingers*) throw (cSDHLibraryException*)

Get enabled/disabled state of axis controllers of finger(s).

The enabled/disabled state of the controllers of the selected fingers is read from the SDH. A finger is reported disabled if any of its axes is disabled and reported enabled if all its axes are enabled.

Parameters:

fingers - A vector of finger indices to access.

- The indices in *fingers* are checked if they are valid finger indices.
- If **any** finger index is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of enabled/disabled states as doubles (0=disabled, 1.0=enabled) of the selected axes.
- The order of the elements of the *axes* vector and the returned values vector *rv* matches. I.e. `rv`[i] will be the value of axis `axes`[i] (not axis `i`).

See also GetAxisEnable(int) for an overloaded variant to access a single axis.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get enabled state of all fingers:
std::vector<double> v = hand.GetFingerEnable( hand.all_fingers );
// now v is something like {0.0, 1.0, 0.0}

// Get enabled state of finger 0 and 2
std::vector<int> fingers02;
fingers02.push_back( 0 );
fingers02.push_back( 2 );

v = hand.GetFingerEnable( fingers02 );
// now v is something like {0.0, 0.0}

// Get enabled state of finger 1
double v1 = hand.GetFingerEnable( 1 );
// now v1 is something like 1.0
```

10.17.4.76 double cSDH::GetFingerEnable (int *iFinger*) throw (cSDHLibraryException*)

Like GetFingerEnable(std::vector<int>const&), just for a single finger *iFinger* and returning a single double value

10.17.4.77 void cSDH::SetFingerTargetAngle (int *iFinger*, std::vector< double > const & *angles*) throw (cSDHLibraryException*)

Set the target angle(s) for a single finger.

The target axis angles *angle* of finger *iFinger* are stored in the SDH. The movement is not executed until an additional move command is sent.

Parameters:

iFinger - index of finger to access. This must be a single index.

angles - the angle(s) to set or `None` to set the current actual axis angles of the finger as target angle. This can be a single number or a vector of numbers. The value(s) are expected in the configured angle unit system uc_angle.

Remarks:

- Setting the target angles will **not** make the finger move.
- The *iFinger* index is checked if it is a valid finger index.

- The angles are checked if they are in the allowed range [0 .. f_max_angle_v], i.e. it is checked that `angles`[i], converted to internal units, is in [0 .. `f_max_angle_v`[finger_axis_-index[iFinger][i]]].
- If **any** index or value is invalid then **none** of the specified values is sent to the SDH, instead a SDH::cSDHErrorInvalidParameter* exception is thrown.

See also SetFingerTargetAngle(int,double,double,double) for an overloaded variant to set finger axis target angles from single `double` values.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Set target axis angles of finger 0 to { 10.0°, -08.15°, 47.11° }
std::vector<double> angles;
angles.push_back( 10.0 );
angles.push_back( -08.15 );
angles.push_back(  47.11 );

hand.SetFingerTargetAngle( 0, angles );


// Set target axis angles of finger 1 to { 0.0°, 24.7°, 17.4° }
angles[0] = 0.0;   // "virtual" base axis of finger 1
angles[1] = 24.7;
angles[2] = 17.4;
hand.SetFingerTargetAngle( 1, { 0.0, 24.7, 17.4 } );


// Set target axis angles of all axes of finger 0 to 12.34°
hand.SetFingerTargetAngle( 0, 12.34, 12.34, 12.34 );


// REMARK: the last command changed the previously set target axis
// angle for axis 0, since axis 0 is used as base axis for both
// finger 0 and 2!
```

10.17.4.78 void cSDH::SetFingerTargetAngle (int *iFinger*, double *a0*, double *a1*, double *a2*) throw (cSDHLibraryException*)

Like SetFingerTargetAngle(int,std::vector<double>const&), just with individual finger axis angles *a0*, *a1* and *a2*.

10.17.4.79 std::vector< double > cSDH::GetFingerTargetAngle (int *iFinger*) throw (cSDHLibraryException*)

Get the target axis angles of a single finger.

The target axis angles of finger *iFinger* are read from the SDH.

Parameters:

iFinger - index of finger to access. This must be a single index

Remarks:

- The *iFinger* index is checked if it is a valid finger index.

- If *iFinger* is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A list of the selected fingers target axis angles
- The values are returned in the configured angle unit system uc_angle.

See also GetFingerTargetAngle(int,double&,double&,double&) for an overloaded variant to get finger axis target angles into single `double` values.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get target axis angles of finger 0
std::vector<double> v = hand.GetFingerTargetAngle( 0 );
// now v is something like {42.0, -10.0, 47.11}

// Get target axis angles of finger 1
double a0, a1, a2;
hand.GetFingerTargetAngle( 1, a0, a1, a2 );
// now a0, a1, a2 are something like 0.0, 24.7 and -5.5 respectively.
```

10.17.4.80 void cSDH::GetFingerTargetAngle (int *iFinger*, double & *a0*, double & *a1*, double & *a2*) throw (cSDHLibraryException*)

Like GetFingerTargetAngle(int), just returning the target axis angles in the *a0*, *a1* and *a2* parameters which are given by reference.

10.17.4.81 std::vector< double > cSDH::GetFingerActualAngle (int *iFinger*) throw (cSDHLibraryException*)

Get the current actual axis angles of a single finger.

The current actual axis angles of finger *iFinger* are read from the SDH.

Parameters:

iFinger - index of finger to access. This must be a single index.

Remarks:

- The *iFinger* index is checked if it is a valid finger index.
- If *iFinger* is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A list of the current actual axis angles of the selected finger
- The values are returned in the configured angle unit system uc_angle.

See also GetFingerActualAngle(int,double&,double&,double&) for an overloaded variant to get finger axis actual angles into single `double` values.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get actual axis angles of finger 0
std::vector<double> v = hand.GetFingerActualAngle( 0 );
// v is now something like {42.0, -10.0, 47.11}

// Get actual axis angles of finger 1
double a0, a1, a2;
hand.GetFingerTargetAngle( 1, a0, a1, a2 );
// now a0, a1, a2 are something like 0.0, 24.7 and -5.5 respectively.
```

10.17.4.82 void cSDH::GetFingerActualAngle (int *iFinger*, double & *a0*, double & *a1*, double & *a2*) throw (cSDHLibraryException*)

Like GetFingerActualAngle(int), just returning the actual axis angles in the *a0*, *a1* and *a2* parameters which are given by reference.

10.17.4.83 std::vector< double > cSDH::GetFingerMinAngle (int *iFinger*) throw (cSDHLibraryException*)

Get the minimum axis angles of a single finger.

The minimum axis angles of finger *iFingers* axes, stored in the library, are returned.

Parameters:

iFinger - index of finger to access. This must be a single index

Remarks:

- The *iFinger* index is checked if it is a valid finger index.
- If *iFinger* is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A list of the selected fingers minimum axis angles
- The values are returned in the configured angle unit system uc_angle.

See also GetFingerMinAngle(int,double&,double&,double&) for an overloaded variant to get finger axis min angles into single `double` values.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get minimum axis angles of finger 0
std::vector<double> v = hand.GetFingerMinAngle( 0 );
// now v is something like {0.0, -90.0, -90.0}


// Get target axis angles of finger 1
double a0, a1, a2;
hand.GetFingerMinAngle( 1, a0, a1, a2 );
// now a0, a1, a2 are something like 0.0, -90.0, -90.0 respectively.
```

```
// Or if you change the angle unit system:
hand.UseRadians();
v = hand.GetFingerMinAngle( 0 );
// now v is something like {0.0, -1.5707963267948966, -1.5707963267948966}
```

10.17.4.84 void cSDH::GetFingerMinAngle (int *iFinger*, double & *a0*, double & *a1*, double & *a2*) throw (cSDHLibraryException*)

Like GetFingerMinAngle(int), just returning the finger axis min angles in the *a0*, *a1* and *a2* parameters which are given by reference.

10.17.4.85 std::vector< double > cSDH::GetFingerMaxAngle (int *iFinger*) throw (cSDHLibraryException*)

Get the maximum axis angles of a single finger.

The maximum axis angles of finger *iFingers* axes, stored in the library, are returned.

Parameters:

iFinger - index of finger to access. This must be a single index

Remarks:

- The *iFinger* index is checked if it is a valid finger index.
- If *iFinger* is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A list of the selected fingers maximum axis angles
- The values are returned in the configured angle unit system uc_angle.

See also GetFingerMaxAngle(int,double&,double&,double&) for an overloaded variant to get finger axis max angles into single `double` values.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get maximum axis angles of finger 0
std::vector<double> v = hand.GetFingerMaxAngle( 0 );
// now v is something like {90.0, 90.0, 90.0}


// Get target axis angles of finger 1
double a0, a1, a2;
hand.GetFingerMaxAngle( 1, a0, a1, a2 );
// now a0, a1, a2 are something like 90.0, 90.0, 90.0 respectively.


// Or if you change the angle unit system:
hand.UseRadians();
v = hand.GetFingerMaxAngle( 0 );
// now v is something like {1.5707963267948966, 1.5707963267948966, 1.5707963267948966}
```

10.17.4.86 void cSDH::GetFingerMaxAngle (int *iFinger*, double & *a0*, double & *a1*, double & *a2*) throw (cSDHLibraryException*)

Like GetFingerMaxAngle(int), just returning the finger axis max angles in the *a0*, *a1* and *a2* parameters which are given by reference.

10.17.4.87 std::vector< double > cSDH::GetFingerXYZ (int *iFinger*, std::vector< double > const & *angles*) throw (cSDHLibraryException*)

Get the cartesian xyz finger tip position of a single finger from the given axis angles (forward kinematics).

Parameters:

iFinger - index of finger to access. This must be a single index

angles - a vector of NUMBER_OF_AXES_PER_FINGER angles. The values are expected in the configured angle unit system uc_angle.

Remarks:

- The *iFinger* index is checked if it is a valid finger index.
- The angles are checked if they are in the allowed range [0 .. f_max_angle_v], i.e. it is checked that `angles`[i], converted to internal units, is in [0 .. `f_max_angle_v`[finger_axis_index[iFinger][i]]].
- If any index or value is invalid then a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

- A vector of the x,y,z values of the finger tip position
- The values are returned in the configured position unit system uc_position.

See also GetFingerXYZ(int,double,double,double) for an overloaded variant to get finger tip position from single `double` values.

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get actual finger angles of finger 0:
std::vector<double> angles = hand.GetFingerActualAngle( 0 );

// Get actual finger tip position of finger 0:
std::vector<double> position = hand.GetFingerXYZ( 0, angles );
// now position is something like {18.821618775581801, 32.600000000000001, 174.0}
// (assuming that finger 0 is at axis angles {0,0,0})

// Get finger tip position of finger 2 at axis angles {90,-90,-90}:
position = hand.GetFingerXYZ( 2, 90, -90, -90 );
// now position is something like {18.821618775581804, 119.60000000000002, -53.0}

// Or if you change the angle unit system:
hand.UseRadians();
position = hand.GetFingerXYZ( 0, 1.5707963267948966, -1.5707963267948966, -1.570796326794896
// now position is still something like {18.821618775581804, 119.60000000000002, -53.0}

// Or if you change the position unit system too:
hand.uc_position = &cSDH::uc_position_meter
position = hand.GetFingerXYZ( 0, 1.5707963267948966, -1.5707963267948966, -1.570796326794896
// now position is still something like {0.018821618775581, 0.119.60000000000002, -0.0529999
```

10.17.4.88 std::vector< double > cSDH::GetFingerXYZ (int *iFinger*, double *a0*, double *a1*, double *a2*) throw (cSDHLibraryException*)

Like SetFingerTargetAngle(int,std::vector<double>const&), just with individual finger axis angles *a0*, *a1* and *a2*.

10.17.4.89 double cSDH::MoveFinger (std::vector< int >const & *fingers*, bool *sequ* = `true`) throw (cSDHLibraryException*)

Move selected finger(s) to the previously set target pose with the previously set velocity profile, (maximum) target velocities and target accelerations.

Parameters:

fingers - A vector of finger indices to access.

sequ - flag: if true (default) then the function executes sequentially and returns not until after the SDH has finished the movement. If false then the function returns immediately after the movement command has been sent to the SDH (the currently set target axis angles for other fingers will then be **overwritten** with their current actual axis angles).

- The indices in *fingers* are checked if they are valid finger indices.
- If any index is invalid then no movement is performed, instead a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

The expected/elapsed execution time for the movement in the configured time unit system uc_time

Remarks:

- The axes will be enabled automatically.
- Currently the actual movement velocity of an axis is determined by the SDH firmware to make the movements of all involved axes start and end synchronously at the same time. Therefore the axis that needs the longest time for its movement at its given maximum velocity determines the velocities of all the other axes.
- Other fingers than *iFinger* will **NOT** move, even if target axis angles for their axes have been set. (Exception: as axis 0 is used by finger 0 and 2 these two fingers cannot be moved completely idependent of each other.)
- If *sequ* is true then the currently set target axis angles for other fingers will be restored upon return of the function.
- If *sequ* is false then the currently set target axis angles for other fingers will be **overwritten** with their current actual axis angles

See also MoveFinger(int,bool) for an overloaded variant to move a single finger.

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Set a new target pose for finger 0:
hand.SetFingerTargetAngle( 0, 0.0, 0.0, 0.0 );

// Set a new target pose for finger 1
```

```
hand.SetFingerTargetAngle( 1, 0.0, -10.0, -10.0 );

// Set a new target pose for finger 2
hand.SetFingerTargetAngle( 2, 20.0, -20.0, -20.0 );

// Move finger 0 only (and finger 2 partly as axis 0 also belongs to finger 2);
hand.MoveFinger( 0, true );
// The finger 0 has been moved to {20,0,0}
// (axis 0 is 'wrong' since the target angle for axis 0 has been overwritten
//  while setting the target angles for finger 2);

// The target poses for finger 1 and 2 are still set since the
// last MoveFinger() call was sequentially.
// So move finger 1 now:
double t = hand.MoveFinger( 1, false );

// wait until the non-sequential call has finished:
SleepSec( t );

// The finger 1 has been moved to {0,-10,-10}.

// The target angles for finger 2 have been overwritten since the
// last MoveFinger() call was non-sequentially.

// Therefore this next call will just keep the fingers in their
// current positions:
hand.MoveFinger( hand.All, true );


// Set new target angles for all axes ("home pose");
hand.SetAxisTargetAngle( hand.All, 0.0 );

// Now move all axes back to home pose:
hand.MoveHand();
```

10.17.4.90 double cSDH::MoveFinger (int *iFinger*, bool *sequ* = `true`) throw (cSDHLibraryException*)

Like MoveFinger(std::vector<int>const&,bool), just for a single finger *iFinger* (or all fingers if All is given).

10.17.4.91 double cSDH::MoveHand (bool *sequ* = `true`) throw (cSDHLibraryException*)

Move all fingers to the previously set target pose with the previously set (maximum) velocities.

This is just a shortcut to MoveFinger(int,bool) with *iFinger* set to `hand.All` and *sequ* as indicated, so see there for details and examples.

10.17.4.92 double cSDH::GetGripMaxVelocity (void)

Get the maximum velocity of grip skills

The maximum velocity is currently not read from the SDH, but is stored in the library.

Returns:

- a single double value is returned representing the velocity in the uc_angular_velocity unit system

Examples:

```
// Assuming "hand" is a cSDH object ...

// Get maximum grip skill velocity
double v = hand.GetGripMaxVelocity();
// v is now something like 100.0

// Or if you change the velocity unit system:
hand.UseRadians();
v = hand.GetGripMaxVelocity();
// now v is something like 1.7453292519943295
```

10.17.4.93 double cSDH::GripHand (eGraspId *grip*, double *close*, double *velocity*, bool *sequ* = `true`) throw (cSDHLibraryException*)

Perform one of the internal eGraspId "grips" or "grasps"

Parameters:

grip - The index of the grip to perform [0..eGID_DIMENSION-1] (s.a. eGraspId)

close - close-ratio: [0.0 .. 1.0] where 0.0 is 'fully opened' and 1.0 is 'fully closed'

velocity - maximum allowed angular axis velocity in the chosen external unit system uc_angular_velocity

sequ - flag: if true (default) then the function executes sequentially and returns not until after the SDH has finished the movement. If false then the function returns immediately after the movement command has been sent to the SDH.

- The *close* and *velocity* values are checked if they are in their allowed range.
- If **any** value is invalid then **no** grip is perfomed, instead a SDH::cSDHErrorInvalidParameter* exception is thrown.

Returns:

The expected/elapsed execution time for the movement in the configured time unit system uc_time.

Remarks:

- Currently the actual movement velocity of an axis is determined by the SDH firmware to make the movements of all involved axes start and end synchronously at the same time. Therefore the axis that needs the longest time for its movement at its given maximum velocity determines the velocities of all the other axes.
- The currently set target axis angles are not changed by this command
- The movement uses the eMotorCurrentMode motor current modes "eMCM_GRIP" while gripping and then changes the motor current mode to "eMCM_HOLD". After the movement previously set motor currents set for mode "eMCM_MOVE" are **overwritten!**

Warning:

Bug

!!! Currently the performing of a skill or grip can **NOT** be interrupted!!! Even if the command is sent with *sequ=false* it **cannot** be stoped or emergency stopped.

Examples:

```
// Assuming 'hand' is a cSDH object ...

// Perform a fully opened centrical grip at 50°/s:
hand.GripHand( hand.eGID_CENTRICAL, 0.0, 50.0, true );

// Now close it 50% with 30°/s:
hand.GripHand( hand.eGID_CENTRICAL, 0.5, 30.0, true );

// Then close it completely with 20°/s:
hand.GripHand( hand.eGID_CENTRICAL, 1.0, 20.0, true );
```

10.17.5 Member Data Documentation

10.17.5.1 cUnitConverter const cSDH::uc_angle_degrees [static]

Default converter for angles (internal unit == external unit): degrees.

10.17.5.2 cUnitConverter const cSDH::uc_angle_radians [static]

Converter for angles: external unit = radians.

10.17.5.3 cUnitConverter const cSDH::uc_time_seconds [static]

Default converter for times (internal unit == external unit): seconds.

10.17.5.4 cUnitConverter const cSDH::uc_time_milliseconds [static]

Converter for times: external unit = milliseconds.

10.17.5.5 cUnitConverter const cSDH::uc_temperature_celsius [static]

Default converter for temparatures (internal unit == external unit): degrees celsius.

10.17.5.6 cUnitConverter const cSDH::uc_temperature_fahrenheit [static]

Converter for temperatures: external unit = degrees fahrenheit.

10.17.5.7 cUnitConverter const cSDH::uc_angular_velocity_degrees_per_second [static]

Default converter for angular velocities (internal unit == external unit): degrees / second.

10.17.5.8 cUnitConverter const cSDH::uc_angular_velocity_radians_per_second [static]

Converter for angular velocieties: external unit = radians/second.

10.17.5.9 cUnitConverter const cSDH::uc_angular_acceleration_degrees_per_second_squared [static]

Default converter for angular accelerations (internal unit == external unit): degrees / second.

10.17.5.10 cUnitConverter const cSDH::uc_angular_acceleration_radians_per_second_squared [static]

Converter for angular velocieties: external unit = radians/second.

10.17.5.11 cUnitConverter const cSDH::uc_motor_current_ampere [static]

Default converter for motor current (internal unit == external unit): Ampere.

10.17.5.12 cUnitConverter const cSDH::uc_motor_current_milliampere [static]

Converter for motor current: external unit = milli Ampere.

10.17.5.13 cUnitConverter const cSDH::uc_position_millimeter [static]

Default converter for position (internal unit == external unit): millimeter.

10.17.5.14 cUnitConverter const cSDH::uc_position_meter [static]

Converter for position: external unit = meter.

10.17.5.15 int SDH::cSDH::NUMBER_OF_AXES_PER_FINGER [protected]

The number of axis per finger (for finger 1 this includes the "virtual" base axis).

10.17.5.16 int SDH::cSDH::NUMBER_OF_VIRTUAL_AXES [protected]

The number of virtual axes.

10.17.5.17 int SDH::cSDH::nb_all_axes [protected]

The number of all axes including virtual axes.

10.17.5.18 std::vector<int> SDH::cSDH::finger_number_of_axes [protected]

Mapping of finger index to number of real axes of fingers:.

10.17.5.19 std::vector<std::vector<int> > SDH::cSDH::finger_axis_index [protected]

Mapping of finger index, finger axis index to axis index:.

10.17.5.20 std::vector<double> SDH::cSDH::f_zeros_v [protected]

Vector of 3 epsilon values.

Vector of 3 0.0 values

10.17.5.21 std::vector<double> SDH::cSDH::f_ones_v [protected]

Vector of 3 1.0 values.

10.17.5.22 std::vector<double> SDH::cSDH::zeros_v [protected]

Vector of nb_all_axes 0.0 values.

10.17.5.23 std::vector<double> SDH::cSDH::ones_v [protected]

Vector of nb_all_axes 1.0 values.

10.17.5.24 std::vector<double> SDH::cSDH::f_min_motor_current_v [protected]

Minimum allowed motor currents (in internal units (Ampere)), including the virtual axis.

10.17.5.25 std::vector<double> SDH::cSDH::f_max_motor_current_v [protected]

Maximum allowed motor currents (in internal units (Ampere)), including the virtual axis.

10.17.5.26 std::vector<double> SDH::cSDH::f_min_angle_v [protected]

Minimum allowed axis angles (in internal units (degrees)), including the virtual axis.

10.17.5.27 std::vector<double> SDH::cSDH::f_max_angle_v [protected]

Maximum allowed axis angles (in internal units (degrees)), including the virtual axis.

10.17.5.28 std::vector<double> SDH::cSDH::f_min_velocity_v [protected]

Minimum allowed axis velocity (in internal units (degrees/second)), including the virtual axis.

10.17.5.29 std::vector<double> SDH::cSDH::f_max_velocity_v [protected]

Maximum allowed axis velocity (in internal units (degrees/second)), including the virtual axis.

10.17.5.30 std::vector<double> SDH::cSDH::f_min_acceleration_v [protected]

Minimum allowed axis acceleration (in internal units (degrees/(second $*$ second))), including the virtual axis.

10.17.5.31 std::vector<double> SDH::cSDH::f_max_acceleration_v [protected]

Maximum allowed axis acceleration (in internal units (degrees/(second * second))), including the virtual axis.

10.17.5.32 double SDH::cSDH::grip_max_velocity [protected]

Maximum allowed grip velocity (in internal units (degrees/second)).

10.17.5.33 double SDH::cSDH::l1 [protected]

length of limb 1 (proximal joint to distal joint) in mm

10.17.5.34 double SDH::cSDH::l2 [protected]

length of limb 2 (distal joint to fingertip) in mm

10.17.5.35 double SDH::cSDH::d [protected]

10.17.5.36 double SDH::cSDH::h [protected]

10.17.5.37 std::vector<std::vector<double> > SDH::cSDH::offset [protected]

list of xyz-vectors for all fingers with offset from (0,0,0) of proximal joint in mm

10.17.5.38 cSerialBase* SDH::cSDH::com [protected]

10.17.5.39 cSDHSerial SDH::cSDH::comm_interface

The object to interface with the SDH attached via serial RS232 or CAN.

10.17.5.40 std::vector<int> SDH::cSDH::all_axes

A vector with indices of all axes (in natural order), including the virtual axis.

10.17.5.41 std::vector<int> SDH::cSDH::all_fingers

A vector with indices of all fingers (in natural order).

10.17.5.42 std::vector<int> SDH::cSDH::all_temperature_sensors

A vector with indices of all temperature sensors.

10.17.5.43 const cUnitConverter* SDH::cSDH::uc_angle

unit convert for (axis) angles: default = SDH::cSDH::uc_angle_degrees

10.17.5.44 const cUnitConverter* SDH::cSDH::uc_angular_velocity

unit convert for (axis) angular velocities: default = SDH::cSDH::uc_angular_velocity_degrees_per_second

10.17.5.45 const cUnitConverter* SDH::cSDH::uc_angular_acceleration

unit convert for (axis) angular accelerations: default = SDH::cSDH::uc_angular_acceleration_degrees_-per_second_squared

10.17.5.46 const cUnitConverter* SDH::cSDH::uc_time

unit convert for times: default = uc_time_seconds

10.17.5.47 const cUnitConverter* SDH::cSDH::uc_temperature

unit convert for temperatures: default = SDH::cSDH::uc_temperature_celsius

10.17.5.48 const cUnitConverter* SDH::cSDH::uc_motor_current

unit converter for motor curent: default = SDH::cSDH::uc_motor_current_ampere

10.17.5.49 const cUnitConverter* SDH::cSDH::uc_position

unit converter for position: default = SDH::cSDH::uc_position_millimeter

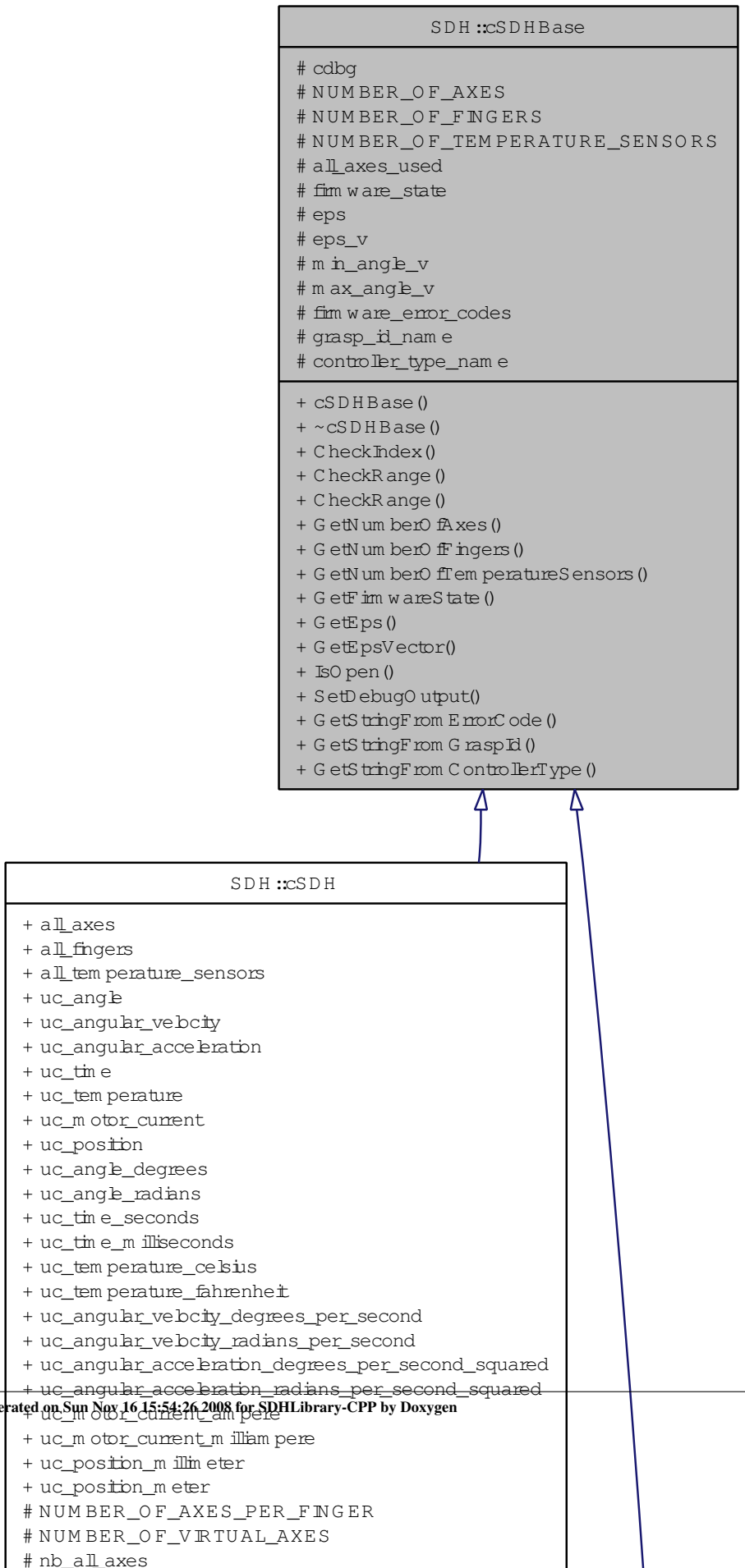
The documentation for this class was generated from the following files:

- sdh/sdh.h
- sdh/sdh.cpp

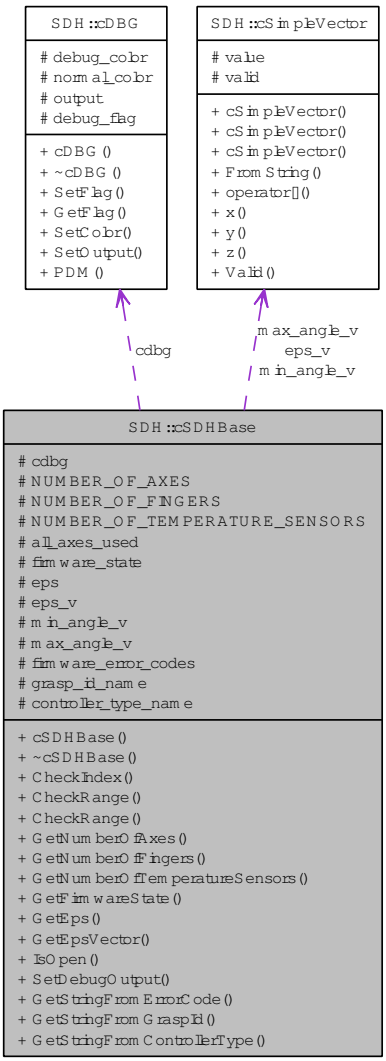
10.18 SDH::cSDHBase Class Reference

`#include <sdhbase.h>`

Inheritance diagram for SDH::cSDHBase:



Collaboration diagram for SDH::cSDHBase:



10.18.1 Detailed Description

The base class to control the SCHUNK Dexterous Hand.

