Indradrive API

0.5

Generated by Doxygen 1.8.13

Contents

1 Main Page

Author

"hokyo" (Robert Hahn)

1.1 Introduction

The Indradrive API provides an universal programming interface to the Indradrive M devices. A dedicated DLL (IndradriveAPI.dll, or IndradriveAPI-LV.dll for LabVIEW) handles the user inputs and converts them to SIS protocol telegrams. These telegrams are transfered to the Indradrive device via RS232 interface (refer to Indradrive User's Manual for more information). The API uses the reply telegram to extract the required data or identifies potentials errors and provides it back to the user.

1.1.1 Drive modes

The API is designed to support two dedicated drive modes:

- Speed Control
- Sequencer

1.1.1.1 Speed Control

The principle of the Speed Control is depicted below:

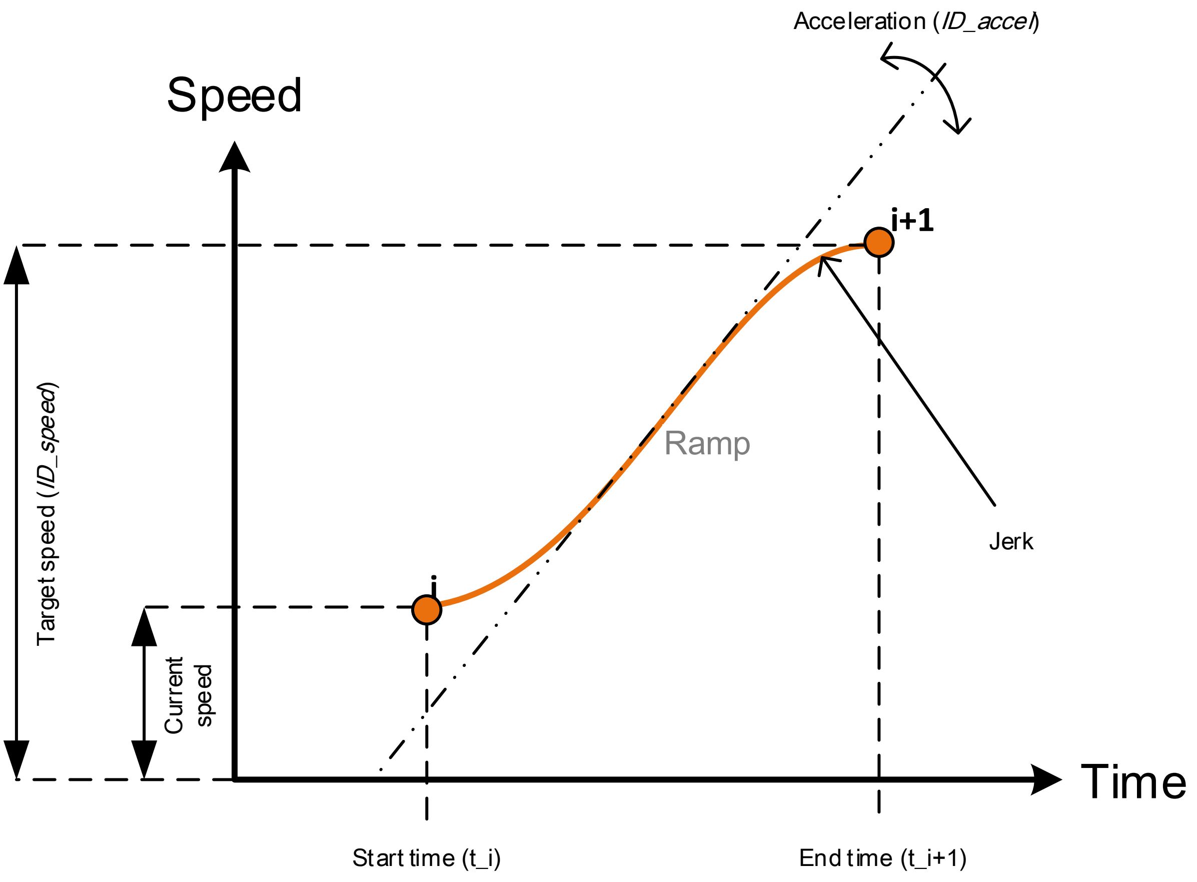


Figure 1 Principle of Speed Control

Baiscally, Speed Control offers non-realtime-capable way to quickly setup a new kinematic point (controlled via speed and acceleration).

Based on the requested speed and acceleration, the motor connected to the Indradrive system is cranking or down to the respective kinematic point.

The time between providing the data to the API and reaction of the motor depends on the Operating System (most likely Windows), calculation and creation of the SIS telegram and the baudrate to transfer the telegram. The time to go from the current kinematic point to the requested kinematic point can be determined as the following:

$$t = t_{i+1} - t_i = \frac{v_\text{target} - v_\text{current}}{a}$$

whereas a is the acceleration and $v_\text{target} - v_\text{current}$ the difference between current and targeted speed.

Remarks

The Speed Control drive mode cannot be used for real-time applications, since the jitter caused by OS and telegram transmission is unpredictable. Use the Sequencer drive mode for real-time applications instead.

The Speed Control drive mode is properly controlled in the following order:

1. Check the current drive mode by using get_drivemode()
   - If drive mode "Sequencer" is selected, proceed like this:
     (a) Check, if Indradrive is in "bb" operation state by using get_opstate()
     (b) Call speedcontrol_activate()
   - If drive mode "Speed Control" is selected, do not do anything and proceed with the next point
2. Initialize the right units by using speedcontrol_init()
3. Write the target kinematic point by using speedcontrol_write()

Attention

Speed Control commands the Indradrive to control the next kinematic point. This kinematic operates continuously until the next kinematic point is given or the emergency brake has been used. There is no automatic or time-limited stop system implemented.

1.1.1.2 Sequencer

The principle of the Sequencer is depicted below:

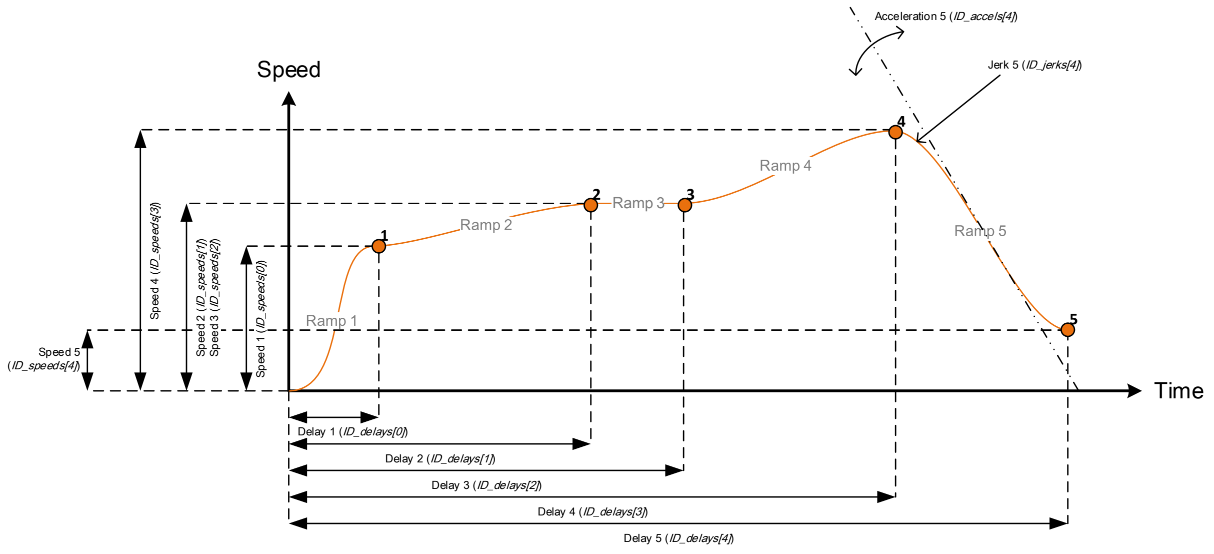


Figure 2 Principle of Sequencer

Sequencer offers real-time capable operation of a pre-programmed kinematic sequence upon receiving a trigger signal. Thus, Sequencer can be used if operations in a time-critical application is required.

The Sequencer routine is implemented into Indradrive's built-in PLC. If the routine is neither properly programmed nor running, the Sequencer operation mode is not working correctly.

In contrast to Speed Control, the Sequencer will be pre-programmed with a specific kinematic sequence (an example is shown in the figure above). Upon receiving an hardware or a software trigger, the Sequencer routine within the PLC immediately starts operating based on the first given kinematic point. After the pre-programmed elapsed time delay, the next kinematic point will be operated accordingly. As soon as the last kinematic point has been processed, the Indradrive motor goes back into standstill state (stop mode).

Attention

> If the PLC routine for the Sequencer is neither properly programmed nor running, the Sequencer drive mode cannot correctly operate.

Planning the kinematic sequence premises some calculations to be done for the jerk, if the delay, speed and acceleration is know for each sequence element. The following formula can be used for calculing the respective jerk, r:

$$r_i = \frac{a_i^2}{a_i(t_i - t_{i-1}) - v_i}$$

whereas $t_i - t_{i-1}$ is the Delay i to get from the previous kinematic point to the next requested kinematic point, a_i is the acceleration and v_i is the speed.

The Sequencer drive mode is properly controlled in the following order:

1. Check the current drive mode by using get_drivemode()
   - If drive mode "Speed Control" is selected, proceed like this:
     (a) Check, if Indradrive is in "bb" operation state by using get_opstate()
     (b) Call sequencerl_activate()
   - If drive mode "Sequencer" is selected, do not do anything and proceed with the next point
2. Initialize the right units by using sequencer_init()
3. Write the whole kinematic sequence by using sequencer_write()
4. Trigger the operation by using sequencer_softtrigger(), or use the hardware trigger (refer to Indradrive's User's Manual)

1.1.2 API Modules

As an overview, the API provides following modules:

Module	Description
Fundamentals	Provides functions for communication establishment
Status	Get information for diagnostic, drive modes, operation states, or even actual speed information
Configuration	Setting up essential required configurations
Sequencer	Programming functions for "Sequencer" drive mode
Speed Control	Programming functions for "Speed Control" drive mode

1.2 Installation

The API package consists of:

- IndradriveAPI.dll, or IndradriveAPI-LV.dll (for LabVIEW)
- msvcp140.dll
- vcruntime140.dll

Installation is very easy, if IndradriveAPI.dll, or IndradriveAPI-LV.dll is already present: Just copy all the mentioned DLLs from above into your binary folder, where your target application will be started from.

If the DLL IndradriveAPI.dll, or IndradriveAPI-LV.dll is missing, you have to compile the respective file first by doing the following steps:

1. Install Visual Studio 2015, or later (alternatively, install Visual Studio 2015 Express for Desktop)
2. Fetch the source code repository
3. Open the text editor of your choice and copy in the following text:
   - If you have LabVIEW installed on your computer, use this code (and adjust the cintools folder to your LabVIEW version)

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
<ImportGroup Label="PropertySheets" />
<PropertyGroup Label="UserMacros" />
<PropertyGroup />
<ItemDefinitionGroup>
<ClCompile>
<AdditionalIncludeDirectories>C:\Program Files\National Instruments\LabVIEW 2015\cintools;sis;serial;..\..\
      sis;..\..\serial;..\..;..;.;%(AdditionalIncludeDirectories)</AdditionalIncludeDirectories>
</ClCompile>
<Link>
<AdditionalLibraryDirectories>C:\Program Files\National Instruments\LabVIEW 2015\cintools;serial;%(
      AdditionalLibraryDirectories)</AdditionalLibraryDirectories>
<AdditionalDependencies>%(AdditionalDependencies)</AdditionalDependencies>
</Link>
</ItemDefinitionGroup>
<ItemGroup />
</Project>
```

   - If LabVIEW is not installed on your computer, use this code:

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion = "4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
<ImportGroup Label="PropertySheets" />
<PropertyGroup Label="UserMacros" />
<PropertyGroup />
<ItemDefinitionGroup>
<ClCompile>
<AdditionalIncludeDirectories>sis;serial;..\..\sis;..\..\serial;..\..;..;.;%(AdditionalIncludeDirectories)
      </AdditionalIncludeDirectories>
</ClCompile>
<Link>
<AdditionalLibraryDirectories>serial;%(AdditionalLibraryDirectories)</AdditionalLibraryDirectories>
<AdditionalDependencies>%(AdditionalDependencies)</AdditionalDependencies>
</Link>
</ItemDefinitionGroup>
<ItemGroup />
</Project>
```

4. Save this file as `UserDirectories.props` to the root directory of the source code (same level as `IndradriveAPI.vcxproj`)
5. Open the Visual Studio solution called `Indradrive.sln`
6. Choose configuration "Release" or "ReleaseLabview" (for LabView specific build)
7. Build the solution
   - For "Release", the final DLLs are located in the bin/ folder
   - For "ReleaseLabview", the final DLL are located in the ../ folder

1.3 Usage

1.3.1 API Function Overview

The following tables provides an overview of exported functions that can be accessed through the API DLL:

Module	API function	Brief description
Fundamentals	init()	Creates API reference.
Fundamentals	open()	Opens the communication port to the Indradrive device.
Fundamentals	close()	Closes the communication port at the Indradrive device.
Sequencer	sequencer_activate()	Activates the drive mode "Sequencer".
Sequencer	sequencer_init()	Initializes limits and sets the right scaling/unit factors for operation of "Sequencer" drive mode.
Sequencer	sequencer_write()	Writes the whole run sequence into the device.
Sequencer	sequencer_softtrigger()	Software-Trigger to start operation of the "Sequencer" drive mode.
Sequencer	sequencer_hardtrigger()	Hardware-Trigger to start operation of the "Sequencer" drive mode.
Sequencer	sequencer_getstatus()	Get the status of the "Sequencer" drive mode.
Speed Control	speedcontrol_activate()	Activates the drive mode "Speed Control".
Speed Control	speedcontrol_init()	Initializes limits and sets the right scaling/unit factors for operation of "Speed Control" drive mode.
Speed Control	speedcontrol_write()	Writes the current kinematic (speed and acceleration) into the device.
Configuration	set_stdenvironment()	Sets the proper unit and language environment.
Status	get_drivemode()	Retrieve information about the drive mode: Speed Control or Sequencer.
Status	get_opstate()	Retrieve information about the operation states: bb, Ab, or AF.
Status	get_speed()	Gets the actual rotation speed.
Status	get_diagnostic_msg()	Gets diagnostic message string of the current Indradrive status.
Status	get_diagnostic_num()	Gets diagnostic number of the current Indradrive status.
Status	clear_error()	Clears a latched error in the Indradrive device.

1.4 Examples

This sections gives some examples for C# and Python. However, through the nature of DLL, the API can be also called by other programming languages and development environments, such as LabVIEW, Matlab, etc.

1.4.1 C# Examples

The following code defines a C# class than can be copied in into a seperated .cs file. The `Indradrive` is accessible within the WpfApplication1 namespace (or whatever namespace you are writing).

1.4.2 Python Examples

2 Todo List

Member SISProtocol::BAUDRATE

This enum is not yet implemented into the code.

3 Bug List

Member sequencer_write (SISProtocol ∗ID_ref, double_t ID_speeds[], double_t ID_accels[], double_t ID_↩ jerks[], uint32_t ID_delays[], const uint16_t ID_set_length, ErrHandle ID_err=ErrHandle())

List length will not be extended automatically. In case of list length is set too short, programming of all values might fail. This may cause an improper operation of the "Sequencer" drive mode.

Member sequencer_write (SISProtocol ∗ID_ref, double_t ID_speeds[], double_t ID_accels[], double_t ID_↩ jerks[], uint32_t ID_delays[], const uint16_t ID_set_length, ErrHandle ID_err=ErrHandle())

List length will not be extended automatically. In case of list length is set too short, programming of all values might fail. This may cause an improper operation of the "Sequencer" drive mode.

4 Namespace Documentation

4.1 TGM Namespace Reference

Grouping structs/enums/unions for a SIS Telegram.

Namespaces

- Bitfields

  Grouping unions that merge together both raw and structured information.
- Commands

  Grouping SIS Telegram Payload struct definitions for commands.
- Reactions

  Grouping SIS Telegram Payload struct definitions for reception.

Classes

- struct Bytestream

  Container for Telegram in raw Bytes.
- struct Data

  Struct to hold payload Bytes in a command payload.
- struct Header

  The Telegram Header contains all information required for conducting orderly telegram traffic..
- struct HeaderExt

  Extended Telegram Header to be used for Routing and Sequential Telegrams.
- union Map

  Templated mapping union to transfer raw TGM Bytes from/to specialized Bytes class.

Typedefs

- typedef struct TGM::Data Data

  Struct to hold payload Bytes in a command payload.
- typedef struct TGM::Bytestream Bytestream

  Container for Telegram in raw Bytes.
- typedef struct TGM::Header Header

  The Telegram Header contains all information required for conducting orderly telegram traffic..
- typedef TGM::HeaderExt HeaderExt

  Extended Telegram Header to be used for Routing and Sequential Telegrams.

Enumerations

- enum HeaderType : BYTE { TypeCommand, TypeReaction }

  Values that represent Telegram header types.
- enum SercosParamVar : BYTE { SercosParamS, SercosParamP }

  Values that represent SERCOS Parameter variants.
- enum SercosDatablock : BYTE {
  Datablock_ChannelNotActive, Datablock_IdentNumber, Datablock_Name, Datablock_Attribute,
  Datablock_Unit, Datablock_Minval, Datablock_Maxval, Datablock_OperationData }

  Values that represent SERCOS Parameter Bytes block to be processed.
- enum SercosCommandrequest : BYTE { Commandrequest_NotSet = 0x0, Commandrequest_Cancel = 0x1,
  Commandrequest_Set = 0x3 }

  Values that represent SERCOS command requests value.
- enum SercosCommandstatus : BYTE {
  Commandstatus_NotSet = 0x0, Commandstatus_OK = 0x3, Commandstatus_Canceled = 0x5,
  Commandstatus_Busy = 0x7,
  Commandstatus_Error = 0xF }

  Values that represent SERCOS command status.
- enum SercosTxProgress : BYTE { TxProgress_InProgress, TxProgress_Final }

  Values that represent information in the SIS Telegram's Control Byte about the type of the Command Telegram or Reception Telegram.
- enum SercosDatalen : UINT32 {
  Datalen_Res1 = 0b000, Datalen_2ByteParam = 0b001, Datalen_4ByteParam = 0b010, Datalen_8ByteParam
  = 0b011,
  Datalen_1ByteList = 0b100, Datalen_2ByteList = 0b101, Datalen_4ByteList = 0b110, Datalen_8ByteList =
  0b111 }

  Values that represent the information stored in a Parameter attributes (can be retrieved by attribute datablock).

4.1.1 Detailed Description

Grouping structs/enums/unions for a SIS Telegram.

4.1.2 Typedef Documentation

4.1.2.1 Data

```
typedef struct TGM::Data TGM::Data
```

Struct to hold payload Bytes in a command payload.

Payload Bytes is third part of a regular Telegram: Header + Payload Bytes + Payload header.

4.1.2.2 Bytestream

```
typedef struct TGM::Bytestream TGM::Bytestream
```

Container for Telegram in raw Bytes.

4.1.2.3 Header

```
typedef struct TGM::Header TGM::Header
```

The Telegram Header contains all information required for conducting orderly telegram traffic..

4.1.2.4 HeaderExt

```
typedef TGM::HeaderExt TGM::HeaderExt
```

Extended Telegram Header to be used for Routing and Sequential Telegrams.

See also

Header

4.1.3 Enumeration Type Documentation

4.1.3.1 HeaderType

```
enum TGM::HeaderType : BYTE
```

Values that represent Telegram header types.

Enumerator

TypeCommand	Telegram for command.
TypeReaction	Telegram for reception.

Definition at line 16 of file Telegrams_Bitfields.h.

4.1.3.2 SercosParamVar

```
enum TGM::SercosParamVar :  BYTE
```

Values that represent SERCOS Parameter variants.

Enumerator

SercosParamS	SERCOS S Parameter (e.g. S-0-xxxx)
SercosParamP	SERCOS P Parameter (e.g. P-0-xxxx)

Definition at line 24 of file Telegrams_Bitfields.h.

4.1.3.3 SercosDatablock

```
enum TGM::SercosDatablock :  BYTE
```

Values that represent SERCOS Parameter Bytes block to be processed.

Using this in the Telegram's control byte will inform or request what is/should stored in the payload.

Enumerator

Datablock_ChannelNotActive	Channel not active (read-only)
Datablock_IdentNumber	Getting the SERCOS parameter identification number (read-only)
Datablock_Name	Getting the SERCOS parameter name (read-only)
Datablock_Attribute	Getting the SERCOS parameter Datablock (read-only). Response be represented by SercosParamAttribute.
Datablock_Unit	Getting the SERCOS parameter unit information.
Datablock_Minval	Getting the SERCOS parameter possible min Value.
Datablock_Maxval	Getting the SERCOS parameter possible max Value.
Datablock_OperationData	Getting the SERCOS operation Bytes (actual content of the parameter)

Definition at line 33 of file Telegrams_Bitfields.h.

4.1.3.4 SercosCommandrequest

```
enum TGM::SercosCommandrequest :  BYTE
```

Values that represent SERCOS command requests value.

Mainly used for write_parameter() in SISProtocol class to initiate or cancel processing a command (e.g. entering parametrization level).

See also

SISProtocol
write_parameter()

Enumerator

Commandrequest_NotSet	
Commandrequest_Cancel	
Commandrequest_Set	

Definition at line 58 of file Telegrams_Bitfields.h.

4.1.3.5 SercosCommandstatus

```
enum TGM::SercosCommandstatus : BYTE
```

Values that represent SERCOS command status.

Mainly used for get_parameter_status() in SISProtocol class to retrieve feedback of the command processing (e.g. entering parametrization level finished?).

See also

SISProtocol
get_parameter_status()

Enumerator

Commandstatus_NotSet	
Commandstatus_OK	
Commandstatus_Canceled	
Commandstatus_Busy	
Commandstatus_Error	

Definition at line 69 of file Telegrams_Bitfields.h.

4.1.3.6 SercosTxProgress

```
enum TGM::SercosTxProgress : BYTE
```

Values that represent information in the SIS Telegram's Control Byte about the type of the Command Telegram or Reception Telegram.

See also

SercosParamControl

Enumerator

TxProgress_InProgress	An enum constant representing that Telegram will be followed by another Telegram.
TxProgress_Final	An enum constant representing that this is a single Telegram (not followed by another Telegram).

Definition at line 81 of file Telegrams_Bitfields.h.

4.1.3.7 SercosDatalen

```
enum TGM::SercosDatalen : UINT32
```

Values that represent the information stored in a Parameter attributes (can be retrieved by attribute datablock).

See also

SercosParamAttribute

Enumerator

Datalen_Res1	
Datalen_2ByteParam	
Datalen_4ByteParam	
Datalen_8ByteParam	
Datalen_1ByteList	
Datalen_2ByteList	
Datalen_4ByteList	
Datalen_8ByteList	

Definition at line 90 of file Telegrams_Bitfields.h.

4.2 TGM::Bitfields Namespace Reference

Grouping unions that merge together both raw and structured information.

Classes

- struct HeaderControl

  Control byte consisting of several bit fields. Size: 8 bit.
- struct SercosParamAttribute

  Attribute for a SERCOS parameter that is callable via SercosDatablock.
- struct SercosParamControl

  The control byte specifies how a Bytes block element of a parameter is accessed.
- struct SercosParamIdent

  Identification of the parameter. Size: 16 bit.

Typedefs

- typedef struct TGM::Bitfields::HeaderControl HeaderControl

  Control byte consisting of several bit fields. Size: 8 bit.
- typedef struct TGM::Bitfields::SercosParamControl SercosParamControl

  The control byte specifies how a Bytes block element of a parameter is accessed.
- typedef struct TGM::Bitfields::SercosParamIdent SercosParamIdent

  Identification of the parameter. Size: 16 bit.
- typedef struct TGM::Bitfields::SercosParamAttribute SercosParamAttribute

  Attribute for a SERCOS parameter that is callable via SercosDatablock.

4.2.1 Detailed Description

Grouping unions that merge together both raw and structured information.

4.2.2 Typedef Documentation

4.2.2.1 HeaderControl

```
typedef struct TGM::Bitfields::HeaderControl TGM::Bitfields::HeaderControl
```

Control byte consisting of several bit fields. Size: 8 bit.

4.2.2.2 SercosParamControl

```
typedef struct TGM::Bitfields::SercosParamControl TGM::Bitfields::SercosParamControl
```

The control byte specifies how a Bytes block element of a parameter is accessed.

The control byte is read out of the command telegram and copied into the response telegram.

4.2.2.3 SercosParamIdent

```
typedef struct TGM::Bitfields::SercosParamIdent TGM::Bitfields::SercosParamIdent
```

Identification of the parameter. Size: 16 bit.

4.2.2.4 SercosParamAttribute

```
typedef struct TGM::Bitfields::SercosParamAttribute TGM::Bitfields::SercosParamAttribute
```

Attribute for a SERCOS parameter that is callable via SercosDatablock.

See also

SercosDatablock

4.3 TGM::Commands Namespace Reference

Grouping SIS Telegram Payload struct definitions for commands.

Classes

- struct SercosList

  Sercos Command Telegram used for reading/writing single elements in lists from/to slave.
- struct SercosParam

  Sercos Command Telegram used for reading/writing single parameter from/to slave.
- struct Subservice

  Representation of the PAYLOAD for a Subservice command.

Typedefs

- typedef struct TGM::Commands::Subservice Subservice

  Representation of the PAYLOAD for a Subservice command.
- typedef struct TGM::Commands::SercosParam SercosParam

  Sercos Command Telegram used for reading/writing single parameter from/to slave.
- typedef struct TGM::Commands::SercosList SercosList

  Sercos Command Telegram used for reading/writing single elements in lists from/to slave.

4.3.1 Detailed Description

Grouping SIS Telegram Payload struct definitions for commands.

4.3.2 Typedef Documentation

4.3.2.1 Subservice

```
typedef struct TGM::Commands::Subservice TGM::Commands::Subservice
```

Representation of the PAYLOAD for a Subservice command.

A Command Telegram is for regular subservices, such communication init, or device identification. User for master communication (active communicator).

4.3.2.2 SercosParam

```
typedef struct TGM::Commands::SercosParam TGM::Commands::SercosParam
```

Sercos Command Telegram used for reading/writing single parameter from/to slave.

4.3.2.3 SercosList

```
typedef struct TGM::Commands::SercosList TGM::Commands::SercosList
```

Sercos Command Telegram used for reading/writing single elements in lists from/to slave.

4.4 TGM::Reactions Namespace Reference

Grouping SIS Telegram Payload struct definitions for reception.

Classes

- struct SercosList

  Sercos Command Telegram used for reading/writing single elements in lists from/to slave..
- struct SercosParam

  Representation of the payload for a Sercos Parameter reaction.
- struct Subservice

  Representation of the payload for a Subservice reaction.

Typedefs

- typedef struct TGM::Reactions::Subservice Subservice

  Representation of the payload for a Subservice reaction.
- typedef struct TGM::Reactions::SercosParam SercosParam

  Representation of the payload for a Sercos Parameter reaction.
- typedef struct TGM::Reactions::SercosList SercosList

  Sercos Command Telegram used for reading/writing single elements in lists from/to slave..

4.4.1 Detailed Description

Grouping SIS Telegram Payload struct definitions for reception.

4.4.2 Typedef Documentation

4.4.2.1 Subservice

```
typedef struct TGM::Reactions::Subservice TGM::Reactions::Subservice
```

Representation of the payload for a Subservice reaction.

A Reaction Telegram is for regular subservices, such communication init, or device identification. This telegram is responded after successful execution of previous Command Telegram.

4.4.2.2 SercosParam

```
typedef struct TGM::Reactions::SercosParam TGM::Reactions::SercosParam
```

Representation of the payload for a Sercos Parameter reaction.

A Reaction Telegram is for regular subservices, such communication init, or device identification. This telegram is responded after successful execution of previous Command Telegram.

4.4.2.3 SercosList

```
typedef struct TGM::Reactions::SercosList TGM::Reactions::SercosList
```

Sercos Command Telegram used for reading/writing single elements in lists from/to slave..

5 Class Documentation

5.1 TGM::Bytestream Struct Reference

Container for Telegram in raw Bytes.

Public Member Functions

- Bytestream ()

  Default constructor.
- void clear ()

  Clears this object to its blank/initial state.

Public Attributes

- BYTE Bytes [TGM_SIZEMAX]

  The raw Bytes. Size: 254 bytes.

5.1.1 Detailed Description

Container for Telegram in raw Bytes.

Definition at line 210 of file Telegrams.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Bytestream()

```
TGM::Bytestream::Bytestream ( )  [inline]
```

Default constructor.

Definition at line 216 of file Telegrams.h.

5.1.3 Member Function Documentation

5.1.3.1 clear()

```
void TGM::Bytestream::clear ( )  [inline]
```

Clears this object to its blank/initial state.

Definition at line 219 of file Telegrams.h.

5.1.4 Member Data Documentation

5.1.4.1 Bytes

```
BYTE TGM::Bytestream::Bytes[TGM_SIZEMAX]
```

The raw Bytes. Size: 254 bytes.

Definition at line 213 of file Telegrams.h.

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

- Telegrams.h

5.2 TGM::Data Struct Reference

Struct to hold payload Bytes in a command payload.

Public Member Functions

- Data (std::vector< BYTE > _data=std::vector< BYTE >())

  Default constructor.
- Data (UINT8 _data)

  Constructor.
- Data (UINT16 _data)

  Constructor.
- Data (UINT32 _data)

  Constructor.
- Data (UINT64 _data)

  Constructor.
- BYTE at (UINT32 _idx)

  Ats the given index.
- std::vector< BYTE > toVector ()

  Converts this object to a vector.
- UINT64 toUINT64 ()

  Converts this object to an uint 64.
- UINT32 toUINT32 ()

  Converts this object to an uint 32.
- UINT16 toUINT16 ()

Converts this object to an uint 16.
- UINT8 toUINT8 ()

  Converts this object to an uint 8.
- BYTE toBYTE ()

  Converts this object to a byte.
- void clear ()

  Clears this object to its blank/initial state.
- Data & operator<< (const BYTE &rhs)

  Bitwise left shift operator.
- size_t get_size ()

  Gets the size.
- void set_size (size_t _size)

  Sets a size.

Public Attributes

- BYTE Bytes [TGM_SIZEMAX_PAYLOAD]

  Actual payload Bytes [TGM_SIZEMAX_PAYLOAD].
- size_t Size

  Size of the payload Bytes.

5.2.1 Detailed Description

Struct to hold payload Bytes in a command payload.

Payload Bytes is third part of a regular Telegram: Header + Payload Bytes + Payload header.

Definition at line 29 of file Telegrams.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Data() [1/5]

```
TGM::Data::Data (
             std::vector< BYTE > _data = std::vector<BYTE>() )  [inline]
```

Default constructor.

Parameters

_PayloadData	(Optional) The data vector.

Definition at line 39 of file Telegrams.h.

5.2.2.2 Data() [2/5]

```
TGM::Data::Data (
             UINT8 _data )  [inline]
```

Constructor.

Parameters

_PayloadData	Single data byte.

Definition at line 52 of file Telegrams.h.

5.2.2.3 Data() [3/5]

```
TGM::Data::Data (
             UINT16 _data )  [inline]
```

Constructor.

Parameters

_PayloadData	Single data word (2 bytes).

Definition at line 62 of file Telegrams.h.

5.2.2.4 Data() [4/5]

```
TGM::Data::Data (
             UINT32 _data )  [inline]
```

Constructor.

Parameters

_PayloadData	Single data integer (4 bytes).

Definition at line 73 of file Telegrams.h.

5.2.2.5 Data() [5/5]

```
TGM::Data::Data (
             UINT64 _data )  [inline]
```

Constructor.

Parameters

_PayloadData	Single UINT64 data (8 bytes).

Definition at line 86 of file Telegrams.h.

5.2.3 Member Function Documentation

5.2.3.1 at()

```
BYTE TGM::Data::at (
            UINT32 _idx )  [inline]
```

Ats the given index.

Parameters

_idx	The index.

Returns

Data byte.

Definition at line 105 of file Telegrams.h.

5.2.3.2 toVector()

```
std::vector<BYTE> TGM::Data::toVector ( )  [inline]
```

Converts this object to a vector.

Returns

This object as a std::vector<BYTE>

Definition at line 113 of file Telegrams.h.

5.2.3.3 toUINT64()

```
UINT64 TGM::Data::toUINT64 ( )  [inline]
```

Converts this object to an uint 64.

Returns

This object as an UINT64.

Definition at line 126 of file Telegrams.h.

5.2.3.4 toUINT32()

```
UINT32 TGM::Data::toUINT32 ( )  [inline]
```

Converts this object to an uint 32.

Returns

This object as an UINT32.

Definition at line 139 of file Telegrams.h.

5.2.3.5 toUINT16()

```
UINT16 TGM::Data::toUINT16 ( )  [inline]
```

Converts this object to an uint 16.

Returns

This object as an UINT16.

Definition at line 152 of file Telegrams.h.

5.2.3.6 toUINT8()

```
UINT8 TGM::Data::toUINT8 ( )  [inline]
```

Converts this object to an uint 8.

Returns

This object as an UINT8.

Definition at line 165 of file Telegrams.h.

5.2.3.7 toBYTE()

```
BYTE TGM::Data::toBYTE ( )  [inline]
```

Converts this object to a byte.

Returns

This object as a BYTE.

Definition at line 173 of file Telegrams.h.

5.2.3.8 clear()

```
void TGM::Data::clear ( )  [inline]
```

Clears this object to its blank/initial state.

Definition at line 179 of file Telegrams.h.

5.2.3.9 operator<<()

```
Data& TGM::Data::operator<< (
             const BYTE & rhs )  [inline]
```

Bitwise left shift operator.

Parameters

rhs	The right hand side.

Returns

The shifted result.

Definition at line 190 of file Telegrams.h.

5.2.3.10 get_size()

```
size_t TGM::Data::get_size ( )  [inline]
```

Gets the size.

Returns

The size.

Definition at line 199 of file Telegrams.h.

5.2.3.11 set_size()

```
void TGM::Data::set_size (
             size_t _size )  [inline]
```

Sets a size.

Parameters

_size	The size.

Definition at line 204 of file Telegrams.h.

5.2.4 Member Data Documentation

5.2.4.1 Bytes

```
BYTE TGM::Data::Bytes[TGM_SIZEMAX_PAYLOAD]
```

Actual payload Bytes [TGM_SIZEMAX_PAYLOAD].

Definition at line 32 of file Telegrams.h.

5.2.4.2 Size

```
size_t TGM::Data::Size
```

Size of the payload Bytes.

Definition at line 34 of file Telegrams.h.

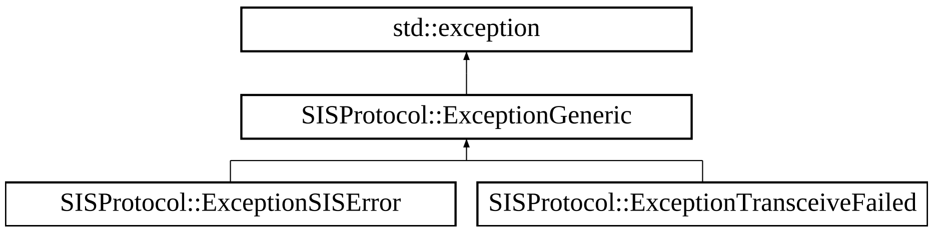
The documentation for this struct was generated from the following file:

- Telegrams.h

5.3 SISProtocol::ExceptionGeneric Class Reference

Generic exceptions for SIS protocol.

Inheritance diagram for SISProtocol::ExceptionGeneric:


std::exception
SISProtocol::ExceptionGeneric
SISProtocol::ExceptionSISError
SISProtocol::ExceptionTransceiveFailed

Public Member Functions

- ExceptionGeneric (int _status, const std::string _trace_log, bool _warning=false)
- virtual const char $*$ what () const throw ()
- int get_status ()

Public Attributes

- bool warning

Protected Attributes

- int m_status
- std::string m_message

5.3.1 Detailed Description

Generic exceptions for SIS protocol.

See also

std::exception

Definition at line 157 of file SISProtocol.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ExceptionGeneric()

```
SISProtocol::ExceptionGeneric::ExceptionGeneric (
             int _status,
             const std::string _trace_log,
             bool _warning = false )  [inline]
```

Definition at line 162 of file SISProtocol.h.

5.3.3 Member Function Documentation

5.3.3.1 what()

```
virtual const char* SISProtocol::ExceptionGeneric::what ( ) const throw )   [inline], [virtual]
```

Reimplemented in SISProtocol::ExceptionSISError, and SISProtocol::ExceptionTransceiveFailed.

Definition at line 172 of file SISProtocol.h.

5.3.3.2 get_status()

```
int SISProtocol::ExceptionGeneric::get_status ( )  [inline]
```

Definition at line 183 of file SISProtocol.h.

5.3.4 Member Data Documentation

5.3.4.1 warning

```
bool SISProtocol::ExceptionGeneric::warning
```

Definition at line 160 of file SISProtocol.h.

5.3.4.2 m_status

```
int SISProtocol::ExceptionGeneric::m_status  [protected]
```

Definition at line 186 of file SISProtocol.h.

5.3.4.3 m_message

```
std::string SISProtocol::ExceptionGeneric::m_message  [protected]
```

Definition at line 188 of file SISProtocol.h.

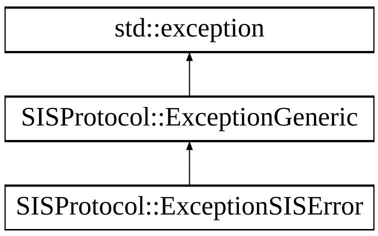
The documentation for this class was generated from the following file:

- SISProtocol.h

5.4 SISProtocol::ExceptionSISError Class Reference

Specific exception handling of SIS Protocol error codes.

Inheritance diagram for SISProtocol::ExceptionSISError:



Public Member Functions

- ExceptionSISError (int _status, int _code, const std::string _bytestream, bool _warning=false)
- ∼ExceptionSISError () throw ()
- virtual const char ∗ what () const throw ()
- int get_errorcode ()

Protected Attributes

- int m_errorcode
- std::string m_bytestream

Additional Inherited Members

5.4.1 Detailed Description

Specific exception handling of SIS Protocol error codes.

See also

SISProtocol::ExceptionGeneric

Definition at line 221 of file SISProtocol.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 ExceptionSISError()

```
SISProtocol::ExceptionSISError::ExceptionSISError (
             int _status,
             int _code,
             const std::string _bytestream,
             bool _warning = false )  [inline]
```

Definition at line 224 of file SISProtocol.h.

5.4.2.2 ∼ExceptionSISError()

```
SISProtocol::ExceptionSISError::~ExceptionSISError ( ) throw )   [inline]
```

Definition at line 234 of file SISProtocol.h.

5.4.3 Member Function Documentation

5.4.3.1 what()

```
virtual const char* SISProtocol::ExceptionSISError::what ( ) const throw )   [inline], [virtual]
```

Reimplemented from SISProtocol::ExceptionGeneric.

Definition at line 236 of file SISProtocol.h.

5.4.3.2 get_errorcode()

```
int SISProtocol::ExceptionSISError::get_errorcode ( )  [inline]
```

Definition at line 247 of file SISProtocol.h.

5.4.4 Member Data Documentation

5.4.4.1 m_errorcode

```
int SISProtocol::ExceptionSISError::m_errorcode  [protected]
```

Definition at line 250 of file SISProtocol.h.

5.4.4.2 m_bytestream

```
std::string SISProtocol::ExceptionSISError::m_bytestream  [protected]
```

Definition at line 251 of file SISProtocol.h.

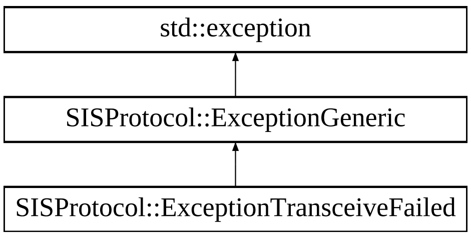
The documentation for this class was generated from the following file:

- SISProtocol.h

5.5 SISProtocol::ExceptionTransceiveFailed Class Reference

Specific exception handling of SIS Protocol transceiving failed.

Inheritance diagram for SISProtocol::ExceptionTransceiveFailed:


std::exception
SISProtocol::ExceptionGeneric
SISProtocol::ExceptionTransceiveFailed

Public Member Functions

- ExceptionTransceiveFailed (int _status, const std::string _message, bool _warning=false)
- ∼ExceptionTransceiveFailed () throw ()
- virtual const char ∗ what () const throw ()

Additional Inherited Members

5.5.1 Detailed Description

Specific exception handling of SIS Protocol transceiving failed.

See also

SISProtocol::ExceptionGeneric

Definition at line 194 of file SISProtocol.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 ExceptionTransceiveFailed()

```
SISProtocol::ExceptionTransceiveFailed::ExceptionTransceiveFailed (
            int _status,
            const std::string _message,
            bool _warning = false )  [inline]
```

Definition at line 197 of file SISProtocol.h.

5.5.2.2 ∼ExceptionTransceiveFailed()

```
SISProtocol::ExceptionTransceiveFailed::∼ExceptionTransceiveFailed ( ) throw )   [inline]
```

Definition at line 204 of file SISProtocol.h.