End-Users should **NOT** use this class directly, as it only provides some common settings and no function interface. End users should use the class cSDH instead, as it provides the end-user functions to control the SDH.

Public Types

- enum { All = -1 }

  Anonymous enum (instead of define like macros).

- enum eErrorCode {

eEC_SUCCESS = 0, eEC_NOT_AVAILABLE = 1, eEC_NO_SENSOR = 2, eEC_NOT_-INITIALIZED = 3,

eEC_ALREADY_RUNNING = 4, eEC_FEATURE_NOT_SUPPORTED = 5, eEC_-INCONSISTENT_DATA = 6, eEC_TIMEOUT = 7,

eEC_READ_ERROR = 8, eEC_WRITE_ERROR = 9, eEC_INSUFFICIENT_RESOURCES = 10, eEC_CHECKSUM_ERROR = 11,

eEC_NOT_ENOUGH_PARAMS = 12, eEC_NO_PARAMS_EXPECTED = 13, eEC_CMD_-UNKNOWN = 14, eEC_CMD_FORMAT_ERROR = 15,

eEC_ACCESS_DENIED = 16, eEC_ALREADY_OPEN = 17, eEC_CMD_FAILED = 18, eEC_-CMD_ABORTED = 19,

eEC_INVALID_HANDLE = 20, eEC_DEVICE_NOT_FOUND = 21, eEC_DEVICE_NOT_-OPENED = 22, eEC_IO_ERROR = 23,

eEC_INVALID_PARAMETER = 24, eEC_RANGE_ERROR = 25, eEC_NO_DATAPIPE = 26, eEC_INDEX_OUT_OF_BOUNDS = 27,

eEC_HOMING_ERROR = 28, eEC_AXIS_DISABLED = 29, eEC_OVER_TEMPERATURE = 30, eEC_DIMENSION }

- enum eGraspId {

  eGID_INVALID = -1, eGID_CENTRICAL = 0, eGID_PARALLEL = 1, eGID_CYLINDRICAL = 2,

  eGID_SPHERICAL = 3, eGID_DIMENSION }

  The enum values of the known grasps.

- enum eControllerType { eCT_POSE = 0, eCT_DIMENSION }

  An enum for all possible SDH internal controller types.

- enum eVelocityProfile { eVP_INVALID = -1, eVP_SIN_SQUARE, eVP_RAMP, eVP_-DIMENSION }

  An enum for all possible SDH internal velocity profile types.

Public Member Functions

- cSDHBase (int debug_level)
- virtual ~cSDHBase ()
- void CheckIndex (int index, int maxindex, char const *name="") throw (cSDHErrorInvalidParameter*)

  Check if index *is in [0 ..* maxindex-1*] or All. Throw a cSDHErrorInvalidParameter exception if not.*

- void CheckRange (double value, double minvalue, double maxvalue, char const *name="") throw (cSDHErrorInvalidParameter*)

  Check if value *is in [*minvalue *..* maxvalue*]. Throw a cSDHErrorInvalidParameter exception if not.*

- void CheckRange (double *values, double *minvalues, double *maxvalues, char const *name="") throw (cSDHErrorInvalidParameter*)

  Check if any value[i] in array values *is in [*minvalue*[i] ..* maxvalue*[i]]. Throw a cSDHErrorInvalidParameter exception if not.*

- int GetNumberOfAxes (void)

Return the number of axes of the SDH.

- int GetNumberOfFingers (void)

  Return the number of fingers of the SDH.

- int GetNumberOfTemperatureSensors (void)

  Return the number of temperature sensors of the SDH.

- eErrorCode GetFirmwareState (void)

  Return the last known state of the SDH firmware.

- double GetEps (void)

  Return the eps value.

- cSimpleVector const & GetEpsVector (void)

  Return simple vector of number of axes epsilon values.

- virtual bool IsOpen (void)=0

  Return true if connection to SDH firmware/hardware is open.

- virtual void SetDebugOutput (std::ostream *debuglog)

  change the stream to use for debug messages

Static Public Member Functions

- static char const * GetStringFromErrorCode (eErrorCode error_code)

  Return a ptr to a (static) string describing error code error_code.

- static char const * GetStringFromGraspId (eGraspId grasp_id)

  Return a ptr to a (static) string describing grasp id grasp_id.

- static char const * GetStringFromControllerType (eControllerType controller_type)

  Return a ptr to a (static) string describing controller type controller_Type.

Protected Attributes

- cDBG cdbg

  debug stream to print colored debug messages

- int NUMBER_OF_AXES

  The number of axes.

- int NUMBER_OF_FINGERS

  The number of fingers.

- int NUMBER_OF_TEMPERATURE_SENSORS

  The number of temperature sensors.

- int all_axes_used

  Bit field with the bits for all axes set.

- eErrorCode firmware_state

  the last known state of the SDH firmware

- double eps

  epsilon value (max absolute deviation of reported values from actual hardware values) (needed since firmware limits number of digits reported)

- cSimpleVector eps_v

  simple vector of 7 epsilon values

- cSimpleVector min_angle_v

  simple vector of 7 0 values ???

- cSimpleVector max_angle_v

  Maximum allowed axis angles (in internal units (degrees)).

Static Protected Attributes

- static char const * firmware_error_codes [ ]

  A mapping from eErrorCode error code enums to strings with human readable error messages.

- static char const * grasp_id_name [ ]

  A mapping from eGraspId grasp id enums to strings with human readable grasp id names.

- static char const * controller_type_name [ ]

  A mapping from eControllerType controller type enums to strings with human readable controller type names.

10.18.2 Member Enumeration Documentation

10.18.2.1 anonymous enum

Anonymous enum (instead of define like macros).

Enumerator:

All A meta-value that means "access all possible values".

10.18.2.2 enum SDH::cSDHBase::eErrorCode

The error codes of the SDH firmware

Enumerator:

eEC_SUCCESS

eEC_NOT_AVAILABLE
eEC_NO_SENSOR
eEC_NOT_INITIALIZED
eEC_ALREADY_RUNNING
eEC_FEATURE_NOT_SUPPORTED
eEC_INCONSISTENT_DATA
eEC_TIMEOUT
eEC_READ_ERROR
eEC_WRITE_ERROR
eEC_INSUFFICIENT_RESOURCES
eEC_CHECKSUM_ERROR
eEC_NOT_ENOUGH_PARAMS
eEC_NO_PARAMS_EXPECTED
eEC_CMD_UNKNOWN
eEC_CMD_FORMAT_ERROR
eEC_ACCESS_DENIED
eEC_ALREADY_OPEN
eEC_CMD_FAILED
eEC_CMD_ABORTED
eEC_INVALID_HANDLE
eEC_DEVICE_NOT_FOUND
eEC_DEVICE_NOT_OPENED
eEC_IO_ERROR
eEC_INVALID_PARAMETER
eEC_RANGE_ERROR
eEC_NO_DATAPIPE
eEC_INDEX_OUT_OF_BOUNDS
eEC_HOMING_ERROR
eEC_AXIS_DISABLED
eEC_OVER_TEMPERATURE
eEC_DIMENSION Endmarker and dimension.

10.18.2.3 enum SDH::cSDHBase::eGraspId

The enum values of the known grasps.

Enumerator:

eGID_INVALID invalid grasp id
eGID_CENTRICAL centrical grasp: ???
eGID_PARALLEL parallel grasp: ???
eGID_CYLINDRICAL cylindrical grasp: ???
eGID_SPHERICAL spherecial grasp: ???
eGID_DIMENSION Endmarker and dimension.

10.18.2.4 enum SDH::cSDHBase::eControllerType

An enum for all possible SDH internal controller types.

Enumerator:

eCT_POSE position controller (position per axis => "pose controller")

eCT_DIMENSION Endmarker and dimension.

10.18.2.5 enum SDH::cSDHBase::eVelocityProfile

An enum for all possible SDH internal velocity profile types.

Enumerator:

eVP_INVALID not a valid velocity profile, used to make SDH::cSDHSerial::vp() read the velocity profile from the firmware

eVP_SIN_SQUARE sin square velocity profile

eVP_RAMP ramp velocity profile

eVP_DIMENSION endmarker and dimension

10.18.3 Constructor & Destructor Documentation

10.18.3.1 cSDHBase::cSDHBase (int *debug_level*)

Constructor of cSDHBase class, initilize internal variables and settings

Parameters:

debug_level : debug level of the created object. If the *debug_level* of an object is > 0 then it will output debug messages.

- (Subclasses of cSDHBase like cSDH or cSDHSerial use additional settings, see there.)

10.18.3.2 virtual SDH::cSDHBase::~cSDHBase () [inline, virtual]

virtual destructor to make compiler happy

10.18.4 Member Function Documentation

10.18.4.1 void cSDHBase::CheckIndex (int *index*, int *maxindex*, char const * *name* = "") throw (cSDHErrorInvalidParameter*)

Check if *index* is in [0 .. *maxindex-1*] or All. Throw a cSDHErrorInvalidParameter exception if not.

10.18.4.2 void cSDHBase::CheckRange (double *value*, double *minvalue*, double *maxvalue*, char const * *name* = "") throw (cSDHErrorInvalidParameter*)

Check if *value* is in [*minvalue* .. *maxvalue*]. Throw a cSDHErrorInvalidParameter exception if not.

10.18.4.3 void cSDHBase::CheckRange (double * *values*, double * *minvalues*, double * *maxvalues*, char const * *name* = "") throw (cSDHErrorInvalidParameter*)

Check if any value[i] in array *values* is in [*minvalue*[i] .. *maxvalue*[i]]. Throw a cSDHErrorInvalidParameter exception if not.

10.18.4.4 char const * cSDHBase::GetStringFromErrorCode (eErrorCode *error_code*) [static]

Return a ptr to a (static) string describing error code *error_code*.

10.18.4.5 char const * cSDHBase::GetStringFromGraspId (eGraspId *grasp_id*) [static]

Return a ptr to a (static) string describing grasp id *grasp_id*.

10.18.4.6 char const * cSDHBase::GetStringFromControllerType (eControllerType *controller_type*) [static]

Return a ptr to a (static) string describing controller type *controller_Type*.

10.18.4.7 int cSDHBase::GetNumberOfAxes (void)

Return the number of axes of the SDH.

10.18.4.8 int cSDHBase::GetNumberOfFingers (void)

Return the number of fingers of the SDH.

10.18.4.9 int cSDHBase::GetNumberOfTemperatureSensors (void)

Return the number of temperature sensors of the SDH.

10.18.4.10 cSDHBase::eErrorCode cSDHBase::GetFirmwareState (void)

Return the last known state of the SDH firmware.

10.18.4.11 double cSDHBase::GetEps (void)

Return the eps value.

10.18.4.12 cSimpleVector const & cSDHBase::GetEpsVector (void)

Return simple vector of number of axes epsilon values.

10.18.4.13 virtual bool SDH::cSDHBase::IsOpen (void) [pure virtual]

Return true if connection to SDH firmware/hardware is open.

Implemented in SDH::cSDH, and SDH::cSDHSerial.

10.18.4.14 virtual void SDH::cSDHBase::SetDebugOutput (std::ostream * *debuglog*) [inline, virtual]

change the stream to use for debug messages

Reimplemented in SDH::cSDH.

10.18.5 Member Data Documentation

10.18.5.1 cDBG SDH::cSDHBase::cdbg [protected]

debug stream to print colored debug messages

10.18.5.2 char const * cSDHBase::firmware_error_codes [static, protected]

A mapping from eErrorCode error code enums to strings with human readable error messages.

10.18.5.3 char const * cSDHBase::grasp_id_name [static, protected]

Initial value:

```
{

    "eGID_CENTRICAL: centrical grasp",
    "eGID_PARALLEL: parallel grasp",
    "eGID_CYLINDRICAL: cylindrical grasp",
    "eGID_SPHERICAL: spherecial grasp",

    "eGID_DIMENSION: number of predefined grasp ids"
}
```

A mapping from eGraspId grasp id enums to strings with human readable grasp id names.

10.18.5.4 char const * cSDHBase::controller_type_name [static, protected]

Initial value:

```
{

    "eCT_POSE: position/pose controller",

    "eCT_DIMENSION: number of controller types"
}
```

A mapping from eControllerType controller type enums to strings with human readable controller type names.

10.18.5.5 int SDH::cSDHBase::NUMBER_OF_AXES [protected]

The number of axes.

10.18.5.6 int SDH::cSDHBase::NUMBER_OF_FINGERS [protected]

The number of fingers.

10.18.5.7 int SDH::cSDHBase::NUMBER_OF_TEMPERATURE_SENSORS [protected]

The number of temperature sensors.

10.18.5.8 int SDH::cSDHBase::all_axes_used [protected]

Bit field with the bits for all axes set.

10.18.5.9 eErrorCode SDH::cSDHBase::firmware_state [protected]

the last known state of the SDH firmware

10.18.5.10 double SDH::cSDHBase::eps [protected]

epsilon value (max absolute deviation of reported values from actual hardware values) (needed since firmware limits number of digits reported)

10.18.5.11 cSimpleVector SDH::cSDHBase::eps_v [protected]

simple vector of 7 epsilon values

10.18.5.12 cSimpleVector SDH::cSDHBase::min_angle_v [protected]

simple vector of 7 0 values ???

simple vector of 7 1 values ??? Minimum allowed axis angles (in internal units (degrees))

10.18.5.13 cSimpleVector SDH::cSDHBase::max_angle_v [protected]

Maximum allowed axis angles (in internal units (degrees)).

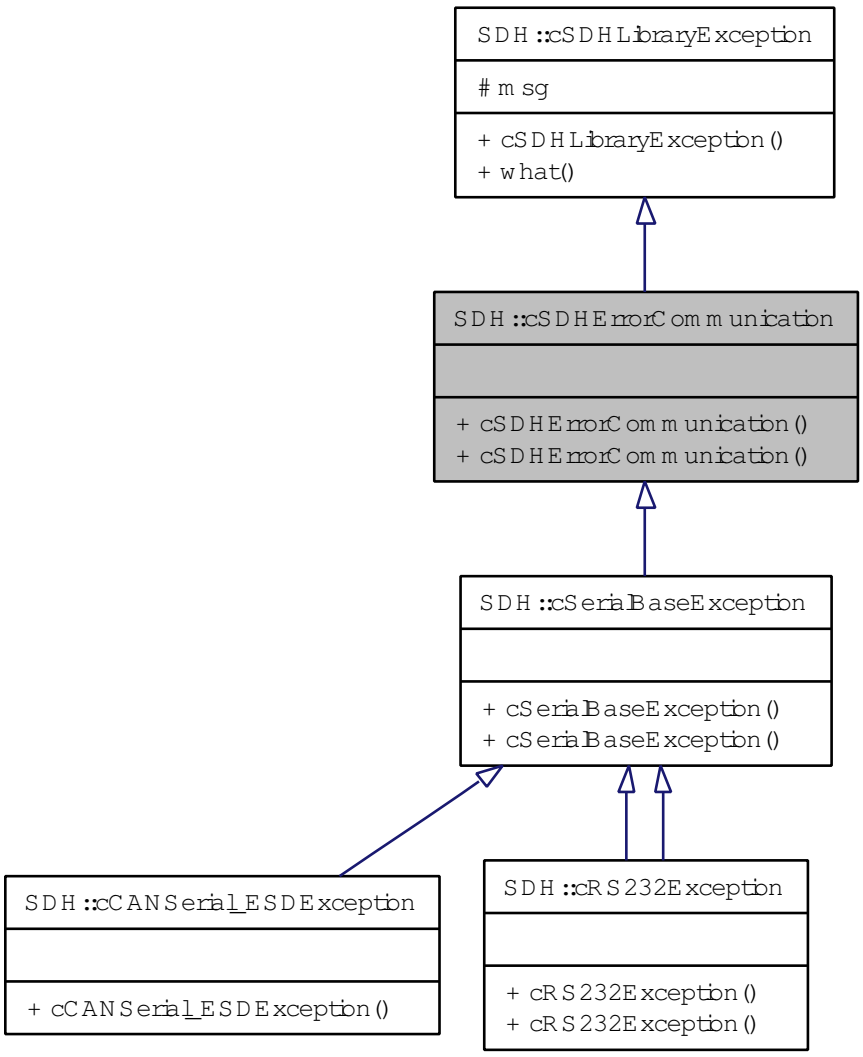
The documentation for this class was generated from the following files:

- sdh/sdhbase.h
- sdh/sdhbase.cpp

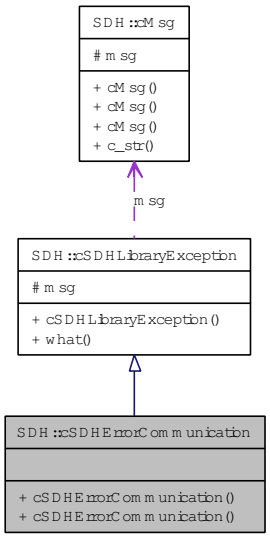
10.19 SDH::cSDHErrorCommunication Class Reference

`#include <sdhexception.h>`

Inheritance diagram for SDH::cSDHErrorCommunication:



Collaboration diagram for SDH::cSDHErrorCommunication:



10.19.1 Detailed Description

Derived exception class for exceptions related to communication between the SDHLibrary and the SDH.

Public Member Functions

- cSDHErrorCommunication (cMsg const &_msg)
- cSDHErrorCommunication (char const *_type, cMsg const &_msg)

10.19.2 Constructor & Destructor Documentation

10.19.2.1 SDH::cSDHErrorCommunication::cSDHErrorCommunication (cMsg const & *_msg*) [inline]

10.19.2.2 SDH::cSDHErrorCommunication::cSDHErrorCommunication (char const * *_type*, cMsg const & *_msg*) [inline]

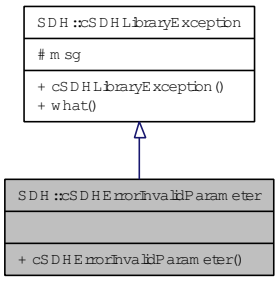
The documentation for this class was generated from the following file:

- sdh/sdhexception.h

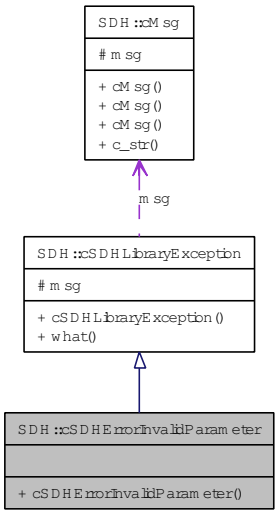
10.20 SDH::cSDHErrorInvalidParameter Class Reference

`#include <sdhbase.h>`

Inheritance diagram for SDH::cSDHErrorInvalidParameter:



Collaboration diagram for SDH::cSDHErrorInvalidParameter:



10.20.1 Detailed Description

Derived exception class for exceptions related to invalid parameters.

Public Member Functions

- cSDHErrorInvalidParameter (cMsg const &_msg)

10.20.2 Constructor & Destructor Documentation

10.20.2.1 SDH::cSDHErrorInvalidParameter::cSDHErrorInvalidParameter (cMsg const & *_msg*) [inline]

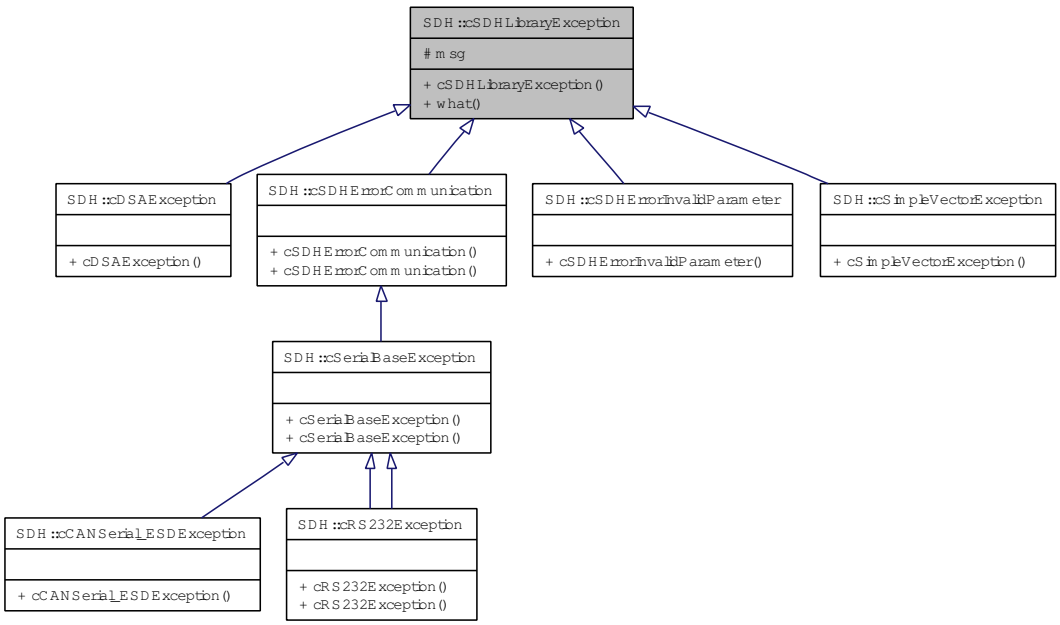
The documentation for this class was generated from the following file:

- sdh/sdhbase.h

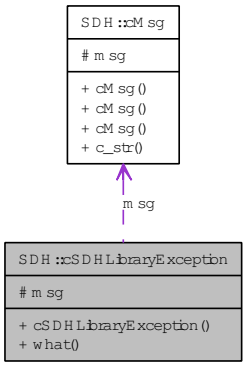
10.21 SDH::cSDHLibraryException Class Reference

`#include <sdhexception.h>`

Inheritance diagram for SDH::cSDHLibraryException:



Collaboration diagram for SDH::cSDHLibraryException:



10.21.1 Detailed Description

Base class for exceptions in the SDHLibrary-CPP.

At construction time a cMsg object is stored in the msg member of the cSDHLibraryException object. The cMsg object should contain a string wich further describes the actual cause of the exception thrown. The string in the cMsg object can be queried with the overloaded what() member function, just like in the std::exception class.

See the verbose description of the constructor SDH::cSDHLibraryException::cSDHLibraryException() for examplary use.

Public Member Functions

- cSDHLibraryException (char const *_type, cMsg const &_msg)
- virtual const char * what () const throw ()

Protected Attributes

- cMsg msg

  The message object.

10.21.2 Constructor & Destructor Documentation

10.21.2.1 cSDHLibraryException::cSDHLibraryException (char const * *_type*, cMsg const & *_msg*)

Constructor of sdh exception base class.

Parameters:

_type - the type name of the exception. By convention this is the class name of the exception

_msg - a reference to a cMsg object that further describes the exception.

Remarks:

- The *_type* parameter is mainly usefull in derived classes
- The *_msg* given as parameter is copied to the msg member. Thus the given *_msg* object can be an anonymous object, like in:

```
...
if ( v > v_max )
  throw new cSDHLibraryException( "cSDHLibraryException", cMsg( "Failed since v is invalid (v=%d > %
```

- But exceptions of the base will hardly ever be thrown. Instead objects of derived, more specific classes will be thrown. This looks like:

```
// Derived exception class for more specific exceptions:
class cDerivedException : public cSDHLibraryException
{
public:
  cDerivedException( cMsg const & _msg )
    : cSDHLibraryException( "cDerivedException", _msg )
  {}
}
// (Yes that is really all that must be done here!)

...

try
{
  ...
  if ( v > v_max )
    throw new cDerivedException( cMsg( "Failed since v is invalid (v=%d > %d=v_max)", v, v_max
  ...
}
catch ( cDerivedException *e )
```

```
{
  cerr << "Caught exception " << e->what() << "\n";
  // handle exception
  ...

  // finally delete the caught exception
  delete e;
}
```

10.21.3 Member Function Documentation

10.21.3.1 const char * cSDHLibraryException::what () const throw () [virtual]

Return the msg member

10.21.4 Member Data Documentation

10.21.4.1 cMsg SDH::cSDHLibraryException::msg [protected]

The message object.

The documentation for this class was generated from the following files:

- sdh/sdhexception.h
- sdh/sdhexception.cpp

10.22 cSDHOptions Class Reference

`#include <sdhoptions.h>`

Public Member Functions

- cSDHOptions (void)

  constructor: init members to their default values

- int Parse (int argc, char **argv, char const *helptext, char const *progname, char const *version, char const *libname, char const *librelease)

Public Attributes

- char * usage
- int port
- unsigned long rs232_baudrate
- int debug_level
- std::ostream * debuglog
- bool use_radians
- bool use_fahrenheit
- double period
- double timeout
- bool use_can
- int net
- unsigned long can_baudrate
- unsigned int id_read
- unsigned int id_write

10.22.1 Constructor & Destructor Documentation

10.22.1.1 cSDHOptions::cSDHOptions (void)

constructor: init members to their default values

10.22.2 Member Function Documentation

10.22.2.1 int cSDHOptions::Parse (int *argc*, char ** *argv*, char const * *helptext*, char const * *progname*, char const * *version*, char const * *libname*, char const * *librelease*)

parse the command line parameters *argc*, *argv* into members. *helptext*, *progname*, *version*, *libname* and *librelease* are used when printing online help. start parsing at option with index *p_option_index parse all options if parse_all is true, else only one option is parsed

Returns:

the optind index of the first non option argument in argv

10.22.3 Member Data Documentation

10.22.3.1 char* cSDHOptions::usage

10.22.3.2 int cSDHOptions::port

10.22.3.3 unsigned long cSDHOptions::rs232_baudrate

10.22.3.4 int cSDHOptions::debug_level

10.22.3.5 std::ostream* cSDHOptions::debuglog

10.22.3.6 bool cSDHOptions::use_radians

10.22.3.7 bool cSDHOptions::use_fahrenheit

10.22.3.8 double cSDHOptions::period

10.22.3.9 double cSDHOptions::timeout

10.22.3.10 bool cSDHOptions::use_can

10.22.3.11 int cSDHOptions::net

10.22.3.12 unsigned long cSDHOptions::can_baudrate

10.22.3.13 unsigned int cSDHOptions::id_read

10.22.3.14 unsigned int cSDHOptions::id_write

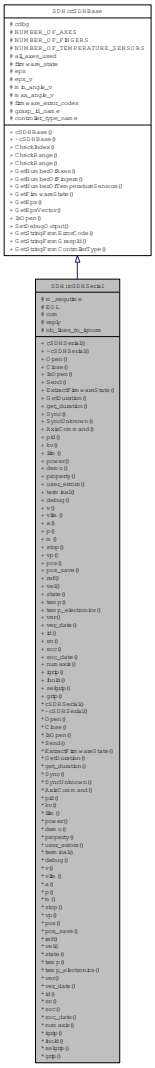
The documentation for this class was generated from the following files:

- demo/sdhoptions.h
- demo/sdhoptions.cpp

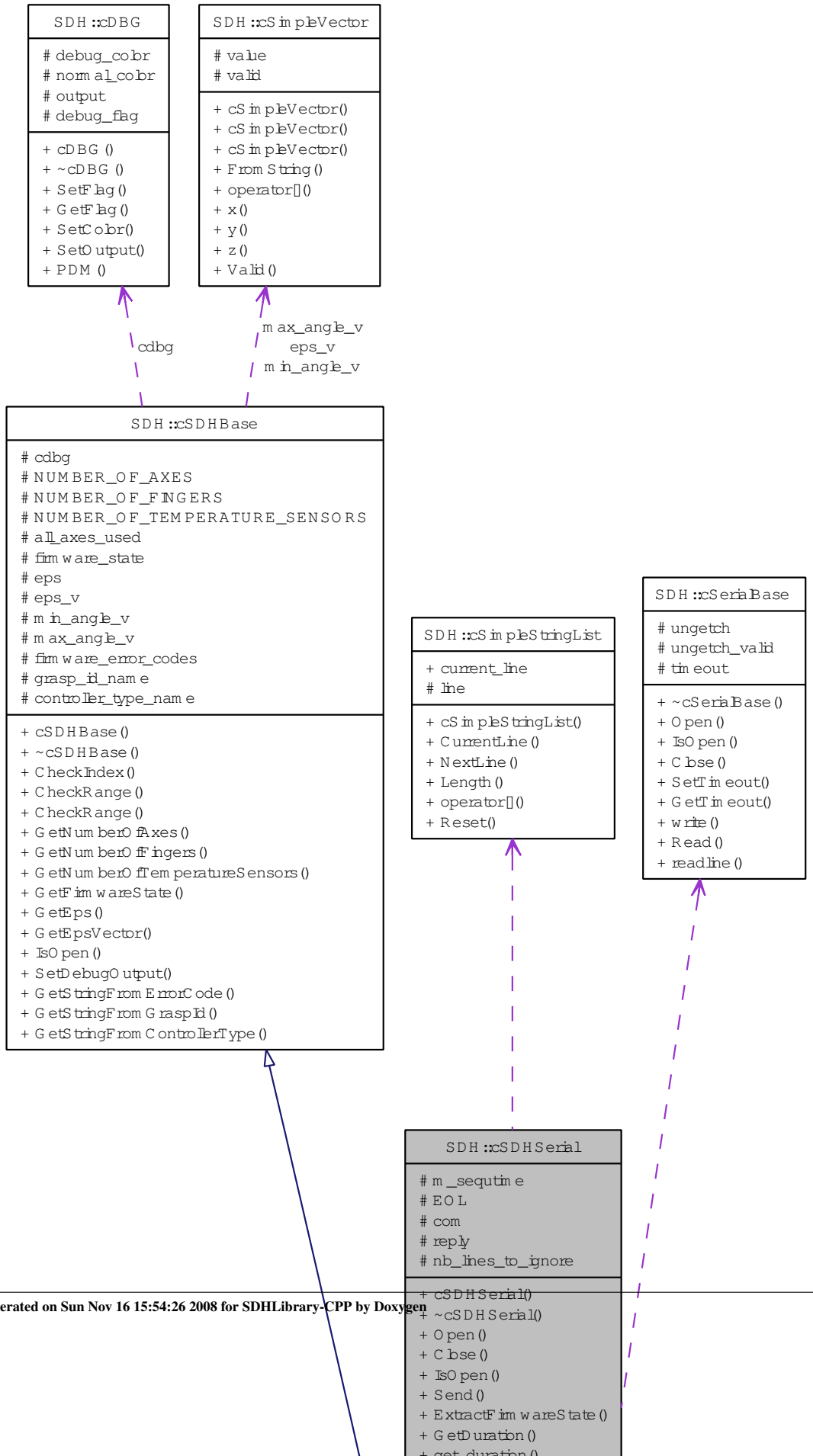
10.23 SDH::cSDHSerial Class Reference

`#include <sdhserial.h>`

Inheritance diagram for SDH::cSDHSerial:



Collaboration diagram for SDH::cSDHSerial:



10.23.1 Detailed Description

The class to communicate with a SDH via RS232.