5.5.3 Member Function Documentation

5.5.3.1 what()

```
virtual const char* SISProtocol::ExceptionTransceiveFailed::what ( ) const throw )   [inline],
[virtual]
```

Reimplemented from SISProtocol::ExceptionGeneric.

Definition at line 206 of file SISProtocol.h.

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

- SISProtocol.h

5.6 GenericErrHandle Struct Reference

Generic error handle that is returned from each API function.

Public Member Functions

- GenericErrHandle (uint32_t _code=0, const char ∗_msg="")

    Constructor.
- void set (uint32_t _code, const char ∗_msg)

    Sets error code and error message.
- void set_msg (const char ∗_msg)

    Sets an error message.
- void set_code (uint32_t _code)

    Sets an error code.

Public Attributes

- uint32_t code

  Error code.
- char msg [2048]

  Error message [2048].

5.6.1 Detailed Description

Generic error handle that is returned from each API function.

In contrast to a LabVIEW error handle (LVErrorCluster_t) that includes a specific type of Error string, the Generic Error Handle (GenericErrHandle) consists of generic C types (int and char∗) for both error code and error message.

GenericErrHandle is used as pointer for all Indradrive API Functions (see API Function Overview).

Remarks

Depending on the USE_LABVIEW_ENV switch, the GenericErrHandle can be replaced by LStrHandle.

Definition at line 27 of file errors.h.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 GenericErrHandle()

```
GenericErrHandle::GenericErrHandle (
             uint32_t _code = 0,
             const char * _msg = "" )  [inline]
```

Constructor.

Parameters

_code	(Optional) Error code. Can be later on set also via set() function.
_msg	(Optional) Error message. Parameter will not be used.

Definition at line 38 of file errors.h.

5.6.3 Member Function Documentation

5.6.3.1 set()

```
void GenericErrHandle::set (
             uint32_t _code,
             const char * _msg )  [inline]
```

Sets error code and error message.

Parameters

_code	Error code.
_msg	Error message.

Definition at line 46 of file errors.h.

5.6.3.2 set_msg()

```
void GenericErrHandle::set_msg (
             const char * _msg )  [inline]
```

Sets an error message.

Parameters

_msg	Error message.

Definition at line 57 of file errors.h.

5.6.3.3 set_code()

```
void GenericErrHandle::set_code (
             uint32_t _code )  [inline]
```

Sets an error code.

Parameters

_code	Error code.

Definition at line 65 of file errors.h.

5.6.4 Member Data Documentation

5.6.4.1 code

```
uint32_t GenericErrHandle::code
```

Error code.

Definition at line 30 of file errors.h.

5.6.4.2 msg

```
char GenericErrHandle::msg[2048]
```

Error message [2048].

Definition at line 32 of file errors.h.

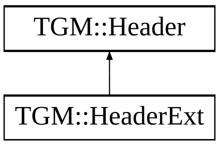
The documentation for this struct was generated from the following file:

- errors.h

5.7 TGM::Header Struct Reference

The Telegram Header contains all information required for conducting orderly telegram traffic..

Inheritance diagram for TGM::Header:



Public Member Functions

- Header (BYTE _addr_master=0, BYTE _addr_slave=0, BYTE _service=0, TGM::Bitfields::HeaderControl ↩ _cntrl=TGM::Bitfields::HeaderControl())

    Default constructor.
- BYTE get_sum (bool exclude_cs=true)

    Gets the sum without carry of all header bytes for checksum calculation.
- size_t get_size ()

    Gets the size.
- void set_DatL (size_t _payload_len)

    Sets length of Telegram, stored in DatL and DatLW (copy).
- size_t get_DatL ()

    Gets Telegram's length.
- void calc_checksum (TGM::Bytestream $*$_payload)

    Calculates the Telegram checksum, stored in CS.

Public Attributes

- BYTE StZ = 0x02

  Start symbol: STX (0x02).
- BYTE CS

  The checksum byte.
- BYTE DatL

  The length of the sub-sequential user Bytes and the variable part are in the frame protocol.
- BYTE DatLW

  Repetition of DatL takes place here.
- BYTE Cntrl

  Control byte consisting of several bit fields. Use TGM::Bitfields::Cntrl and toByte() for configuration.
- BYTE Service

  This specifies the service that the sender requests from the recipient or that the recipient has executed.
- BYTE AdrS

  Address of sender:
- BYTE AdrE

  Address of Recipient:

5.7.1 Detailed Description

The Telegram Header contains all information required for conducting orderly telegram traffic..

Definition at line 277 of file Telegrams.h.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 Header()

```
TGM::Header::Header (
             BYTE _addr_master = 0,
             BYTE _addr_slave = 0,
             BYTE _service = 0,
             TGM::Bitfields::HeaderControl _cntrl = TGM::Bitfields::HeaderControl() )  [inline]
```

Default constructor.

Parameters

_addr_master	(Optional) The address master id.
_addr_slave	(Optional) The address slave id.
_service	(Optional) The service id.
_cntrl	(Optional) The Control Byte, represented by TGM::Bitfields::HeaderControl.

See also

TGM::Bitfields::HeaderControl

Definition at line 343 of file Telegrams.h.

5.7.3 Member Function Documentation

5.7.3.1 get_sum()

```
BYTE TGM::Header::get_sum (
             bool exclude_cs = true )  [inline]
```

Gets the sum without carry of all header bytes for checksum calculation.

Parameters

exclude_cs	(Optional) true to exclude checksum from calculation.

Returns

The sum.

Definition at line 359 of file Telegrams.h.

5.7.3.2 get_size()

```
size_t TGM::Header::get_size ( )  [inline]
```

Gets the size.

Returns

The size.

Definition at line 371 of file Telegrams.h.

5.7.3.3 set_DatL()

```
void TGM::Header::set_DatL (
             size_t _payload_len )  [inline]
```

Sets length of Telegram, stored in DatL and DatLW (copy).

By default, the length of the telegram is defined by the payload length (head + Bytes).

Parameters

_payload_len	Length of the payload.

Definition at line 377 of file Telegrams.h.

5.7.3.4 get_DatL()

```
size_t TGM::Header::get_DatL ( )  [inline]
```

Gets Telegram's length.

Returns

The length of Telegram.

Definition at line 382 of file Telegrams.h.

5.7.3.5 calc_checksum()

```
void TGM::Header::calc_checksum (
            TGM::Bytestream * _payload )  [inline]
```

Calculates the Telegram checksum, stored in CS.

The calculated checksum will automatically assigned to CS. This function will use DatL parameter for the appropriate length determination.

Parameters

	_payload_len	Length of the payload.
in	*_payload*	Bytestream of payload (head + Bytes) with the raw Bytes.

Definition at line 390 of file Telegrams.h.

5.7.4 Member Data Documentation

5.7.4.1 StZ

```
BYTE TGM::Header::StZ = 0x02
```

Start symbol: STX (0x02).

Definition at line 280 of file Telegrams.h.

5.7.4.2 CS

```
BYTE TGM::Header::CS
```

The checksum byte.

It is generated by adding all sub-sequential telegram symbols as well as the start symbol StZ and concluding negation. In other words, the sum of all telegram symbols always equals 0 if the transmission was successful.

Definition at line 285 of file Telegrams.h.

5.7.4.3 DatL

```
BYTE TGM::Header::DatL
```

The length of the sub-sequential user Bytes and the variable part are in the frame protocol.

Up to 247 bytes (255 - 7{subaddresses} - 1{running telegram number}) user Bytes can be transmitted in one telegram.

Definition at line 289 of file Telegrams.h.

5.7.4.4 DatLW

```
BYTE TGM::Header::DatLW
```

Repetition of DatL takes place here.

The telegram length is generated from the DatLW and the fixed part of the frame protocol (byte 1-8), i.e. telegram length = DatLW + 8.

Definition at line 293 of file Telegrams.h.

5.7.4.5 Cntrl

```
BYTE TGM::Header::Cntrl
```

Control byte consisting of several bit fields. Use TGM::Bitfields::Cntrl and toByte() for configuration.

Definition at line 296 of file Telegrams.h.

5.7.4.6 Service

```
BYTE TGM::Header::Service
```

This specifies the service that the sender requests from the recipient or that the recipient has executed.

- 0x00 ... 0x0F General services:
- 0x00 User identification
- 0x01 Data transmission aborted
- 0x02 Flash operation
- 0x03 Initializing SIS communication
- 0x04 Executing a list of SIS services
- 0x0F Token passing
- 0x10 ... 0x7F temporarily reserved
- 0x80 ... 0x8F Special services for ECODRIVE
- 0x90 ... 0x9F Special services for SYNAX
- 0xA0 ... 0xAF Special services for MT - CNC or .MTC200
- 0xB0 ... 0xBF Special services for ISP200
- 0xC0 ... 0xCF Special services for CLC - GPS
- 0xD0 ... 0xDF Special services for HMI - System
- 0xE0 ... 0xEF Special services for DISC
- 0xF0 ... 0xFF temporarily reserved.

Definition at line 315 of file Telegrams.h.

5.7.4.7 AdrS

```
BYTE TGM::Header::AdrS
```

Address of sender:

- AdrS = [0..126]: specifies a single station
- AdrS = 127: Special address for a SIS master in case of service or emergencies (this address may not be used during active communication).

Definition at line 321 of file Telegrams.h.

5.7.4.8 AdrE

```
BYTE TGM::Header::AdrE
```

Address of Recipient:

- AdrE = [0..126]: specifies a single station,
- AdrE = 128: Special address for point-to-point communication (the recipient's response is not dependent on its actual station number with this special address).
- AdrE = [129..199]: reserved,
- AdrE = [200..253]: addresses logical groups,
- AdrE = 254: specifies a broadcast to all stations on a hierarchical level(this address can only be listed once, as the last address in the list),
- AdrE = 255: specifies a global broadcast. Telegrams with AdrE = [200..255] are not answered with a response telegram.

Definition at line 333 of file Telegrams.h.

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

- Telegrams.h

5.8 TGM::Bitfields::HeaderControl Struct Reference

Control byte consisting of several bit fields. Size: 8 bit.

Public Member Functions

- HeaderControl (HeaderType type=TypeCommand)

  Constructor.

Public Attributes

- union {
  struct **Bits** {
  BYTE NumSubAddresses: 3
  Bit 0-2 of Control Byte: Number of sub-addresses in the address block: NumSubAddresses=[0..7].
  BYTE NumRunningTgm: 1
  Bit 3 of Control Byte: Running telegram number.
  HeaderType Type: 1
  Bit 4 of Control Byte: Telegram Type, represented by HeaderType.
  BYTE StatusReactionTgm: 3
  Bit 5-7 of Control Byte: Status Bytes for the reaction telegram.
  } Bits
  BYTE Value
  Representation of the raw value.
  };

5.8.1 Detailed Description

Control byte consisting of several bit fields. Size: 8 bit.

Definition at line 106 of file Telegrams_Bitfields.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 HeaderControl()

```
TGM::Bitfields::HeaderControl::HeaderControl (
            HeaderType type = TypeCommand )  [inline]
```

Constructor.

Parameters

type	(Optional) Header type, represented by HeaderType.

See also

HeaderType

Definition at line 153 of file Telegrams_Bitfields.h.

5.8.3 Member Data Documentation

5.8.3.1 NumSubAddresses

```
BYTE TGM::Bitfields::HeaderControl::NumSubAddresses
```

Bit 0-2 of Control Byte: Number of sub-addresses in the address block: NumSubAddresses=[0..7].

Definition at line 113 of file Telegrams_Bitfields.h.

5.8.3.2 NumRunningTgm

```
BYTE TGM::Bitfields::HeaderControl::NumRunningTgm
```

Bit 3 of Control Byte: Running telegram number.

Byte represents:

- 0: not support
- 1: additional byte

Definition at line 118 of file Telegrams_Bitfields.h.

5.8.3.3 Type

`HeaderType TGM::Bitfields::HeaderControl::Type`

Bit 4 of Control Byte: Telegram Type, represented by HeaderType.

Definition at line 121 of file Telegrams_Bitfields.h.

5.8.3.4 StatusReactionTgm

`BYTE TGM::Bitfields::HeaderControl::StatusReactionTgm`

Bit 5-7 of Control Byte: Status Bytes for the reaction telegram.

Byte represents:

- 000: no error, request was processed
- 001: transmission request being processed
- 010: transmission cannot presently be processed
- 100: warning
- 110: error.

Definition at line 129 of file Telegrams_Bitfields.h.

5.8.3.5 Bits

`struct { ... } ::Bits TGM::Bitfields::HeaderControl::Bits`

5.8.3.6 Value

`BYTE TGM::Bitfields::HeaderControl::Value`

Representation of the raw value.

Definition at line 145 of file Telegrams_Bitfields.h.

5.8.3.7 "@7

`union { ... }`

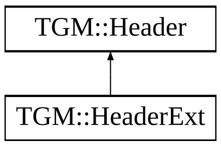
The documentation for this struct was generated from the following file:

- Telegrams_Bitfields.h

5.9 TGM::HeaderExt Struct Reference

Extended Telegram Header to be used for Routing and Sequential Telegrams.

Inheritance diagram for TGM::HeaderExt:



Public Attributes

- BYTE AdrES1

  Expanded part of the telegram header. Subaddress 1 of recipient. Bit 0-2 of Cntrl byte: 000.
- BYTE AdrES2

  Expanded part of the telegram header. Subaddress 2 of recipient. Bit 0-2 of Cntrl byte: 001.
- BYTE AdrES3

  Expanded part of the telegram header. Subaddress 3 of recipient. Bit 0-2 of Cntrl byte: 010.
- BYTE AdrES4

  Expanded part of the telegram header. Subaddress 4 of recipient. Bit 0-2 of Cntrl byte: 011.
- BYTE AdrES5

  Expanded part of the telegram header. Subaddress 5 of recipient. Bit 0-2 of Cntrl byte: 100.
- BYTE AdrES6

  Expanded part of the telegram header. Subaddress 6 of recipient. Bit 0-2 of Cntrl byte: 101.
- BYTE AdrES7

  Expanded part of the telegram header. Subaddress 7 of recipient. Bit 0-2 of Cntrl byte: 110.
- BYTE PaketN

  Expanded part of the telegram header.

Additional Inherited Members

5.9.1 Detailed Description

Extended Telegram Header to be used for Routing and Sequential Telegrams.

See also

Header

Definition at line 411 of file Telegrams.h.

5.9.2 Member Data Documentation

5.9.2.1 AdrES1

```
BYTE TGM::HeaderExt::AdrES1
```

Expanded part of the telegram header. Subaddress 1 of recipient. Bit 0-2 of Cntrl byte: 000.

Definition at line 414 of file Telegrams.h.

5.9.2.2 AdrES2

```
BYTE TGM::HeaderExt::AdrES2
```

Expanded part of the telegram header. Subaddress 2 of recipient. Bit 0-2 of Cntrl byte: 001.

Definition at line 417 of file Telegrams.h.

5.9.2.3 AdrES3

```
BYTE TGM::HeaderExt::AdrES3
```

Expanded part of the telegram header. Subaddress 3 of recipient. Bit 0-2 of Cntrl byte: 010.

Definition at line 420 of file Telegrams.h.

5.9.2.4 AdrES4

```
BYTE TGM::HeaderExt::AdrES4
```

Expanded part of the telegram header. Subaddress 4 of recipient. Bit 0-2 of Cntrl byte: 011.

Definition at line 423 of file Telegrams.h.

5.9.2.5 AdrES5

```
BYTE TGM::HeaderExt::AdrES5
```

Expanded part of the telegram header. Subaddress 5 of recipient. Bit 0-2 of Cntrl byte: 100.

Definition at line 426 of file Telegrams.h.

5.9.2.6 AdrES6

```
BYTE TGM::HeaderExt::AdrES6
```

Expanded part of the telegram header. Subaddress 6 of recipient. Bit 0-2 of Cntrl byte: 101.

Definition at line 429 of file Telegrams.h.

5.9.2.7 AdrES7

```
BYTE TGM::HeaderExt::AdrES7
```

Expanded part of the telegram header. Subaddress 7 of recipient. Bit 0-2 of Cntrl byte: 110.

Definition at line 432 of file Telegrams.h.

5.9.2.8 PaketN

```
BYTE TGM::HeaderExt::PaketN
```

Expanded part of the telegram header.

Sequential telegram number (packet number) , if bit 3 in Cntrl byte is set.

Definition at line 436 of file Telegrams.h.

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

- Telegrams.h

5.10 TGM::Map< THeader, TPayload > Union Template Reference

Templated mapping union to transfer raw TGM Bytes from/to specialized Bytes class.

Classes

- struct Mapping
  Specialized Bytes class, comprising structure payload head and Bytes.

Public Member Functions

- Map (THeader &_header=THeader(), TPayload &_payload=TPayload())
  Default constructor.
- ∼Map ()
  Destructor.
- void set (THeader &_header, TPayload &_payload)
  Sets the header/payload even after initialization.

Public Attributes

- Bytestream Raw

    Generic raw Bytes, comprising byte arrays.
- struct TGM::Map::Mapping Mapping

5.10.1 Detailed Description

template<class THeader, class TPayload>
union TGM::Map< THeader, TPayload >

Templated mapping union to transfer raw TGM Bytes from/to specialized Bytes class.

Definition at line 228 of file Telegrams.h.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Map()

```
template<class THeader, class TPayload>
TGM::Map< THeader, TPayload >::Map (
            THeader & _header = THeader(),
            TPayload & _payload = TPayload() )  [inline]
```

Default constructor.

Parameters

in	*_header*	(Optional) The Telegram header.
in	*_payload*	(Optional) The Telegram payload.

Definition at line 258 of file Telegrams.h.

5.10.2.2 ∼Map()

```
template<class THeader, class TPayload>
TGM::Map< THeader, TPayload >::∼Map ( )  [inline]
```

Destructor.

Definition at line 262 of file Telegrams.h.

5.10.3 Member Function Documentation

5.10.3.1 set()

```
template<class THeader, class TPayload>
void TGM::Map< THeader, TPayload >::set (
            THeader & _header,
            TPayload & _payload )  [inline]
```

Sets the header/payload even after initialization.

Parameters

in	*_header*	The Telegram header.
in	*_payload*	The Telegram payload.

Definition at line 268 of file Telegrams.h.

5.10.4 Member Data Documentation

5.10.4.1 Raw

```
template<class THeader, class TPayload>
Bytestream TGM::Map< THeader, TPayload >::Raw
```

Generic raw Bytes, comprising byte arrays.

Definition at line 232 of file Telegrams.h.

5.10.4.2 Mapping

```
template<class THeader, class TPayload>
struct TGM::Map::Mapping TGM::Map< THeader, TPayload >::Mapping
```

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

- Telegrams.h

5.11 TGM::Map< THeader, TPayload >::Mapping Struct Reference

Specialized Bytes class, comprising structure payload head and Bytes.

Public Member Functions

- Mapping (THeader &_header, TPayload _payload)

  Constructor.

Public Attributes

- THeader Header

    The Telegram header.
- TPayload Payload

    The Telegram payload.

5.11.1 Detailed Description

template<class THeader, class TPayload>
struct TGM::Map< THeader, TPayload >::Mapping

Specialized Bytes class, comprising structure payload head and Bytes.

Definition at line 236 of file Telegrams.h.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Mapping()

```
template<class THeader, class TPayload>
TGM::Map< THeader, TPayload >::Mapping::Mapping (
             THeader & _header,
             TPayload _payload )  [inline]
```

Constructor.

Parameters

in	*_header*	The Telegram header.
in	*_payload*	The Telegram payload.

Definition at line 247 of file Telegrams.h.

5.11.3 Member Data Documentation

5.11.3.1 Header

```
template<class THeader, class TPayload>
THeader TGM::Map< THeader, TPayload >::Mapping::Header
```

The Telegram header.

Definition at line 239 of file Telegrams.h.

5.11.3.2 Payload

```
template<class THeader, class TPayload>
TPayload TGM::Map< THeader, TPayload >::Mapping::Payload
```

The Telegram payload.

Definition at line 241 of file Telegrams.h.

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

- Telegrams.h

5.12 OPSTATE Struct Reference

Structure is used for loading the payload of the Reception Telegram from the Indradrive SERCOS parameter P-0-0115.

Public Member Functions

- OPSTATE (uint16_t P_0_0115=0)

  Constructor.

Public Attributes

- union {
  struct **Bits** {
  uint8_t OperateState: 2
  Bit 0-1 of parameter's payload:
  uint8_t DriveHalted: 1
  Bit 2 of parameter's payload: Drive Halt acknowledgment.
  uint8_t DriveError: 1
  Bit 3 of parameter's payload: Drive error.
  } Bits
  uint8_t Value
  Raw and unstructured data value.
  };

5.12.1 Detailed Description

Structure is used for loading the payload of the Reception Telegram from the Indradrive SERCOS parameter P-0-0115.

The structure is designed to be loaded with an integer, but automatically structured into its components. Thus, it is possible extract the exact information that are requested (e.g. Operate State of Indradrive M device).

The following code demonstrates a possible usage of this struct:

```
uint64_t curopstate;
SISProtocol_ref->read_parameter(TGM::SercosParamP, 115, curopstate);

OPSTATE opstate(static_cast<uint16_t>(curopstate));
int foo = opstate.Value;
```

.

See also

SISProtocol
SISProtocol::read_parameter

Definition at line 64 of file Wrapper.h.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 OPSTATE()

```
OPSTATE::OPSTATE (
             uint16_t P_0_0115 = 0 )  [inline]
```

Constructor.

Parameters

P_0_0115	(Optional) Payload data of SERCOS P-0-0115 parameter feedback. Default: 0.

Definition at line 107 of file Wrapper.h.

5.12.3 Member Data Documentation

5.12.3.1 OperateState

```
uint8_t OPSTATE::OperateState
```

Bit 0-1 of parameter's payload:

- 0b00: Control section / power section not ready for operation(e.g., drive error or phase 2)
- 0b01 : Control section ready for operation "bb"
- 0b10 : Control section and power section ready for op. "Ab"
- 0b11 : Drive with torque "AF".

Definition at line 75 of file Wrapper.h.

5.12.3.2 DriveHalted

```
uint8_t OPSTATE::DriveHalted
```

Bit 2 of parameter's payload: Drive Halt acknowledgment.

- 0: Drive Halt not active
- 1: Drive Halt is active and axis is in standstill

Definition at line 80 of file Wrapper.h.

5.12.3.3 DriveError

```
uint8_t OPSTATE::DriveError
```

Bit 3 of parameter's payload: Drive error.

- 0: No error
- 1: Drive error

Definition at line 85 of file Wrapper.h.

5.12.3.4 Bits

```
struct { ... } ::Bits OPSTATE::Bits
```

5.12.3.5 Value

```
uint8_t OPSTATE::Value
```

Raw and unstructured data value.

Definition at line 101 of file Wrapper.h.

5.12.3.6 "@15

```
union { ... }
```

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

- Wrapper.h

5.13 TGM::Reactions::SercosList Struct Reference

Sercos Command Telegram used for reading/writing single elements in lists from/to slave..

Public Member Functions

- SercosList ()
  Default constructor.
- void clear ()
  Clears this object to its blank/initial state.
- size_t get_head_size ()
  Gets payload header size.
- size_t get_size ()
  Gets the Payload size including Payload Header size.

Public Attributes

- BYTE Status

  Recipient status.
- BYTE Control

  Sercos control. Size: 8 bit. Set coding by TGM::Bitfields::SercosParamControl and toByte().
- BYTE UnitAddr

  The unit address of a drive is read in the command telegram and copied into the response telegram.
- union {
    Data Bytes
    USHORT Error
  };

  Payload Bytes, or error byte.

5.13.1 Detailed Description

Sercos Command Telegram used for reading/writing single elements in lists from/to slave..

Definition at line 740 of file Telegrams.h.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 SercosList()

```
TGM::Reactions::SercosList::SercosList ( )  [inline]
```

Default constructor.

Definition at line 761 of file Telegrams.h.

5.13.3 Member Function Documentation

5.13.3.1 clear()

```
void TGM::Reactions::SercosList::clear ( )  [inline]
```

Clears this object to its blank/initial state.

Definition at line 769 of file Telegrams.h.

5.13.3.2 get_head_size()

```
size_t TGM::Reactions::SercosList::get_head_size ( )  [inline]
```

Gets payload header size.

Returns

The payload header size.

Definition at line 779 of file Telegrams.h.

5.13.3.3 get_size()

```
size_t TGM::Reactions::SercosList::get_size ( )  [inline]
```

Gets the Payload size including Payload Header size.

Returns

The Payload size.

Definition at line 784 of file Telegrams.h.

5.13.4 Member Data Documentation

5.13.4.1 Status

```
BYTE TGM::Reactions::SercosList::Status
```

Recipient status.

Definition at line 743 of file Telegrams.h.

5.13.4.2 Control

```
BYTE TGM::Reactions::SercosList::Control
```

Sercos control. Size: 8 bit. Set coding by TGM::Bitfields::SercosParamControl and toByte().

Definition at line 746 of file Telegrams.h.

5.13.4.3 UnitAddr

`BYTE TGM::Reactions::SercosList::UnitAddr`

The unit address of a drive is read in the command telegram and copied into the response telegram.

For direct SIS communication with drives supporting SIS interface, unit address is the same as the SIS address of the receiver. Otherwise, the SIS address is related to the motion control and the unit address to the drive.

Definition at line 751 of file Telegrams.h.

5.13.4.4 Bytes

`Data TGM::Reactions::SercosList::Bytes`

Definition at line 756 of file Telegrams.h.

5.13.4.5 Error

`USHORT TGM::Reactions::SercosList::Error`

Definition at line 757 of file Telegrams.h.

5.13.4.6 "@5

`union { ... }`

Payload Bytes, or error byte.

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

- Telegrams.h

5.14 TGM::Commands::SercosList Struct Reference

Sercos Command Telegram used for reading/writing single elements in lists from/to slave.

Public Member Functions

- SercosList (TGM::Bitfields::SercosParamControl _ControlByte=TGM::Bitfields::SercosParamControl(), BYTE _unit_addr=0, TGM::Bitfields::SercosParamIdent _ParamIdent=TGM::Bitfields::SercosParamIdent(), USH↩ ORT _ListOffset=0, USHORT _SegmentSize=0, TGM::Data _PayloadData=Data())

  Constructor.
- void clear ()

  Clears this object to its blank/initial state.
- size_t get_head_size ()

  Gets size of payload header.
- size_t get_size ()

  Gets the Payload size including Payload Header size.

Public Attributes

- BYTE Control

    Sercos control. Size: 8 bit. Set coding by TGM::Bitfields::SercosParamControl and toByte().
- BYTE UnitAddr

    The unit address of a drive is read in the command telegram and copied into the response telegram.
- BYTE ParamType
- USHORT ParamNum

    Identifier for the parameter.
- USHORT ListOffset

    Defines the offset in bytes of the segment that has to be read.
- USHORT SegmentSize

    Size of the element to be handeled.
- Data Bytes

    Payload Bytes.

5.14.1 Detailed Description

Sercos Command Telegram used for reading/writing single elements in lists from/to slave.

Definition at line 554 of file Telegrams.h.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 SercosList()

```
TGM::Commands::SercosList::SercosList (
             TGM::Bitfields::SercosParamControl _ControlByte = TGM::Bitfields::SercosParam↵
Control(),
             BYTE _unit_addr = 0,
             TGM::Bitfields::SercosParamIdent _ParamIdent = TGM::Bitfields::SercosParam↵
Ident(),
             USHORT _ListOffset = 0,
             USHORT _SegmentSize = 0,
             TGM::Data _PayloadData = Data() )  [inline]
```

Constructor.

Parameters

_ControlByte	(Optional) Control Byte.
_unit_addr	(Optional) Unit address, which is the same as the SIS address of the receiver.
_ParamIdent	(Optional) Parameter Identifier.
_ListOffset	(Optional) List offset.
_SegmentSize	(Optional) Size of a single segment.
_PayloadData	(Optional) Payload data.

Definition at line 589 of file Telegrams.h.

5.14.3 Member Function Documentation

5.14.3.1 clear()

```
void TGM::Commands::SercosList::clear ( )  [inline]
```

Clears this object to its blank/initial state.

Definition at line 607 of file Telegrams.h.

5.14.3.2 get_head_size()

```
size_t TGM::Commands::SercosList::get_head_size ( )  [inline]
```

Gets size of payload header.

Returns

Size of payload header.

Definition at line 616 of file Telegrams.h.

5.14.3.3 get_size()

```
size_t TGM::Commands::SercosList::get_size ( )  [inline]
```

Gets the Payload size including Payload Header size.

Returns

The Payload size.

Definition at line 621 of file Telegrams.h.

5.14.4 Member Data Documentation

5.14.4.1 Control

```
BYTE TGM::Commands::SercosList::Control
```

Sercos control. Size: 8 bit. Set coding by TGM::Bitfields::SercosParamControl and toByte().

Definition at line 557 of file Telegrams.h.

5.14.4.2 UnitAddr

```
BYTE TGM::Commands::SercosList::UnitAddr
```

The unit address of a drive is read in the command telegram and copied into the response telegram.

For direct SIS communication with drives supporting SIS interface, unit address is the same as the SIS address of the receiver. Otherwise, the SIS address is related to the motion control and the unit address to the drive.

Definition at line 562 of file Telegrams.h.

5.14.4.3 ParamType

```
BYTE TGM::Commands::SercosList::ParamType
```

Definition at line 564 of file Telegrams.h.

5.14.4.4 ParamNum

```
USHORT TGM::Commands::SercosList::ParamNum
```

Identifier for the parameter.

Size: 16 bit. Set coding by TGM::Bitfields::SercosParamIdentification and toByte().

Definition at line 568 of file Telegrams.h.

5.14.4.5 ListOffset

```
USHORT TGM::Commands::SercosList::ListOffset
```

Defines the offset in bytes of the segment that has to be read.

For example: The 11th element of a list consisting of 4-byte elements should be handeled –> ListOffset=0x0028.

Definition at line 572 of file Telegrams.h.

5.14.4.6 SegmentSize

```
USHORT TGM::Commands::SercosList::SegmentSize
```

Size of the element to be handeled.

For example: The 11th element of a list consisting of 4-byte elements should be handeled –> Segment↩
Size=0x0004.

Definition at line 576 of file Telegrams.h.

5.14.4.7 Bytes

```
Data TGM::Commands::SercosList::Bytes
```

Payload Bytes.

Definition at line 579 of file Telegrams.h.

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

- Telegrams.h

5.15 TGM::Reactions::SercosParam Struct Reference

Representation of the payload for a Sercos Parameter reaction.

Public Member Functions

- SercosParam ()
  Default constructor.
- void clear ()
  Clears this object to its blank/initial state.
- size_t get_head_size ()
  Gets payload header size.
- size_t get_size ()
  Gets the Payload size including Payload Header size.

Public Attributes

- BYTE Status
  Recipient status.
- BYTE Control
  Sercos control. Size: 8 bit. Set coding by TGM::Bitfields::SercosParamControl and toByte().
- BYTE UnitAddr
  The unit address of a drive is read in the command telegram and copied into the response telegram.
- union {
  Data Bytes
  USHORT Error
  };
  Payload Bytes, or error byte.

5.15.1 Detailed Description

Representation of the payload for a Sercos Parameter reaction.

A Reaction Telegram is for regular subservices, such communication init, or device identification. This telegram is responded after successful execution of previous Command Telegram.

Definition at line 688 of file Telegrams.h.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 SercosParam()

```
TGM::Reactions::SercosParam::SercosParam ( )  [inline]
```

Default constructor.

Definition at line 709 of file Telegrams.h.

5.15.3 Member Function Documentation

5.15.3.1 clear()

```
void TGM::Reactions::SercosParam::clear ( )  [inline]
```

Clears this object to its blank/initial state.

Definition at line 717 of file Telegrams.h.

5.15.3.2 get_head_size()

```
size_t TGM::Reactions::SercosParam::get_head_size ( )  [inline]
```

Gets payload header size.

Returns

The payload header size.

Definition at line 727 of file Telegrams.h.

5.15.3.3 get_size()

```
size_t TGM::Reactions::SercosParam::get_size ( )  [inline]
```

Gets the Payload size including Payload Header size.

Returns

The Payload size.

Definition at line 732 of file Telegrams.h.

5.15.4 Member Data Documentation

5.15.4.1 Status

```
BYTE TGM::Reactions::SercosParam::Status
```

Recipient status.

Definition at line 691 of file Telegrams.h.

5.15.4.2 Control

```
BYTE TGM::Reactions::SercosParam::Control
```

Sercos control. Size: 8 bit. Set coding by TGM::Bitfields::SercosParamControl and toByte().

Definition at line 694 of file Telegrams.h.

5.15.4.3 UnitAddr

```
BYTE TGM::Reactions::SercosParam::UnitAddr
```

The unit address of a drive is read in the command telegram and copied into the response telegram.

For direct SIS communication with drives supporting SIS interface, unit address is the same as the SIS address of the receiver. Otherwise, the SIS address is related to the motion control and the unit address to the drive.

Definition at line 699 of file Telegrams.h.

5.15.4.4 Bytes

```
Data TGM::Reactions::SercosParam::Bytes
```

Definition at line 704 of file Telegrams.h.

5.15.4.5 Error

```
USHORT TGM::Reactions::SercosParam::Error
```

Definition at line 705 of file Telegrams.h.

5.15.4.6 "@3

```
union { ... }
```

Payload Bytes, or error byte.

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

- Telegrams.h

5.16 TGM::Commands::SercosParam Struct Reference

Sercos Command Telegram used for reading/writing single parameter from/to slave.

Public Member Functions

- SercosParam (TGM::Bitfields::SercosParamControl _control=TGM::Bitfields::SercosParamControl(), BYTE _unit_addr=0, TGM::Bitfields::SercosParamIdent _param_ident=TGM::Bitfields::SercosParamIdent(), TGM↩ ::Data _data=Data())

    Constructor.
- void clear ()

    Clears this object to its blank/initial state.
- size_t get_head_size ()

    Gets size of Payload Header.
- size_t get_size ()

    Gets the Payload size including Payload Header size.

Public Attributes

- BYTE Control

    Sercos control. Size: 8 bit. Set coding by TGM::Bitfields::SercosParamControl and toByte().
- BYTE UnitAddr

    The unit address of a drive is read in the command telegram and copied into the response telegram.
- BYTE ParamType
- USHORT ParamNum

    Identifier for the parameter.
- Data Bytes

    Payload Bytes.

5.16.1 Detailed Description

Sercos Command Telegram used for reading/writing single parameter from/to slave.

Definition at line 491 of file Telegrams.h.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 SercosParam()

```
TGM::Commands::SercosParam::SercosParam (
            TGM::Bitfields::SercosParamControl _control = TGM::Bitfields::SercosParamControl(),
            BYTE _unit_addr = 0,
            TGM::Bitfields::SercosParamIdent _param_ident = TGM::Bitfields::SercosParam↩
Ident(),
            TGM::Data _data = Data() )  [inline]
```

Constructor.

Parameters

_ControlByte	(Optional) Control Byte.
_unit_addr	(Optional) Unit address, which is the same as the SIS address of the receiver.
_ParamIdent	(Optional) Parameter Identifier (e.g. S-0-4000).
_PayloadData	(Optional) Payload data.

Definition at line 516 of file Telegrams.h.

5.16.3 Member Function Documentation

5.16.3.1 clear()

```
void TGM::Commands::SercosParam::clear ( )  [inline]
```

Clears this object to its blank/initial state.

Definition at line 529 of file Telegrams.h.

5.16.3.2 get_head_size()

```
size_t TGM::Commands::SercosParam::get_head_size ( )  [inline]
```

Gets size of Payload Header.

Returns

The Payload Header size.

Definition at line 541 of file Telegrams.h.

5.16.3.3 get_size()

```
size_t TGM::Commands::SercosParam::get_size ( )  [inline]
```

Gets the Payload size including Payload Header size.

Returns

The Payload size.

Definition at line 546 of file Telegrams.h.

5.16.4 Member Data Documentation

5.16.4.1 Control

```
BYTE TGM::Commands::SercosParam::Control
```

Sercos control. Size: 8 bit. Set coding by TGM::Bitfields::SercosParamControl and toByte().

Definition at line 494 of file Telegrams.h.

5.16.4.2 UnitAddr

```
BYTE TGM::Commands::SercosParam::UnitAddr
```

The unit address of a drive is read in the command telegram and copied into the response telegram.

For direct SIS communication with drives supporting SIS interface, unit address is the same as the SIS address of the receiver. Otherwise, the SIS address is related to the motion control and the unit address to the drive.

Definition at line 499 of file Telegrams.h.

5.16.4.3 ParamType

```
BYTE TGM::Commands::SercosParam::ParamType
```

Definition at line 501 of file Telegrams.h.

5.16.4.4 ParamNum

```
USHORT TGM::Commands::SercosParam::ParamNum
```

Identifier for the parameter.

Size: 16 bit. Set coding by TGM::Bitfields::SercosParamIdent and toByte().

Definition at line 505 of file Telegrams.h.

5.16.4.5 Bytes

```
Data TGM::Commands::SercosParam::Bytes
```

Payload Bytes.

Definition at line 508 of file Telegrams.h.

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

- Telegrams.h

5.17 TGM::Bitfields::SercosParamAttribute Struct Reference

Attribute for a SERCOS parameter that is callable via SercosDatablock.

Public Member Functions

- SercosParamAttribute (UINT32 _value=0)

  Constructor.

Public Attributes

- union {
  struct **Bits** {
  UINT32 ConversionFactor: 16
  Bit 0-15 of Reception Telegram's payload: Conversion factor: The conversion factor is an unsigned integer used to conv
  SercosDatalen DataLen: 3
  Bit 16-18 of Reception Telegram's payload: The Bytes length is required so that the Master is able to complete Service
  UINT32 DataFunction: 1
  Bit 19 of Reception Telegram's payload: Indicates whether this Bytes calls a procedure in a drive:
  UINT32 DataDisplay: 3
  Bit 20-22 of Reception Telegram's payload: Format used to convert the operation Bytes, and min/max input values to th
  UINT32 res5: 1
  Bit 23 of Reception Telegram's payload.
  UINT32 ScaleFactor: 4
  Bit 24-27 of Reception Telegram's payload: Decimal point: Places after the decimal point indicates the position of the de
  UINT32 is_writeonly_phase2: 1
  Bit 28 of Reception Telegram's payload.
  UINT32 is_writeonly_phase3: 1
  Bit 29 of Reception Telegram's payload.
  UINT32 is_writeonly_phase4: 1
  Bit 30 of Reception Telegram's payload.
  UINT32 res10: 1
  Bit 31 of Reception Telegram's payload.
  } Bits
  UINT32 Value
  Raw data value.
  };

5.17.1 Detailed Description

Attribute for a SERCOS parameter that is callable via SercosDatablock.

See also

SercosDatablock

Definition at line 250 of file Telegrams_Bitfields.h.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 SercosParamAttribute()

```
TGM::Bitfields::SercosParamAttribute::SercosParamAttribute (
            UINT32 _value = 0 )  [inline]
```

Constructor.

Parameters

_value	(Optional) Raw data value of the Reception Telegram's payload.

Definition at line 315 of file Telegrams_Bitfields.h.

5.17.3 Member Data Documentation

5.17.3.1 ConversionFactor

```
UINT32 TGM::Bitfields::SercosParamAttribute::ConversionFactor
```

Bit 0-15 of Reception Telegram's payload: Conversion factor: The conversion factor is an unsigned integer used to convert numeric Bytes to display format.

The conversion factor shall be set to a Value of 1, if a conversion is not required (e.g. for binary numbers, character strings or floating - point numbers).

Definition at line 259 of file Telegrams_Bitfields.h.

5.17.3.2 DataLen

```
SercosDatalen TGM::Bitfields::SercosParamAttribute::DataLen
```

Bit 16-18 of Reception Telegram's payload: The Bytes length is required so that the Master is able to complete Service Channel Bytes transfers correctly.

Definition at line 263 of file Telegrams_Bitfields.h.

5.17.3.3 DataFunction

```
UINT32 TGM::Bitfields::SercosParamAttribute::DataFunction
```

Bit 19 of Reception Telegram's payload: Indicates whether this Bytes calls a procedure in a drive:

- 0 Operation Bytes or parameter
- 1 Procedure command.

Definition at line 268 of file Telegrams_Bitfields.h.

5.17.3.4 DataDisplay

`UINT32 TGM::Bitfields::SercosParamAttribute::DataDisplay`

Bit 20-22 of Reception Telegram's payload: Format used to convert the operation Bytes, and min/max input values to the correct display format.

Definition at line 271 of file Telegrams_Bitfields.h.

5.17.3.5 res5

`UINT32 TGM::Bitfields::SercosParamAttribute::res5`

Bit 23 of Reception Telegram's payload.

Definition at line 274 of file Telegrams_Bitfields.h.

5.17.3.6 ScaleFactor

`UINT32 TGM::Bitfields::SercosParamAttribute::ScaleFactor`

Bit 24-27 of Reception Telegram's payload: Decimal point: Places after the decimal point indicates the position of the decimal point of appropriate operation Bytes.

Decimal point is used to define fixed point decimal numbers. For all other display formats the decimal point shall be = 0.

Definition at line 279 of file Telegrams_Bitfields.h.

5.17.3.7 is_writeonly_phase2

`UINT32 TGM::Bitfields::SercosParamAttribute::is_writeonly_phase2`

Bit 28 of Reception Telegram's payload.

Definition at line 282 of file Telegrams_Bitfields.h.

5.17.3.8 is_writeonly_phase3

`UINT32 TGM::Bitfields::SercosParamAttribute::is_writeonly_phase3`

Bit 29 of Reception Telegram's payload.

Definition at line 285 of file Telegrams_Bitfields.h.