End-Users should **NOT** use this class directly! The interface of cSDHSerial is subject to change in future releases. End users should use the class cSDH instead, as that interface is considered more stable.

Public Member Functions

Internal methods

- cSDHSerial (int _debug_level=0)
  Constructor of cSDHSerial.

- virtual ~cSDHSerial ()
- void Open (cSerialBase *_com) throw (cSDHLibraryException*)
- void Close () throw (cSDHLibraryException*)
- virtual bool IsOpen (void)
- void Send (char const *s, int nb_lines=All, int nb_lines_total=All, int max_retries=3) throw (cSDHLibraryException*)
- void ExtractFirmwareState () throw (cSDHErrorCommunication*)
- double GetDuration (char *line) throw (cSDHErrorCommunication*)
- double get_duration (void)
- void Sync () throw (cSDHErrorCommunication*)
- void SyncUnknown () throw (cSDHErrorCommunication*)
- cSimpleVector AxisCommand (char *command, int axis=All, double *value=NULL) throw (cSDHLibraryException*)

Setup and configuration methods

- cSimpleVector pid (int axis, double *p=NULL, double *i=NULL, double *d=NULL) throw (cSDHLibraryException*)
- cSimpleVector kv (int axis=All, double *kv=NULL) throw (cSDHLibraryException*)
- cSimpleVector ilim (int axis=All, double *limit=NULL) throw (cSDHLibraryException*)
- cSimpleVector power (int axis=All, double *flag=NULL) throw (cSDHLibraryException*)

Misc. methods

- void demo (bool onoff)
- int property (char *propname, int value)
- int user_errors (int value)
- int terminal (int value)
- int debug (int value)

Movement methods

- cSimpleVector v (int axis=All, double *velocity=NULL) throw (cSDHLibraryException*)
- cSimpleVector vlim (int axis=All, double *dummy=NULL) throw (cSDHLibraryException*)
- cSimpleVector a (int axis=All, double *acceleration=NULL) throw (cSDHLibraryException*)
- cSimpleVector p (int axis=All, double *angle=NULL) throw (cSDHLibraryException*)
- double m (bool sequ) throw (cSDHLibraryException*)
- void stop (void) throw (cSDHLibraryException*)
- eVelocityProfile vp (eVelocityProfile velocity_profile=eVP_INVALID) throw (cSDHLibraryException*)

Diagnostic and identification methods

- cSimpleVector pos (int axis=All, double *dummy=NULL) throw (cSDHLibraryException*)
- cSimpleVector pos_save (int axis=All, double *value=NULL) throw (cSDHLibraryException*)
- cSimpleVector ref (int axis=All, double *value=NULL) throw (cSDHLibraryException*)
- cSimpleVector vel (int axis=All, double *dummy=NULL) throw (cSDHLibraryException*)
- cSimpleVector state (int axis=All, double *dummy=NULL) throw (cSDHLibraryException*)
- cSimpleVector temp (void) throw (cSDHLibraryException*)
- cSimpleVector temp_electronics (void) throw (cSDHLibraryException*)
- char * ver (void) throw (cSDHLibraryException*)
- char * ver_date (void) throw (cSDHLibraryException*)
- char * id (void) throw (cSDHLibraryException*)
- char * sn (void) throw (cSDHLibraryException*)
- char * soc (void) throw (cSDHLibraryException*)
- char * soc_date (void) throw (cSDHLibraryException*)
- int numaxis (void) throw (cSDHLibraryException*)

Grip methods

- cSimpleVector igrip (int axis=All, double *limit=NULL) throw (cSDHLibraryException*)
- cSimpleVector ihold (int axis=All, double *limit=NULL) throw (cSDHLibraryException*)
- double selgrip (eGraspId grip, bool sequ) throw (cSDHLibraryException*)
- double grip (double close, double velocity, bool sequ) throw (cSDHLibraryException*)

Protected Attributes

- double m_sequtime

  additional time in seconds to wait for sequential execution of m command (as these are always executed non-sequentially by the firmware

- char * EOL

  String to use as "End Of Line" marker when sending to SDH.

- cSerialBase * com

  The communication object to the serial device (RS232 port or ESD CAN net).

- cSimpleStringList reply

  Space for the replies from the SDH.

- int nb_lines_to_ignore

  number of remaining reply lines of a previous (non-sequential) command

10.23.2 Constructor & Destructor Documentation

10.23.2.1 cSDHSerial::cSDHSerial (int *_debug_level* = 0)

Constructor of cSDHSerial.

Parameters:

_debug_level : debug level of the created object. If the *debug_level* of an object is > 0 then it will output debug messages. (forwared to constructor of base class)

String to use as "End Of Line" marker when sending to SDH

10.23.2.2 virtual SDH::cSDHSerial::~cSDHSerial () [inline, virtual]

virtual destructor to make compiler happy

10.23.3 Member Function Documentation

10.23.3.1 void cSDHSerial::Open (cSerialBase * *_com*) throw (cSDHLibraryException*)

Open the serial device and check connection to SDH by querying the SDH firmware version

Parameters:

_com - ptr to the serial device to use

This may throw an exception on failure.

The serial port on the PC-side can be opened successfully even if no SDH is attached. Therefore this routine tries to read the SDH firmware version with a 1s timeout after the port is opened. If the SDH does not reply in time then

- an error message is printed on stderr,
- the port is closed
- and a cSerialBaseException* exception is thrown.

10.23.3.2 void cSDHSerial::Close () throw (cSDHLibraryException*)

Close connection to serial port.

10.23.3.3 bool cSDHSerial::IsOpen (void) [virtual]

Return true if connection to SDH firmware/hardware is open

Implements SDH::cSDHBase.

10.23.3.4 void cSDHSerial::Send (char const * *s*, int *nb_lines* = All, int *nb_lines_total* = All, int *max_retries* = 3) throw (cSDHLibraryException*)

Send command string s+EOL to com and read reply according to nb_lines.

If nb_lines == All then reply lines are read until a line without "@" prefix is found. If nb_lines != All it is the number of lines to read.

firmware_state is set according to reply (if read) nb_lines_total contains the total number of lines replied for the s command. If fewer lines are read then nb_lines_total-nb_lines will be remembered to be ignored before the next command can be sent.

Return a list of all read lines of the reply from the SDH hardware.

10.23.3.5 void cSDHSerial::ExtractFirmwareState () throw (cSDHErrorCommunication*)

Try to extract the state of the SDH firmware from the last reply

10.23.3.6 double cSDHSerial::GetDuration (char * *line*) throw (cSDHErrorCommunication*)

Return duration of the execution of a SDH command as reported by line

10.23.3.7 double cSDHSerial::get_duration (void)

Send get_duration command. Returns the calculated duration of the currently configured movement (target positions, velocities, accelerations and velocity profile.

return the expected duration of the execution of the command in seconds

10.23.3.8 void cSDHSerial::Sync () throw (cSDHErrorCommunication*)

Read all pending lines from SDH to resync execution of PC and SDH.

10.23.3.9 void cSDHSerial::SyncUnknown () throw (cSDHErrorCommunication*)

Read an unknown number of lines from SDH to resync execution of PC and SDH.

10.23.3.10 cSimpleVector cSDHSerial::AxisCommand (char * *command*, int *axis* = `All`, double * *value* = `NULL`) throw (cSDHLibraryException*)

Get/Set values.

- If axis is All and value is None then a NUMBER_OF_AXES-list of the actual values read from the SDH is returned
- If axis is a single number and value is None then the actual value for that axis is read from the SDH and is returned
- If axis and value are single numbers then that value is set for that axis and returned.
- If axis is All and value is a NUMBER_OF_AXES-vector then all axes values are set accordingly, a NUMBER_OF_AXES-list is returned.

10.23.3.11 cSimpleVector cSDHSerial::pid (int *axis*, double * *p* = `NULL`, double * *i* = `NULL`, double * *d* = `NULL`) throw (cSDHLibraryException*)

Get/Set PID controller parameters

- axis must be a single number: the index of the axis to get/set
- If p,i,d are None then a list of the actually set PID controller parameters of the axis is returned
- If p,i,d are numbers then the PID controller parameters for that axis are set (and returned).

10.23.3.12 cSimpleVector cSDHSerial::kv (int *axis* = All, double * *kv* = NULL) throw (cSDHLibraryException*)

Get/Set kv parameter

- If axis is All and kv is None then a NUMBER_OF_AXES-list of the actually set kv parameters is returned
- If axis is a single number and kv is None then the kv parameter for that axis is returned.
- If axis and kv are single numbers then the kv parameter for that axis is set (and returned).
- If axis is All and kv is a NUMBER_OF_AXES-vector then all axes kv parameters are set accordingly, NUMBER_OF_AXES-list is returned.

10.23.3.13 cSimpleVector cSDHSerial::ilim (int *axis* = All, double * *limit* = NULL) throw (cSDHLibraryException*)

Get/Set actual motor current limit for m command

- If axis is All and limit is None then a NUMBER_OF_AXES-list of the actually set motor current limits is returned
- If axis is a single number and limit is None then the motor current limit for that axis is returned.
- If axis and limit are single numbers then the motor current limit for that axis is set (and returned).
- If axis is All and limit is a NUMBER_OF_AXES-vector then all axes motor current limits are set accordingly, the NUMBER_OF_AXES-list is returned.

10.23.3.14 cSimpleVector cSDHSerial::power (int *axis* = All, double * *flag* = NULL) throw (cSDHLibraryException*)

Get/Set actual power state

- If axis is All and flag is None then a NUMBER_OF_AXES-list of the actually set power states is returned
- If axis is a single number and flag is None then the power state for that axis is returned.
- If axis is a single number and flag is a single number or a boolean value then the power state for that axis is set (and returned).
- If axis is All and flag is a NUMBER_OF_AXES-vector then all axes power states are set accordingly, the NUMBER_OF_AXES-list is returned.
- If axis is All and flag is a a single number or a boolean value then all axes power states are set to that value, the NUMBER_OF_AXES-list is returned.

10.23.3.15 void cSDHSerial::demo (bool *onoff*)

Enable/disable SCHUNK demo

10.23.3.16 int cSDHSerial::property (char * *propname*, int *value*)

Set named property

Valid propnames are:

- "user_errors"
- "terminal"
- "debug"

10.23.3.17 int cSDHSerial::user_errors (int *value*)

10.23.3.18 int cSDHSerial::terminal (int *value*)

10.23.3.19 int cSDHSerial::debug (int *value*)

10.23.3.20 cSimpleVector cSDHSerial::v (int *axis* = `All`, double * *velocity* = `NULL`) throw (cSDHLibraryException*)

Get/Set target velocity. (NOT the actual velocity!)

The default velocity is 40 deg/s for axes 0-6. Due to some firmware bug axis 0 can go no faster than 14 deg/s

- If axis is All and velocity is None then a NUMBER_OF_AXES-list of the actually set target velocities is returned
- If axis is a single number and velocity is None then the target velocity for that axis is returned.
- If axis and velocity are single numbers then the target velocity for that axis is set (and returned).
- If axis is All and velocity is a NUMBER_OF_AXES-vector then all axes target velocities are set accordingly, the NUMBER_OF_AXES-list is returned.

Velocities are set/reported in degrees per second.

10.23.3.21 cSimpleVector cSDHSerial::vlim (int *axis* = `All`, double * *dummy* = `NULL`) throw (cSDHLibraryException*)

Get velocity limits.

- If axis is All then a NUMBER_OF_AXES-list of the velocity limits is returned
- If axis is a single number then the velocity limit for that axis is returned.

dummy parameter is just needed to make this function have the same signature as e.g. v(), so it can be used as a function pointer.

Velocity limits are reported in degrees per second.

10.23.3.22 cSimpleVector cSDHSerial::a (int *axis* = `All`, double * *acceleration* = `NULL`) throw (cSDHLibraryException*)

Get/Set target acceleration for axis. (NOT the actual acceleration!)

- If axis is All and acceleration is None then a NUMBER_OF_AXES-list of the actually set target accelerations is returned
- If axis is a single number and acceleration is None then the target acceleration for that axis is returned.
- If axis and acceleration are single numbers then the target acceleration for that axis is set (and returned).
- If axis is All and acceleration is a NUMBER_OF_AXES-vector then all axes target accelerations are set accordingly, the NUMBER_OF_AXES-list is returned.

Accelerations are set/reported in degrees per second squared.

10.23.3.23 cSimpleVector cSDHSerial::p (int *axis* = `All`, double * *angle* = `NULL`) throw (cSDHLibraryException*)

Get/Set target angle for axis. (NOT the actual angle!)

- If axis is All and angle is None then a NUMBER_OF_AXES-list of the actually set target angles is returned
- If axis is a single number and angle is None then the target angle for that axis is returned.
- If axis and angle are single numbers then the target angle for that axis is set (and returned).
- If axis is All and angle is a NUMBER_OF_AXES-vector then all axes target angles are set accordingly, the NUMBER_OF_AXES-list is returned.

Angles are set/reported in degrees.

10.23.3.24 double cSDHSerial::m (bool *sequ*) throw (cSDHLibraryException*)

Send move command. Moves all enabled axes to their previously set target angle. The movement duration is determined by that axis that takes longest with its actually set velocity. The actual velocity of all other axes is set so that all axes begin and end their movements synchronously.

If sequ is True then wait until SDH hardware fully executed the command. Else return immediately and do not wait until SDH hardware fully executed the command.

return the expected duration of the execution of the command in seconds

10.23.3.25 void cSDHSerial::stop (void) throw (cSDHLibraryException*)

Stop sdh.

Will NOT interrupt a previous "selgrip" or "grip" command, only an "m" command!

10.23.3.26 cSDHBase::eVelocityProfile cSDHSerial::vp (eVelocityProfile *velocity_profile* = eVP_INVALID) throw (cSDHLibraryException*)

Get/set velocity profile.

If *velocity_profile* is < 0 then the actually set velocity profile is read from the firmware and returned. Else the given velocity_profile type is set in the firmware if valid.

10.23.3.27 cSimpleVector cSDHSerial::pos (int *axis* = All, double * *dummy* = NULL) throw (cSDHLibraryException*)

Get actual angle/s of axis/axes.

- If axis is All then a NUMBER_OF_AXES-vector of the actual axis angles is returned
- If axis is a single number then the actual angle of that axis is returned.

Angles are reported in degrees.

Remarks:

dummy ptr is never used, but needed nonetheless to make the signature of the function the same as for the other axis-access functions. This way a pointer to it can be used as a pointer to the other functions, which is needed by the generic cSDH::SetAxisValue and cSDH::GetAxisValue functions.

10.23.3.28 cSimpleVector cSDHSerial::pos_save (int *axis* = All, double * *value* = NULL) throw (cSDHLibraryException*)

Save actual angle/s to non volatile memory. (Usefull for axes that dont have an absolute encoder)

- If value is None then an exception is thrown since this is NOT usefull if any axis has an absolute encoder that the LLC knows about since these positions will be invalidated at the next start
- If axis and value are single numbers then that axis is saved.
- If axis is All and value is a NUMBER_OF_AXES-vector then all axes are saved if the corresponding value is 1.
- This will yield a E_RANGE_ERROR if any of the given values is not 0 or 1

10.23.3.29 cSimpleVector cSDHSerial::ref (int *axis* = All, double * *value* = NULL) throw (cSDHLibraryException*)

Do reference movements with selected axes. (Usefull for axes that dont have an absolute encoder)

each *value* must be either

- 0 : do not reference
- 1 : reference till mechanical block in positive direction
- 2 : reference till mechanical block in negative direction

- If *axis* and *value* are single numbers then that axis is referenced as requested
- If *axis* and *value* is a NUMBER_OF_AXES-vector then all axes are referenced as requested.
- This will yield a E_RANGE_ERROR if any of the given values is not 0 or 1 or 2

10.23.3.30 cSimpleVector cSDHSerial::vel (int *axis* = `All`, double * *dummy* = `NULL`) throw (cSDHLibraryException*)

Get actual angular velocity/s of axis/axes.

- If axis is All then a NUMBER_OF_AXES-vector of the actual axis angular velocities is returned
- If axis is a single number then the actual angular velocity of that axis is returned.

Angular velocities are reported in degrees/second.

Remarks:

dummy ptr is never used, but needed nonetheless to make the signature of the function the same as for the other axis-access functions. This way a pointer to it can be used as a pointer to the other functions, which is needed by the generic cSDH::SetAxisValue and cSDH::GetAxisValue functions.

10.23.3.31 cSimpleVector cSDHSerial::state (int *axis* = `All`, double * *dummy* = `NULL`) throw (cSDHLibraryException*)

Get actual state/s of axis/axes.

A state of 0 means "not moving" while 1 means "moving".

- If axis is All then a NUMBER_OF_AXES-vector of the actual axis states is returned
- If axis is a single number then the actual state of that axis is returned.

10.23.3.32 cSimpleVector cSDHSerial::temp (void) throw (cSDHLibraryException*)

Get actual temperatures of the axis motors

Returns a list of the actual controller and driver temperature in degrees celsius.

10.23.3.33 cSimpleVector cSDHSerial::temp_electronics (void) throw (cSDHLibraryException*)

Get actual temperatures of the electronics, i.e. teh FPGA and the PCB. (FPGA = Field Programmable Gate Array = the reprogrammable chip with the soft processors) (PCB = Printed Circuit Board)

Returns a list of the actual controller and driver temperature in degrees celsius.

10.23.3.34 char * cSDHSerial::ver (void) throw (cSDHLibraryException*)

Return version of SDH firmware

Attention:

The string returned is stored internally in this object and might be overwritten by the next command to this object

10.23.3.35 char * cSDHSerial::ver_date (void) throw (cSDHLibraryException*)

Return date of SDH firmware

Attention:

The string returned is stored internally in this object and might be overwritten by the next command to this object

10.23.3.36 char * cSDHSerial::id (void) throw (cSDHLibraryException*)

Return id of SDH

Attention:

The string returned is stored internally in this object and might be overwritten by the next command to this object

10.23.3.37 char * cSDHSerial::sn (void) throw (cSDHLibraryException*)

Return sn of SDH

Attention:

The string returned is stored internally in this object and might be overwritten by the next command to this object

10.23.3.38 char * cSDHSerial::soc (void) throw (cSDHLibraryException*)

Return soc (System On Chip) ID of SDH

Attention:

The string returned is stored internally in this object and might be overwritten by the next command to this object

10.23.3.39 char * cSDHSerial::soc_date (void) throw (cSDHLibraryException*)

Return date of soc (System On Chip) ID of SDH

Attention:

The string returned is stored internally in this object and might be overwritten by the next command to this object

10.23.3.40 int cSDHSerial::numaxis (void) throw (cSDHLibraryException*)

Return number of axis of SDH

10.23.3.41 cSimpleVector cSDHSerial::igrip (int *axis* = `All`, double * *limit* = `NULL`) throw (cSDHLibraryException*)

Get/Set motor current limits for grip commands

- If axis is All and limit is None then a NUMBER_OF_AXES-list of the actually set motor current limits is returned
- If axis is a single number and limit is None then the motor current limit for that axis is returned.
- If axis and limit are single numbers then the motor current limit for that axis is set (and returned).
- If axis is All and limit is a NUMBER_OF_AXES-vector then all axes motor current limits are set accordingly, the NUMBER_OF_AXES-list is returned.

10.23.3.42 cSimpleVector cSDHSerial::ihold (int *axis* = `All`, double * *limit* = `NULL`) throw (cSDHLibraryException*)

Get/Set motor current limits for hold commands

- If axis is All and limit is None then a NUMBER_OF_AXES-list of the actually set motor current limits is returned
- If axis is a single number and limit is None then the motor current limit for that axis is returned.
- If axis and limit are single numbers then the motor current limit for that axis is set (and returned).
- If axis is All and limit is a NUMBER_OF_AXES-vector then all axes motor current limits are set accordingly, the NUMBER_OF_AXES-list is returned.

10.23.3.43 double cSDHSerial::selgrip (eGraspId *grip*, bool *sequ*) throw (cSDHLibraryException*)

Send "selgrip grip" command to SDH. Where grip is in [0..eGID_DIMENSION-1] or one of the eGraspId enums.

If sequ is True then wait until SDH hardware fully executed the command. Else return immediately and do not wait until SDH hardware fully executed the command.

This seems to work with sin square velocity profile only, so the velocity profile is switched to that if necessary.

return the expected duration of the execution of the command in seconds

10.23.3.44 double cSDHSerial::grip (double *close*, double *velocity*, bool *sequ*) throw (cSDHLibraryException*)

send "grip=close,velocity" command to SDH close : [0.0 .. 1.0] where 0.0 is 'fully opened' and 1.0 is 'fully closed' velocity :]0.0 .. 100.0] where 0.0 (not allowed) is very slow and 100.0 is very fast

If sequ is True then wait until SDH hardware fully executed the command. Else return immediately and do not wait until SDH hardware fully executed the command.

This seems to work with sin square velocity profile only, so the velocity profile is switched to that if necessary.

return the expected duration of the execution of the command in seconds

10.23.4 Member Data Documentation

10.23.4.1 double SDH::cSDHSerial::m_sequtime [protected]

additional time in seconds to wait for sequential execution of m command (as these are always executed non-sequentially by the firmware

10.23.4.2 char* SDH::cSDHSerial::EOL [protected]

String to use as "End Of Line" marker when sending to SDH.

10.23.4.3 cSerialBase* SDH::cSDHSerial::com [protected]

The communication object to the serial device (RS232 port or ESD CAN net).

10.23.4.4 cSimpleStringList SDH::cSDHSerial::reply [protected]

Space for the replies from the SDH.

10.23.4.5 int SDH::cSDHSerial::nb_lines_to_ignore [protected]

number of remaining reply lines of a previous (non-sequential) command

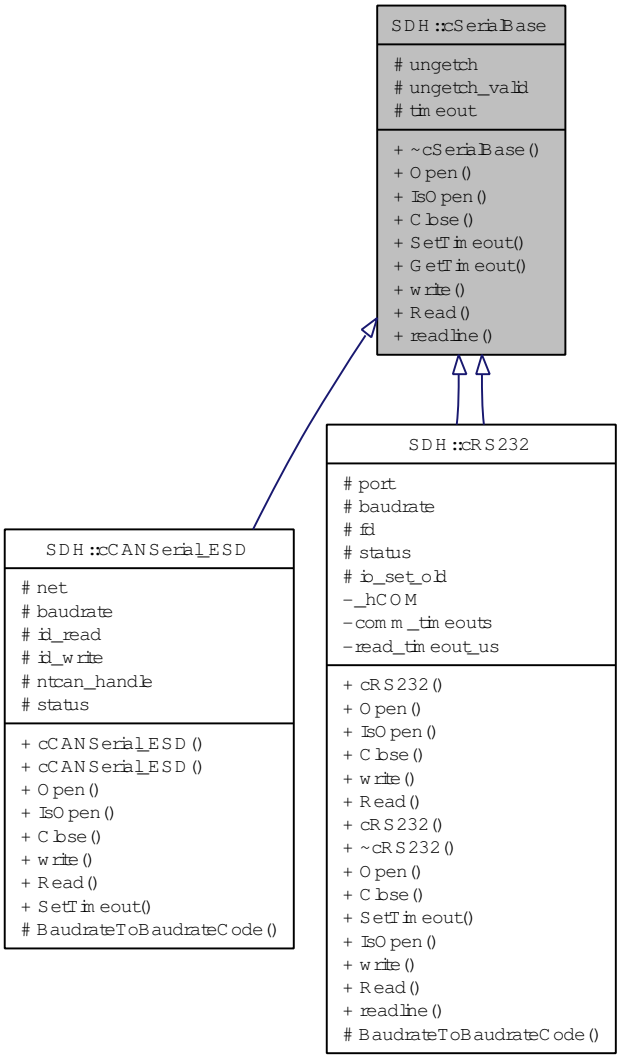
The documentation for this class was generated from the following files:

- sdh/sdhserial.h
- sdh/sdhserial.cpp

10.24 SDH::cSerialBase Class Reference

`#include <serialbase.h>`

Inheritance diagram for SDH::cSerialBase:



10.24.1 Detailed Description

Low-level communication class to access a serial port.

(This is an abstract base class with pure virtual functions)

Public Member Functions

- virtual ~cSerialBase (void)
- virtual void Open (void)=0 throw (cSerialBaseException*)

  Open rs232 port port.

- virtual bool IsOpen (void)=0 throw ()

  Return true if communication channel is open.

- virtual void Close (void)=0 throw (cSerialBaseException∗)

  Close the previously opened communication channel.

- virtual void SetTimeout (double _timeout) throw (cSerialBaseException∗)

  set the timeout for next readline() calls (negative value means: no timeout, wait for ever)

- virtual double GetTimeout ()

  set the timeout for next readline() calls (negative value means: no timeout, wait for ever)

- virtual int write (char const ∗ptr, int len=0)=0 throw (cSerialBaseException∗)

  Write data to a previously opened port.

- virtual ssize_t Read (void ∗data, ssize_t size, long timeout_us, bool return_on_less_data)=0 throw (cSerialBaseException∗)
- virtual char ∗ readline (char ∗line, int size, char ∗eol="\n", bool return_on_less_data=false) throw (cSerialBaseException∗)

  Read a line from the device.

Protected Attributes

- char ungetch

  an already read data byte of the next line

- bool ungetch_valid

  Flag, true if ungetch is valid.

- double timeout

  timeout in seconds

10.24.2 Constructor & Destructor Documentation

10.24.2.1 virtual SDH::cSerialBase::∼cSerialBase (void) [inline, virtual]

10.24.3 Member Function Documentation

10.24.3.1 virtual void SDH::cSerialBase::Open (void) throw (cSerialBaseException∗) [pure virtual]

Open rs232 port *port*.

Open the device with the parameters provided in the constructor

Implemented in SDH::cCANSerial_ESD, SDH::cRS232, and SDH::cRS232.

10.24.3.2 virtual bool SDH::cSerialBase::IsOpen (void) throw () [pure virtual]

Return true if communication channel is open.

Implemented in SDH::cCANSerial_ESD, SDH::cRS232, and SDH::cRS232.

10.24.3.3 virtual void SDH::cSerialBase::Close (void) throw (cSerialBaseException*) [pure virtual]

Close the previously opened communication channel.

Implemented in SDH::cCANSerial_ESD, SDH::cRS232, and SDH::cRS232.

10.24.3.4 virtual void SDH::cSerialBase::SetTimeout (double *_timeout*) throw (cSerialBaseException*) [inline, virtual]

set the timeout for next readline() calls (negative value means: no timeout, wait for ever)

Reimplemented in SDH::cCANSerial_ESD, and SDH::cRS232.