5.17.3.9 is_writeonly_phase4

```
UINT32 TGM::Bitfields::SercosParamAttribute::is_writeonly_phase4
```

Bit 30 of Reception Telegram's payload.

Definition at line 288 of file Telegrams_Bitfields.h.

5.17.3.10 res10

```
UINT32 TGM::Bitfields::SercosParamAttribute::res10
```

Bit 31 of Reception Telegram's payload.

Definition at line 291 of file Telegrams_Bitfields.h.

5.17.3.11 Bits

```
struct { ...  } ::Bits TGM::Bitfields::SercosParamAttribute::Bits
```

5.17.3.12 Value

```
UINT32 TGM::Bitfields::SercosParamAttribute::Value
```

Raw data value.

Definition at line 309 of file Telegrams_Bitfields.h.

5.17.3.13 "@13

```
union { ...  }
```

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

- Telegrams_Bitfields.h

5.18 TGM::Bitfields::SercosParamControl Struct Reference

The control byte specifies how a Bytes block element of a parameter is accessed.

Public Member Functions

- SercosParamControl (SercosDatablock datablock=Datablock_OperationData)

  Constructor.
- SercosParamControl (BYTE value)

  Constructor.

Public Attributes

```
union {
  struct Bits {
    BYTE res1: 1
    BYTE res2: 1
    SercosTxProgress TxProgress: 1
      The transmission of a consecutive telegram is controlled with this bit (lists are written in several steps):
    SercosDatablock Datablock: 3
      SERCOS parameter datablock, represented by SercosDatablock.
    BYTE res6: 1
    BYTE res7: 1
  } Bits
  BYTE Value
      Representation of the raw value.
};
```

5.18.1 Detailed Description

The control byte specifies how a Bytes block element of a parameter is accessed.

The control byte is read out of the command telegram and copied into the response telegram.

Definition at line 159 of file Telegrams_Bitfields.h.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 SercosParamControl() [1/2]

```
TGM::Bitfields::SercosParamControl::SercosParamControl (
             SercosDatablock datablock = Datablock_OperationData )  [inline]
```

Constructor.

Parameters

datablock	(Optional) SERCOS Datablock, represented by SercosDatablock..

Definition at line 196 of file Telegrams_Bitfields.h.

5.18.2.2 SercosParamControl() [2/2]

```
TGM::Bitfields::SercosParamControl::SercosParamControl (
            BYTE value )  [inline]
```

Constructor.

Parameters

value	Raw byte data of the Control Byte.

Definition at line 201 of file Telegrams_Bitfields.h.

5.18.3 Member Data Documentation

5.18.3.1 res1

```
BYTE TGM::Bitfields::SercosParamControl::res1
```

Definition at line 165 of file Telegrams_Bitfields.h.

5.18.3.2 res2

```
BYTE TGM::Bitfields::SercosParamControl::res2
```

Definition at line 166 of file Telegrams_Bitfields.h.

5.18.3.3 TxProgress

```
SercosTxProgress TGM::Bitfields::SercosParamControl::TxProgress
```

The transmission of a consecutive telegram is controlled with this bit (lists are written in several steps):

- 0: transmission in progress
- 1: final transmission.

Definition at line 171 of file Telegrams_Bitfields.h.

5.18.3.4 Datablock

```
SercosDatablock TGM::Bitfields::SercosParamControl::Datablock
```

SERCOS parameter datablock, represented by SercosDatablock.

Definition at line 174 of file Telegrams_Bitfields.h.

5.18.3.5 res6

```
BYTE TGM::Bitfields::SercosParamControl::res6
```

Definition at line 176 of file Telegrams_Bitfields.h.

5.18.3.6 res7

```
BYTE TGM::Bitfields::SercosParamControl::res7
```

Definition at line 177 of file Telegrams_Bitfields.h.

5.18.3.7 Bits

```
struct { ...  } ::Bits TGM::Bitfields::SercosParamControl::Bits
```

5.18.3.8 Value

```
BYTE TGM::Bitfields::SercosParamControl::Value
```

Representation of the raw value.

Definition at line 190 of file Telegrams_Bitfields.h.

5.18.3.9 "@9

```
union { ...  }
```

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

- Telegrams_Bitfields.h

5.19 TGM::Bitfields::SercosParamIdent Struct Reference

Identification of the parameter. Size: 16 bit.

Public Member Functions

- SercosParamIdent (SercosParamVar param_variant=TGM::SercosParamS, USHORT param_num=0)

  Constructor.

Public Attributes

- union {
    struct **Bits** {
      USHORT ParamNumber: 12
        Bit 0-11: The parameter number [0..4095], e.g. P-0-∗1177∗, includes 1177 as ParamNumber.
      USHORT ParamSet: 3
        Bit 12-15: The parameter block [0..7], e.g. P-∗0∗-1177, includes 0 as ParamSet.
      USHORT ParamVariant: 1
        Bit 16: Parameter variant:
    } Bits
    USHORT Value
  };

5.19.1 Detailed Description

Identification of the parameter. Size: 16 bit.

Definition at line 206 of file Telegrams_Bitfields.h.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 SercosParamIdent()

```
TGM::Bitfields::SercosParamIdent::SercosParamIdent (
            SercosParamVar param_variant = TGM::SercosParamS,
            USHORT param_num = 0 )  [inline]
```

Constructor.

Parameters

param_variant	(Optional) The parameter variant, represented by SercosParamVar.
param_num	(Optional) The parameter number.

Definition at line 241 of file Telegrams_Bitfields.h.

5.19.3 Member Data Documentation

5.19.3.1 ParamNumber

```
USHORT TGM::Bitfields::SercosParamIdent::ParamNumber
```

Bit 0-11: The parameter number [0..4095], e.g. P-0-∗1177∗, includes 1177 as ParamNumber.

Definition at line 213 of file Telegrams_Bitfields.h.

5.19.3.2 ParamSet

```
USHORT TGM::Bitfields::SercosParamIdent::ParamSet
```

Bit 12-15: The parameter block [0..7], e.g. P-∗0∗-1177, includes 0 as ParamSet.

Definition at line 216 of file Telegrams_Bitfields.h.

5.19.3.3 ParamVariant

```
USHORT TGM::Bitfields::SercosParamIdent::ParamVariant
```

Bit 16: Parameter variant:

- 0: S-Parameter (drive)
- 1: P-Parameter (drive).

Definition at line 221 of file Telegrams_Bitfields.h.

5.19.3.4 Bits

```
struct { ... } ::Bits TGM::Bitfields::SercosParamIdent::Bits
```

5.19.3.5 Value

```
USHORT TGM::Bitfields::SercosParamIdent::Value
```

Definition at line 234 of file Telegrams_Bitfields.h.

5.19.3.6 "@11

```
union { ... }
```

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

- Telegrams_Bitfields.h

5.20 SISProtocol Class Reference

Class to hold functions an members for the SIS protocol support.

Classes

- class ExceptionGeneric

  Generic exceptions for SIS protocol.
- class ExceptionSISError

  Specific exception handling of SIS Protocol error codes.
- class ExceptionTransceiveFailed

  Specific exception handling of SIS Protocol transceiving failed.

Public Types

- enum SIS_SERVICES {
  SIS_SERVICE_INIT_COMM = 0x03, SIS_SERVICE_SEQUENTIALOP = 0x04, SIS_SERVICE_SERCOS↩
  _PARAM_READ = 0x10, SIS_SERVICE_SERCOS_LIST_READ = 0x11,
  SIS_SERVICE_SERCOS_READ_PHASE = 0x12, SIS_SERVICE_SERCOS_SWITCH_PHASE = 0x1D, S↩
  IS_SERVICE_SERCOS_LIST_WRITE = 0x1E, SIS_SERVICE_SERCOS_PARAM_WRITE = 0x1F }

  Values that represent identifiers to be used for SIS services.
- enum BAUDRATE {
  Baud_9600 = 0b00000000, Baud_19200 = 0b00000001, Baud_38400 = 0b00000010, Baud_57600 =
  0b00000100,
  Baud_115200 = 0b00001000 }

  Baudrate mask that can be utilized for the TypeCommand Telegram Subservice 0x07.
- typedef enum SISProtocol::SIS_SERVICES SIS_SERVICES

  Values that represent identifiers to be used for SIS services.
- typedef enum SISProtocol::BAUDRATE BAUDRATE

  Baudrate mask that can be utilized for the TypeCommand Telegram Subservice 0x07.

Public Member Functions

- SISProtocol ()

  Default constructor.
- virtual ∼SISProtocol ()

  Destructor.
- void open (const wchar_t ∗_port=L"COM1")
- void close ()
- void set_baudrate (BAUDRATE baudrate)
- void read_parameter (TGM::SercosParamVar _paramvar, USHORT _paramnum, UINT32 &_rcvddata)
- void read_parameter (TGM::SercosParamVar _paramvar, USHORT _paramnum, UINT64 &_rcvddata)
- void read_parameter (TGM::SercosParamVar _paramvar, USHORT _paramnum, DOUBLE &_rcvddata)
- void read_parameter (TGM::SercosParamVar _paramvar, USHORT _paramnum, char _rcvddata[TGM_SI↩
  ZEMAX_PAYLOAD])
- void read_listelm (TGM::SercosParamVar _paramvar, USHORT _paramnum, USHORT _elm_pos, UINT32
  &_rcvdelm)
- void read_listelm (TGM::SercosParamVar _paramvar, USHORT _paramnum, USHORT _elm_pos, UINT64
  &_rcvdelm)
- void read_listelm (TGM::SercosParamVar _paramvar, USHORT _paramnum, USHORT _elm_pos, DOUBLE
  &_rcvdelm)

- void write_parameter (TGM::SercosParamVar _paramvar, USHORT _paramnum, const UINT32 _data)
- void write_parameter (TGM::SercosParamVar _paramvar, USHORT _paramnum, const UINT64 _data)
- void write_parameter (TGM::SercosParamVar _paramvar, USHORT _paramnum, const DOUBLE _data)
- void write_listelm (TGM::SercosParamVar _paramvar, USHORT _paramnum, USHORT _elm_pos, const U↩ INT32 _rcvdelm)
- void write_listelm (TGM::SercosParamVar _paramvar, USHORT _paramnum, USHORT _elm_pos, const U↩ INT64 _rcvdelm)
- void write_listelm (TGM::SercosParamVar _paramvar, USHORT _paramnum, USHORT _elm_pos, const D↩ OUBLE _rcvdelm)
- void execute_command (TGM::SercosParamVar _paramvar, USHORT _paramnum)

5.20.1 Detailed Description

Class to hold functions an members for the SIS protocol support.

Definition at line 33 of file SISProtocol.h.

5.20.2 Member Typedef Documentation

5.20.2.1 SIS_SERVICES

```
typedef enum SISProtocol::SIS_SERVICES SISProtocol::SIS_SERVICES
```

Values that represent identifiers to be used for SIS services.

5.20.2.2 BAUDRATE

```
typedef enum SISProtocol::BAUDRATE SISProtocol::BAUDRATE
```

Baudrate mask that can be utilized for the TypeCommand Telegram Subservice 0x07.

Todo This enum is not yet implemented into the code.

5.20.3 Member Enumeration Documentation

5.20.3.1 SIS_SERVICES

```
enum SISProtocol::SIS_SERVICES
```

Values that represent identifiers to be used for SIS services.

Enumerator

SIS_SERVICE_INIT_COMM	
SIS_SERVICE_SEQUENTIALOP	
SIS_SERVICE_SERCOS_PARAM_READ	
SIS_SERVICE_SERCOS_LIST_READ	
SIS_SERVICE_SERCOS_READ_PHASE	
SIS_SERVICE_SERCOS_SWITCH_PHASE	
SIS_SERVICE_SERCOS_LIST_WRITE	
SIS_SERVICE_SERCOS_PARAM_WRITE	

Definition at line 44 of file SISProtocol.h.

5.20.3.2 BAUDRATE

```
enum SISProtocol::BAUDRATE
```

Baudrate mask that can be utilized for the TypeCommand Telegram Subservice 0x07.

Todo This enum is not yet implemented into the code.

Enumerator

Baud_9600	An enum constant representing the option for 9600 baud.
Baud_19200	An enum constant representing the option for 19200 baud.
Baud_38400	An enum constant representing the option for 38400 baud.
Baud_57600	An enum constant representing the option for 57600 baud.
Baud_115200	An enum constant representing the option for 115200 baud.

Definition at line 59 of file SISProtocol.h.

5.20.4 Constructor & Destructor Documentation

5.20.4.1 SISProtocol()

```
SISProtocol::SISProtocol ( )
```

Default constructor.

Definition at line 5 of file SISProtocol.cpp.

5.20.4.2 ∼SISProtocol()

```
SISProtocol::∼SISProtocol ( )  [virtual]
```

Destructor.

Definition at line 10 of file SISProtocol.cpp.

5.20.5 Member Function Documentation

5.20.5.1 open()

```
void SISProtocol::open (
             const wchar_t * _port = L"COM1" )
```

Definition at line 15 of file SISProtocol.cpp.

5.20.5.2 close()

```
void SISProtocol::close ( )
```

Definition at line 47 of file SISProtocol.cpp.

5.20.5.3 set_baudrate()

```
void SISProtocol::set_baudrate (
             BAUDRATE baudrate )
```

Definition at line 62 of file SISProtocol.cpp.

5.20.5.4 read_parameter() [1/4]

```
void SISProtocol::read_parameter (
             TGM::SercosParamVar _paramvar,
             USHORT _paramnum,
             UINT32 & _rcvddata )
```

Definition at line 88 of file SISProtocol.cpp.

5.20.5.5 read_parameter() [2/4]

```
void SISProtocol::read_parameter (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            UINT64 & _rcvddata )
```

Definition at line 108 of file SISProtocol.cpp.

5.20.5.6 read_parameter() [3/4]

```
void SISProtocol::read_parameter (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            DOUBLE & _rcvddata )
```

Definition at line 128 of file SISProtocol.cpp.

5.20.5.7 read_parameter() [4/4]

```
void SISProtocol::read_parameter (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            char _rcvddata[TGM_SIZEMAX_PAYLOAD] )
```

Definition at line 148 of file SISProtocol.cpp.

5.20.5.8 read_listelm() [1/3]

```
void SISProtocol::read_listelm (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            USHORT _elm_pos,
            UINT32 & _rcvdelm )
```

Definition at line 161 of file SISProtocol.cpp.

5.20.5.9 read_listelm() [2/3]

```
void SISProtocol::read_listelm (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            USHORT _elm_pos,
            UINT64 & _rcvdelm )
```

Definition at line 184 of file SISProtocol.cpp.

5.20.5.10 read_listelm() [3/3]

```
void SISProtocol::read_listelm (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            USHORT _elm_pos,
            DOUBLE & _rcvdelm )
```

Definition at line 207 of file SISProtocol.cpp.

5.20.5.11 write_parameter() [1/3]

```
void SISProtocol::write_parameter (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            const UINT32 _data )
```

Definition at line 259 of file SISProtocol.cpp.

5.20.5.12 write_parameter() [2/3]

```
void SISProtocol::write_parameter (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            const UINT64 _data )
```

Definition at line 267 of file SISProtocol.cpp.

5.20.5.13 write_parameter() [3/3]

```
void SISProtocol::write_parameter (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            const DOUBLE _data )
```

Definition at line 275 of file SISProtocol.cpp.

5.20.5.14 write_listelm() [1/3]

```
void SISProtocol::write_listelm (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            USHORT _elm_pos,
            const UINT32 _rcvdelm )
```

Definition at line 297 of file SISProtocol.cpp.

5.20.5.15 write_listelm() [2/3]

```
void SISProtocol::write_listelm (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            USHORT _elm_pos,
            const UINT64 _rcvdelm )
```

Definition at line 306 of file SISProtocol.cpp.

5.20.5.16 write_listelm() [3/3]

```
void SISProtocol::write_listelm (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum,
            USHORT _elm_pos,
            const DOUBLE _rcvdelm )
```

Definition at line 315 of file SISProtocol.cpp.

5.20.5.17 execute_command()

```
void SISProtocol::execute_command (
            TGM::SercosParamVar _paramvar,
            USHORT _paramnum )
```

Definition at line 342 of file SISProtocol.cpp.

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

- SISProtocol.h
- SISProtocol.cpp

5.21 SPEEDUNITS Struct Reference

Structure is used for loading the payload of the Reception Telegram from the Indradrive SERCOS parameter S-0-0044.

Public Member Functions

- SPEEDUNITS (uint16_t S_0_0044=0)

  Constructor.

Public Attributes

- union {
  struct **Bits** {
  uint16_t type_of_scaling: 3
  Bit 0-2 of parameter's payload: Type of scaling.
  uint16_t automode: 1
  Bit 3 of parameter's payload: Auto mode.
  uint16_t scale_units: 1
  Bit 4 of parameter's payload: Units for translational/rotatory scaling.
  uint16_t time_units: 1
  Bit 5 of parameter's payload: Time units.
  uint16_t data_rel: 1
  Bit 6 of parameter's payload: Data relation.
  uint16_t res7: 9
  Bit 7-15 of parameter's payload: reserved.
  } Bits
  uint16_t Value
  Raw and unstructured data value.
  };

5.21.1 Detailed Description

Structure is used for loading the payload of the Reception Telegram from the Indradrive SERCOS parameter S-0-0044.

The structure is designed to be loaded with an integer, but automatically structured into its components. Thus, it is possible extract the exact information that are requested (e.g. Operate State of Indradrive M device).

Definition at line 116 of file Wrapper.h.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 SPEEDUNITS()

```
SPEEDUNITS::SPEEDUNITS (
            uint16_t S_0_0044 = 0 )  [inline]
```

Constructor.

Parameters

S_0_0044	(Optional) Reception Telegram's payload data

Definition at line 176 of file Wrapper.h.

5.21.3 Member Data Documentation

5.21.3.1 type_of_scaling

```
uint16_t SPEEDUNITS::type_of_scaling
```

Bit 0-2 of parameter's payload: Type of scaling.

- 0b001: Translational scaling
- 0b010: Rotatory scaling.

Definition at line 125 of file Wrapper.h.

5.21.3.2 automode

```
uint16_t SPEEDUNITS::automode
```

Bit 3 of parameter's payload: Auto mode.

- 0: Preferred scaling
- 1: Scaling by parameters

Definition at line 130 of file Wrapper.h.

5.21.3.3 scale_units

```
uint16_t SPEEDUNITS::scale_units
```

Bit 4 of parameter's payload: Units for translational/rotatory scaling.

- 0: Millimeter/Revolutions
- 1: Inch/reserved

Definition at line 135 of file Wrapper.h.

5.21.3.4 time_units

```
uint16_t SPEEDUNITS::time_units
```

Bit 5 of parameter's payload: Time units.

- 0: Minute
- 1: Second

Definition at line 140 of file Wrapper.h.

5.21.3.5 data_rel

```
uint16_t SPEEDUNITS::data_rel
```

Bit 6 of parameter's payload: Data relation.

- 0: At motor shaft
- 1: At load

Definition at line 145 of file Wrapper.h.

5.21.3.6 res7

```
uint16_t SPEEDUNITS::res7
```

Bit 7-15 of parameter's payload: reserved.

Definition at line 148 of file Wrapper.h.

5.21.3.7 Bits

```
struct { ... } ::Bits SPEEDUNITS::Bits
```

5.21.3.8 Value

```
uint16_t SPEEDUNITS::Value
```

Raw and unstructured data value.

Definition at line 170 of file Wrapper.h.

5.21.3.9 "@17

```
union { ... }
```

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

- Wrapper.h

5.22 TGM::Reactions::Subservice Struct Reference

Representation of the payload for a Subservice reaction.

Public Member Functions

- Subservice ()

  Default constructor.
- void clear ()

  Clears this object to its blank/initial state.
- size_t get_head_size ()

  Gets payload header size.
- size_t get_size ()

  Gets the Payload size including Payload Header size.

Public Attributes

- BYTE Status

  Recipient status.
- BYTE RecipientAddr

  Address of the recipient.
- BYTE ServiceNumber

  SIS service number.
- union {
  Data Bytes
  BYTE Error
  };

  Payload Bytes, or error byte.

5.22.1 Detailed Description

Representation of the payload for a Subservice reaction.

A Reaction Telegram is for regular subservices, such communication init, or device identification. This telegram is responded after successful execution of previous Command Telegram.

Definition at line 636 of file Telegrams.h.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 Subservice()

```
TGM::Reactions::Subservice::Subservice ( )  [inline]
```

Default constructor.

Definition at line 655 of file Telegrams.h.

5.22.3 Member Function Documentation

5.22.3.1 clear()

```
void TGM::Reactions::Subservice::clear ( )  [inline]
```

Clears this object to its blank/initial state.

Definition at line 663 of file Telegrams.h.

5.22.3.2 get_head_size()

```
size_t TGM::Reactions::Subservice::get_head_size ( )  [inline]
```

Gets payload header size.

Returns

The payload head size.

Definition at line 673 of file Telegrams.h.