10.24.3.5 virtual double SDH::cSerialBase::GetTimeout () [inline, virtual]

set the timeout for next readline() calls (negative value means: no timeout, wait for ever)

10.24.3.6 virtual int SDH::cSerialBase::write (char const * *ptr*, int *len* = 0) throw (cSerialBaseException*) [pure virtual]

Write data to a previously opened port.

Write *len* bytes from **ptr* to the rs232 device

Parameters:

ptr - pointer the byte array to send in memory

len - number of bytes to send

Returns:

the number of bytes actually written

Implemented in SDH::cCANSerial_ESD, SDH::cRS232, and SDH::cRS232.

10.24.3.7 virtual ssize_t SDH::cSerialBase::Read (void * *data*, ssize_t *size*, long *timeout_us*, bool *return_on_less_data*) throw (cSerialBaseException*) [pure virtual]

Read data from device. This function waits until *max_time_us* us passed or the expected number of bytes are received via serial line. if (*return_on_less_data* is true (default value), the number of bytes that have been received are returned and the data is stored in *data* If the *return_on_less_data* is false, data is only read from serial line, if at least *size* bytes are available.

Implemented in SDH::cCANSerial_ESD, SDH::cRS232, and SDH::cRS232.

10.24.3.8 char * cSerialBase::readline (char * *line*, int *size*, char * *eol* = "\n", bool *return_on_less_data* = false) throw (cSerialBaseException*) [virtual]

Read a line from the device.

A line is terminated with one of the end-of-line (eol) characters (’

’ by default) or until timeout

Parameters:

line - ptr to where to store the read line

size - space available in line (bytes)

eol - a string containing all the chars that mark an end of line

return_on_less_data - if (*return_on_less_data* is true (default value), the number of bytes that have been received are returned and the data is stored in *data* If the *return_on_less_data* is false, data is only read from serial line, if at least *size* bytes are available.

A pointer to the line read is returned.

Reimplemented in SDH::cRS232.

10.24.4 Member Data Documentation

10.24.4.1 char SDH::cSerialBase::ungetch [protected]

an already read data byte of the next line

10.24.4.2 bool SDH::cSerialBase::ungetch_valid [protected]

Flag, true if ungetch is valid.

10.24.4.3 double SDH::cSerialBase::timeout [protected]

timeout in seconds

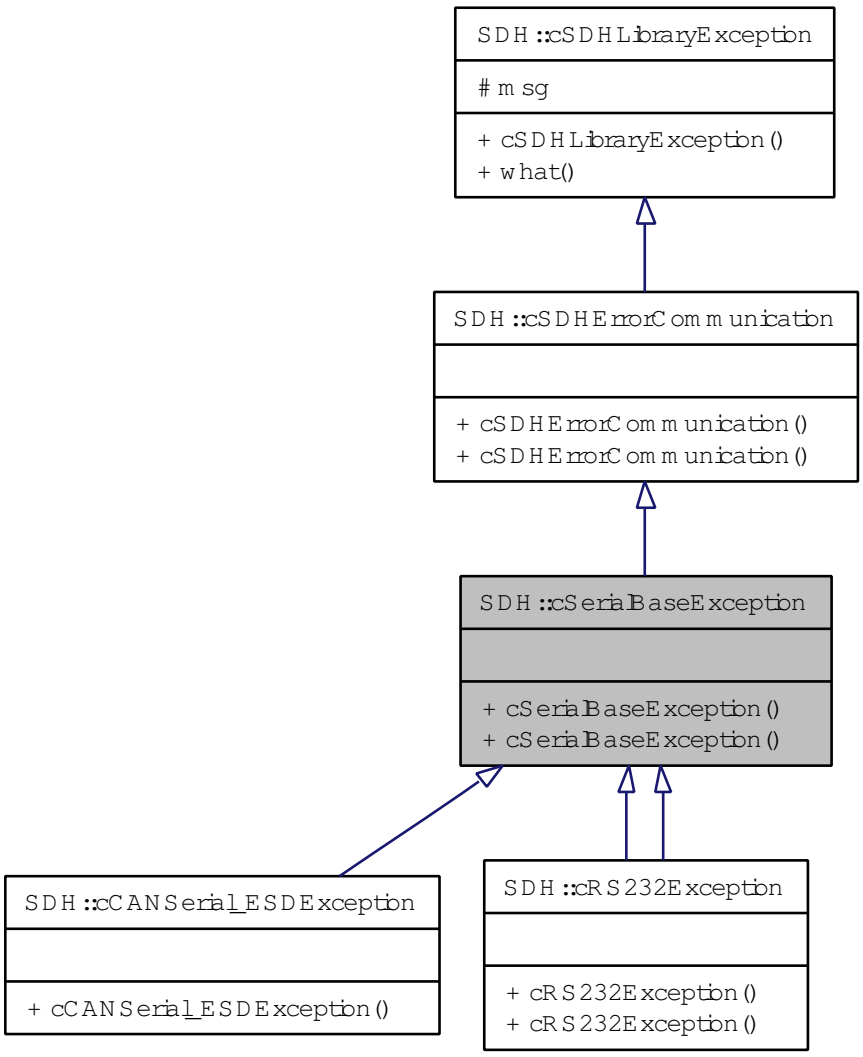
The documentation for this class was generated from the following files:

- sdh/serialbase.h
- sdh/serialbase.cpp

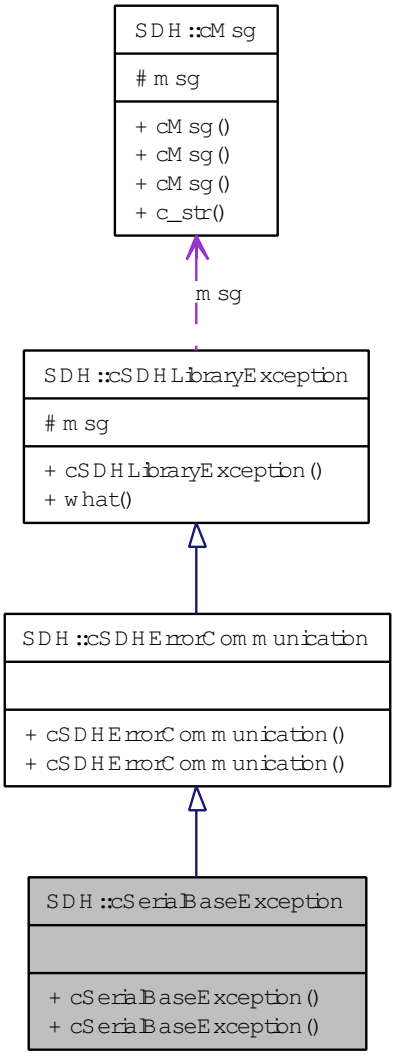
10.25 SDH::cSerialBaseException Class Reference

`#include <serialbase.h>`

Inheritance diagram for SDH::cSerialBaseException:



Collaboration diagram for SDH::cSerialBaseException:



10.25.1 Detailed Description

Derived exception class for low-level serial communication related exceptions.

Public Member Functions

- cSerialBaseException (cMsg const &_msg)
- cSerialBaseException (char const *_type, cMsg const &_msg)

10.25.2 Constructor & Destructor Documentation

10.25.2.1 SDH::cSerialBaseException::cSerialBaseException (cMsg const & *_msg*) [inline]

10.25.2.2 SDH::cSerialBaseException::cSerialBaseException (char const * *_type*, cMsg const & *_msg*) [inline]

The documentation for this class was generated from the following file:

- sdh/serialbase.h

10.26 SDH::cSimpleStringList Class Reference

`#include <simplestringlist.h>`

10.26.1 Detailed Description

A simple string list. (Fixed maximum number of strings of fixed maximum length).

Public Types

- enum { eMAX_LINES = 256, eMAX_CHARS = 256 }

  anonymous enum instead of define macros

Public Member Functions

- cSimpleStringList ()

  Default constructor: init members.

- char * CurrentLine ()

  Return the current line.

- char * NextLine ()

  Return the next line, this increases current_line.

- int Length () const

  Return number of lines stored.

- char * operator[ ] (int index)

  return ptr to line with index.

- void Reset ()

  reset list

Public Attributes

- int current_line

  the index of the current line. For empty cSimpleStringLists this is -1.

Protected Attributes

- char line [eMAX_LINES][eMAX_CHARS]

  a fixed length array of lines with fixed length

10.26.2 Member Enumeration Documentation

10.26.2.1 anonymous enum

anonymous enum instead of define macros

Enumerator:

eMAX_LINES

eMAX_CHARS

10.26.3 Constructor & Destructor Documentation

10.26.3.1 cSimpleStringList::cSimpleStringList ()

Default constructor: init members.

10.26.4 Member Function Documentation

10.26.4.1 char * cSimpleStringList::CurrentLine ()

Return the current line.

10.26.4.2 char * cSimpleStringList::NextLine ()

Return the next line, this increases current_line.

10.26.4.3 int cSimpleStringList::Length () const

Return number of lines stored.

10.26.4.4 char * cSimpleStringList::operator[ ] (int *index*)

return ptr to line with index.
if index < 0 then the numbering starts from the end, thus [-1] gives the last line, [-2] the next to last, ...

10.26.4.5 void cSimpleStringList::Reset ()

reset list

10.26.5 Member Data Documentation

10.26.5.1 int SDH::cSimpleStringList::current_line

the index of the current line. For empty cSimpleStringLists this is -1.

10.26.5.2 char SDH::cSimpleStringList::line[eMAX_LINES][eMAX_CHARS] [protected]

a fixed length array of lines with fixed length

The documentation for this class was generated from the following files:

- sdh/simplestringlist.h
- sdh/simplestringlist.cpp

10.27 SDH::cSimpleTime Class Reference

`#include <simpletime.h>`

10.27.1 Detailed Description

Very simple class to measure elapsed time.

Public Member Functions

- cSimpleTime ()

  Constructor: store current time ("now") internally.

- void StoreNow (void)

  Store current time internally.

- double Elapsed (void) const

  Return time in seconds elapsed between the time stored in the object and now.

- long Elapsed_us (void) const

  Return time in micro seconds elapsed between the time stored in the object and now.

- double Elapsed (cSimpleTime const &other) const

  Return time in seconds elapsed between the time stored in the object and other.

- long Elapsed_us (cSimpleTime const &other) const

  Return time in micro seconds elapsed between the time stored in the object and other.

- timeval Timeval (void)

  Return the time stored as C struct timeval.

Protected Attributes

- struct timeval a_time

10.27.2 Constructor & Destructor Documentation

10.27.2.1 SDH::cSimpleTime::cSimpleTime () `[inline]`

Constructor: store current time ("now") internally.

10.27.3 Member Function Documentation

10.27.3.1 void SDH::cSimpleTime::StoreNow (void) `[inline]`

Store current time internally.

10.27.3.2 double SDH::cSimpleTime::Elapsed (void) const [inline]

Return time in seconds elapsed between the time stored in the object and now.

10.27.3.3 long SDH::cSimpleTime::Elapsed_us (void) const [inline]

Return time in micro seconds elapsed between the time stored in the object and now.

10.27.3.4 double SDH::cSimpleTime::Elapsed (cSimpleTime const & *other*) const [inline]

Return time in seconds elapsed between the time stored in the object and *other*.

10.27.3.5 long SDH::cSimpleTime::Elapsed_us (cSimpleTime const & *other*) const [inline]

Return time in micro seconds elapsed between the time stored in the object and *other*.

10.27.3.6 timeval SDH::cSimpleTime::Timeval (void) [inline]

Return the time stored as C struct timeval.

10.27.4 Member Data Documentation

10.27.4.1 struct timeval SDH::cSimpleTime::a_time [read, protected]

The documentation for this class was generated from the following file:

- sdh/simpletime.h

10.28 SDH::cSimpleVector Class Reference

`#include <simplevector.h>`

10.28.1 Detailed Description

A simple vector implementation.

Objects of this class are used to return vector like answers from the SDH firmware to the cSDHBase class. End users need not use this class, as cSDH, the real end user interface class provides a more convenient way using STL vectors.

Public Types

- enum { eNUMBER_OF_ELEMENTS = 7 }

  anonymous enum (instead of define like macros)

Public Member Functions

- cSimpleVector () throw (cSimpleVectorException∗)

  Default constructor: init members to zero.

- cSimpleVector (int _nb_values, char const ∗str) throw (cSimpleVectorException∗)

  Constructor: init members from _nb_values *comma separated values in the give string* str.

- cSimpleVector (int _nb_values, int start_index, char const ∗str) throw (cSimpleVectorException∗)

  Constructor: init members from _nb_values *comma separated values in the give string* str.

- void FromString (int nb_values, int start_index, char const ∗str) throw (cSimpleVectorException∗)

  init _nb_values *starting from index* start_index *from comma separated values in* str

- double & operator[ ] (unsigned int index)

  index operator, return a reference to the index-th *element of this*

- double & x (void)

  Interpret object as x/y/z vector: return x = the first element, if that is valid.

- double & y (void)

  Interpret object as x/y/z vector: return x = the first element, if that is valid.

- double & z (void)

  Interpret object as x/y/z vector: return x = the first element, if that is valid.

- bool Valid (unsigned int index) const

  Return true if vector element index *is valid (has been accessed at least once).*

Protected Attributes

- double value [eNUMBER_OF_ELEMENTS]
- int valid

  bit mask which values in value are valid

10.28.2 Member Enumeration Documentation

10.28.2.1 anonymous enum

anonymous enum (instead of define like macros)

Enumerator:

eNUMBER_OF_ELEMENTS number of elements in vector

10.28.3 Constructor & Destructor Documentation

10.28.3.1 cSimpleVector::cSimpleVector () throw (cSimpleVectorException*)

Default constructor: init members to zero.

10.28.3.2 cSimpleVector::cSimpleVector (int *_nb_values*, char const * *str*) throw (cSimpleVectorException*)

Constructor: init members from *_nb_values* comma separated values in the give string *str*.

10.28.3.3 cSimpleVector::cSimpleVector (int *_nb_values*, int *start_index*, char const * *str*) throw (cSimpleVectorException*)

Constructor: init members from *_nb_values* comma separated values in the give string *str*.

10.28.4 Member Function Documentation

10.28.4.1 void cSimpleVector::FromString (int *nb_values*, int *start_index*, char const * *str*) throw (cSimpleVectorException*)

init *_nb_values* starting from index *start_index* from comma separated values in *str*

10.28.4.2 double & cSimpleVector::operator[ ] (unsigned int *index*)

index operator, return a reference to the *index-th* element of this

10.28.4.3 double & cSimpleVector::x (void)

Interpret object as x/y/z vector: return x = the first element, if that is valid.

10.28.4.4 double & cSimpleVector::y (void)

Interpret object as x/y/z vector: return x = the first element, if that is valid.

10.28.4.5 double & cSimpleVector::z (void)

Interpret object as x/y/z vector: return x = the first element, if that is valid.

10.28.4.6 bool cSimpleVector::Valid (unsigned int *index*) const

Return true if vector element *index* is valid (has been accessed at least once).

10.28.5 Member Data Documentation

10.28.5.1 double SDH::cSimpleVector::value[eNUMBER_OF_ELEMENTS] [protected]

10.28.5.2 int SDH::cSimpleVector::valid [protected]

bit mask which values in value are valid

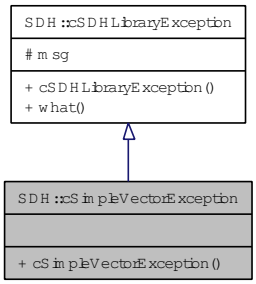
The documentation for this class was generated from the following files:

- sdh/simplevector.h
- sdh/simplevector.cpp

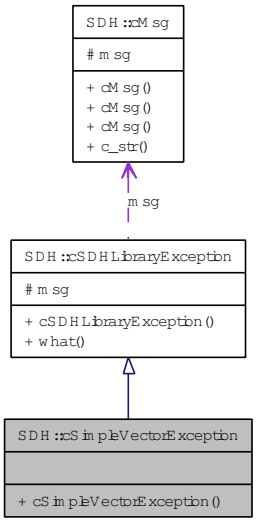
10.29 SDH::cSimpleVectorException Class Reference

`#include <simplevector.h>`

Inheritance diagram for SDH::cSimpleVectorException:



Collaboration diagram for SDH::cSimpleVectorException:



10.29.1 Detailed Description

Derived exception class for low-level simple vector related exceptions.

Public Member Functions

- cSimpleVectorException (cMsg const &_msg)

10.29.2 Constructor & Destructor Documentation

10.29.2.1 SDH::cSimpleVectorException::cSimpleVectorException (cMsg const & *_msg*) [inline]

The documentation for this class was generated from the following file:

- sdh/simplevector.h

10.30 SDH::cUnitConverter Class Reference

`#include <unit_converter.h>`

10.30.1 Detailed Description

Unit conversion class to convert values between physical unit systems.

An object of this class can be configured to convert values of a physical unit between 2 physical unit systems. An angle value given in degrees can e.g. be converted from/to radians or vice versa by an object of this class.

Public Member Functions

- cUnitConverter (char *_kind, char *_name, char *_symbol, double _factor=1.0, double _offset=0.0, int _decimal_places=1)
- double ToExternal (double internal) const
- cSimpleVector ToExternal (cSimpleVector &internal) const
- std::vector< double > ToExternal (std::vector< double > const &internal) const
- double ToInternal (double external) const
- cSimpleVector ToInternal (cSimpleVector &external) const
- std::vector< double > ToInternal (std::vector< double > const &external) const
- char const * GetKind (void) const

  Return the kind of unit converted (something like "angle" or "time").

- char const * GetName (void) const

  Return the name of the external unit (something like "degrees" or "milliseconds").

- char const * GetSymbol (void) const

  Return the symbol of the external unit (something like "deg" or "ms").

- double GetFactor (void) const

  Return the conversion factor from internal to external units.

- double GetOffset (void) const

  Return the conversion offset from internal to external units.

- int GetDecimalPlaces (void) const

  Return the number of decimal places for printing values in the external unit system.

Protected Attributes

- char * kind

  the kind of unit to be converted (something like "angle" or "time")

- char * name

  the name of the external unit (something like "degrees" or "milliseconds")

- char * symbol

  the symbol of the external unit (something like "deg" or "ms")

- double factor

  the conversion factor from internal to external units

- double offset

  the conversion offset from internal to external units

- int decimal_places

  A usefull number of decimal places for printing values in the external unit system.

10.30.2 Constructor & Destructor Documentation

10.30.2.1 cUnitConverter::cUnitConverter (char * *_kind*, char * *_name*, char * *_symbol*, double *_factor* = 1.0, double *_offset* = 0.0, int *_decimal_places* = 1)

Constructor of cUnitConverter class.

At construction time the conversion parameters - a *factor* and an *offset* - must be provided along with elements that describe the unit of a value

Parameters:

_kind - a string describing the kind of unit to be converted (something like "angle" or "time")

_name - the name of the external unit (something like "degrees" or "milliseconds")

_symbol - the symbol of the external unit (something like "deg" or "ms")

_factor - the conversion factor from internal to external units

_offset - the conversion offset from internal to external units

_decimal_places - A usefull number of decimal places for printing values in the external unit system

Attention:

The strings given _kind, _name and _symbol are **NOT** copied, just their address is stored

10.30.3 Member Function Documentation

10.30.3.1 double cUnitConverter::ToExternal (double *internal*) const

Convert single value *internal* given in internal units into external units. Returns *internal* * factor + offset

10.30.3.2 cSimpleVector cUnitConverter::ToExternal (cSimpleVector & *internal*) const

Convert values in simple array *internal*, each given in internal units into external units. Returns a simple array with each valid element converted with *internal*[i] * factor + offset

Only valid entries of the *internal* vector are converted.

10.30.3.3 std::vector< double > cUnitConverter::ToExternal (std::vector< double > const & *internal*) const

Convert values in vector *internal*, each given in internal units into external units. Returns a vector with each element converted with *internal*[i] ∗ factor + offset

10.30.3.4 double cUnitConverter::ToInternal (double *external*) const

Convert value *external* given in external units into internal units. Returns (*external* - offset) / factor

10.30.3.5 cSimpleVector cUnitConverter::ToInternal (cSimpleVector & *external*) const

Convert values in simple array *external*, each given in external units into internal units. Returns a simple array with each valid element converted with *external*[i] ∗ factor + offset

Only valid entries of the *external* vector are converted.

10.30.3.6 std::vector< double > cUnitConverter::ToInternal (std::vector< double > const & *external*) const

Convert values in vector *external*, each given in external units into internal units. Returns a vector with each valid element converted with *external*[i] ∗ factor + offset

10.30.3.7 char const ∗ cUnitConverter::GetKind (void) const

Return the kind of unit converted (something like "angle" or "time").

10.30.3.8 char const ∗ cUnitConverter::GetName (void) const

Return the name of the external unit (something like "degrees" or "milliseconds").

10.30.3.9 char const ∗ cUnitConverter::GetSymbol (void) const

Return the symbol of the external unit (something like "deg" or "ms").

10.30.3.10 double cUnitConverter::GetFactor (void) const

Return the conversion factor from internal to external units.

10.30.3.11 double cUnitConverter::GetOffset (void) const

Return the conversion offset from internal to external units.

10.30.3.12 int cUnitConverter::GetDecimalPlaces (void) const

Return the number of decimal places for printing values in the external unit system.

10.30.4 Member Data Documentation

10.30.4.1 char* SDH::cUnitConverter::kind [protected]

the kind of unit to be converted (something like "angle" or "time")

10.30.4.2 char* SDH::cUnitConverter::name [protected]

the name of the external unit (something like "degrees" or "milliseconds")

10.30.4.3 char* SDH::cUnitConverter::symbol [protected]

the symbol of the external unit (something like "deg" or "ms")

10.30.4.4 double SDH::cUnitConverter::factor [protected]

the conversion factor from internal to external units

10.30.4.5 double SDH::cUnitConverter::offset [protected]

the conversion offset from internal to external units

10.30.4.6 int SDH::cUnitConverter::decimal_places [protected]

A usefull number of decimal places for printing values in the external unit system.

The documentation for this class was generated from the following files:

- sdh/unit_converter.h
- sdh/unit_converter.cpp

10.31 option Struct Reference

`#include <getopt.h>`

Public Attributes

- char * name
- int has_arg
- int * flag
- int val

10.31.1 Member Data Documentation

10.31.1.1 char* option::name

10.31.1.2 int option::has_arg

10.31.1.3 int* option::flag

10.31.1.4 int option::val

The documentation for this struct was generated from the following file:

- vcc/getopt.h

Chapter 11

File Documentation

11.1 architecture.dox File Reference

11.1.1 Detailed Description

Short overview of the SDHLibrary-CPP and SDH architecture.

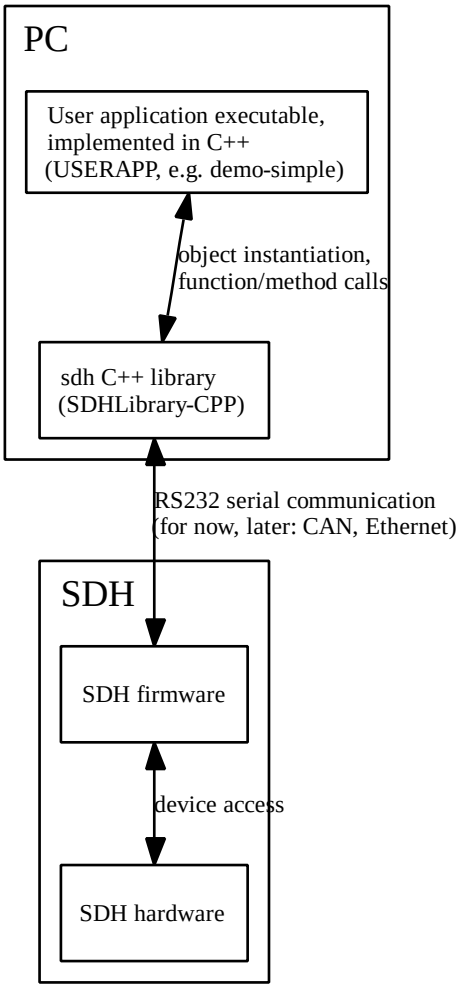
11.1.2 Overview

Naming convention:

As a convention **"SDH"** (capital letters) is used to refer to the physical device, the three fingered SCHUNK Dexterous Hand, while **"sdh"** (small letters) refers to the PC-software that communicates with the physical SDH device. Within the "sdh" PC-software further entities can be distinguished: The C++ library **SDHLibrary-CPP.so** (on Linux) or **SDHLibrary-CPP.dll** (on Windows/cygwin) that contains the complete sdh library including the user interface class **SDH::cSDH**. This SDH::cSDH class will be described in detail below.

Basic structure:

The basic structure of the components looks like this:



Basic architecture:

There are several classes defined in SDHLibrary-CPP:

- SDH::cSDH the class used to communicate with the SDH. This class provides the functional interface of the SDH. It should be used by end users, as its interface is considered more stable.
- Other classes, like cSDHBase and cSDHSerial, are used by SDH::cSDH and provide more low level services and should **NOT** be used directly, as their interfaces are subject to change in future releases.
- cSDHLibraryException and derivatives: these are used when an exception is raised

Example use:

An exemplary use of the sdh module in a user application in C++ might look like this:

```
...
// Include the cSDH interface
include <sdh.h>

// Create an instance "hand" of the class cSDH:
cSDH hand;
```

```
// Open communication to the SDH device via default serial port 0 == "COM1"
hand.OpenRS232();

// Perform some action:
//   get the current actual axis angles of finger 0
std::vector<double> faa = hand.GetFingerActualAngle( 0 );

//   modify these by decreasing the proximal and the distal axis angles:
std::vector<double> fta = faa;
fta[1] -= 10;
fta[2] -= 10;

//   set modified angles as new target angles:
hand.SetFingerTargetAngle( 0, fta );

//   now make the finger move there:
hand.MoveFinger( 0 );

// Finally close connection to SDH again (This automatically
// switches off the axis controllers to prevent overheating):
hand.Close();
```

Real example code is available in the demo-∗.cpp code files, see e.g.

- demo-simple.cpp
- demo-temperature.cpp
- demo-GetAxisActualAngle.cpp

11.2 connectors.dox File Reference

11.2.1 Detailed Description

11.2.2 General project information

Author:

Dirk Osswald

Date:

2007-02-19

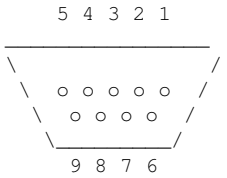
11.2.3 Purpose

This page discribes the connectors of the SDH / LWA

SDH

RS232 Communication connector cable of the SDH at the base of the LWA:

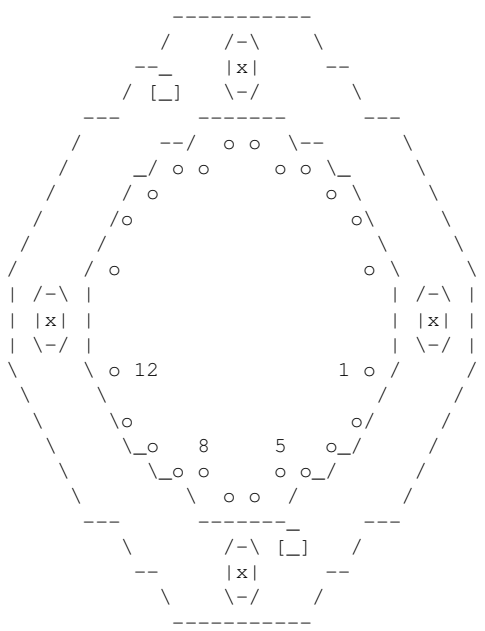
- 9pin Sub-D female connector (View from connecting side):



- 1 : here: nc, (normally: Data Carrier Detect)
- 2 : RxD of PC = TxD of SDH
- 3 : TxD 0f PC = RxD of SDH
- 4 : here: nc, (normally: Data Terminal Ready)
- 5 : GND
- 6 : here: nc, (normally: Data Set Ready)
- 7 : here: nc, (normally: Request To Send)
- 8 : here: nc, (normally: Clear To Send)
- 9 : here: nc, (Ring Indicator)

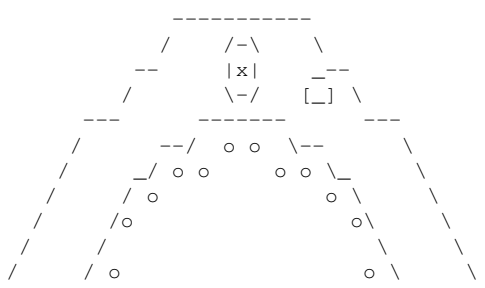
- RS232 + Power Communication connector:

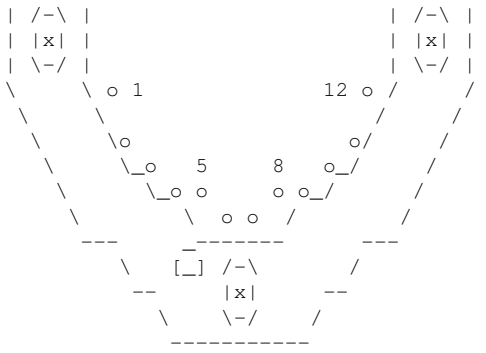
- 2x12pin FWS connector, view from connecting side of FWS part mounted on LWA: only one half is used for the SDH



- 1 : Pwr (RD) 24V supply for power and logic of SDH2
- 2 : TxD (BN) RS232 TxD of PC = RxD of SDH2
- 3 : RxD (WH) RS232 RxD of PC = TxD of SDH2
- 4 : TxD2 (RD) RS232 TxD of PC = RxD of SDH2, second RS232 channel of SDH2
- 5 : RxD2 (PK) RS232 RxD of PC = TxD of SDH2, second RS232 channel of SDH2
- 6 : CAN_H (GN) CAN High
- 7 : CAN_L (YE) CAN Low
- 8 : ENET TPI+ Ethernet
- 9 : ENET TPI- Ethernet
- 10 : ENET TPO+ Ethernet
- 11 : ENET TPO- Ethernet
- 12 : GND (BK) Ground

- 2x12pin FWS connector view from connecting side of FWS part mounted on SDH: only one half is used by the SDH



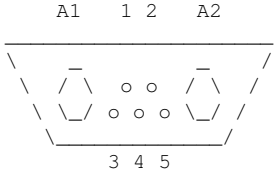


- 1 : Pwr (RD) 24V supply for power and logic of SDH2
- 2 : TxD (BN) RS232 TxD of PC = RxD of SDH2
- 3 : RxD (WH) RS232 RxD of PC = TxD of SDH2
- 4 : TxD2 (RD) RS232 TxD of PC = RxD of SDH2, second RS232 channel of SDH2
- 5 : RxD2 (PK) RS232 RxD of PC = TxD of SDH2, second RS232 channel of SDH2
- 6 : CAN_H (GN) CAN High
- 7 : CAN_L (YE) CAN Low
- 8 : ENET TPI+ Ethernet
- 9 : ENET TPI- Ethernet
- 10 : ENET TPO+ Ethernet
- 11 : ENET TPO- Ethernet
- 12 : GND (BK) Ground

LWA

Combined power / CAN connector of the LWA:

- Male connector (View from connecting side):



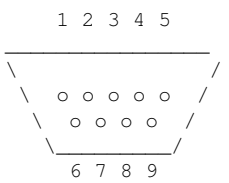
- A1 : +24V
- A2 : 0V (blue)
- 1 : CAN-High (Profibus-A1) (green)
- 2 : RxD

- 3 : CAN-Low (profibus-B1) (yellow)
- 4 : Shield
- 5 : TxD

ESD CAN-USB mini

On the PC-side CAN interface cards are often connected as follows:

- 9pin Sub-D Male connector (View from connecting side):



- 1 : reserved
- 2 : CAN_L (yellow)
- 3 : CAN_GND (blue)
- 4 : reserved
- 5 : Shield
- 6 : (CAN_GND)
- 7 : CAN_H (green)
- 8 : reserved
- 9 : reserved

11.2.4 Copyright

11.3 demo/cancat.cpp File Reference

11.3.1 Detailed Description

Very simple C++ programm to make the SDH move.