5.22.3.3 get_size()

```
size_t TGM::Reactions::Subservice::get_size ( )  [inline]
```

Gets the Payload size including Payload Header size.

Returns

The Payload size.

Definition at line 678 of file Telegrams.h.

5.22.4 Member Data Documentation

5.22.4.1 Status

```
BYTE TGM::Reactions::Subservice::Status
```

Recipient status.

Definition at line 639 of file Telegrams.h.

5.22.4.2 RecipientAddr

```
BYTE TGM::Reactions::Subservice::RecipientAddr
```

Address of the recipient.

Definition at line 642 of file Telegrams.h.

5.22.4.3 ServiceNumber

```
BYTE TGM::Reactions::Subservice::ServiceNumber
```

SIS service number.

Definition at line 645 of file Telegrams.h.

5.22.4.4 Bytes

```
Data TGM::Reactions::Subservice::Bytes
```

Definition at line 650 of file Telegrams.h.

5.22.4.5 Error

```
BYTE TGM::Reactions::Subservice::Error
```

Definition at line 651 of file Telegrams.h.

5.22.4.6 "@1

```
union { ... }
```

Payload Bytes, or error byte.

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

- Telegrams.h

5.23 TGM::Commands::Subservice Struct Reference

Representation of the PAYLOAD for a Subservice command.

Public Member Functions

- Subservice (BYTE _addr=0, BYTE _subservice=0, Data _data=Data())

    Constructor.
- void clear ()

    Clears this object to its blank/initial state.
- size_t get_head_size ()

    Gets size of Payload Header.
- size_t get_size ()

    Gets the Payload size including Payload Header size.

Public Attributes

- BYTE RecipientAddr

    The recipient address.
- BYTE ServiceNumber

    The subservice number.
- Data Bytes

    The Payload content.

5.23.1 Detailed Description

Representation of the PAYLOAD for a Subservice command.

A Command Telegram is for regular subservices, such communication init, or device identification. User for master communication (active communicator).

Definition at line 449 of file Telegrams.h.

5.23.2 Constructor & Destructor Documentation

5.23.2.1 Subservice()

```
TGM::Commands::Subservice::Subservice (
            BYTE _addr = 0,
            BYTE _subservice = 0,
            Data _data = Data() )  [inline]
```

Constructor.

Parameters

_addr	(Optional) The recipient address.
_subservice	(Optional) The subservice number.
_PayloadData	(Optional) The data.

Definition at line 463 of file Telegrams.h.

5.23.3 Member Function Documentation

5.23.3.1 clear()

```
void TGM::Commands::Subservice::clear ( )  [inline]
```

Clears this object to its blank/initial state.

Definition at line 473 of file Telegrams.h.

5.23.3.2 get_head_size()

```
size_t TGM::Commands::Subservice::get_head_size ( )  [inline]
```

Gets size of Payload Header.

Returns

The Payload Header size.

Definition at line 478 of file Telegrams.h.

5.23.3.3 get_size()

```
size_t TGM::Commands::Subservice::get_size ( )  [inline]
```

Gets the Payload size including Payload Header size.

Returns

The Payload size.

Definition at line 483 of file Telegrams.h.

5.23.4 Member Data Documentation

5.23.4.1 RecipientAddr

```
BYTE TGM::Commands::Subservice::RecipientAddr
```

The recipient address.

Definition at line 452 of file Telegrams.h.

5.23.4.2 ServiceNumber

```
BYTE TGM::Commands::Subservice::ServiceNumber
```

The subservice number.

Definition at line 454 of file Telegrams.h.

5.23.4.3 Bytes

```
Data TGM::Commands::Subservice::Bytes
```

The Payload content.

Definition at line 456 of file Telegrams.h.

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

- Telegrams.h

6 File Documentation

6.1 errors.h File Reference

Implementation of Error handle that is used within each API call function.

Classes

- struct GenericErrHandle

  Generic error handle that is returned from each API function.

Macros

- #define Err_Base (0x0)

  A macro that defines Error base that is used for coding the final error code.

Typedefs

- typedef struct GenericErrHandle GenericErrHandle

  Generic error handle that is returned from each API function.
- typedef GenericErrHandle $*$ ErrHandle

  Defines an alias representing a pointer to GenericErrHandle.
- typedef enum EErrorBlocks EErrorBlocks

  Values that represent error blocks to be used as block_code paramater for set_error() function.

Enumerations

- enum EErrorBlocks {
  Err_NoError = 0, Err_Block_OpenByCOM = 1, Err_Block_Close = 2, Err_Block_Test = 3,
  Err_Block_SeqInit = 6, Err_Block_SeqWrite = 7, Err_Block_VelCInit = 8, Err_Block_VelCWrite = 9,
  Err_Block_GetStatus = 10, Err_Block_SetControl = 11, Err_Invalid_Pointer = 12 }

  Values that represent error blocks to be used as block_code paramater for set_error() function.

Functions

- static int32_t set_error (ErrHandle errhndl, std::string errstr, int32_t block_code, int32_t issue_code=1)

  Sets an error handle to the errhndl parameter.

6.1.1 Detailed Description

Implementation of Error handle that is used within each API call function.

Definition in file errors.h.

6.1.2 Macro Definition Documentation

6.1.2.1 Err_Base

```
#define Err_Base (0x0)
```

A macro that defines Error base that is used for coding the final error code.

See also

set_error()

Definition at line 114 of file errors.h.

6.1.3 Typedef Documentation

6.1.3.1 GenericErrHandle

```
typedef struct GenericErrHandle GenericErrHandle
```

Generic error handle that is returned from each API function.

In contrast to a LabVIEW error handle (LVErrorCluster_t) that includes a specific type of Error string, the Generic Error Handle (GenericErrHandle) consists of generic C types (int and char∗) for both error code and error message.

GenericErrHandle is used as pointer for all Indradrive API Functions (see API Function Overview).

Remarks

Depending on the USE_LABVIEW_ENV switch, the GenericErrHandle can be replaced by LStrHandle.

6.1.3.2 ErrHandle

```
typedef GenericErrHandle* ErrHandle
```

Defines an alias representing a pointer to GenericErrHandle.

Remarks

The alias is used since the USE_LABVIEW_ENV switch can the GenericErrHandle by LStrHandle.

Definition at line 81 of file errors.h.

6.1.3.3 EErrorBlocks

```
typedef enum EErrorBlocks EErrorBlocks
```

Values that represent error blocks to be used as block_code paramater for set_error() function.

See also

set_error()

6.1.4 Enumeration Type Documentation

6.1.4.1 EErrorBlocks

```
enum EErrorBlocks
```

Values that represent error blocks to be used as block_code paramater for set_error() function.

See also

set_error()

Enumerator

Err_NoError	An enum constant representing the Error: no error.
Err_Block_OpenByCOM	An enum constant representing the Error on open by com.
Err_Block_Close	An enum constant representing the Error on close.
Err_Block_Test	An enum constant representing the Error on test.
Err_Block_SeqInit	An enum constant representing the Error on Sequence init.
Err_Block_SeqWrite	An enum constant representing the Error on Sequence write.
Err_Block_VelCInit	An enum constant representing the Error on Speed Contrl init.
Err_Block_VelCWrite	An enum constant representing the Error on Speed Control write.
Err_Block_GetStatus	An enum constant representing the Error on get status.
Err_Block_SetControl	An enum constant representing the Error on set control.
Err_Invalid_Pointer	An enum constant representing the Error of invalid API reference.

Definition at line 120 of file errors.h.

6.1.5 Function Documentation

6.1.5.1 set_error()

```
static int32_t set_error (
            ErrHandle errhndl,
            std::string errstr,
            int32_t block_code,
            int32_t issue_code = 1 )  [inline], [static]
```

Sets an error handle to the errhndl parameter.

This static function can be utilized to set an error message as well as a error code in the following scheme to an Error handle: Error code: 0 $<<$ 8 $|$ block_code $<<$ 4 $|$ issue_code, whereas "|" indicates an OR- concatenation.

Parameters

out	*errhndl*	Error handle pointer.
in	*errstr*	Error message.
in	*block_code*	Error block code defined by EErrorBlocks enum.
in	*issue_code*	(Optional) The issue code.

Returns

The final error code.

See also

EErrorBlocks

Definition at line 183 of file errors.h.

6.2 errors.h

```

#ifndef LABVIEW_ERRORS_H
#define LABVIEW_ERRORS_H

#include <cstring>
#include <string>


#ifdef USE_LABVIEW_ENV
// Including Labviewv.lib in case of project is compiled for LabVIEW
#pragma comment(lib,"labviewv.lib")
// Including extcode.h in case of project is compiled for LabVIEW
#include "extcode.h"
#endif

typedef struct GenericErrHandle {

    uint32_t    code;
    char        msg[2048];

    GenericErrHandle(uint32_t _code = 0, const char* _msg = "") :
        code(_code)
    {}

    void set(uint32_t _code, const char* _msg)
    {
        code = _code;

        for (size_t i = 0; i < strlen(_msg); i++)
            msg[i] = _msg[i];
    }

    void set_msg(const char* _msg)
    {
        set(code, _msg);
    }

    void set_code(uint32_t _code)
    {
        set(_code, msg);
    }
} GenericErrHandle;


#ifdef USE_LABVIEW_ENV
typedef LStrHandle ErrHandle;
#else
typedef GenericErrHandle* ErrHandle;
#endif


#ifdef USE_LABVIEW_ENV
#pragma pack(push,1)
#include "extcode.h"
#pragma pack(pop)
#endif

#ifdef USE_LABVIEW_ENV
#pragma pack(push,1)
typedef struct {
    LVBoolean   status;
    int32       code;
    LStrHandle  msg;
} LVErrorCluster_t;
#pragma pack(pop)
#endif


#ifdef USE_LABVIEW_ENV
#define Err_Base (0x08EF)
#else

#define Err_Base (0x0)
#endif

typedef enum EErrorBlocks
{
    Err_NoError              = 0,
    Err_Block_OpenByCOM      = 1,
    Err_Block_Close          = 2,
    Err_Block_Test            = 3,
    Err_Block_SeqInit      = 6,
    Err_Block_SeqWrite        = 7,
    Err_Block_VelCInit        = 8,
    Err_Block_VelCWrite      = 9,
    Err_Block_GetStatus      = 10,
    Err_Block_SetControl    = 11,
    Err_Invalid_Pointer      = 12
} EErrorBlocks;

#ifdef USE_LABVIEW_ENV
static MgErr write_string(ErrHandle lvhandle, std::string str)
{
    //Initializes the buffer
    MgErr err = NumericArrayResize(uB, 1, (UHandle*)&lvhandle, str.length());
    if (err) return err;

    //Informs the LabVIEW string handle about the size of the size
    (**lvhandle).cnt = str.length();

    //Fills the string buffer with str
    strcpy((char*)(**lvhandle).str, str.c_str());

    return noErr;
}
```

```
#endif

inline static int32_t set_error(ErrHandle errhndl, std::string errstr, int32_t block_code
, int32_t issue_code = 1)
{
    int32_t retcode = (Err_Base << 8) | (block_code << 4) | issue_code;

#ifdef USE_LABVIEW_ENV
    write_string(errhndl, errstr);
#else
    errhndl->set(retcode, errstr.c_str());
#endif

    return retcode;
}


#endif // LABVIEW_ERRORS_H