11.3.2 General file information

Author:

Dirk Osswald

Date:

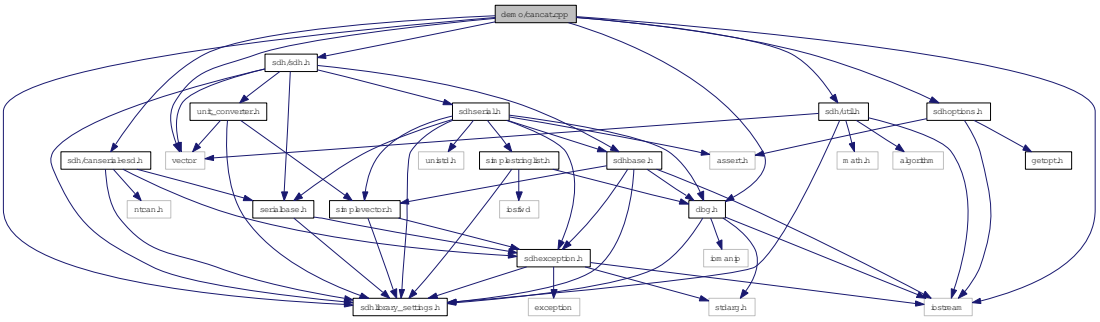
2007-01-18

This code contains only the very basicst use of the features provided by the SDHLibrary-CPP. For more sophisticated applications see the other demo-∗.cpp programms, or of course the html/pdf documentation.

11.3.3 Copyright


```
#include <iostream>
#include <vector>
#include "sdh/sdh.h"
#include "sdh/dbg.h"
#include "sdh/canserial-esd.h"
#include "sdh/util.h"
#include "sdh/sdhlibrary_settings.h"
#include "sdhoptions.h"
```

Include dependency graph for cancat.cpp:



Functions

- int main (int argc, char ∗∗argv)

Variables

- char * usage

Some informative variables

- char const * __help__ = "Send data from command line via ESD CAN and display replies until CTRL-C is pressed."
- char const * __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"
- char const * __url__ = "http://www.schunk.com"
- char const * __version__ = "$Id: cancat.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"
- char const * __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.3.4 Function Documentation

11.3.4.1 int main (int *argc*, char ** *argv*)

11.3.5 Variable Documentation

11.3.5.1 char const* __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"

11.3.5.2 char const* __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.3.5.3 char const* __help__ = "Send data from command line via ESD CAN and display replies until CTRL-C is pressed."

11.3.5.4 char const* __url__ = "http://www.schunk.com"

11.3.5.5 char const* __version__ = "$Id: cancat.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"

11.3.5.6 char* usage

Initial value:

```
"usage: cancat [options]\n"
```

11.4 demo/demo-dsa.cpp File Reference

11.4.1 Detailed Description

Simple program to test class cDSA. See online help ("-h" # or "–help") for available options. # # ##

11.4.2 General file information

# #

Author:

Dirk Osswald #

Date:

2008-06-12 # #

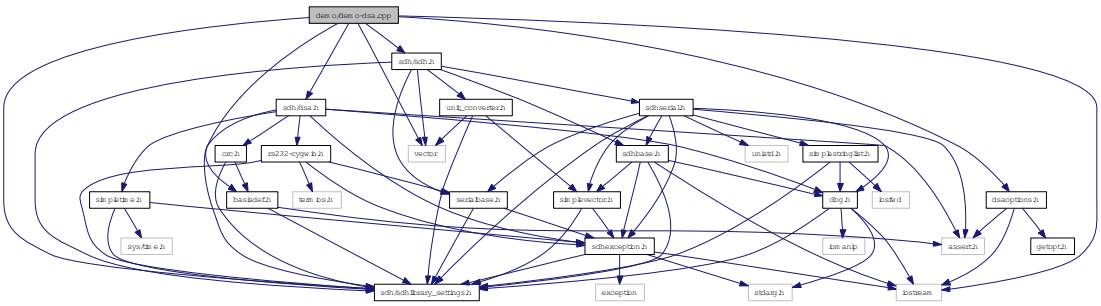
11.4.3 Copyright

- Copyright (c) 2008 SCHUNK GmbH & Co. KG #

#

```
#include "sdh/sdhlibrary_settings.h"
#include <iostream>
#include <vector>
#include "sdh/sdh.h"
#include "sdh/dsa.h"
#include "sdh/basisdef.h"
#include "dsaoptions.h"
```

Include dependency graph for demo-dsa.cpp:



Functions

- int main (int argc, char **argv)

Variables

Some informative variables

Some definitions that describe the demo program

- char const * __help__
- char const * __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"
- char const * __url__ = "http://www.schunk.com"
- char const * __version__ = "$Id: demo-dsa.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"
- char const * __copyright__ = "Copyright (c) 2008 SCHUNK GmbH & Co. KG"
- char const * usage

11.4.4 Function Documentation

11.4.4.1 int main (int *argc*, char ** *argv*)

11.4.5 Variable Documentation

11.4.5.1 char const* __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"

11.4.5.2 char const* __copyright__ = "Copyright (c) 2008 SCHUNK GmbH & Co. KG"

11.4.5.3 char const* __help__

Initial value:

```
"Simple demo to test cDSA class of SDHLibrary-cpp.\n"
"\n"
"Remarks:\n"
"- You must specify at least one of the following options to to see\n"
"  some output: --fullframe, --sensorinfo, --controllerinfo, --matrixinfo\n"
"- Use --framerate=5 --fullframe to print a full frame 5 times per second. \n"
```

11.4.5.4 char const* __url__ = "http://www.schunk.com"

11.4.5.5 char const* __version__ = "$Id: demo-dsa.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"

11.4.5.6 char const* usage

Initial value:

```
"usage: demo-dsa [options]\n"
```

11.5 demo/demo-GetAxisActualAngle.cpp File Reference

11.5.1 Detailed Description

Print measured actual axis angles of SDH. (C++ demo application using the SDHLibrary-CPP library.).

11.5.2 General file information

Author:

Dirk Osswald

Date:

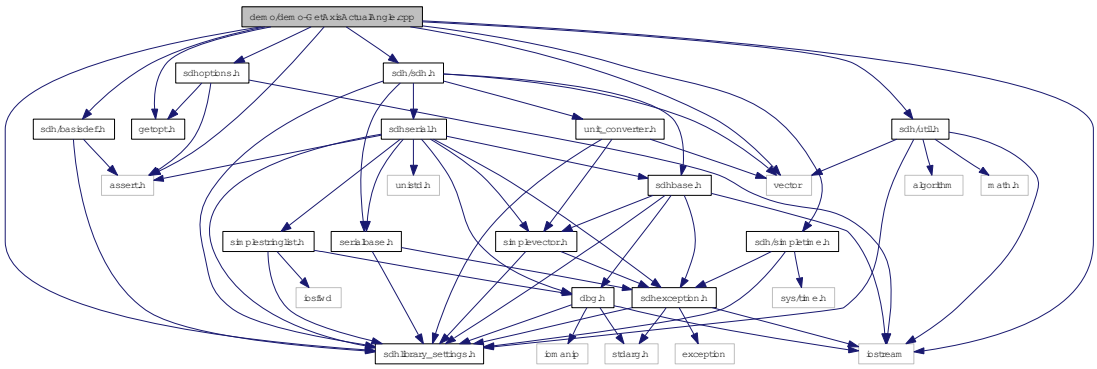
2007-03-07

11.5.3 Copyright

Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include <getopt.h>
#include <assert.h>
#include <iostream>
#include <vector>
#include "sdh/sdh.h"
#include "sdh/simpletime.h"
#include "sdh/util.h"
#include "sdh/sdhlibrary_settings.h"
#include "sdh/basisdef.h"
#include "sdhoptions.h"
```

Include dependency graph for demo-GetAxisActualAngle.cpp:



Functions

- int main (int argc, char **argv)

Variables

- char * usage

Some informative variables

- char const * __help__ = "Print measured actual axis angles of SDH.\n(C++ demo application using the SDHLibrary-CPP library.)"
- char const * __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"
- char const * __url__ = "http://www.schunk.com"
- char const * __version__ = "$Id: demo-GetAxisActualAngle.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"
- char const * __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.5.4 Function Documentation

11.5.4.1 int main (int *argc*, char ** *argv*)

11.5.5 Variable Documentation

11.5.5.1 char const* __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"

11.5.5.2 char const* __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.5.5.3 char const* __help__ = "Print measured actual axis angles of SDH.\n(C++ demo application using the SDHLibrary-CPP library.)"

11.5.5.4 char const* __url__ = "http://www.schunk.com"

11.5.5.5 char const* __version__ = "$Id: demo-GetAxisActualAngle.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"

11.5.5.6 char* usage

Initial value:

```
"usage: demo-GetAxisActualAngle [options]\n"
```

11.6 demo/demo-GetFingerXYZ.cpp File Reference

11.6.1 Detailed Description

Print measured actual axis angles of SDH. (C++ demo application using the SDHLibrary-CPP library.).

11.6.2 General file information

Author:

Dirk Osswald

Date:

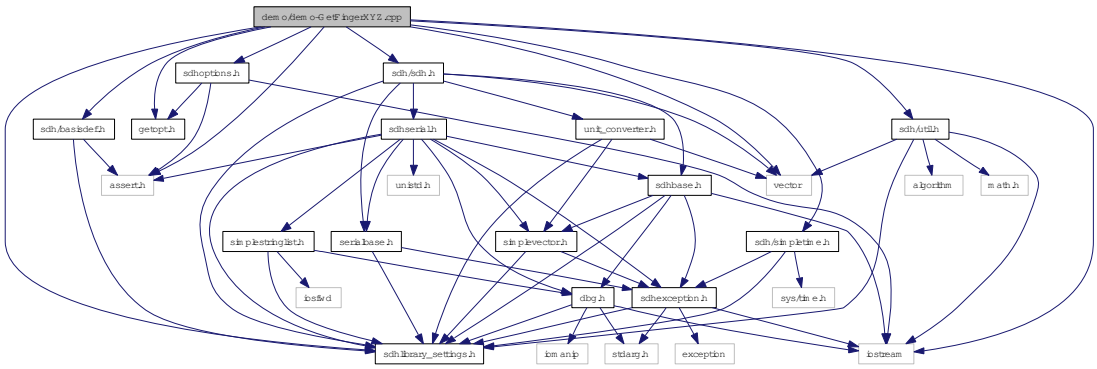
2007-03-07

11.6.3 Copyright

Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include <getopt.h>
#include <assert.h>
#include <iostream>
#include <vector>
#include "sdh/sdh.h"
#include "sdh/simpletime.h"
#include "sdh/util.h"
#include "sdh/sdhlibrary_settings.h"
#include "sdh/basisdef.h"
#include "sdhoptions.h"
```

Include dependency graph for demo-GetFingerXYZ.cpp:



Functions

- int main (int argc, char **argv)

Variables

- char * usage

Some informative variables

- char const * __help__ = "Print measured XYZ position of fingertips of SDH.\n(C++ demo application using the SDHLibrary-CPP library.)"
- char const * __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"
- char const * __url__ = "http://www.schunk.com"
- char const * __version__ = "$Id: demo-GetFingerXYZ.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"
- char const * __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.6.4 Function Documentation

11.6.4.1 int main (int *argc*, char ** *argv*)

11.6.5 Variable Documentation

11.6.5.1 char const* __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"

11.6.5.2 char const* __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.6.5.3 char const* __help__ = "Print measured XYZ position of fingertips of SDH.\n(C++ demo application using the SDHLibrary-CPP library.)"

11.6.5.4 char const* __url__ = "http://www.schunk.com"

11.6.5.5 char const* __version__ = "$Id: demo-GetFingerXYZ.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"

11.6.5.6 char* usage

Initial value:

```
"usage: demo-GetFingerXYZ [options]\n"
```

11.7 demo/demo-ref.cpp File Reference

11.7.1 Detailed Description

Very simple C++ programm to make the SDH move.

11.7.2 General file information

Author:

Dirk Osswald

Date:

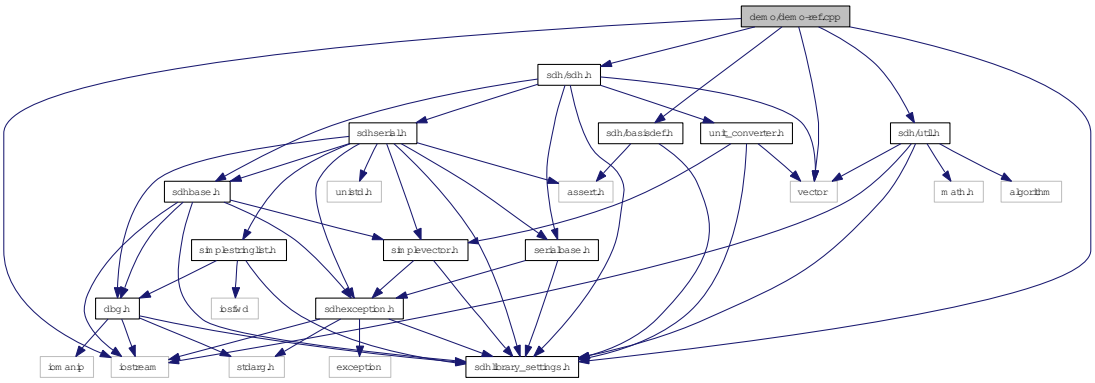
2007-01-18

This code contains only the very basicst use of the features provided by the SDHLibrary-CPP. For more sophisticated applications see the other demo-∗.cpp programms, or of course the html/pdf documentation.

11.7.3 Copyright


```
#include <iostream>
#include <vector>
#include "sdh/sdh.h"
#include "sdh/util.h"
#include "sdh/sdhlibrary_settings.h"
#include "sdh/basisdef.h"
```

Include dependency graph for demo-ref.cpp:



Functions

- int main ()

11.7.4 Function Documentation

11.7.4.1 int main (void)

11.8 demo/demo-simple-withtiming.cpp File Reference

11.8.1 Detailed Description

Very simple C++ programm to make the SDH move.

11.8.2 General file information

Author:

Dirk Osswald

Date:

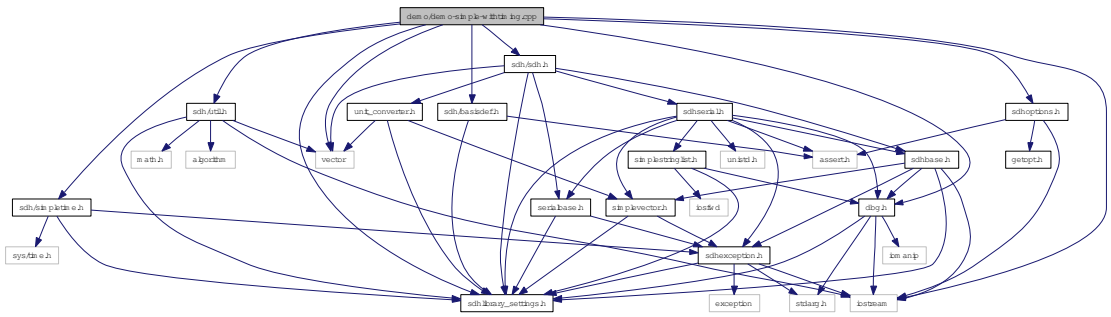
2007-01-18

This code contains only the very basicst use of the features provided by the SDHLibrary-CPP. For more sophisticated applications see the other demo-∗.cpp programms, or of course the html/pdf documentation.

11.8.3 Copyright


```
#include <iostream>
#include <vector>
#include "sdh/sdh.h"
#include "sdh/util.h"
#include "sdh/sdhlibrary_settings.h"
#include "sdh/basisdef.h"
#include "sdhoptions.h"
#include "sdh/simpletime.h"
#include "sdh/dbg.h"
```

Include dependency graph for demo-simple-withtiming.cpp:



Functions

- int main (int argc, char ∗∗argv)

Variables

- char * usage

Some informative variables

- char const * __help__ = "Move proximal and distal joints of finger 1 three times by 10 degrees and measure time for these actions.\n(C++ demo application using the SDHLibrary-CPP library.)"
- char const * __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"
- char const * __url__ = "http://www.schunk.com"
- char const * __version__ = "$Id: demo-simple-withtiming.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"
- char const * __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.8.4 Function Documentation

11.8.4.1 int main (int *argc*, char ** *argv*)

11.8.5 Variable Documentation

11.8.5.1 char const* __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"

11.8.5.2 char const* __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.8.5.3 char const* __help__ = "Move proximal and distal joints of finger 1 three times by 10 degrees and measure time for these actions.\n(C++ demo application using the SDHLibrary-CPP library.)"

11.8.5.4 char const* __url__ = "http://www.schunk.com"

11.8.5.5 char const* __version__ = "$Id: demo-simple-withtiming.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"

11.8.5.6 char* usage

Initial value:

```
"usage: demo-simple-withtiming [options]\n"
```

11.9 demo/demo-simple.cpp File Reference

11.9.1 Detailed Description

Very simple C++ programm to make the SDH move.

11.9.2 General file information

Author:

Dirk Osswald

Date:

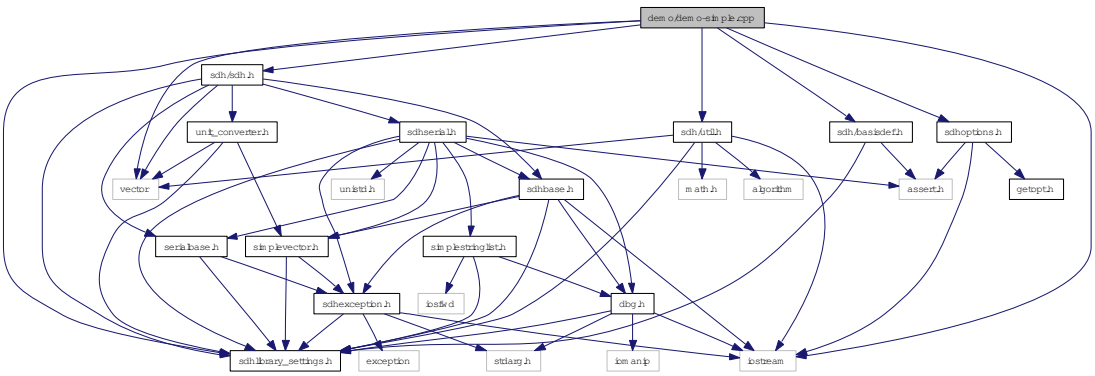
2007-01-18

This code contains only the very basicst use of the features provided by the SDHLibrary-CPP. For more sophisticated applications see the other demo-∗.cpp programms, or of course the html/pdf documentation.

11.9.3 Copyright


```
#include <iostream>
#include <vector>
#include "sdh/sdh.h"
#include "sdh/util.h"
#include "sdh/sdhlibrary_settings.h"
#include "sdh/basisdef.h"
#include "sdhoptions.h"
```

Include dependency graph for demo-simple.cpp:



Functions

- int main (int argc, char ∗∗argv)

Variables

- char * usage

Some informative variables

- char const * __help__ = "Move proximal and distal joints of finger 1 three times by 10 degrees.\n(C++ demo application using the SDHLibrary-CPP library.)"
- char const * __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"
- char const * __url__ = "http://www.schunk.com"
- char const * __version__ = "$Id: demo-simple.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"
- char const * __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.9.4 Function Documentation

11.9.4.1 int main (int *argc*, char ** *argv*)

11.9.5 Variable Documentation

11.9.5.1 char const* __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"

11.9.5.2 char const* __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.9.5.3 char const* __help__ = "Move proximal and distal joints of finger 1 three times by 10 degrees.\n(C++ demo application using the SDHLibrary-CPP library.)"

11.9.5.4 char const* __url__ = "http://www.schunk.com"

11.9.5.5 char const* __version__ = "$Id: demo-simple.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"

11.9.5.6 char* usage

Initial value:

```
"usage: demo-simple [options]\n"
```

11.10 demo/demo-simple2.cpp File Reference

11.10.1 Detailed Description

Very simple C++ programm to make the SDH move. With non-sequential call of move and Stop.

11.10.2 General file information

Author:

Dirk Osswald

Date:

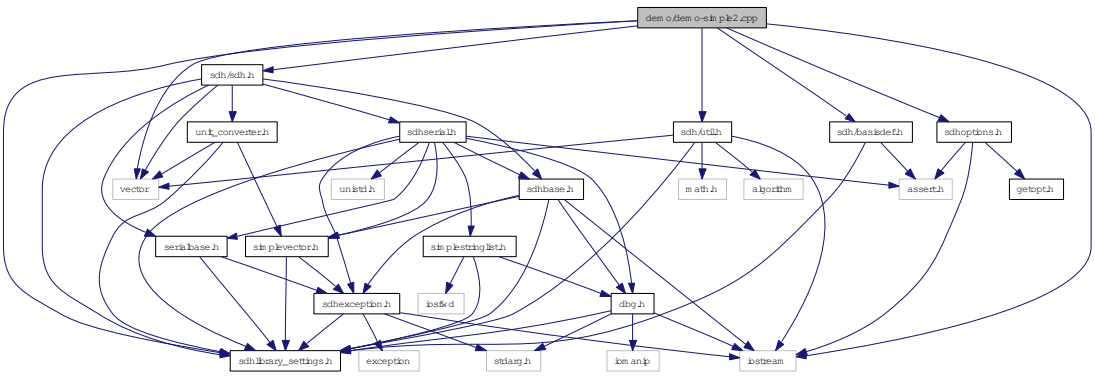
2007-01-18

This code contains only the very basicst use of the features provided by the SDHLibrary-CPP. For more sophisticated applications see the other demo-∗.cpp programms, or of course the html/pdf documentation.

11.10.3 Copyright


```
#include <iostream>
#include <vector>
#include "sdh/sdh.h"
#include "sdh/util.h"
#include "sdh/sdhlibrary_settings.h"
#include "sdh/basisdef.h"
#include "sdhoptions.h"
```

Include dependency graph for demo-simple2.cpp:



Functions

- int main (int argc, char ∗∗argv)

Variables

Some informative variables

- char const * __help__ = "Move proximal and distal joints of finger 1 three times by 10 degrees, stop movement when halfway done.\n(C++ demo application using the SDHLibrary-CPP library.)"
- char const * __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"
- char const * __url__ = "http://www.schunk.com"
- char const * __version__ = "$Id: demo-simple2.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"
- char const * __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.10.4 Function Documentation

11.10.4.1 int main (int *argc*, char ** *argv*)

11.10.5 Variable Documentation

11.10.5.1 char const* __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"

11.10.5.2 char const* __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.10.5.3 char const* __help__ = "Move proximal and distal joints of finger 1 three times by 10 degrees, stop movement when halfway done.\n(C++ demo application using the SDHLibrary-CPP library.)"

11.10.5.4 char const* __url__ = "http://www.schunk.com"

11.10.5.5 char const* __version__ = "$Id: demo-simple2.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"

11.11 demo/demo-simple3.cpp File Reference

11.11.1 Detailed Description

Very simple C++ programm to make the SDH move. With non-sequential call of move and WaitAxis.

11.11.2 General file information

Author:

Dirk Osswald

Date:

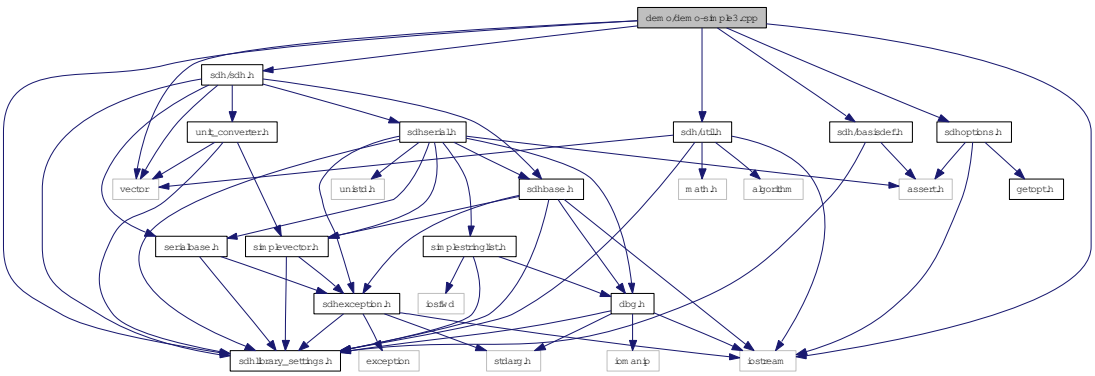
2007-01-18

This code contains only the very basicst use of the features provided by the SDHLibrary-CPP. For more sophisticated applications see the other demo-∗.cpp programms, or of course the html/pdf documentation.

11.11.3 Copyright


```
#include <iostream>
#include <vector>
#include "sdh/sdh.h"
#include "sdh/util.h"
#include "sdh/sdhlibrary_settings.h"
#include "sdh/basisdef.h"
#include "sdhoptions.h"
```

Include dependency graph for demo-simple3.cpp:



Functions

- int main (int argc, char ∗∗argv)

Variables

Some informative variables

- char const * __help__ = "Move axes 1,2 and 3 to a specific point.\n(C++ demo application using the SDHLibrary-CPP library.)"
- char const * __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"
- char const * __url__ = "http://www.schunk.com"
- char const * __version__ = "$Id: demo-simple3.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"
- char const * __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.11.4 Function Documentation

11.11.4.1 int main (int *argc*, char ** *argv*)

11.11.5 Variable Documentation

11.11.5.1 char const* __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"

11.11.5.2 char const* __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.11.5.3 char const* __help__ = "Move axes 1,2 and 3 to a specific point.\n(C++ demo application using the SDHLibrary-CPP library.)"

11.11.5.4 char const* __url__ = "http://www.schunk.com"

11.11.5.5 char const* __version__ = "$Id: demo-simple3.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"

11.12 demo/demo-temperature.cpp File Reference

11.12.1 Detailed Description

Print measured temperatures of SDH. (C++ demo application using the SDHLibrary-CPP library.).

11.12.2 General file information

Author:

Dirk Osswald

Date:

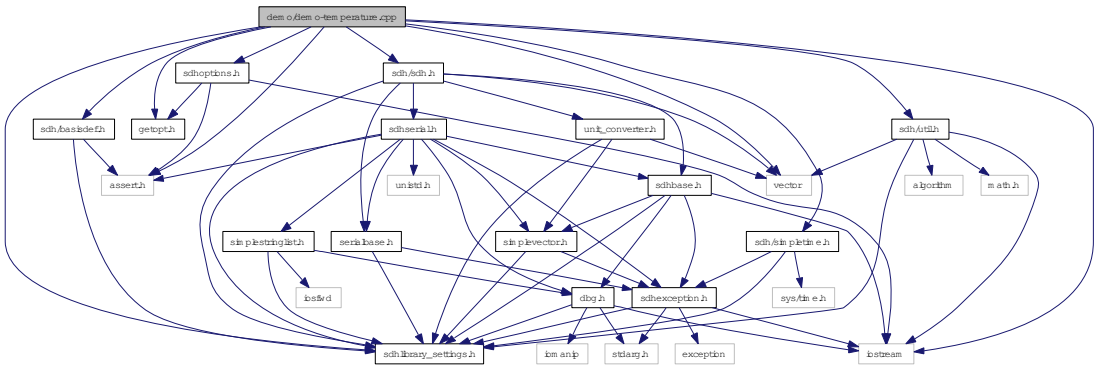
2007-01-18

11.12.3 Copyright

Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include <getopt.h>
#include <assert.h>
#include <iostream>
#include <vector>
#include "sdh/sdh.h"
#include "sdh/simpletime.h"
#include "sdh/util.h"
#include "sdh/sdhlibrary_settings.h"
#include "sdh/basisdef.h"
#include "sdhoptions.h"
```

Include dependency graph for demo-temperature.cpp:



Functions

- int main (int argc, char **argv)

Variables

Some informative variables

- char const * __help__ = "Print measured temperatures of SDH.\n(C++ demo application using the SDHLibrary-CPP library.)"
- char const * __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"
- char const * __url__ = "http://www.schunk.com"
- char const * __version__ = "$Id: demo-temperature.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"
- char const * __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.12.4 Function Documentation

11.12.4.1 int main (int *argc*, char ** *argv*)

11.12.5 Variable Documentation

11.12.5.1 char const* __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"

11.12.5.2 char const* __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.12.5.3 char const* __help__ = "Print measured temperatures of SDH.\n(C++ demo application using the SDHLibrary-CPP library.)"

11.12.5.4 char const* __url__ = "http://www.schunk.com"

11.12.5.5 char const* __version__ = "$Id: demo-temperature.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"

11.13 demo/demo-test.cpp File Reference

11.13.1 Detailed Description

Print measured actual axis angles of SDH. (C++ demo application using the SDHLibrary-CPP library.).

11.13.2 General file information

Author:

Dirk Osswald

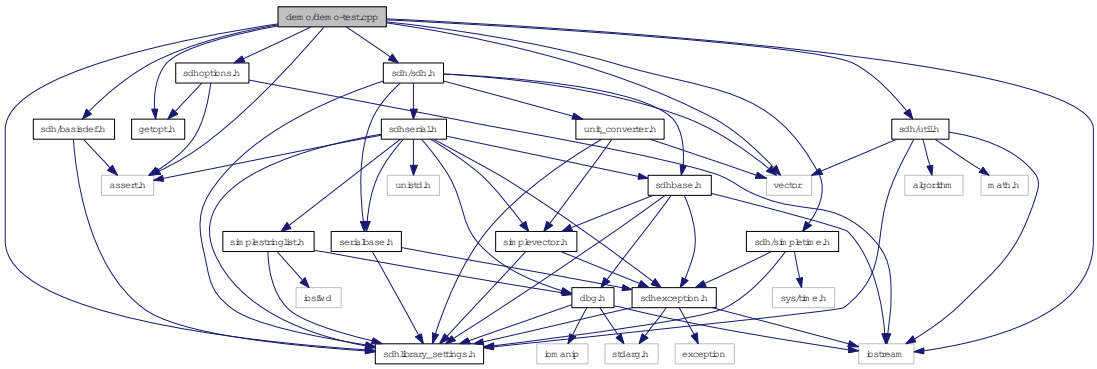
Date:

2007-03-07

11.13.3 Copyright


```
#include <getopt.h>
#include <assert.h>
#include <iostream>
#include <vector>
#include "sdh/sdh.h"
#include "sdh/simpletime.h"
#include "sdh/util.h"
#include "sdh/sdhlibrary_settings.h"
#include "sdh/basisdef.h"
#include "sdhoptions.h"
```

Include dependency graph for demo-test.cpp:



Functions

- int main (int argc, char **argv)

Variables

Some informative variables

- char const * __help__ = "Tries to connect to SDH, read actual angles and exits.\n(C++ demo application using the SDHLibrary-CPP library.)"
- char const * __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"
- char const * __url__ = "http://www.schunk.com"
- char const * __version__ = "$Id: demo-test.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"
- char const * __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.13.4 Function Documentation

11.13.4.1 int main (int *argc*, char ** *argv*)

11.13.5 Variable Documentation

11.13.5.1 char const* __author__ = "Dirk Osswald: dirk.osswald@de.schunk.com"

11.13.5.2 char const* __copyright__ = "Copyright (c) 2007 SCHUNK GmbH & Co. KG"

11.13.5.3 char const* __help__ = "Tries to connect to SDH, read actual angles and exits.\n(C++ demo application using the SDHLibrary-CPP library.)"

11.13.5.4 char const* __url__ = "http://www.schunk.com"

11.13.5.5 char const* __version__ = "$Id: demo-test.cpp 3686 2008-10-13 15:07:24Z Osswald2 $"

11.14 demo/dsaoptions.cpp File Reference

11.14.1 Detailed Description

Implementation of a class to parse common SDH related command line options.

11.14.2 General file information

Author:

Dirk Osswald

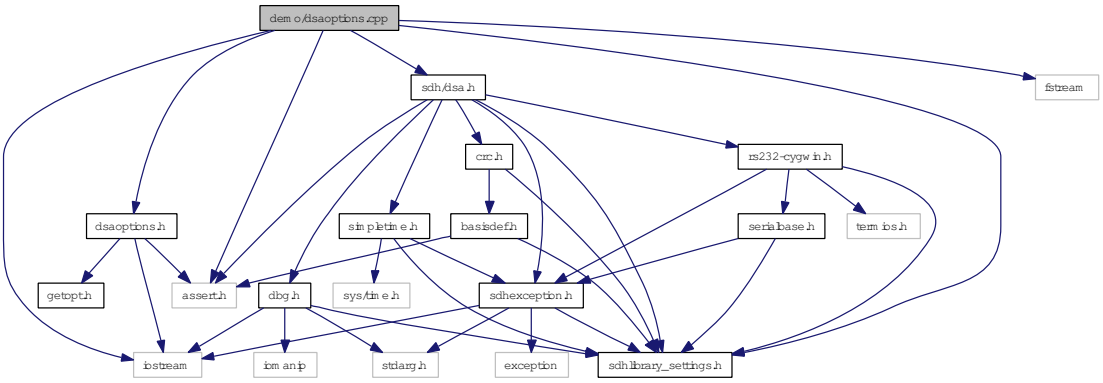
Date:

2008-06-13

11.14.3 Copyright


```
#include <assert.h>
#include <iostream>
#include <fstream>
#include "sdh/dsa.h"
#include "sdh/sdhlibrary_settings.h"
#include "dsaoptions.h"
```

Include dependency graph for dsaoptions.cpp:



Variables

- static char * dsaoptions_usage
- static char * dsaoptions_short_options = "hp:r:fscmd:v"
- static struct option dsaoptions_long_options [ ]

11.14.4 Variable Documentation

11.14.4.1 struct option dsaoptions_long_options[ ] [static]

Initial value:

```
{

    {"help",           0, 0, 'h'},
    {"dsaport",        1, 0, 256+'p'},
    {"port",           1, 0, 'p'},
    {"framerate",      1, 0, 'r' },
    {"fullframe",      0, 0, 'f'},
    {"sensorinfo",     0, 0, 's'},
    {"controllerinfo", 0, 0, 'c' },
    {"matrixinfo",     1, 0, 'm'},
    {"no_rle",         0, 0, 256+'r' },
    {"debug",          1, 0, 'd'},
    {"debuglog",       1, 0, 'l'},
    {"version",        0, 0, 'v'},
    {0, 0, 0, 0}
}
```

11.14.4.2 char* dsaoptions_short_options = "hp:r:fscmd:v" [static]

11.14.4.3 char* dsaoptions_usage [static]

Initial value:

```
"options:\n"
"  -h, --help            show this help message and exit\n"
"  -p, --port=PORT, --dsaport=PORT \n"
"                        use RS232 communication PORT to connect to to \n"
"                        tactile sensor controller of SDH \n"
"                        instead of default 0='COM1'='/dev/ttyS0'.\n"
"  -d LEVEL, --debug=LEVEL\n"
"                        Print debug messages of level LEVEL or lower while\n"
"                        execting the programm. Level 0 (default): No\n"
"                        messages,  1: application-level messages, 2: cSDH-level\n"
"                        messages, 3: cSDHSerial-level messages\n"
"  -l LOGFILE, --debuglog=LOGFILE\n"
"                        Redirect the printed debug messages to LOGFILE instead\n"
"                        of standard error (default). If LOGFILE starts with\n"
"                        '+' then the output will be appended to the file\n"
"                        (without the leading '+'), else the file will be\n"
"  -v, --version         Print the version (revision/release names of application,\n"
"                        library and firmware then exit.\n"
"  -r, --framerate=FRAMERATE \n"
"                        report tactile sensor frames with FRAMERATE frames per second\n"
"  -f, --fullframe       Print acquired full frames numerically.\n"
"  -s, --sensorinfo      Print sensor info from DSA (texel dimensions, number of texels...).\n"
"  -c, --controllerinfo  Print controller info from DSA (version...).\n"
"  -m, --matrixinfo=MATRIX_INDEX\n"
"                        Print matrix info for matrix with index MATRIX_INDEX from DSA.\n"
"  --no_rle              Do not use the RunLengthEncoding\n"
```

11.15 demo/dsaoptions.h File Reference

11.15.1 Detailed Description

Implementation of a class to parse common SDH related command line options.

11.15.2 General file information

Author:

Dirk Osswald

Date:

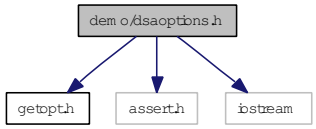
2008-05-05

11.15.3 Copyright

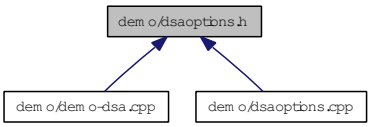
Copyright (c) 2008 SCHUNK GmbH & Co. KG

```
#include <getopt.h>
#include <assert.h>
#include <iostream>
```

Include dependency graph for dsaoptions.h:



This graph shows which files directly or indirectly include this file:



Classes

- class cDSAOptions

11.16 demo/sdhoptions.cpp File Reference

11.16.1 Detailed Description

Implementation of a class to parse common SDH related command line options.

Author:

Dirk Osswald

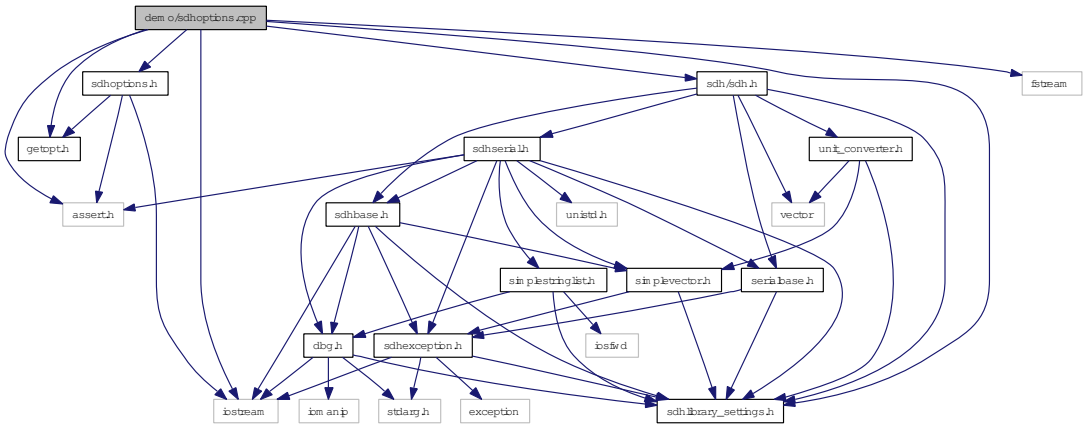
Date:

2008-05-05

11.16.2 Copyright


```
#include <getopt.h>
#include <assert.h>
#include <iostream>
#include <fstream>
#include "sdh/sdh.h"
#include "sdh/sdhlibrary_settings.h"
#include "sdhoptions.h"
```

Include dependency graph for sdhoptions.cpp:



Variables

- static char * sdhoptions_usage
- static char * sdhoptions_short_options = "hp:d:RFvt:T:cn:b:r:w:l:"
- static struct option sdhoptions_long_options [ ]

11.16.3 Variable Documentation

11.16.3.1 struct option sdhoptions_long_options[ ] [static]

Initial value:

```
{

  {"help",       0, 0, 'h'},
  {"port",       1, 0, 'p'},
  {"debug",      1, 0, 'd'},
  {"debuglog",   1, 0, 'l'},
  {"radians",    0, 0, 'R'},
  {"fahrenheit", 0, 0, 'F'},
  {"version",    0, 0, 'v'},
  {"period",     1, 0, 't'},
  {"timeout",    1, 0, 'T'},
  {"can",        0, 0, 'c'},
  {"net",        1, 0, 'n'},
  {"baud",       1, 0, 'b'},
  {"id_read",    1, 0, 'r'},
  {"id_write",   1, 0, 'w'},

  {0, 0, 0, 0}
}
```

11.16.3.2 char* sdhoptions_short_options = "hp:d:RFvt:T:cn:b:r:w:l:" [static]

11.16.3.3 char* sdhoptions_usage [static]

11.17 demo/sdhoptions.h File Reference

11.17.1 Detailed Description

Implementation of a class to parse common SDH related command line options.

11.17.2 General file information

Author:

Dirk Osswald

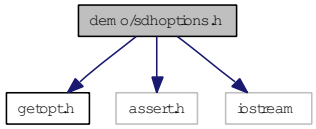
Date:

2008-05-05

11.17.3 Copyright


```
#include <getopt.h>
#include <assert.h>
#include <iostream>
```

Include dependency graph for sdhoptions.h:



This graph shows which files directly or indirectly include this file:

Classes

- class cSDHOptions

11.18 Doxyfile File Reference

11.19 Makefile File Reference

11.19.1 Detailed Description

Makefile for SDH SDHLibrary C project.

11.19.2 General file information

Author:

Dirk Osswald

Date:

2007-01-03

This makefile can generate the C library itself, demo-programs and auxiliary stuff like **doxygen** documentation or generate a distribution for delivery to end users.

For a general description of the project see general project information.

11.19.3 Makefile variables

The variables defined here state project specific settings which are then used by the goals and/or by the included, more generic sub makefiles like:

- **Makefile-common**
- **Makefile-doc**
- **Makefile-rules**

11.19.4 Makefile targets

- `all` : generate everything
  - `build` : generate library and demo programs
  - `doc` : generate all documentation
- `clean` : clean up generated program files, but not TAGS or **doxygen** doc
- `mrproper` : clean up all generated files, including TAGS and **doxygen** doc
- `tags` : generate emacs TAGS file
- `dist` : create a distribution
  - `dist_only` : create a distribution without regenerating doc or plot

11.19.5 Links

- The online documentation for `gnu make` can be found at `http://www.gnu.org/software/make/manual/make.html`

11.19.6 Copyright

11.20 sdh/basisdef.h File Reference

11.20.1 Detailed Description

This file contains some basic definitions (defines, macros, datatypes).

11.20.2 General file information

Author:

Jan Grewe, Dirk Osswald

Date:

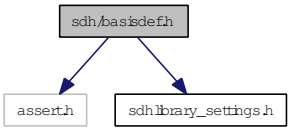
08.10.2004

- Datatypes: SDH::Int8, SDH::UInt8, SDH::Int16, SDH::UInt16, SDH::Int32, SDH::UInt32

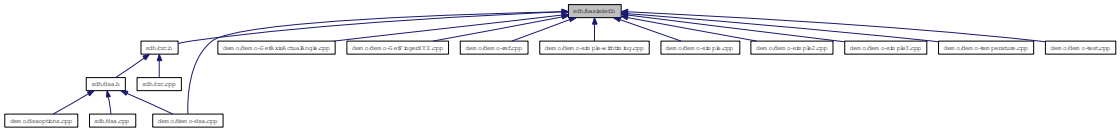
11.20.3 Copyright


```
#include <assert.h>
#include "sdhlibrary_settings.h"
```

Include dependency graph for basisdef.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Defines

- #define SDH_ASSERT_TYPESIZES()

  macro to assert that the defined typedefs have the expected sizes

Typedefs

- typedef char SDH::Int8

  signed integer, size 1 Byte (8 Bit)

- typedef unsigned char SDH::UInt8

  unsigned integer, size 1 Byte (8 Bit)

- typedef short SDH::Int16

  signed integer, size 2 Byte (16 Bit)

- typedef unsigned short SDH::UInt16

  unsigned integer, size 2 Byte (16 Bit)

- typedef long SDH::Int32

  signed integer, size 4 Byte (32 Bit)

- typedef unsigned long SDH::UInt32

  unsigned integer, size 4 Byte (32 Bit)

11.20.4 Define Documentation

11.20.4.1 #define SDH_ASSERT_TYPESIZES()

Value:

```
do {                                    \
        assert( sizeof( Int8 )   == 1 );    \
        assert( sizeof( UInt8 )  == 1 );    \
        assert( sizeof( Int16 )  == 2 );    \
        assert( sizeof( UInt16 ) == 2 );    \
        assert( sizeof( Int32 )  == 4 );    \
        assert( sizeof( UInt32 ) == 4 );    \
    } while (0)
```

macro to assert that the defined typedefs have the expected sizes

11.21 sdh/canserial-esd.cpp File Reference

11.21.1 Detailed Description

Implementation of class SDH::cCANSerial_ESD, a class to access an ESD CAN interface on cygwin/linux and Visual Studio.

11.21.2 General file information

Author:

Dirk Osswald

Date:

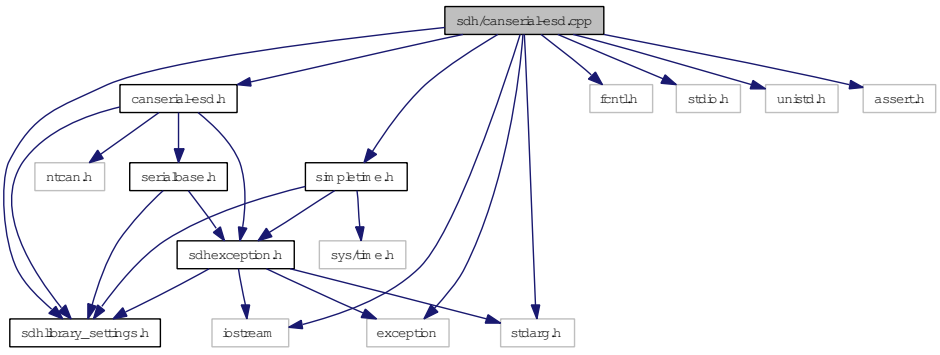
2007-02-20

11.21.3 Copyright

Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include "sdhlibrary_settings.h"
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <exception>
#include <stdarg.h>
#include <assert.h>
#include "canserial-esd.h"
#include "simpletime.h"
```

Include dependency graph for canserial-esd.cpp:



Defines

- #define DEFINE_TO_CASECOMMAND(_c) case _c: return (#_c)
- #define DEFINE_TO_CASECOMMAND_MSG(_c,...) case _c: return (#_c ": " __VA_ARGS__)

Functions

- char const * ESD_strerror (NTCAN_RESULT rc)

11.21.4 Define Documentation

11.21.4.1 #define DEFINE_TO_CASECOMMAND(_c) case _c: return (#_c)

Just a macro for the very lazy programmer to convert an enum or a DEFINE macro into a case command that returns the name of the macro as string.

Usage:

```
char const* eSomeEnumType_ToString( eSomeEnumType rc )
{
switch (rc)
{
DEFINE_TO_CASECOMMAND( AN_ENUM );
DEFINE_TO_CASECOMMAND( AN_OTHER_ENUM );
...
default:                return "unknown return code";
}
}
```

Remarks:

You must use the enum or macro directly (not a variable with that value) since CPP-stringification is used.

See also DEFINE_TO_CASECOMMAND_MSG

11.21.4.2 #define DEFINE_TO_CASECOMMAND_MSG(_c, ...) case _c: return (#_c ": " __VA_ARGS__)

Just a macro for the very lazy programmer to convert an enum or a DEFINE macro and a message into a case command that returns the name of the macro and the message as string.

Usage:

```
char const* eSomeEnumType_ToString( eSomeEnumType rc )
{
switch (rc)
{
DEFINE_TO_CASECOMMAND_MSG( AN_ENUM, "some mighty descriptive message" );
DEFINE_TO_CASECOMMAND_MSG( AN_OTHER_ENUM, "guess what" );
...
default:                return "unknown return code";
}
}
```

Remarks:

You must use the enum or macro directly (not a variable with that value) since CPP-stringification is used.

See also DEFINE_TO_CASECOMMAND

11.21.5 Function Documentation

11.21.5.1 char const* ESD_strerror (NTCAN_RESULT *rc*)

11.22 sdh/canserial-esd.h File Reference

11.22.1 Detailed Description

Interface of class SDH::cCANSerial_ESD, class to access CAN bus via ESD card on cygwin/linux.

11.22.2 General file information

Author:

Dirk Osswald

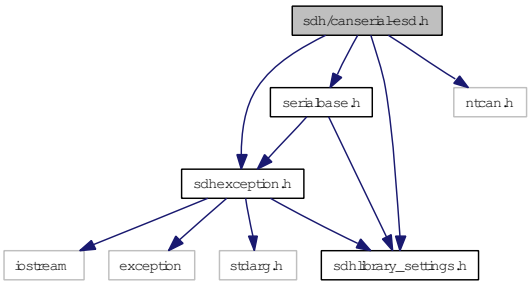
Date:

2008-05-02

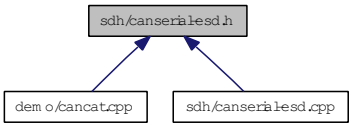
11.22.3 Copyright


```
#include "sdhexception.h"
#include "serialbase.h"
#include "ntcan.h"
#include "sdhlibrary_settings.h"
```

Include dependency graph for canserial-esd.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cCANSerial_ESDException

  Derived exception class for low-level CAN ESD related exceptions.

- class SDH::cCANSerial_ESD

  Low-level communication class to access a CAN port.

Defines

- #define CAN_ESD_TXQUEUESIZE 32

  transmit queue size for CAN frames

- #define CAN_ESD_RXQUEUESIZE 512

  receive queue size for CAN frames

11.22.4 Define Documentation

11.22.4.1 #define CAN_ESD_RXQUEUESIZE 512

receive queue size for CAN frames

11.22.4.2 #define CAN_ESD_TXQUEUESIZE 32

transmit queue size for CAN frames

11.23 sdh/crc.cpp File Reference

11.23.1 Detailed Description

Implementation of class SDH::cCRC_DSACON32m (actually only the static members all other is derived).

11.23.2 General file information

Author:

Dirk Osswald

Date:

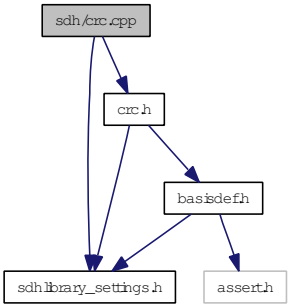
2007-02-19

11.23.3 Copyright

- Copyright (c) 2008 SCHUNK GmbH & Co. KG

```
#include "sdhlibrary_settings.h"
#include "crc.h"
```

Include dependency graph for crc.cpp:



11.24 sdh/crc.h File Reference

11.24.1 Detailed Description

This file contains interface to cCRC, a class to handle CRC calculation.

11.24.2 General file information

Author:

Dirk Osswald

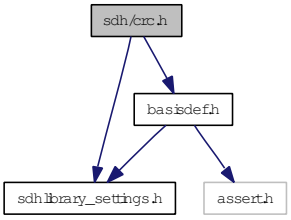
Date:

2008-06-09

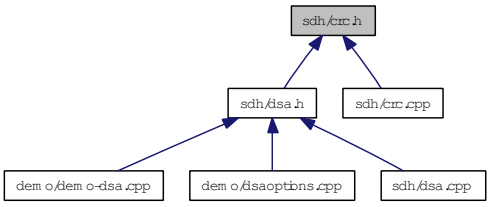
11.24.3 Copyright


```
#include "sdhlibrary_settings.h"
#include "basisdef.h"
```

Include dependency graph for crc.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cCRC

- class SDH::cCRC_DSACON32m

  A derived CRC class that uses a CRC table and initial value suitable for the Weiss Robotics DSACON32m controller.

Typedefs

- typedef **UInt16** SDH::tCRCValue

  the data type used to calculate and exchange CRC values with DSACON32m (16 bit integer)

11.25 sdh/dbg.h File Reference

11.25.1 Detailed Description

This file contains interface and implementation of class SDH::cDBG, a class for colorfull debug messages.

11.25.2 General file information

Author:

Dirk Osswald

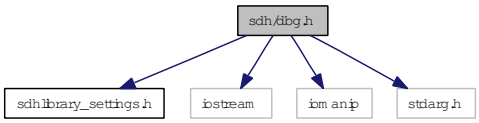
Date:

2007-02-22

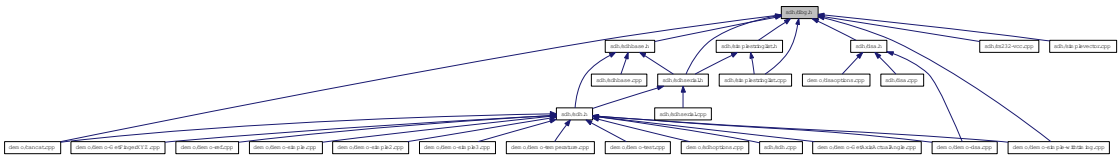
11.25.3 Copyright


```
#include "sdhlibrary_settings.h"
#include <iostream>
#include <iomanip>
#include <stdarg.h>
```

Include dependency graph for dbg.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cDBG

  A class to print colored debug messages.

Defines

- #define VAR(_d, _var) (_d) << #_var << "=’" << _var << "’\n"
- #define V(_var) #_var << "=’" << _var << "’ "

11.25.4 Define Documentation

11.25.4.1 #define V(_var) #_var << "=’" << _var << "’ "

11.25.4.2 #define VAR(_d, _var) (_d) << #_var << "=’" << _var << "’\n"

11.26 sdh/dsa.cpp File Reference

11.26.1 Detailed Description

This file contains definition of SDH::cDSA, a class to communicate with the tactile sensors of the SDH.

11.26.2 General file information

Author:

Dirk Osswald

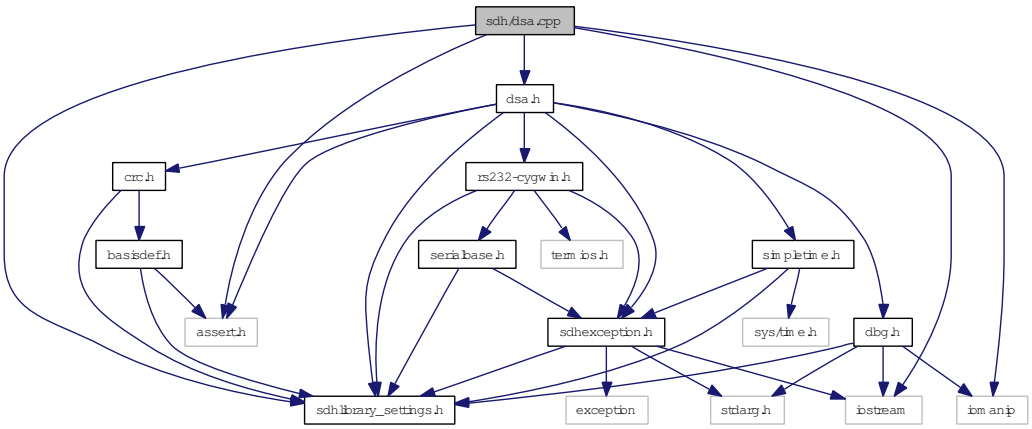
Date:

2008-06-09

11.26.3 Copyright


```
#include "sdhlibrary_settings.h"
#include <assert.h>
#include <iostream>
#include <iomanip>
#include "dsa.h"
```

Include dependency graph for dsa.cpp:



Namespaces

- namespace SDH

Defines

- #define PRINT_MEMBER(_s, _var, _member) (_s) << " " << #_member << "=’" << _var._member << "’\n"

- #define SDH_NAMESPACE_PREFIX SDH::

Functions

- std::ostream & SDH::operator<< (std::ostream &stream, cDSA::sControllerInfo const &controller_info)
- std::ostream & SDH::operator<< (std::ostream &stream, cDSA::sSensorInfo const &sensor_info)
- std::ostream & SDH::operator<< (std::ostream &stream, cDSA::sMatrixInfo const &matrix_info)
- std::ostream & SDH::operator<< (std::ostream &stream, cDSA::sResponse const &response)
- std::ostream & SDH::operator<< (std::ostream &stream, cDSA const &dsa)

11.26.4 Define Documentation

11.26.4.1 #define PRINT_MEMBER(_s, _var, _member) (_s) << " " << #_member << "=’" << _var._member << "’\n"

11.26.4.2 #define SDH_NAMESPACE_PREFIX SDH::

11.27 sdh/dsa.h File Reference

11.27.1 Detailed Description

This file contains interface to SDH::cDSA, a class to communicate with the tactile sensors of the SDH.