```

6.3 mainpage.dox File Reference

6.4 SISProtocol.cpp File Reference

6.5 SISProtocol.cpp

```
#include "SISProtocol.h"



SISProtocol::SISProtocol()
{
}


SISProtocol::~SISProtocol()
{
}


void SISProtocol::open(const wchar_t * _port)
{
    STACK;

    LPCTSTR cport = (LPCTSTR)_port;
    CSerial::EBaudrate cbaudrate    = CSerial::EBaud19200;
    CSerial::EDataBits cdata        = CSerial::EData8;
    CSerial::EParity cparity        = CSerial::EParNone;
    CSerial::EStopBits cstopbits    = CSerial::EStop1;
    CSerial::EHandshake chandshake  = CSerial::EHandshakeOff;

    try
    {
        CSerial::CheckPort(cport);

        m_serial.Open(cport, RS232_BUFFER, RS232_BUFFER, true /* overlapped */);
        m_serial.Setup(cbaudrate, cdata, cparity, cstopbits);
        m_serial.SetupHandshaking(chandshake);

        m_serial.SetMask(CSerial::EEventBreak |
            CSerial::EEventError |
            CSerial::EEventRecv);

        m_serial.SetupReadTimeouts(CSerial::EReadTimeoutNonblocking);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        throw;
    }
}


void SISProtocol::close()
{
    STACK;

    try
    {
        m_serial.Close();
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        throw;
    }
}


void SISProtocol::set_baudrate(BAUDRATE baudrate)
{
    STACK;

    // Build Telegrams ...

    // Mapping for SEND Telegram
    TGM::Map<TGM::Header, TGM::Commands::Subservice>
        tx_tgm(
            // Init header
            TGM::Header(SIS_ADDR_MASTER, SIS_ADDR_SLAVE, SIS_SERVICE_INIT_COMM, TGM::Bitfields::HeaderControl(TGM::TypeCommand)),
            // Init payload
            TGM::Commands::Subservice(SIS_ADDR_UNIT, 0x07, TGM::Data({ (BYTE)baudrate }))
        );

    // Mapping for RECEPTION Telegram
    TGM::Map<TGM::Header, TGM::Reactions::Subservice> rx_tgm;

    // Set payload size
    tx_tgm.Mapping.Header.set_DatL(tx_tgm.Mapping.Payload.get_size());

    // Transceive ...
    transceiving(tx_tgm, rx_tgm);
}


void SISProtocol::read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum, UINT32 & _rcvddata)
{
    STACK;

    // Fetching attributes for length and scale ...
    size_t datalen = 1;
    UINT8 scalefactor = 0;
    get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);

    // Communication with Telegrams ...
    auto rx_tgm = transceive_param
        <TGM::Header, TGM::Commands::SercosParam, TGM::Header, TGM::Reactions::SercosParam>
        (_paramvar, _paramnum, SIS_SERVICE_SERCOS_PARAM_READ);

    // Convert responsed Bytes ...
    INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
    _rcvddata = static_cast<UINT32>(response);
}


void SISProtocol::read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum, UINT64& _rcvddata)
{
    STACK;

    // Fetching attributes for length and scale ...
    size_t datalen = 1;
    UINT8 scalefactor = 0;
    get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);

    // Communication with Telegrams ...
    auto rx_tgm = transceive_param
        <TGM::Header, TGM::Commands::SercosParam, TGM::Header, TGM::Reactions::SercosParam>
        (_paramvar, _paramnum, SIS_SERVICE_SERCOS_PARAM_READ);

    // Convert responsed Bytes ...
    INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
    _rcvddata = static_cast<UINT64>(response);
}


void SISProtocol::read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum, DOUBLE & _rcvddata)
{
    STACK;

    // Fetching attributes for length and scale ...
    size_t datalen = 1;
    UINT8 scalefactor = 0;
    get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);

    // Communication with Telegrams ...
    auto rx_tgm = transceive_param
                    <TGM::Header, TGM::Commands::SercosParam,
TGM::Header, TGM::Reactions::SercosParam>
                    (_paramvar, _paramnum, SIS_SERVICE_SERCOS_PARAM_READ);

    // Convert responsed Bytes ...
    INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
    _rcvddata = (double)response / std::pow(10, scalefactor);
}


void SISProtocol::read_parameter(TGM::SercosParamVar
_paramvar, USHORT _paramnum, char _rcvddata[TGM_SIZEMAX_PAYLOAD])
{
    // Communication with Telegrams ...
    auto rx_tgm = transceive_param
                    <TGM::Header, TGM::Commands::SercosParam,
TGM::Header, TGM::Reactions::SercosParam>
                    (_paramvar, _paramnum, SIS_SERVICE_SERCOS_PARAM_READ);

    // Convert responsed Bytes ...
    memcpy(_rcvddata, (char*)rx_tgm.Mapping.Payload.Bytes.Bytes, rx_tgm.Mapping.Payload.Bytes.Size);
    _rcvddata[rx_tgm.Mapping.Payload.Bytes.Size] = '\0';
}


void SISProtocol::read_listelm(TGM::SercosParamVar _paramvar,
USHORT _paramnum, USHORT _elm_pos, UINT32 & _rcvdelm)
{
    STACK;

    // Fetching attributes for length and scale ...
    size_t datalen = 1;
    UINT8 scalefactor = 0;
    get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);

    // Communication with Telegrams ...
    USHORT SegmentSize = (USHORT)datalen;
    USHORT ListOffset = _elm_pos * SegmentSize;

    auto rx_tgm = transceive_list
        <TGM::Header, TGM::Commands::SercosList,
TGM::Header, TGM::Reactions::SercosList>
        (_paramvar, _paramnum, SIS_SERVICE_SERCOS_LIST_READ, SegmentSize,
ListOffset);

    // Response Bytes ...
    INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
    _rcvdelm = static_cast<UINT32>(response);
}


void SISProtocol::read_listelm(TGM::SercosParamVar _paramvar,
USHORT _paramnum, USHORT _elm_pos, UINT64& _rcvdelm)
{
    STACK;

    // Fetching attributes for length and scale ...
    size_t datalen = 1;
    UINT8 scalefactor = 0;
    get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);

    // Communication with Telegrams ...
    USHORT SegmentSize = (USHORT)datalen;
    USHORT ListOffset = _elm_pos * SegmentSize;

    auto rx_tgm = transceive_list
        <TGM::Header, TGM::Commands::SercosList,
TGM::Header, TGM::Reactions::SercosList>
        (_paramvar, _paramnum, SIS_SERVICE_SERCOS_LIST_READ, SegmentSize,
ListOffset);

    // Response Bytes ...
    INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
    _rcvdelm = static_cast<UINT64>(response);
}


void SISProtocol::read_listelm(TGM::SercosParamVar _paramvar,
USHORT _paramnum, USHORT _elm_pos, DOUBLE & _rcvdelm)
{
    STACK;

    // Fetching attributes for length and scale ...
    size_t datalen = 1;
    UINT8 scalefactor = 0;
    get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);

    // Communication with Telegrams ...
    USHORT SegmentSize = (USHORT)datalen;
    USHORT ListOffset = _elm_pos * SegmentSize;

    auto rx_tgm = transceive_list
                    <TGM::Header, TGM::Commands::SercosList, TGM::Header, TGM::Reactions::SercosList>
                    (_paramvar, _paramnum, SIS_SERVICE_SERCOS_LIST_READ, SegmentSize, ListOffset);

    // Response Bytes ...
    INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
    _rcvdelm = (double)response / std::pow(10, scalefactor);
}


INT64 SISProtocol::get_sized_data(TGM::Data& rx_data, const size_t &datalen)
{
    STACK;

    if (datalen == 1)
    {
        UINT8 Bytes = rx_data.toUINT8();
        UINT64 mask = ((Bytes >> 7) & 1) ? 0xFFFFFFFFFFFFFF00 : 0;
        return (INT64)(Bytes | mask);
    }
    else if (datalen == 2)
    {
        UINT16 Bytes = rx_data.toUINT16();
        UINT64 mask = ((Bytes >> 15) & 1) ? 0xFFFFFFFFFFFF0000 : 0;
        return (INT64)(Bytes | mask);
    }
    else if (datalen == 8)
    {
        return (INT64)rx_data.toUINT64();
    }
    else
    {
        UINT32 Bytes = rx_data.toUINT32();
        UINT64 mask = ((Bytes >> 31) & 1) ? 0xFFFFFFFF00000000 : 0;
        return (INT64)(Bytes | mask);
    }
}


void SISProtocol::write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum, const UINT32 _data)
{
    STACK;

    write_parameter(_paramvar, _paramnum, static_cast<DOUBLE>(_data));
}


void SISProtocol::write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum, const UINT64 _data)
{
    STACK;

    write_parameter(_paramvar, _paramnum, static_cast<DOUBLE>(_data));
}


void SISProtocol::write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum, const DOUBLE _data)
{
    STACK;

    // Fetching attributes for length and scale ...
    size_t datalen = 1;
    UINT8 scalefactor = 0;
    get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);

    // Preprocess Bytes ...
    UINT64 inval = static_cast<UINT64>(_data * std::pow(10, scalefactor));

    // Communication with Telegrams ...
    TGM::Data Bytes;
    set_sized_data(Bytes, datalen, inval);

    transceive_param
        <TGM::Header, TGM::Commands::SercosParam,
TGM::Header, TGM::Reactions::SercosParam>
        (_paramvar, _paramnum, SIS_SERVICE_SERCOS_PARAM_WRITE, &Bytes);
}


void SISProtocol::write_listelm(TGM::SercosParamVar _paramvar,
 USHORT _paramnum, USHORT _elm_pos, const UINT32 _rcvdelm)
{
    STACK;

    DOUBLE buf = static_cast<DOUBLE>(_rcvdelm);
    write_listelm(_paramvar, _paramnum, _elm_pos, buf);
}


void SISProtocol::write_listelm(TGM::SercosParamVar _paramvar,
 USHORT _paramnum, USHORT _elm_pos, const UINT64 _rcvdelm)
{
    STACK;

    DOUBLE buf = static_cast<DOUBLE>(_rcvdelm);
    write_listelm(_paramvar, _paramnum, _elm_pos, buf);
}


void SISProtocol::write_listelm(TGM::SercosParamVar _paramvar,
 USHORT _paramnum, USHORT _elm_pos, const DOUBLE _rcvdelm)
{
    STACK;

    // Fetching attributes for length and scale ...
    size_t datalen      = 0;
    UINT8 scalefactor   = 0;
    get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);

    UINT64 inval = static_cast<UINT64>(_rcvdelm * std::pow(10, scalefactor));

    // Re-adjusting list size, if needed
    set_parameter_listsize(_paramvar, _paramnum, datalen, _elm_pos);

    TGM::Data Bytes;
    set_sized_data(Bytes, datalen, inval);

    // Communication with Telegrams ...
    USHORT SegmentSize  = (USHORT)datalen;
    USHORT ListOffset   = _elm_pos * SegmentSize;

    transceive_list
        <TGM::Header, TGM::Commands::SercosList,
TGM::Header, TGM::Reactions::SercosList>
        (_paramvar, _paramnum, SIS_SERVICE_SERCOS_LIST_WRITE, SegmentSize,
ListOffset, &Bytes);
}


void SISProtocol::execute_command(
TGM::SercosParamVar _paramvar, USHORT _paramnum)
{
    TGM::SercosCommandrequest cmd;
    TGM::SercosCommandstatus Status =
TGM::Commandstatus_Busy;
    int iterations;

    // Start command ...
    cmd = TGM::Commandrequest_Set;
    try
    {
        write_parameter(_paramvar, _paramnum, static_cast<UINT64>(cmd));
    }
    catch (SISProtocol::ExceptionSISError &ex)
    {
        if (ex.get_errorcode() == 0x700C)
            throw SISProtocol::ExceptionGeneric(-1, "Command cannot be
 executed, because it is write-protected. Release the drive torque (disable drive), or restart the Indradrive
 system.");
        else
            throw;
    }

    iterations = 0;
    do
    {
        get_parameter_status(_paramvar, _paramnum, Status);

        if (iterations > MAX_COMMANDCHECK_ITERATIONS) throw
ExceptionGeneric(-1, "Command execution caused a continuous busy loop. Please restart the Indradrive system.");
    } while (Status == TGM::Commandstatus_Busy);

    if (Status != TGM::Commandstatus_OK)
        throw ExceptionGeneric(static_cast<int>(Status), sformat("Command execution failed with status code %d. Command executation canceled or not possible due to released operation state of the drive.", Status));


    // Delete command ...
    cmd = TGM::Commandrequest_NotSet;
    write_parameter(_paramvar, _paramnum, static_cast<UINT64>(cmd));

    Status = TGM::Commandstatus_Busy;
    iterations = 0;
    do
    {
        get_parameter_status(_paramvar, _paramnum, Status);

        if (iterations > MAX_COMMANDCHECK_ITERATIONS) throw ExceptionGeneric(-1, "Command execution caused a continuous busy loop. Please restart the Indradrive system.");
    } while (Status == TGM::Commandstatus_Busy);

    if (Status != TGM::Commandstatus_NotSet)
        throw ExceptionGeneric(static_cast<int>(Status), sformat("Command execution failed with status code %d. Command executation canceled or not possible due to released operation state of the drive.", Status));
}


void SISProtocol::get_parameter_status(const TGM::SercosParamVar _paramvar, const USHORT & _paramnum, TGM::SercosCommandstatus& _datastatus)
{
    STACK;

    // Communication with Telegrams ...
    BYTE service = SIS_SERVICE_SERCOS_PARAM_WRITE;

    auto rx_tgm = transceive_param
                    <TGM::Header, TGM::Commands::SercosParam, TGM::Header, TGM::Reactions::SercosParam>
                    (_paramvar, _paramnum, service, new TGM::Data(), TGM::Datablock_IdentNumber);

    // Read back Datablock ...
    _datastatus = static_cast<TGM::SercosCommandstatus>(rx_tgm.Mapping.Payload.Bytes.toUINT8());
}


void SISProtocol::set_sized_data(TGM::Data& tx_data, const size_t &datalen, UINT64 & _rcvdelm)
{
    STACK;

    if (datalen == 1) tx_data = TGM::Data((UINT8)_rcvdelm);
    else if (datalen == 2) tx_data = TGM::Data((UINT16)_rcvdelm);
    else if (datalen == 4) tx_data = TGM::Data((UINT32)_rcvdelm);
    else if (datalen == 8) tx_data = TGM::Data((UINT64)_rcvdelm);
    else tx_data = TGM::Data((UINT8&)_rcvdelm);
}


template <class TCHeader, class TCPayload, class TRHeader, class TRPayload>
TGM::Map<TRHeader, TRPayload> SISProtocol::transceive_param(TGM::SercosParamVar _paramvar, const USHORT &_paramnum, BYTE _service, TGM::Data const * const _data, TGM::SercosDatablock _attribute)
{
    // Build Telegrams ...
    TGM::Bitfields::SercosParamControl    ParamControl(_attribute);
    TGM::Bitfields::SercosParamIdent    ParamIdent(_paramvar, _paramnum);

    // Mapping for SEND Telegram
    TGM::Map<TCHeader, TCPayload>
        tx_tgm(
            // Init header
            TCHeader(SIS_ADDR_MASTER, SIS_ADDR_SLAVE, _service, TGM::Bitfields::HeaderControl(TGM::TypeCommand)),
            // Init payload
            TCPayload(ParamControl, SIS_ADDR_SLAVE, ParamIdent, *_data)
        );

    // Set payload size
    tx_tgm.Mapping.Header.set_DatL(tx_tgm.Mapping.Payload.get_size());

    // Calculate Checksum
    tx_tgm.Mapping.Header.calc_checksum(&tx_tgm.Raw);

    if (!check_boundaries(tx_tgm))
        throw SISProtocol::ExceptionGeneric(-1, "Boundaries are out of spec.
 Telegram is not ready to be sent.");

    // Mapping for RECEPTION Telegram
    TGM::Map<TRHeader, TRPayload> rx_tgm;

    //  Transceive ...
    // Send and receive
    transceiving<   TCHeader, TCPayload,
        TRHeader, TRPayload >
        (tx_tgm, rx_tgm);

    return rx_tgm;
}


void SISProtocol::set_parameter_listsize(TGM::SercosParamVar param_variant, USHORT &
param_number, const size_t &datalen, const USHORT & segment_position, bool retain_following_segments)
{
    // Getting Parameter header ...
    USHORT size = datalen;
    USHORT pos = 0;
    auto rx_tgm = transceive_list
        <TGM::Header, TGM::Commands::SercosList,
TGM::Header, TGM::Reactions::SercosList>
        (param_variant, param_number, SIS_SERVICE_SERCOS_LIST_READ, size, pos);

    UINT32 param_header = rx_tgm.Mapping.Payload.Bytes.toUINT32();
    // Maximum possible size of parameter list
    UINT16 param_size_max = param_header >> 16;
    // Actual size of parameter list
    UINT16 param_size_cur = param_header & 0xFFFF;

    // In case of listsize has to be changed due to write-list action from within the list, it is up to the
 caller to decide how to handle list size update
    param_size_cur = retain_following_segments ?
        std::max<UINT16>(param_size_cur, segment_position * datalen) :
        static_cast<UINT16>(segment_position * datalen);

    if (param_size_cur > param_size_max)    return;
    if (param_size_cur == param_size_max)   return;

    // Update the Parameter header ...
    TGM::Data *new_header = new TGM::Data((UINT32)((param_size_max << 16) |
param_size_cur));

    transceive_list
        <TGM::Header, TGM::Commands::SercosList,
TGM::Header, TGM::Reactions::SercosList>
        (param_variant, param_number, SIS_SERVICE_SERCOS_LIST_WRITE, size, pos
, new_header);
}


template<class TCHeader, class TCPayload, class TRHeader, class TRPayload>
TGM::Map<TRHeader, TRPayload> SISProtocol::transceive_list(
TGM::SercosParamVar _paramvar, const USHORT & _paramnum, BYTE _service, USHORT &
_element_size, USHORT & _list_offset, TGM::Data const * const _data,
TGM::SercosDatablock _attribute)
{

    // Build Telegrams ...
    TGM::Bitfields::SercosParamControl    sercos_control(_attribute);
    TGM::Bitfields::SercosParamIdent    ParamNum(_paramvar, _paramnum);

    // Mapping for SEND Telegram
    TGM::Map<TCHeader, TCPayload>
        tx_tgm(
            // Init header
            TCHeader(SIS_ADDR_MASTER, SIS_ADDR_SLAVE, _service,
TGM::Bitfields::HeaderControl(TGM::TypeCommand)),
            // Init payload
            TCPayload(sercos_control, SIS_ADDR_SLAVE, ParamNum, _list_offset, _element_size,
*_data)
        );

    // Set payload size
    tx_tgm.Mapping.Header.set_DatL(tx_tgm.Mapping.Payload.get_size());

    // Calculate Checksum
    tx_tgm.Mapping.Header.calc_checksum(&tx_tgm.Raw);

    if (!check_boundaries(tx_tgm))
        throw SISProtocol::ExceptionGeneric(-1, "Boundaries are out of spec.
 Telegram is not ready to be sent.");

    // Mapping for RECEPTION Telegram
    TGM::Map<TRHeader, TRPayload> rx_tgm;

    //  Transceive ...
    // Send and receive
    transceiving<   TCHeader, TCPayload,
        TRHeader, TRPayload >
        (tx_tgm, rx_tgm);

    return rx_tgm;
}


void SISProtocol::get_parameter_attributes(TGM::SercosParamVar _paramvar, const USHORT &
_paramnum, UINT8& _scalefactor, size_t& _datalen)
{
    STACK;

    // Communication with Telegrams ...
    BYTE service = SIS_SERVICE_SERCOS_PARAM_READ;

    auto rx_tgm = transceive_param
                    <TGM::Header, TGM::Commands::SercosParam,
TGM::Header, TGM::Reactions::SercosParam>
                    (_paramvar, _paramnum, service, new TGM::Data(),
TGM::Datablock_Attribute);

    // Read back Datablock ...
    UINT32 attr = rx_tgm.Mapping.Payload.Bytes.toUINT32();
    TGM::Bitfields::SercosParamAttribute sercos_attribute(attr);

    _datalen = 1;
    if (sercos_attribute.Bits.DataLen == TGM::Datalen_2ByteList) _datalen = 2;
    else if (sercos_attribute.Bits.DataLen == TGM::Datalen_4ByteList) _datalen =
4;
    else if (sercos_attribute.Bits.DataLen == TGM::Datalen_8ByteList) _datalen =
8;
    else if (sercos_attribute.Bits.DataLen == TGM::Datalen_2ByteParam) _datalen
= 2;
    else if (sercos_attribute.Bits.DataLen == TGM::Datalen_4ByteParam) _datalen
= 4;
    else if (sercos_attribute.Bits.DataLen == TGM::Datalen_8ByteParam) _datalen
= 8;

    _scalefactor = 0xFF & sercos_attribute.Bits.ScaleFactor;
}


template <class TCHeader, class TCPayload, class TRHeader, class TRPayload>
void SISProtocol::transceiving(TGM::Map<TCHeader, TCPayload>& tx_tgm,
TGM::Map<TRHeader, TRPayload>& rx_tgm)
{
    STACK;

    // Lock mutex to set the semaphore, so that the SIS access be reentrant
    mutex_sis.lock();

    // Transceiver lengths
    size_t tx_payload_len = tx_tgm.Mapping.Payload.get_size();
    size_t tx_header_len = tx_tgm.Mapping.Header.get_size();

    // Receiver lengths
    size_t rx_header_len = tx_tgm.Mapping.Header.get_size();
    size_t rx_payload_len = 0;

    // Clear buffers
    m_serial.Purge();

    // Write ...
    m_serial.Write(tx_tgm.Raw.Bytes, tx_header_len + tx_payload_len);

    // Read ...
    bool bContd = true;
    DWORD rcvd_cur = 0;
    DWORD rcvd_rcnt = 0;

    do
    {
        // Wait for an event
        m_serial.WaitEvent(0, RS232_READ_TIMEOUT);

        // Save event
        const CSerial::EEvent event = m_serial.GetEventType();

        // Handle Break event
```

```cpp
        if (event & CSerial::EEventBreak)
            throw SISProtocol::ExceptionTransceiveFailed(CSerial::EEventBreak, "Break event occurred. Transceive has been aborted.", true);

        // Handle error event
        if (event & CSerial::EEventError)
            throw_rs232_error_events(m_serial.GetError());

        // Handle Bytes receive event
        if (event & CSerial::EEventRecv)
        {
            // Read header Bytes
            m_serial.Read(rx_tgm.Raw.Bytes + rcvd_rcnt, RS232_BUFFER - rcvd_rcnt, &rcvd_cur, 0, RS232_READ_TIMEOUT);

            // Loop back if nothing received
            if (rcvd_cur == 0) continue;

            // Hold back number of already received bytes
            rcvd_rcnt += rcvd_cur;

            // It is assumed that if the number of received bytes is bigger than 4,
            // which is the position of the payload length, the length can be read out.
            if (rcvd_rcnt > 4)
            {
                rx_payload_len = rx_tgm.Mapping.Header.DatL;
                rx_tgm.Mapping.Payload.Bytes.set_size(rx_payload_len - rx_tgm.Mapping.Payload.get_head_size());
            }

            // Length of payload is zero --> No payload received
            if (rx_payload_len == 0)
            {
                std::string tx_hexstream = hexprint_bytestream(tx_tgm.Raw.Bytes, tx_header_len + tx_payload_len);
                std::string rx_hexstream = hexprint_bytestream(rx_tgm.Raw.Bytes, rx_header_len);
                throw SISProtocol::ExceptionTransceiveFailed(-1, sformat("Reception Telegram received without payload, but just the header.\nRecption Header bytestream: %s.\nCommand Telegram bytestream was: %s.", rx_hexstream.c_str(), tx_hexstream.c_str()), true);
            }


            // Complete Telegram received
            if (rx_header_len + rx_payload_len <= rcvd_rcnt)
            {
                if (rx_tgm.Mapping.Payload.Status)
                {
                    std::string tx_hexstream = hexprint_bytestream(tx_tgm.Raw.Bytes, tx_header_len + tx_payload_len);

                    USHORT error = rx_tgm.Mapping.Payload.Error;

                    if (error == 0x800C || error == 0x800B || error == 0x8001)
                    {
                        mutex_sis.unlock();
                        transceiving<TCHeader, TCPayload, TRHeader, TRPayload>(tx_tgm, rx_tgm);
                    }
                    else
                        throw SISProtocol::ExceptionSISError(rx_tgm.Mapping.Payload.Status, rx_tgm.Mapping.Payload.Error, tx_hexstream);
                }

                bContd = false;
            }
        }

    } while (bContd);

    // Unlock mutex to unset the semaphore
    mutex_sis.unlock();
}


template<class THeader, class TPayload>
bool SISProtocol::check_boundaries(TGM::Map<THeader, TPayload>& _tgm)
{
    STACK;

    size_t tgm_size = _tgm.Mapping.Header.get_size() + _tgm.Mapping.Payload.get_size();
    if (tgm_size <= RS232_BUFFER) return true;

    return false;
}


std::string SISProtocol::hexprint_bytestream(const BYTE * _bytestream, const size_t _len)
```

```
{
    STACK;

    std::string buf;

    for (size_t i = 0; i < _len; i++)
        buf.append(sformat("%02X ", (BYTE)_bytestream[i]));

    return buf;
}

void SISProtocol::throw_rs232_error_events(CSerial::EError _err)
{
    STACK;

    switch (_err)
    {
    case CSerial::EErrorBreak:
        throw SISProtocol::ExceptionTransceiveFailed(CSerial::EErrorBreak, "Break condition occurred. Transceive has been aborted.", true);

    case CSerial::EErrorFrame:
        throw SISProtocol::ExceptionTransceiveFailed(CSerial::EErrorFrame, "Framing error occurred. Transceive has been aborted.", true);

    case CSerial::EErrorIOE:
        throw SISProtocol::ExceptionTransceiveFailed(CSerial::EErrorIOE, "IO device error occurred. Transceive has been aborted.", true);

    case CSerial::EErrorMode:
        throw SISProtocol::ExceptionTransceiveFailed(CSerial::EErrorMode, "Unsupported mode detected. Transceive has been aborted.", true);

    case CSerial::EErrorOverrun:
        throw SISProtocol::ExceptionTransceiveFailed(CSerial::EErrorOverrun, "Buffer overrun detected. Transceive has been aborted.", true);

    case CSerial::EErrorRxOver:
        throw SISProtocol::ExceptionTransceiveFailed(CSerial::EErrorRxOver, "Input buffer overflow detected. Transceive has been aborted.", true);

    case CSerial::EErrorParity:
        throw SISProtocol::ExceptionTransceiveFailed(CSerial::EErrorParity, "Input parity occurred. Transceive has been aborted.", true);

    case CSerial::EErrorTxFull:
        throw SISProtocol::ExceptionTransceiveFailed(CSerial::EErrorTxFull, "Output buffer full. Transceive has been aborted.", true);

    default:
        throw SISProtocol::ExceptionTransceiveFailed(CSerial::EErrorBreak, "Unknown error occurred. Transceive has been aborted.", true);
    }
}
```

6.6 SISProtocol.h File Reference

Classes

- class SISProtocol

    Class to hold functions an members for the SIS protocol support.
- class SISProtocol::ExceptionGeneric

    Generic exceptions for SIS protocol.
- class SISProtocol::ExceptionTransceiveFailed

    Specific exception handling of SIS Protocol transceiving failed.
- class SISProtocol::ExceptionSISError

    Specific exception handling of SIS Protocol error codes.

Macros

- #define RS232_BUFFER 254
- #define RS232_READ_LOOPS_MAX 100

- #define RS232_READ_TIMEOUT 1000
- #define SIS_ADDR_MASTER 0x00

  Defines address master.
- #define SIS_ADDR_SLAVE 0x01

  Defines sis address slave. '128' is used for peer-to-peer communication.
- #define SIS_ADDR_UNIT 0x01

  Address unit. For Indradrive, this value can be found at P-0-4022.
- #define MAX_COMMANDCHECK_ITERATIONS 300

  Defines the maximum iterations of checking the successful executing of a Indradrive command.

6.6.1 Macro Definition Documentation

6.6.1.1 RS232_BUFFER

```
#define RS232_BUFFER 254
```

Definition at line 15 of file SISProtocol.h.

6.6.1.2 RS232_READ_LOOPS_MAX

```
#define RS232_READ_LOOPS_MAX 100
```

Definition at line 16 of file SISProtocol.h.

6.6.1.3 RS232_READ_TIMEOUT

```
#define RS232_READ_TIMEOUT 1000
```

Definition at line 17 of file SISProtocol.h.

6.6.1.4 SIS_ADDR_MASTER

```
#define SIS_ADDR_MASTER 0x00
```

Defines address master.

Definition at line 21 of file SISProtocol.h.

6.6.1.5 SIS_ADDR_SLAVE

```
#define SIS_ADDR_SLAVE 0x01
```

Defines sis address slave. '128' is used for peer-to-peer communication.

Definition at line 23 of file SISProtocol.h.

6.6.1.6 SIS_ADDR_UNIT

```
#define SIS_ADDR_UNIT 0x01
```

Address unit. For Indradrive, this value can be found at P-0-4022.

Definition at line 25 of file SISProtocol.h.

6.6.1.7 MAX_COMMANDCHECK_ITERATIONS

```
#define MAX_COMMANDCHECK_ITERATIONS 300
```

Defines the maximum iterations of checking the successful executing of a Indradrive command.

Definition at line 29 of file SISProtocol.h.