11.27.2 General file information

Author:

Dirk Osswald

Date:

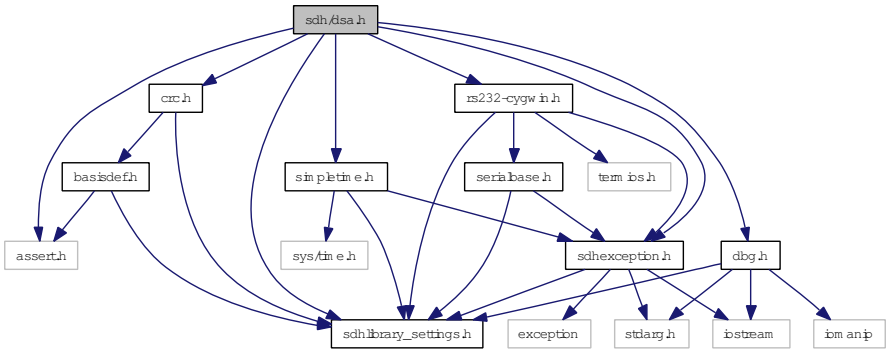
2008-06-09

11.27.3 Copyright

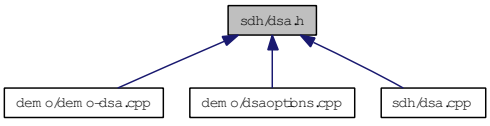
Copyright (c) 2008 SCHUNK GmbH & Co. KG

```
#include "sdhlibrary_settings.h"
#include <assert.h>
#include "sdhexception.h"
#include "dbg.h"
#include "rs232-cygwin.h"
#include "simpletime.h"
#include "crc.h"
```

Include dependency graph for dsa.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cDSAException

  Derived exception class for low-level DSA related exceptions.

- class SDH::cDSA
- struct SDH::cDSA::sControllerInfo

  A data structure describing the controller info about the remote DSACON32m controller.

- struct SDH::cDSA::sSensorInfo

  A data structure describing the sensor info about the remote DSACON32m controller.

- struct SDH::cDSA::sMatrixInfo

  A data structure describing a single sensor matrix connected to the remote DSACON32m controller.

- struct SDH::cDSA::sTactileSensorFrame
- struct SDH::cDSA::sResponse

  data structure for storing responses from the remote DSACON32m controller

Defines

- #define DSA_MAX_PREAMBLE_SEARCH (2∗3∗(6∗(14+13)) + 16)

Functions

- std::ostream & SDH::operator<< (std::ostream &stream, cDSA::sControllerInfo const &controller_info)
- std::ostream & SDH::operator<< (std::ostream &stream, cDSA::sSensorInfo const &sensor_info)
- std::ostream & SDH::operator<< (std::ostream &stream, cDSA::sMatrixInfo const &matrix_info)
- std::ostream & SDH::operator<< (std::ostream &stream, cDSA::sResponse const &response)
- std::ostream & SDH::operator<< (std::ostream &stream, cDSA const &dsa)

11.27.4 Define Documentation

11.27.4.1 #define DSA_MAX_PREAMBLE_SEARCH (2∗3∗(6∗(14+13)) + 16)

11.28 sdh/release.h File Reference

11.28.1 Detailed Description

This file contains nothing but C/C++ defines with the name of the project itself (PROJECT_NAME) and the name of the release (PROJECT_RELEASE) of the whole project.

11.28.2 General file information

Author:

Dirk Osswald

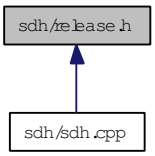
Date:

2006-11-30

For a general description of the project see general project information.

11.28.3 Copyright

This graph shows which files directly or indirectly include this file:



Defines

- #define PROJECT_NAME "SDHLibrary-CPP"

  Name of the software project.

- #define PROJECT_RELEASE "0.0.1.3"

  Release name of the whole software project (a.k.a. as the "version" *of the project).*

- #define PROJECT_DATE "2008-10-16"

  Date of the release of the software project.

- #define PROJECT_COPYRIGHT "(c) SCHUNK GmbH & Co. KG, 2007"

11.28.4 Define Documentation

11.28.4.1 #define PROJECT_COPYRIGHT "(c) SCHUNK GmbH & Co. KG, 2007"

11.28.4.2 #define PROJECT_DATE "2008-10-16"

Date of the release of the software project.

The date of the release of the project.

11.28.4.3 #define PROJECT_NAME "SDHLibrary-CPP"

Name of the software project.

The name of the "SDHLibrary-CPP" (C Library for accessing SDH from a PC) project.

11.28.4.4 #define PROJECT_RELEASE "0.0.1.3"

Release name of the whole software project (a.k.a. as the *"version"* of the project).

The release name of the "SDHLibrary-CPP" project.

A suffix of "-dev" indicates a work in progress, i.e. a not yet finished release. A suffix of "-a", "-b", ... indicates a bugfix release.

From newest to oldest the releases have the following names and features:

- **0.0.1.4**:
  - bufix: `Bug 267: SDHlib cannot be compiled on Linux`
- **0.0.1.3**: 2008-10-16
  - bufix: `Bug 266: demo-dsa does not work in the VCC version`
- **0.0.1.2**: 2008-10-14
  - bugfix: target cancat is now only added in the Makefile if WITH_ESD_CAN is 1
  - bugfix: corrected copy/paste error: removed class qualifiers from member declarations since newer gccs do not accept fully qualified member names within class declarations (like aClass::aMember())
  - bugfix: corrected parameter checking of cSDHSerial::vp()
  - bugfix: corrected error in apply() in util.h (correct result was calculated only locally)
  - made generation of Doxygen-doku work with Doxygen v1.5.5
  - fixed bug in option detection
  - made output of version info more verbose (with SOC and dates)
  - implemented cRS232::Read()
  - made dsa.cpp/h work with VCC
  - corrected TCP calculation: corrected limb lengths and changed coordinate system from left to right handed
  - added cSDH::GetInfo
  - added cSDHSerial::vlim() and cSDH::GetAxisLimitVelocity() to read velocity limits

  - bugfix: Bug 134: cannot generate doxygen documentation if SDH namespace is used
  - bugfix: Bug 261: Header file problems with SDHLibrary-CPP on Linux
  - bugfix: Bug 260: CAN access problems with SDHLibrary-CPP on Linux
  - enhancement: Enable debugging to logfile in SDHLibrary-cpp
  - enhancement: reduced overhead in cDBG::operator<<, no more searching for color strings on each call
  - added demo-simple-withtiming to perform some simple OS-level time measurement
  - change: changed parent class of cSerialBaseException to cSDHErrorCommunication to make the hierarchy more consistent

- **0.0.1.1**:
  - added cancat program for sending inaccessible commands like change_rs232 via CAN
  - added info commands corresponding to new info commands in firmware:
    - in sdhserial:
      - soc - to read the SoC ID
      - soc_date : to read the date string of the SoC
      - ver_date : to read the release date of the firmware
    - in sdh:
      - enhanced GetInfo to read all the above also
    - in the option parser in auxilliary all the info is now printed if "-v" is given
  - added GetDuration command: returns the calculate duration of the currently configured movement (target angle, velocity, acceleration, velocity profile) but does not execute the movement. This simplifies scripts like sdhrecord.py a great deal.
  - made py.test test_sdh work again.
  - while working on Bug 224: Positioning delay of 5s when using SDHLibrary-CPP
    - removed call to SleepSec() in loop in serial for VCC, needed to get rid of delays
    - implemented cSimpleTime in VCC version
  - added support for new firmware v0.0.2.0 commands
    - soc, soc_date, ver_date
    - GetDuration

- **0.0.1.0**: 2008-06-13
  - added basic support for the tactile sensors,
    - new library classes cDSA, cCRC
    - new demo program demo-dsa

- **0.0.0.9**: 2008-06-06
  - added missing files for vcc compilation in distribution
  - removed error in overloaded Sleep function. Internal version renamed to SleepSec.
  - made demo-GetAxisActualAngle and demo-temperature work in periodic mode in Visual Studio

  - untabified all source and header files for use with Visual Studio
  - autostart feature for distribution CD

- **0.0.0.8-a**:
  - added project files for the demo-∗ programs to the Visual Studio solutions file to make the demo programs available under VCC too
  - added forgotten WITH_ESD_CAN=1 to Visual Studio project file for SDHLibrary
  - workaround for accessing RS232 from VCC (WriteFile() does not return number of bytes sent)
  - With VCC the communication via RS232 still has some bugs: long pauses and timeouts,
  - With VCC the "-t" parameter for periodic replies in demos does not work yet

- **0.0.0.8**: 2008-05-26
  - added compatibility code for MS Visual C++ Compiler (VCC)
    - sdhlibrary_defines with compatibility macros
    - pragmas for VCC
    - _attribute_ are switched off for VCC
    - all SDH specific classes can be put in a namespace called "SDH"
  - added index-overview.html with overview of distributed files
  - index.html files in distribution are parsed for ${PROJECT_∗}

- **0.0.0.7-b**: 2008-05-21
  - CAN timeout is now correctly set
  - fixed bug in cpp/Makefile: OSNAME_LINUX=1 was always appended to EXTRACPPFLAGS no matter what OS was used
  - renamed interace member to comm_interface

- **0.0.0.7-a**: 2008-05-17
  - bug fix: minor changes to make compilation work on Linux. (ESDs ntcan.h for Windows is different from that for Linux)
  - still contains a bug that can be fixed without recompilation:
    - The Linux version of ntcan canOpen does not accept timeout values < 0
    - Workaround for demo programs: Use an additional "-T 0.0" command line parameter

- **0.0.0.7**: 2008-05-16
  - added C++ support for CAN using ESD cards
    - restructured low level communication:
      - new base class cSerialBase (+new exception classes cSerialBaseException, cCANSerialESDException)
  - added support for variable baudrate for RS232 communication
  - made command line option handling in c++ demo programs more generic
  - corrected a bug in SDHLibrary-CPP that caused a SEGV (empty throw statement outside of a catch block)

- **0.0.0.6**: 2007-12-27
  - release for RoboCluster, Denmark
    - added ref command and demo-ref program (needed for SDH-003)
    - corrected minor errors to make the above work

- **0.0.0.5-a**: 2007-06-06
  - Included bugfixes from release 0.0.0.3-a (bugfix release for Uni Wales, see below) into release for care-o-bot

- **0.0.0.5**: 2007-05-24
  - Release for care-o-bot (IPA, Stuttgart), mai 2007
  - Restructured files: library stuff into sdh/ and demo programs in demo/ to ease installation on user platform
  - Added library support for the new firmware features:
    - cSDHSerial: a() vp(), vel()
    - cSDH: Get/SetAxisAcceleration(), GetAxisMaxAcceleration(), Get/SetVelocityProfile(), GetAxisCurrentVelocity()
  - while preparing release for IPA care-o-bot:
    - since line endings are corrected in firmware now removed the special EOL treatmnt in readline
    - enhanced generation of distribution
    - extended README files
    - added demo-simple3 in cpp and python
    - made compilation work on linux without warnings (SuSE 8.1 and Knoppix_v5.1.1)
    - added requested functions GetAxisActualState() and WaitAxis() in cpp and python library
    - added eAxisState enums from firmware
    - corrected some yet undetected errors
    - corrected / enhanced some **doxygen** comments
    - tried to find bug:
      - firmware not moving from 5,-5,0,0,0,0,0 to 20,0,0,0,0,0,0:
      - axis 1 is stuck at 1.4...
      - bug could not be resolved (does not happen for for larger movements)

- **0.0.0.4**: 2007-03-19
  - Release for demo at NASA, march 2007
    - adjusted expected lines for "m" command (it now prints one line debug output for every axis)

- **0.0.0.3-a**: 2007-06-05
  - Release modified according to bug report from Martin Huelse
    - A cSDH object could be opened successfully even if no SDH was connected or was connected but not powered:
      - added demo-test program to verify erroneous / repaired behaviour
      - added SetTimeout and GetTimeout to cRS232 class

        - made the code to verify proper connection to SDH work
    - Exceptions could not be caught properly:
        - corrected some real printf-style format string related problems in creations of exceptions
        - added gcc style printf-style format string checking (to enable the compiler to detect errors like the above at compile time)
        - The following piece of information from the C++ Annotations was not considered properly. See http://www.icce.rug.nl/documents/cplusplus/: "A function for which a function throw list is specified may not throw other types of exceptions. A run-time error occurs if it tries to throw other types of exceptions than those mentioned in the function throw list."
        - corrected the function throw lists/exception specification lists so that the most generic type thrown was listed. Thus all the user-level functions now just mention cSDHLibraryException* in their function throw list as all thrown exceptions are derived from it.
  - Further changes
    - added further function throw lists/exception specification lists
    - merged in some changes from newer releases (up to 0.0.0.5):
        - retrying of sending in case of transmission errors
        - naming of sequential / non sequential (formerly called synchronous/asynchronous)
        - fixed many typos in **doxygen** comments

- **0.0.0.3**: 2007-03-09
  - Release modified at visit Uni-Whales
    - Changes to make everything work on Ubuntu-Linux
    - Enhanced Makefile a little bit to be more comfortable for the end user

- **0.0.0.2**: 2007-03-07
  - release, for Uni-Wales

- **0.0.0.1**
  - initial release, works for the first time, but not relyably

11.29 sdh/rs232-cygwin.cpp File Reference

11.29.1 Detailed Description

Implementation of class SDH::cRS232, a class to access serial RS232 port on cygwin/linux.

11.29.2 General file information

Author:

Dirk Osswald

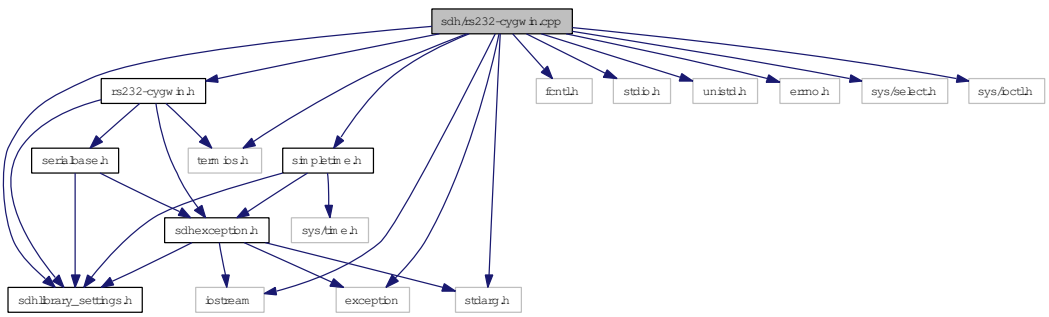
Date:

2007-02-20

11.29.3 Copyright


```
#include "sdhlibrary_settings.h"
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/select.h>
#include <sys/ioctl.h>
#include <iostream>
#include <exception>
#include <stdarg.h>
#include "rs232-cygwin.h"
#include "simpletime.h"
```

Include dependency graph for rs232-cygwin.cpp:



11.30 sdh/rs232-cygwin.h File Reference

11.30.1 Detailed Description

Interface of class SDH::cRS232, a class to access serial RS232 port on cygwin/linux.

11.30.2 General file information

Author:

Dirk Osswald

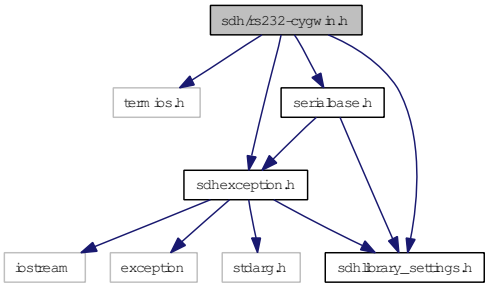
Date:

2007-02-20

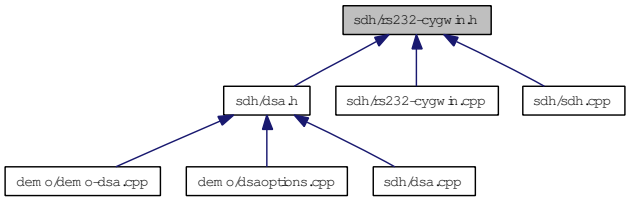
11.30.3 Copyright


```
#include <termios.h>
#include "sdhexception.h"
#include "serialbase.h"
#include "sdhlibrary_settings.h"
```

Include dependency graph for rs232-cygwin.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cRS232Exception

  Derived exception class for low-level RS232 related exceptions.

- class SDH::cRS232

  Low-level communication class to access a serial port on Cygwin and Linux.

11.31 sdh/rs232-vcc.cpp File Reference

11.31.1 Detailed Description

Implementation of class SDH::cRS232, a class to access serial RS232 port with VCC compiler on Windows.

11.31.2 General file information

Author:

Martin

Date:

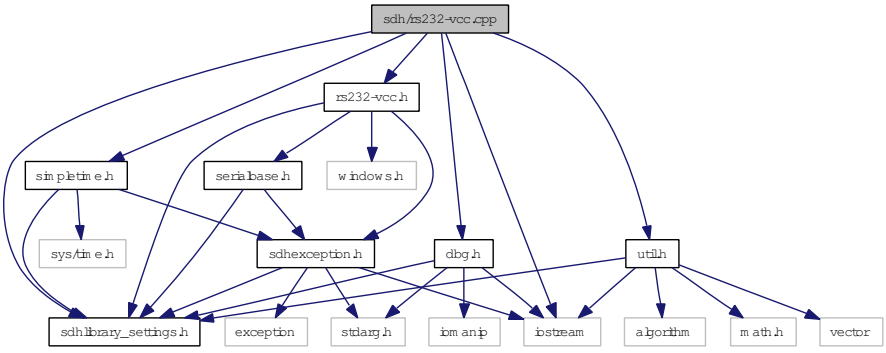
2008-05-23

11.31.3 Copyright

Code kindly provided by Martin from the RoboCluster project Denmark.

```
#include "iostream"
#include "rs232-vcc.h"
#include "simpletime.h"
#include "sdhlibrary_settings.h"
#include "util.h"
#include "dbg.h"
```

Include dependency graph for rs232-vcc.cpp:



Namespaces

- namespace SDH

Defines

- #define _CRT_SECURE_NO_WARNINGS 1

- #define SDH_RS232_VCC_DEBUG 0

  flag, if 1 then debug messages are enabled here in rs232-vcc.cpp. Else messages are disabled without any overhead.

- #define DBG(...) ;

11.31.4 Define Documentation

11.31.4.1 #define _CRT_SECURE_NO_WARNINGS 1

11.31.4.2 #define DBG(...) ;

11.31.4.3 #define SDH_RS232_VCC_DEBUG 0

flag, if 1 then debug messages are enabled here in rs232-vcc.cpp. Else messages are disabled without any overhead.

11.32 sdh/rs232-vcc.h File Reference

11.32.1 Detailed Description

Implementation of class SDH::cRS232, a class to access serial RS232 port with VCC compiler on Windows.

11.32.2 General file information

Author:

Martin

Date:

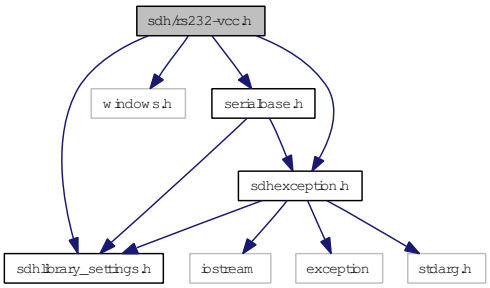
2008-05-23

11.32.3 Copyright

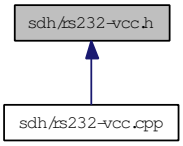
Code kindly provided by Martin from the RoboCluster project Denmark.

```
#include "sdhlibrary_settings.h"
#include <windows.h>
#include "sdhexception.h"
#include "serialbase.h"
```

Include dependency graph for rs232-vcc.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cRS232Exception

  Derived exception class for low-level RS232 related exceptions.

- class SDH::cRS232

  Low-level communication class to access a serial port on Cygwin and Linux.

Defines

- #define SDH_RS232_VCC_ASYNC 0

11.32.4 Define Documentation

11.32.4.1 #define SDH_RS232_VCC_ASYNC 0

11.33 sdh/sdh.cpp File Reference

11.33.1 Detailed Description

This file contains the interface to class SDH::cSDH, the end user class to access the SDH from a PC.

11.33.2 General file information

Author:

Dirk Osswald

Date:

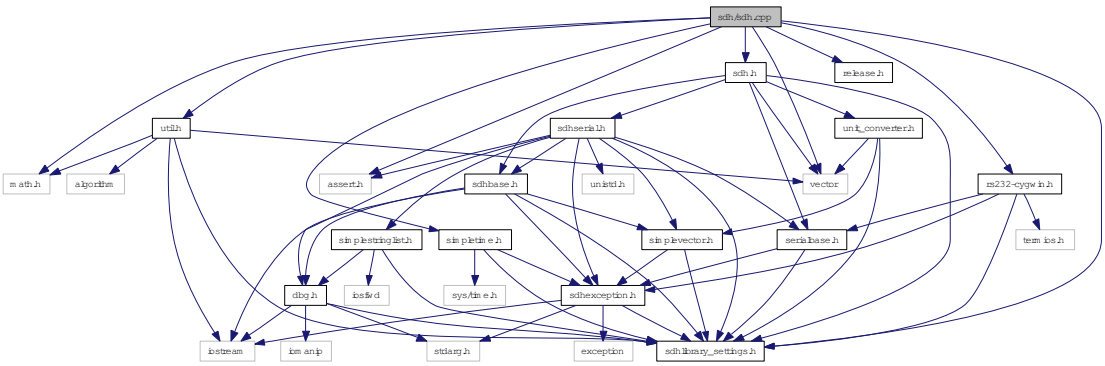
2007-02-20

11.33.3 Copyright

Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include "sdhlibrary_settings.h"
#include <assert.h>
#include <vector>
#include <math.h>
#include <util.h>
#include "sdh.h"
#include "release.h"
#include "simpletime.h"
#include "rs232-cygwin.h"
```

Include dependency graph for sdh.cpp:



11.34 sdh/sdh.h File Reference

11.34.1 Detailed Description

This file contains the interface to class SDH::cSDH, the end user class to access the SDH from a PC.

11.34.2 General file information

Author:

Dirk Osswald

Date:

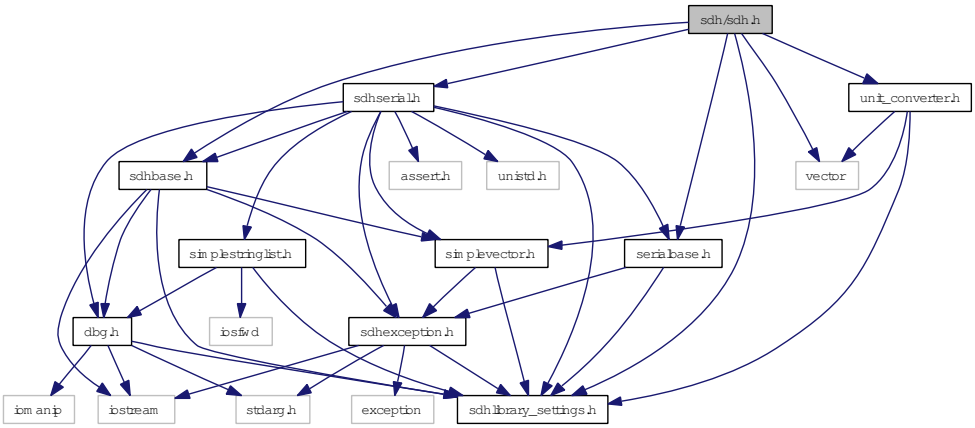
2007-02-20

11.34.3 Copyright

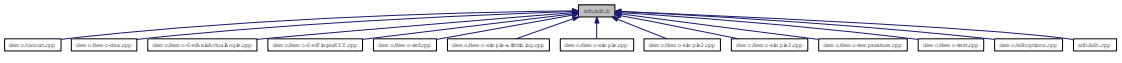
Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include "sdhlibrary_settings.h"
#include <vector>
#include "sdhbase.h"
#include "sdhserial.h"
#include "unit_converter.h"
#include "serialbase.h"
```

Include dependency graph for sdh.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cSDH

  SDH::cSDH is the end user interface class to control a SDH (SCHUNK Dexterous Hand).

Typedefs

- typedef void * SDH::NTCAN_HANDLE

  dummy definition in case ntcan.h is not available

11.35 sdh/sdhbase.cpp File Reference

11.35.1 Detailed Description

Implementation of class SDH::cSDHBase.

11.35.2 General file information

Author:

Dirk Osswald

Date:

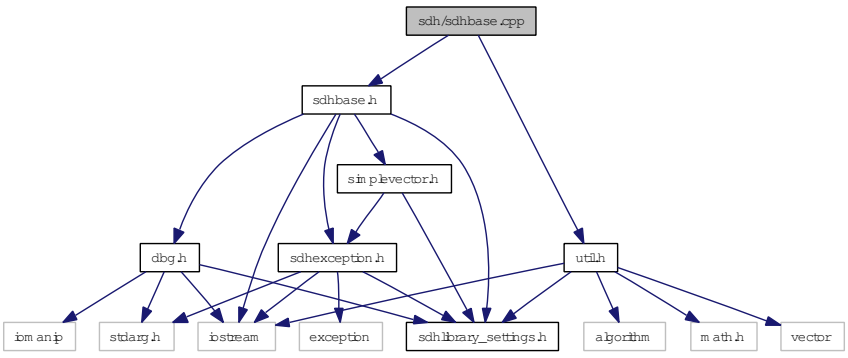
2007-02-19

11.35.3 Copyright

- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include "sdhbase.h"
#include "util.h"
```

Include dependency graph for sdhbase.cpp:



Namespaces

- namespace SDH

Variables

- std::ostream * SDH::g_sdh_debug_log = &std::cerr

11.36 sdh/sdhbase.h File Reference

11.36.1 Detailed Description

Interface of class SDH::cSDHBase.

11.36.2 General file information

Author:

Dirk Osswald

Date:

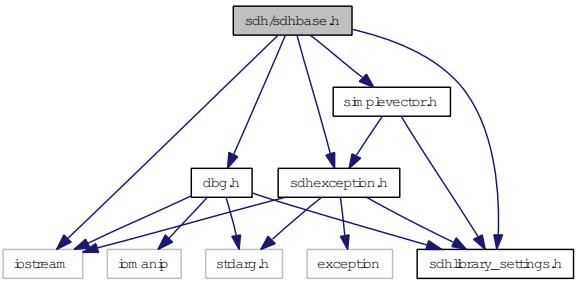
2007-02-19

11.36.3 Copyright

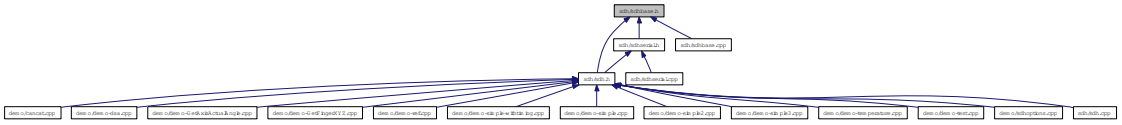
- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include "sdhlibrary_settings.h"
#include "iostream"
#include "dbg.h"
#include "sdhexception.h"
#include "simplevector.h"
```

Include dependency graph for sdhbase.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cSDHErrorInvalidParameter

  Derived exception class for exceptions related to invalid parameters.

- class SDH::cSDHBase

  The base class to control the SCHUNK Dexterous Hand.

11.37 sdh/sdhexception.cpp File Reference

11.37.1 Detailed Description

Implementation of the exception base class SDH::cSDHLibraryException and SDH::cMsg.

11.37.2 General file information

Author:

Dirk Osswald

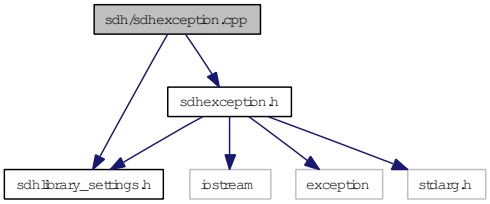
Date:

2007-02-22

11.37.3 Copyright


```
#include "sdhlibrary_settings.h"
#include "sdhexception.h"
```

Include dependency graph for sdhexception.cpp:



Namespaces

- namespace SDH

Functions

- std::ostream & SDH::operator<< (std::ostream &stream, cMsg const &msg)
- std::ostream & SDH::operator<< (std::ostream &stream, cSDHLibraryException const &e)

11.38 sdh/sdhexception.h File Reference

11.38.1 Detailed Description

Interface of the exception base class SDH::cSDHLibraryException and SDH::cMsg.

11.38.2 General file information

Author:

Dirk Osswald

Date:

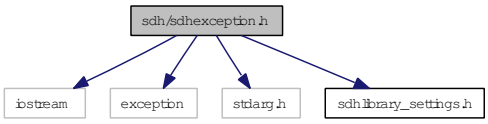
2007-02-22

11.38.3 Copyright

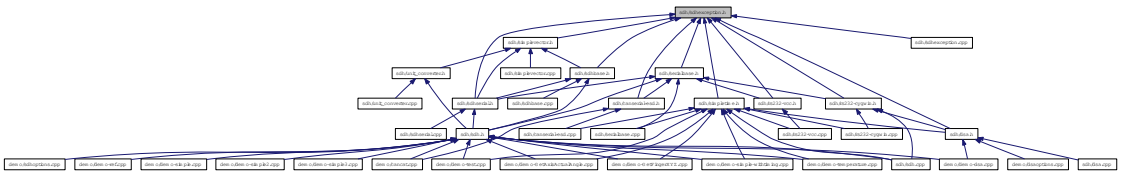
Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include <iostream>
#include <exception>
#include <stdarg.h>
#include "sdhlibrary_settings.h"
```

Include dependency graph for sdhexception.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cMsg

  Class for short, fixed maximum length text messages.

- class SDH::cSDHLibraryException

  Base class for exceptions in the SDHLibrary-CPP.

- class SDH::cSDHErrorCommunication

  Derived exception class for exceptions related to communication between the SDHLibrary and the SDH.

Functions

- std::ostream & SDH::operator<< (std::ostream &stream, cMsg const &msg)
- std::ostream & SDH::operator<< (std::ostream &stream, cSDHLibraryException const &e)

11.39 sdh/sdhlibrary_settings.h File Reference

11.39.1 Detailed Description

This file contains settings to make the SDHLibrary compile on differen systems:

- gcc/Cygwin/Windows
- gcc/Linux
- VisualC++/Windows.

11.39.2 General file information

Author:

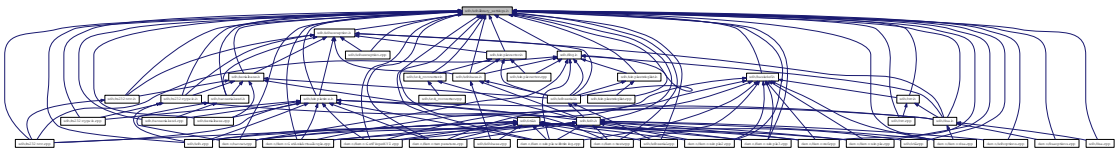
Dirk Osswald

Date:

2008-05-20

11.39.3 Copyright

This graph shows which files directly or indirectly include this file:



Defines

- #define SDH_USE_NAMESPACE 1

  Flag, if 1 then all classes are put into a namespace called SDH. If 0 then the classes are left outside any namespace.

- #define NAMESPACE_SDH_START namespace SDH {
- #define NAMESPACE_SDH_END }
- #define USING_NAMESPACE_SDH using namespace SDH;

11.40 sdh/sdhserial.cpp File Reference

11.40.1 Detailed Description

Interface of class SDH::cSDHSerial.

11.40.2 General file information

Author:

Dirk Osswald

Date:

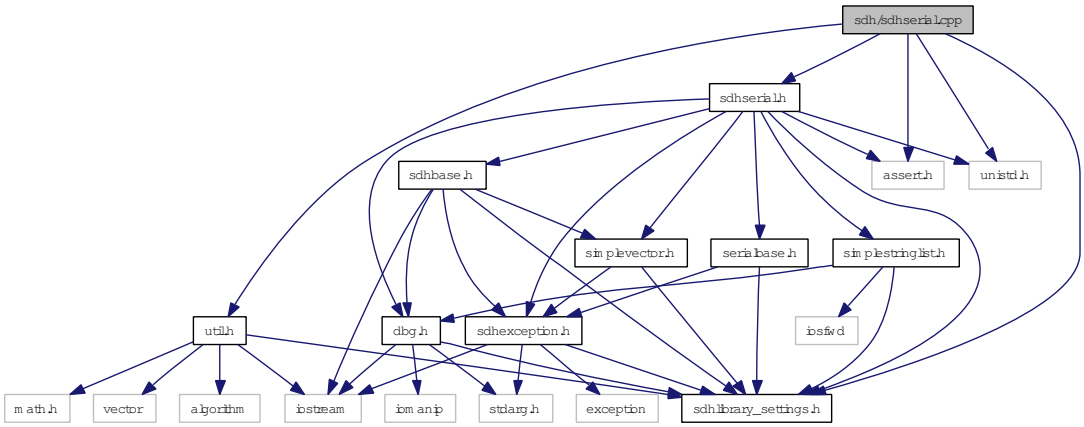
2007-02-19

11.40.3 Copyright

- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include "sdhlibrary_settings.h"
#include <assert.h>
#include <unistd.h>
#include "util.h"
#include "sdhserial.h"
```

Include dependency graph for sdhserial.cpp:



11.41 sdh/sdhserial.h File Reference

11.41.1 Detailed Description

Interface of class SDH::cSDHSerial.

11.41.2 General file information

Author:

Dirk Osswald

Date:

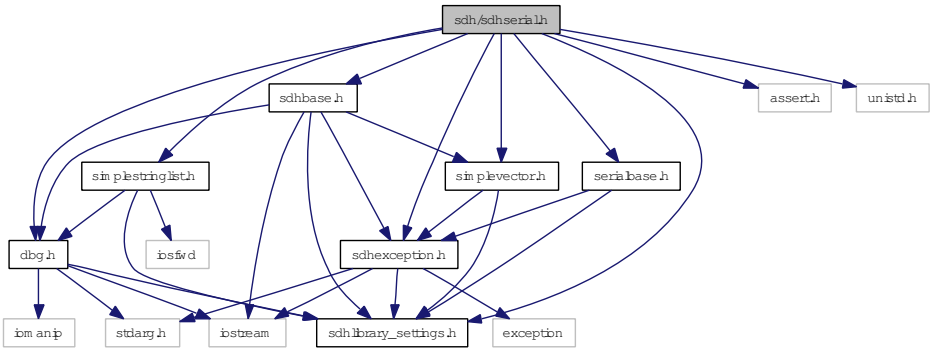
2007-02-19

11.41.3 Copyright

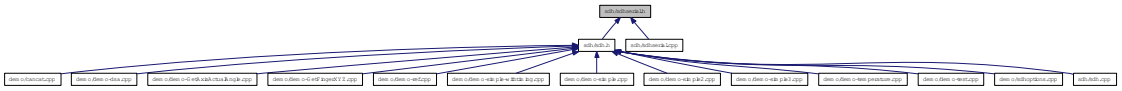
- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include "sdhlibrary_settings.h"
#include <assert.h>
#include <unistd.h>
#include "dbg.h"
#include "sdhexception.h"
#include "simplevector.h"
#include "simplestringlist.h"
#include "sdhbase.h"
#include "serialbase.h"
```

Include dependency graph for sdhserial.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cSDHSerial

  The class to communicate with a SDH via RS232.

Typedefs

- typedef cSimpleVector(cSDHSerial::∗ SDH::pSetFunction)(int, double ∗)

  Type of a pointer to a "set-axis-values" function like cSDHSerial::p, cSDHSerial::pos, ..., cSDHSerial::igrip, cSDHSerial::ihold or cSDHSerial::ilim.

- typedef cSimpleVector(cSDHSerial::∗ SDH::pGetFunction)(int, double ∗)

  Type of a pointer to a "get-axis-values" function like cSDHSerial::p, cSDHSerial::pos, ..., cSDHSerial::igrip, cSDHSerial::ihold or cSDHSerial::ilim.

11.42 sdh/serialbase.cpp File Reference

11.42.1 Detailed Description

Implementation of class SDH::cSerialBase, a virtual base class to access serial interfaces like RS232 or CAN.

11.42.2 General file information

Author:

Dirk Osswald

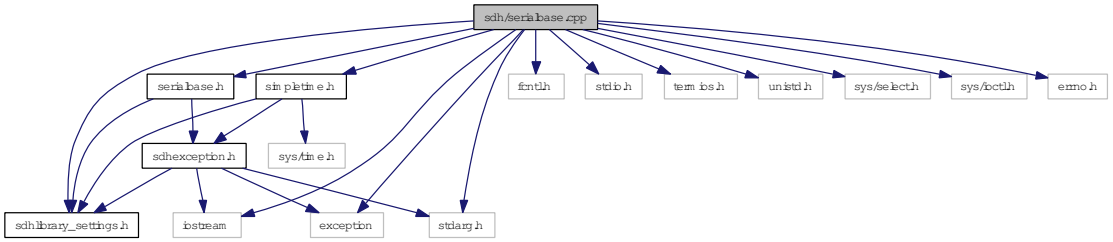
Date:

2007-02-20

11.42.3 Copyright


```
#include "sdhlibrary_settings.h"
#include <fcntl.h>
#include <stdio.h>
#include <termios.h>
#include <unistd.h>
#include <sys/select.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <iostream>
#include <exception>
#include <stdarg.h>
#include "serialbase.h"
#include "simpletime.h"
```

Include dependency graph for serialbase.cpp:



11.43 sdh/serialbase.h File Reference

11.43.1 Detailed Description

Interface of class SDH::cSerialBase, a virtal base class to access serial communication channels like RS232 or CAN.

11.43.2 General file information

Author:

Dirk Osswald

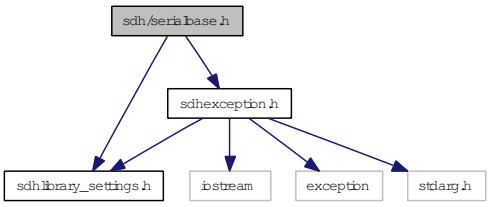
Date:

2008-05-02

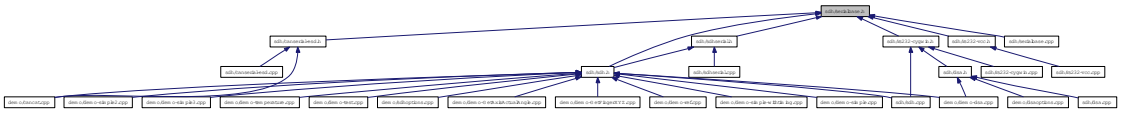
11.43.3 Copyright


```
#include "sdhlibrary_settings.h"
#include "sdhexception.h"
```

Include dependency graph for serialbase.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cSerialBaseException

  Derived exception class for low-level serial communication related exceptions.

- class SDH::cSerialBase

  Low-level communication class to access a serial port.

11.44 sdh/simplestringlist.cpp File Reference

11.44.1 Detailed Description

Implementation of class SDH::cSimpleStringList.

11.44.2 General file information

Author:

Dirk Osswald

Date:

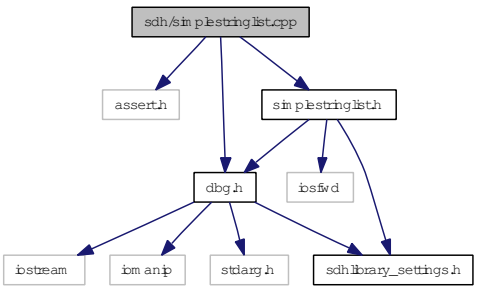
2007-02-19

11.44.3 Copyright

- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include <assert.h>
#include "dbg.h"
#include "simplestringlist.h"
```

Include dependency graph for simplestringlist.cpp:



Namespaces

- namespace SDH

Functions

- std::ostream & SDH::operator<< (std::ostream &stream, cSimpleStringList &ssl)

  Output of cSimpleStringList objects in 'normal' output streams.

11.45 sdh/simplestringlist.h File Reference

11.45.1 Detailed Description

Interface of class SDH::cSimpleStringList.

11.45.2 General file information

Author:

Dirk Osswald

Date:

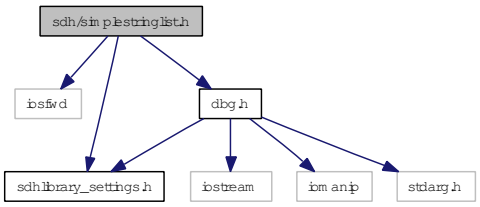
2007-02-19

11.45.3 Copyright

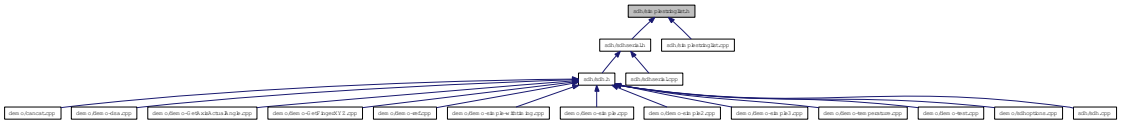
- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include <iosfwd>
#include "dbg.h"
#include "sdhlibrary_settings.h"
```

Include dependency graph for simplestringlist.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cSimpleStringList

  A simple string list. (Fixed maximum number of strings of fixed maximum length).

Functions

- std::ostream & SDH::operator<< (std::ostream &stream, cSimpleStringList &ssl)

  Output of cSimpleStringList objects in 'normal' output streams.

11.46 sdh/simpletime.h File Reference

11.46.1 Detailed Description

Interface of auxilliary utility functions for SDHLibrary-CPP.

11.46.2 General file information

Author:

Dirk Osswald

Date:

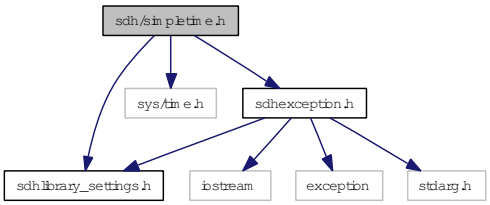
2007-02-19

11.46.3 Copyright

- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include "sdhlibrary_settings.h"
#include <sys/time.h>
#include "sdhexception.h"
```

Include dependency graph for simpletime.h:



This graph shows which files directly or indirectly include this file:

Namespaces

- namespace SDH

Classes

- class SDH::cSimpleTime

  Very simple class to measure elapsed time.

11.47 sdh/simplevector.cpp File Reference

11.47.1 Detailed Description

Implementation of class SDH::cSimpleVector.

11.47.2 General file information

Author:

Dirk Osswald

Date:

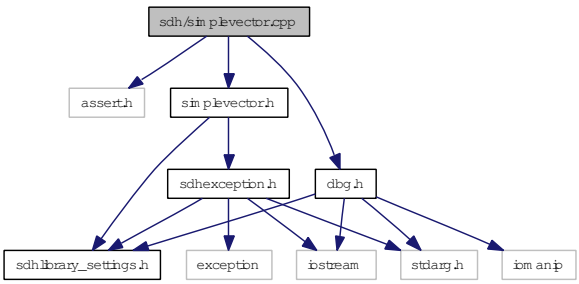
2007-02-19

11.47.3 Copyright

- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include <assert.h>
#include "dbg.h"
#include "simplevector.h"
```

Include dependency graph for simplevector.cpp:



11.48 sdh/simplevector.h File Reference

11.48.1 Detailed Description

Interface of class SDH::cSimpleVector.

11.48.2 General file information

Author:

Dirk Osswald

Date:

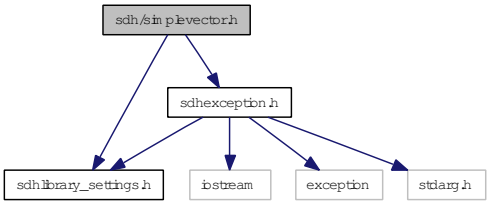
2007-02-19

11.48.3 Copyright

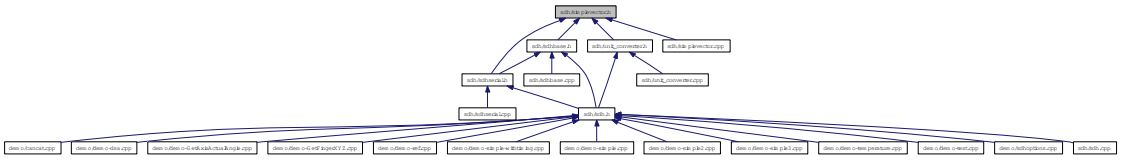
- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include "sdhlibrary_settings.h"
#include "sdhexception.h"
```

Include dependency graph for simplevector.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cSimpleVectorException

  Derived exception class for low-level simple vector related exceptions.

- class SDH::cSimpleVector

  A simple vector implementation.

11.49 sdh/unit_converter.cpp File Reference

11.49.1 Detailed Description

Implementation of class SDH::cUnitConverter.

11.49.2 General file information

Author:

Dirk Osswald

Date:

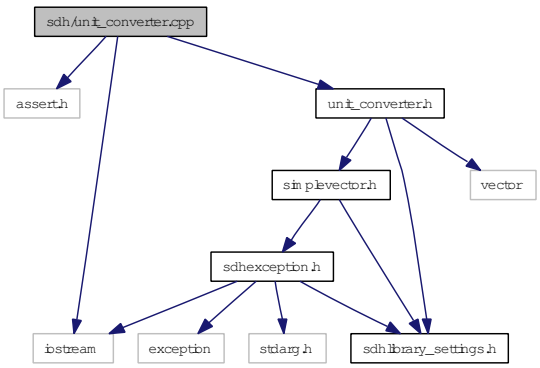
2007-02-19

11.49.3 Copyright

- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include <assert.h>
#include <iostream>
#include "unit_converter.h"
```

Include dependency graph for unit_converter.cpp:



Namespaces

- namespace SDH

Variables

- cUnitConverter const SDH::uc_identity ("any","any","?", 1.0, 0.0, 4)

  Identity converter (internal = external).

11.50 sdh/unit_converter.h File Reference

11.50.1 Detailed Description

Interface of class SDH::cUnitConverter.

11.50.2 General file information

Author:

Dirk Osswald

Date:

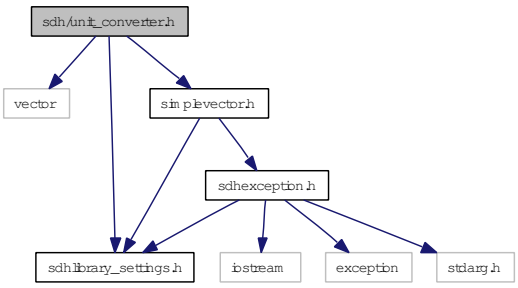
2007-02-19

11.50.3 Copyright

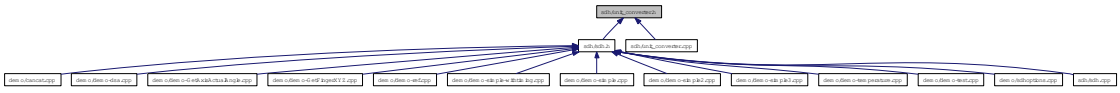
- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include <vector>
#include "simplevector.h"
#include "sdhlibrary_settings.h"
```

Include dependency graph for unit_converter.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Classes

- class SDH::cUnitConverter

Unit conversion class to convert values between physical unit systems.

Typedefs

- typedef double(cUnitConverter::∗ SDH::pDoubleUnitConverterFunction)(double) const

  Type of a pointer to a function like 'double cUnitConverter::ToExternal(double)' or 'cUnitConverterToInternal(double)'.

11.51 sdh/util.cpp File Reference

11.51.1 Detailed Description

Implementation of auxilliary utility functions for SDHLibrary-CPP.

11.51.2 General file information

Author:

Dirk Osswald

Date:

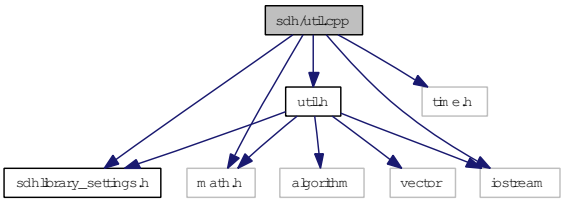
2007-02-19

11.51.3 Copyright

- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include "sdhlibrary_settings.h"
#include <math.h>
#include <time.h>
#include <iostream>
#include "util.h"
```

Include dependency graph for util.cpp:



Namespaces

- namespace SDH

Functions

Auxiliary functions

- bool SDH::InIndex (int v, int max)
- bool SDH::InRange (double v, double min, double max)
- bool SDH::InRange (int n, double const *v, double const *min, double const *max)
- double SDH::ToRange (double v, double min, double max)
- void SDH::ToRange (int n, double *v, double const *min, double const *max)

- void SDH::ToRange (std::vector< double > &v, std::vector< double > const &min, std::vector< double > const &max)
- double SDH::Approx (double a, double b, double eps)
- bool SDH::Approx (int n, double *a, double *b, double *eps)
- double SDH::DegToRad (double d)
- double SDH::RadToDeg (double r)
- void SDH::SleepSec (double t)

11.52 sdh/util.h File Reference

11.52.1 Detailed Description

Interface of auxilliary utility functions for SDHLibrary-CPP.

11.52.2 General file information

Author:

Dirk Osswald

Date:

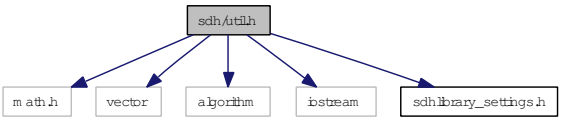
2007-02-19

11.52.3 Copyright

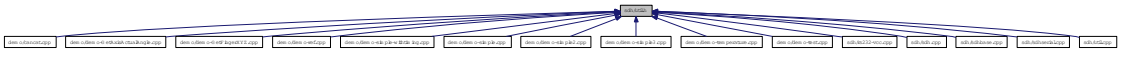
- Copyright (c) 2007 SCHUNK GmbH & Co. KG

```
#include <math.h>
#include <vector>
#include <algorithm>
#include <iostream>
#include "sdhlibrary_settings.h"
```

Include dependency graph for util.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace SDH

Functions

Auxiliary functions

- bool SDH::InIndex (int v, int max)
- bool SDH::InRange (double v, double min, double max)

- bool SDH::InRange (int n, double const *v, double const *min, double const *max)
- double SDH::ToRange (double v, double min, double max)
- void SDH::ToRange (int n, double *v, double const *min, double const *max)
- void SDH::ToRange (std::vector< double > &v, std::vector< double > const &min, std::vector< double > const &max)
- double SDH::Approx (double a, double b, double eps)
- bool SDH::Approx (int n, double *a, double *b, double *eps)
- double SDH::DegToRad (double d)
- double SDH::RadToDeg (double r)
- void SDH::SleepSec (double t)
- template<typename Function, typename Tp>
  void SDH::apply (Function f, Tp &sequence)
- template<typename Function, typename InputIterator>
  Function SDH::apply (Function f, InputIterator first, InputIterator last)
- template<typename Function, typename Tp>
  Tp SDH::map (Function f, Tp sequence)
- template<typename T>
  std::ostream & SDH::operator<< (std::ostream &stream, std::vector< T > v)

Variables

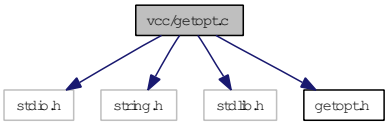
- static double SDH::M_PI = 4.0*atan(1.0)

11.53 sdhlibrary_cpp.dox File Reference

11.54 vcc/getopt.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "getopt.h"
```

Include dependency graph for getopt.c:



Defines

- #define GETOPT_INTERFACE_VERSION 2
- #define _(msgid) (msgid)
- #define SWAP_FLAGS(ch1, ch2)
- #define NONOPTION_P (argv[optind][0] != '-' || argv[optind][1] == '\0')

Enumerations

- enum { REQUIRE_ORDER, PERMUTE, RETURN_IN_ORDER }

Functions

- static char * my_index (const char *str, int chr)
- static void exchange (char **argv)
- static const char * _getopt_initialize (int argc, char *const *argv, const char *optstring)
- int _getopt_internal (int argc, char *const *argv, const char *optstring, const struct option *longopts, int *longind, int long_only)
- int getopt (int argc, char *const *argv, const char *optstring)

Variables

- char * optarg = NULL
- int optind = 1
- int __getopt_initialized = 0
- static char * nextchar
- int opterr = 1
- int optopt = '?'
- static enum { ... } ordering
- static char * posixly_correct
- static int first_nonopt
- static int last_nonopt

11.54.1 Define Documentation

11.54.1.1 #define _(msgid) (msgid)

11.54.1.2 #define GETOPT_INTERFACE_VERSION 2

11.54.1.3 #define NONOPTION_P (argv[optind][0] != '-' || argv[optind][1] == '\0')

11.54.1.4 #define SWAP_FLAGS(ch1, ch2)

11.54.2 Enumeration Type Documentation

11.54.2.1 anonymous enum

Enumerator:

REQUIRE_ORDER

PERMUTE

RETURN_IN_ORDER

11.54.3 Function Documentation

11.54.3.1 static const char* _getopt_initialize (int *argc*, char *const * *argv*, const char * *optstring*) [static]

11.54.3.2 int _getopt_internal (int *argc*, char *const * *argv*, const char * *optstring*, const struct option * *longopts*, int * *longind*, int *long_only*)

11.54.3.3 static void exchange (char ** *argv*) [static]

11.54.3.4 int getopt (int *argc*, char *const * *argv*, const char * *optstring*)

11.54.3.5 static char* my_index (const char * *str*, int *chr*) [static]

11.54.4 Variable Documentation

11.54.4.1 int __getopt_initialized = 0

11.54.4.2 int first_nonopt [static]

11.54.4.3 int last_nonopt [static]

11.54.4.4 char* nextchar [static]

11.54.4.5 char* optarg = NULL

11.54.4.6 int opterr = 1

11.54.4.7 int optind = 1

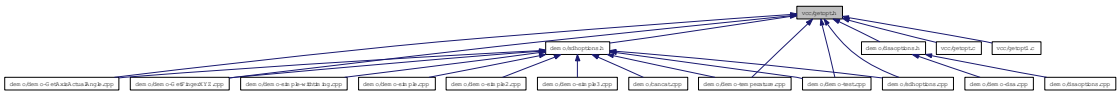
11.54.4.8 int optopt = '?'

11.54.4.9 enum { ... } ordering [static]

11.54.4.10 char* posixly_correct [static]

11.55 vcc/getopt.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- struct option

Defines

- #define _GETOPT_H 1
- #define no_argument 0
- #define required_argument 1
- #define optional_argument 2

Functions

- int getopt ()
- int getopt_long (int argc, char ∗const ∗argv, const char ∗shortopts, const struct option ∗longopts, int ∗longind)
- int getopt_long_only (int argc, char ∗const ∗argv, const char ∗shortopts, const struct option ∗longopts, int ∗longind)
- int _getopt_internal (int argc, char ∗const ∗argv, const char ∗shortopts, const struct option ∗longopts, int ∗longind, int long_only)

Variables

- char ∗ optarg
- int optind
- int opterr
- int optopt

11.55.1 Define Documentation

11.55.1.1 #define _GETOPT_H 1

11.55.1.2 #define no_argument 0

11.55.1.3 #define optional_argument 2

11.55.1.4 #define required_argument 1

11.55.2 Function Documentation

11.55.2.1 int _getopt_internal (int *argc*, char ∗const ∗ *argv*, const char ∗ *shortopts*, const struct option ∗ *longopts*, int ∗ *longind*, int *long_only*)

11.55.2.2 int getopt ()

11.55.2.3 int getopt_long (int *argc*, char ∗const ∗ *argv*, const char ∗ *shortopts*, const struct option ∗ *longopts*, int ∗ *longind*)

11.55.2.4 int getopt_long_only (int *argc*, char ∗const ∗ *argv*, const char ∗ *shortopts*, const struct option ∗ *longopts*, int ∗ *longind*)

11.55.3 Variable Documentation

11.55.3.1 char∗ optarg

11.55.3.2 int opterr

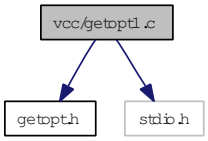
11.55.3.3 int optind

11.55.3.4 int optopt

11.56 vcc/getopt1.c File Reference

```
#include "getopt.h"
#include <stdio.h>
```

Include dependency graph for getopt1.c:



Defines

- #define GETOPT_INTERFACE_VERSION 2
- #define NULL 0

Functions

- int getopt_long (int argc, char ∗const ∗argv, const char ∗options, const struct option ∗long_options, int ∗opt_index)
- int getopt_long_only (int argc, char ∗const ∗argv, const char ∗options, const struct option ∗long_options, int ∗opt_index)

11.56.1 Define Documentation

11.56.1.1 #define GETOPT_INTERFACE_VERSION 2

11.56.1.2 #define NULL 0

11.56.2 Function Documentation

11.56.2.1 int getopt_long (int *argc*, char ∗const ∗ *argv*, const char ∗ *options*, const struct option ∗ *long_options*, int ∗ *opt_index*)

11.56.2.2 int getopt_long_only (int *argc*, char ∗const ∗ *argv*, const char ∗ *options*, const struct option ∗ *long_options*, int ∗ *opt_index*)

Index