6.7 SISProtocol.h

```
00001 #ifndef _SISPROTOCOL_H_
00002 #define _SISPROTOCOL_H_
00003
00004 #include <Windows.h>
00005 #include <string>
00006 #include <mutex>
00007
00008 #include "debug.h"
00009 #include "helpers.h"
00010 #include "RS232.h"
00011 #include "Telegrams.h"
00012
00013
00014
00015 #define RS232_BUFFER            254
00016 #define RS232_READ_LOOPS_MAX    100
00017 #define RS232_READ_TIMEOUT      1000
00018
00019
00021 #define SIS_ADDR_MASTER         0x00
00022 #define SIS_ADDR_SLAVE          0x01
00024 #define SIS_ADDR_UNIT           0x01
00026
00027
00029 #define MAX_COMMANDCHECK_ITERATIONS 300
00030
00031
00033 class SISProtocol
00034 {
00035 public:
00037     class ExceptionGeneric;
00039     class ExceptionTransceiveFailed;
00041     class ExceptionSISError;
00042
00044     typedef enum SIS_SERVICES
00045     {
00046         SIS_SERVICE_INIT_COMM = 0x03,
00047         SIS_SERVICE_SEQUENTIALOP = 0x04,

        SIS_SERVICE_SERCOS_PARAM_READ = 0x10,
        SIS_SERVICE_SERCOS_LIST_READ = 0x11,
        SIS_SERVICE_SERCOS_READ_PHASE = 0x12,
        SIS_SERVICE_SERCOS_SWITCH_PHASE = 0x1D,
        SIS_SERVICE_SERCOS_LIST_WRITE = 0x1E,
        SIS_SERVICE_SERCOS_PARAM_WRITE = 0x1F
    } SIS_SERVICES;

    typedef enum BAUDRATE
    {
        Baud_9600  = 0b00000000,
        Baud_19200    = 0b00000001,
        Baud_38400    = 0b00000010,
        Baud_57600    = 0b00000100,
        Baud_115200 = 0b00001000
    } BAUDRATE;

    SISProtocol();
    virtual ~SISProtocol();



    void open(const wchar_t * _port = L"COM1");
    void close();

    void set_baudrate(BAUDRATE baudrate);

    void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
UINT32& _rcvddata);
    void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
UINT64& _rcvddata);
    void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
DOUBLE& _rcvddata);
    void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
char _rcvddata[TGM_SIZEMAX_PAYLOAD]);

    void read_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
USHORT _elm_pos, UINT32& _rcvdelm);
    void read_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
USHORT _elm_pos, UINT64& _rcvdelm);
    void read_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
USHORT _elm_pos, DOUBLE& _rcvdelm);

    void write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
 const UINT32 _data);
    void write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
 const UINT64 _data);
    void write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
 const DOUBLE _data);

    void write_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
USHORT _elm_pos, const UINT32 _rcvdelm);
    void write_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
USHORT _elm_pos, const UINT64 _rcvdelm);
    void write_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
USHORT _elm_pos, const DOUBLE _rcvdelm);

    void execute_command(TGM::SercosParamVar _paramvar, USHORT _paramnum)
;


private:

    inline void get_parameter_attributes(TGM::SercosParamVar _paramvar, const USHORT &
_paramnum, UINT8& _scalefactor, size_t& _datalen);
    inline void get_parameter_status(const TGM::SercosParamVar _paramvar, const USHORT &
_paramnum, TGM::SercosCommandstatus& _datastatus);

    template <class TCHeader, class TCPayload, class TRHeader, class TRPayload>
    TGM::Map<TRHeader, TRPayload> transceive_param(
TGM::SercosParamVar _paramvar, const USHORT &_paramnum, BYTE _service,
TGM::Data const * const _data = new TGM::Data(),
TGM::SercosDatablock _attribute =
TGM::Datablock_OperationData);

    template <class TCHeader, class TCPayload, class TRHeader, class TRPayload>
    TGM::Map<TRHeader, TRPayload> transceive_list(
TGM::SercosParamVar _paramvar, const USHORT &_paramnum, BYTE _service, USHORT &
_element_size, USHORT & _list_offset, TGM::Data const * const _data = new
TGM::Data(), TGM::SercosDatablock _attribute =
TGM::Datablock_OperationData);

    template <class THeader, class TPayload>
    inline bool check_boundaries(TGM::Map<THeader, TPayload>& _tgm);

    static std::string hexprint_bytestream(const BYTE * _bytestream, const size_t _len);
```

```cpp

    inline INT64 get_sized_data(TGM::Data& rx_data, const size_t &datalen);
    inline void set_sized_data(TGM::Data& tx_data, const size_t &datalen, UINT64& _rcvdelm);
    inline void set_parameter_listsize(TGM::SercosParamVar param_variant, USHORT& param_number, const size_t& datalen, const USHORT& segment_position, bool retain_following_segments = false);

private:

    template <class TCHeader, class TCPayload, class TRHeader, class TRPayload>
    void transceiving(TGM::Map<TCHeader, TCPayload>& tx_tgm, TGM::Map<TRHeader, TRPayload>& rx_tgm);

    static void throw_rs232_error_events(CSerial::EError _err);

private:
    CSerial m_serial;

    std::mutex mutex_sis;
};

class SISProtocol::ExceptionGeneric : public std::exception
{
public:
    bool warning;

    ExceptionGeneric(
        int _status,
        const std::string _trace_log,
        bool _warning = false) :

        m_status(_status),
        m_message(_trace_log),
        warning(_warning)
    {}

    virtual const char* what() const throw ()
    {
#ifdef NDEBUG
        return str2char(sformat("SIS Protocol exception caused: %s ### STATUS=0x%04x (%d) ### MESSAGE='%s'", Stack::GetTraceString().c_str(), m_status, m_status, m_message.c_str()));
#else
        const char* ex = str2char(sformat("SIS Protocol exception caused: %s ### STATUS=0x%04x (%d) ### MESSAGE='%s'", Stack::GetTraceString().c_str(), m_status, m_status, m_message.c_str()));
        OutputDebugStringA((LPCSTR)ex);
        return ex;
#endif
    }

    int get_status() { return m_status; }

protected:
    int m_status;

    std::string m_message;
};

class SISProtocol::ExceptionTransceiveFailed : public SISProtocol::ExceptionGeneric
{
public:
    ExceptionTransceiveFailed(
        int _status,
        const std::string _message,
        bool _warning = false) :

        ExceptionGeneric(_status, _message, _warning)
    {}
    ~ExceptionTransceiveFailed() throw() {}

    virtual const char* what() const throw ()
    {
#ifdef NDEBUG
        return str2char(sformat("SIS Protocol reception fail caused: STATUS=0x%04x (%d) ### MESSAGE='%s'", m_status, m_status, m_message.c_str()));
#else
        const char* ex = str2char(sformat("SIS Protocol reception fail caused: STATUS=0x%04x (%d) ### MESSAGE='%s'", m_status, m_status, m_message.c_str()));
        OutputDebugStringA((LPCSTR)ex);
        return ex;
#endif
    }
};

class SISProtocol::ExceptionSISError : public SISProtocol::ExceptionGeneric
{
```

```
public:
    ExceptionSISError(
        int _status,
        int _code,
        const std::string _bytestream,
        bool _warning = false) :

        ExceptionGeneric(_status, std::string(), _warning),
        m_errorcode(_code),
        m_bytestream(_bytestream)
    {}
    ~ExceptionSISError() throw() {}

    virtual const char* what() const throw ()
    {
#ifdef NDEBUG
        return str2char(sformat("(Return code: %d) SIS Protocol Error code returned has been received:
 0x%04X.\nOriginal Telegram bytestream: %s", m_status, m_errorcode,
m_bytestream.c_str()));
#else
        const char* ex = str2char(sformat("(Return code: %d) SIS Protocol Error code returned has been
 received: 0x%04X.\nOriginal Telegram bytestream: %s", m_status, m_errorcode,
m_bytestream.c_str()));
        OutputDebugStringA((LPCSTR)ex);
        return ex;
#endif
    }

    int get_errorcode() { return m_errorcode; }

protected:
    int m_errorcode;
    std::string m_bytestream;
};

#endif /* _SISPROTOCOL_H_ */
```

6.8 Telegrams.h File Reference

Contains struct definitions for different types of Telegrams.

Classes

- struct TGM::Data

    Struct to hold payload Bytes in a command payload.
- struct TGM::Bytestream

    Container for Telegram in raw Bytes.
- union TGM::Map< THeader, TPayload >

    Templated mapping union to transfer raw TGM Bytes from/to specialized Bytes class.
- struct TGM::Map< THeader, TPayload >::Mapping

    Specialized Bytes class, comprising structure payload head and Bytes.
- struct TGM::Header

    The Telegram Header contains all information required for conducting orderly telegram traffic..
- struct TGM::HeaderExt

    Extended Telegram Header to be used for Routing and Sequential Telegrams.
- struct TGM::Commands::Subservice

    Representation of the PAYLOAD for a Subservice command.
- struct TGM::Commands::SercosParam

    Sercos Command Telegram used for reading/writing single parameter from/to slave.
- struct TGM::Commands::SercosList

    Sercos Command Telegram used for reading/writing single elements in lists from/to slave.
- struct TGM::Reactions::Subservice

    Representation of the payload for a Subservice reaction.
- struct TGM::Reactions::SercosParam

    Representation of the payload for a Sercos Parameter reaction.
- struct TGM::Reactions::SercosList

    Sercos Command Telegram used for reading/writing single elements in lists from/to slave..

Namespaces

- TGM

  Grouping structs/enums/unions for a SIS Telegram.
- TGM::Commands

  Grouping SIS Telegram Payload struct definitions for commands.
- TGM::Reactions

  Grouping SIS Telegram Payload struct definitions for reception.

Macros

- #define TGM_SIZE_HEADER 8
- #define TGM_SIZE_HEADER_EXT 16
- #define TGM_SIZEMAX_PAYLOAD 246
- #define TGM_SIZEMAX 254

Typedefs

- typedef struct TGM::Data TGM::Data

  Struct to hold payload Bytes in a command payload.
- typedef struct TGM::Bytestream TGM::Bytestream

  Container for Telegram in raw Bytes.
- typedef struct TGM::Header TGM::Header

  The Telegram Header contains all information required for conducting orderly telegram traffic..
- typedef TGM::HeaderExt TGM::HeaderExt

  Extended Telegram Header to be used for Routing and Sequential Telegrams.
- typedef struct TGM::Commands::Subservice TGM::Commands::Subservice

  Representation of the PAYLOAD for a Subservice command.
- typedef struct TGM::Commands::SercosParam TGM::Commands::SercosParam

  Sercos Command Telegram used for reading/writing single parameter from/to slave.
- typedef struct TGM::Commands::SercosList TGM::Commands::SercosList

  Sercos Command Telegram used for reading/writing single elements in lists from/to slave.
- typedef struct TGM::Reactions::Subservice TGM::Reactions::Subservice

  Representation of the payload for a Subservice reaction.
- typedef struct TGM::Reactions::SercosParam TGM::Reactions::SercosParam

  Representation of the payload for a Sercos Parameter reaction.
- typedef struct TGM::Reactions::SercosList TGM::Reactions::SercosList

  Sercos Command Telegram used for reading/writing single elements in lists from/to slave..

6.8.1 Detailed Description

Contains struct definitions for different types of Telegrams.

Definition in file Telegrams.h.

6.8.2 Macro Definition Documentation

6.8.2.1 TGM_SIZE_HEADER

```
#define TGM_SIZE_HEADER 8
```

Definition at line 17 of file Telegrams.h.

6.8.2.2 TGM_SIZE_HEADER_EXT

```
#define TGM_SIZE_HEADER_EXT 16
```

Definition at line 18 of file Telegrams.h.

6.8.2.3 TGM_SIZEMAX_PAYLOAD

```
#define TGM_SIZEMAX_PAYLOAD 246
```

Definition at line 19 of file Telegrams.h.

6.8.2.4 TGM_SIZEMAX

```
#define TGM_SIZEMAX 254
```

Definition at line 20 of file Telegrams.h.

6.9 Telegrams.h

```
00001
00004 #ifndef _TELEGRAMS_H_
00005 #define _TELEGRAMS_H_
00006
00007
00008 #include <Windows.h>
00009 #include <vector>
00010 #include <algorithm>
00011 #include <numeric>
00012 #include <type_traits>
00013
00014 #include "Telegrams_Bitfields.h"
00015
00016
00017 #define TGM_SIZE_HEADER     8
00018 #define TGM_SIZE_HEADER_EXT 16
00019 #define TGM_SIZEMAX_PAYLOAD 246
00020 #define TGM_SIZEMAX         254
00021
00022
00023
00025 namespace TGM
00026 {
00029     typedef struct Data
00030     {
00032         BYTE    Bytes[TGM_SIZEMAX_PAYLOAD];
00034         size_t  Size;
00035
00039         Data(std::vector<BYTE> _data = std::vector<BYTE>())
00040         {
00041             clear();
00042
00043             for (std::vector<BYTE>::iterator it = _data.begin(); it != _data.end(); ++it)
00044                 operator<<(*it);
```

```

            Size = _data.size();
        }

        Data(UINT8 _data)
        {
            clear();

            operator<<(_data);
        }

        Data(UINT16 _data)
        {
            clear();

            operator<<(_data & 0xFF);
            operator<<((_data & 0xFF00) >> 8);
        }

        Data(UINT32 _data)
        {
            clear();

            operator<<(_data & 0xFF);
            operator<<((_data & 0xFF00) >> 8);
            operator<<((_data & 0xFF0000) >> 16);
            operator<<((_data & 0xFF000000) >> 24);
        }

        Data(UINT64 _data)
        {
            clear();

            operator<<(_data & 0xFF);
            operator<<((_data & 0xFF00) >> 8);
            operator<<((_data & 0xFF0000) >> 16);
            operator<<((_data & 0xFF000000) >> 24);
            operator<<((_data & 0xFF00000000) >> 32);
            operator<<((_data & 0xFF0000000000) >> 40);
            operator<<((_data & 0xFF000000000000) >> 48);
            operator<<((_data & 0xFF00000000000000) >> 54);
        }

        BYTE at(UINT32 _idx)
        {
            return Bytes[_idx];
        }

        std::vector<BYTE> toVector()
        {
            std::vector<BYTE> out;

            for (int i = 0; i < Size; i++)
                out.push_back(Bytes[i]);

            return out;
        }

        UINT64 toUINT64()
        {
            UINT64 out = 0;

            for (int i = 0; i < std::min<size_t>(Size, 8); i++)
                out |= Bytes[i] << (i * 8);

            return out;
        }

        UINT32 toUINT32()
        {
            UINT32 out = 0;

            for (int i = 0; i < std::min<size_t>(Size, 4); i++)
                out |= Bytes[i] << (i * 8);

            return out;
        }

        UINT16 toUINT16()
        {
            UINT16 out = 0;

            for (int i = 0; i < std::min<size_t>(Size, 2); i++)
                out |= Bytes[i] << (i * 8);

            return out;
        }
        UINT8 toUINT8()
        {
            return toBYTE();
        }

        BYTE toBYTE()
        {
            return Size > 0 ? (BYTE)Bytes[0] : (BYTE)0;
        }

        void clear()
        {
            memset(Bytes, 0, sizeof(Bytes));
            Size = 0;
        }

        Data& operator<<(const BYTE& rhs)
        {
            Bytes[Size++] = rhs;
            return *this;
        }

        size_t get_size() { return Size; }

        void set_size(size_t _size) { Size = _size; }

    } Data;


    typedef struct Bytestream
    {
        BYTE Bytes[TGM_SIZEMAX];

        Bytestream() { clear(); }

        void clear()
        {
            memset(Bytes, 0, sizeof(Bytes));
        }
    } Bytestream;


    template <class THeader, class TPayload>
    union Map
    {
    public:
        Bytestream Raw;

#pragma pack(push,1)
        struct Mapping
        {
            THeader     Header;
            TPayload    Payload;

            Mapping(THeader& _header, TPayload _payload) :
                Header(_header),
                Payload(_payload)
            {};
        } Mapping;
#pragma pack(pop)

        Map(THeader& _header = THeader(), TPayload& _payload = TPayload()) :
            Mapping(_header, _payload)
        {};
        ~Map() {};

        void set(THeader& _header, TPayload& _payload)
        {
            Mapping = Mapping(_header, _payload);
        }
    };


#pragma pack(push,1)
    typedef struct Header
    {
        BYTE StZ = 0x02;

        BYTE CS;

        BYTE DatL;

        BYTE DatLW;

        BYTE Cntrl;

        BYTE Service;

        BYTE AdrS;

        BYTE AdrE;

        Header(BYTE _addr_master = 0, BYTE _addr_slave = 0, BYTE _service = 0,
TGM::Bitfields::HeaderControl _cntrl =
TGM::Bitfields::HeaderControl()) :
            StZ(0x02),
            CS(0),
            DatL(get_size()),
            DatLW(get_size()),
            Cntrl(_cntrl.Value),
            Service(_service),
            AdrS(_addr_master),
            AdrE(_addr_slave)
        {}

        BYTE get_sum(bool exclude_cs = true)
        {
            BYTE res = StZ + DatL + DatLW + Cntrl + Service + AdrS + AdrE;

            if (!exclude_cs) res += CS;

            return res;
        }

        size_t get_size() { return sizeof(*this); }

        inline void set_DatL(size_t _payload_len) { DatL = DatLW = (BYTE)_payload_len; }

        inline size_t get_DatL() { return DatL; }

        void calc_checksum(TGM::Bytestream * _payload)
        {
            // Sum of payload
            BYTE sum_of_payload = 0;
            for (int i = TGM_SIZE_HEADER; i < TGM_SIZE_HEADER + get_DatL(); i
++)
                sum_of_payload += (BYTE)_payload->Bytes[i];

            // Calc difference
            BYTE diff_cs = get_sum() + sum_of_payload;

            // Calc negation and assign to checksum (Byte 1)
            CS = (BYTE)0 - diff_cs;
        }
    } Header;
#pragma pack(pop)


#pragma pack(push,1)
    typedef struct HeaderExt : Header
    {
        BYTE AdrES1;

        BYTE AdrES2;

        BYTE AdrES3;

        BYTE AdrES4;

        BYTE AdrES5;

        BYTE AdrES6;

        BYTE AdrES7;

        BYTE PaketN;

    } HeaderExt;
#pragma pack(pop)


    namespace Commands
    {

#pragma pack(push,1)
        typedef struct Subservice
        {
            BYTE    RecipientAddr;
            BYTE    ServiceNumber;
            Data    Bytes;

            Subservice(
                BYTE _addr = 0,
                BYTE _subservice = 0,
                Data _data = Data()) :
                RecipientAddr(_addr),
                ServiceNumber(_subservice),
                Bytes(_data)
            {}

            void clear() { RecipientAddr = ServiceNumber = 0; }

            size_t get_head_size() { return 2; }

            size_t get_size() { return get_head_size() + Bytes.get_size(); }

        } Subservice;
#pragma pack(pop)


#pragma pack(push,1)
        typedef struct SercosParam
        {
            BYTE Control;

            BYTE UnitAddr;

            BYTE ParamType;

            USHORT ParamNum;

            Data Bytes;

            SercosParam(
                TGM::Bitfields::SercosParamControl _control =
TGM::Bitfields::SercosParamControl(),
                BYTE _unit_addr = 0,
                TGM::Bitfields::SercosParamIdent _param_ident =
TGM::Bitfields::SercosParamIdent(),
                TGM::Data _data = Data()) :
                Control(_control.Value),
                UnitAddr(_unit_addr),
                ParamType(0),
                ParamNum(_param_ident.Value),
                Bytes(_data)
            {}

            void clear()
            {
                Control = 0;
                UnitAddr = 0;
                ParamType = 0;
                ParamNum = 0;
                Bytes.clear();
            }

            size_t get_head_size() { return 5; }

            size_t get_size() { return get_head_size() + Bytes.get_size(); }

        }  SercosParam;
#pragma pack(pop)


#pragma pack(push,1)
        typedef struct SercosList
        {
            BYTE Control;

            BYTE UnitAddr;

            BYTE ParamType;

            USHORT ParamNum;

            USHORT ListOffset;

            USHORT SegmentSize;

            Data Bytes;

            SercosList(
                TGM::Bitfields::SercosParamControl _ControlByte =
TGM::Bitfields::SercosParamControl(),
                BYTE _unit_addr = 0,
                TGM::Bitfields::SercosParamIdent _ParamIdent =
TGM::Bitfields::SercosParamIdent(),
                USHORT _ListOffset = 0,
                USHORT _SegmentSize = 0,
                TGM::Data _PayloadData = Data()) :
```

```

                Control(_ControlByte.Value),
                UnitAddr(_unit_addr),
                ParamType(0),
                ParamNum(_ParamIdent.Value),
                ListOffset(_ListOffset),
                SegmentSize(_SegmentSize),
                Bytes(_PayloadData)
            {}

            void clear()
            {
                Control = UnitAddr = ParamNum = ListOffset = SegmentSize = 0;
                Bytes.clear();
            }

            size_t get_head_size() { return 9; }

            size_t get_size() { return get_head_size() + Bytes.get_size(); }

        }  SercosList;
#pragma pack(pop)
    }



    namespace Reactions
    {
#pragma pack(push,1)
        typedef struct Subservice
        {
            BYTE    Status;

            BYTE    RecipientAddr;

            BYTE    ServiceNumber;

            union
            {
                Data    Bytes;
                BYTE    Error;
            };

            Subservice() :
                Status(1),
                RecipientAddr(0),
                ServiceNumber(0),
                Error(0)
            {}

            void clear()
            {
                Status = 1;
                RecipientAddr = ServiceNumber = 0;
                Bytes.clear();
            }

            size_t get_head_size() { return 3; }

            size_t get_size() { return get_head_size() + Bytes.get_size(); }

        } Subservice;
#pragma pack(pop)


#pragma pack(push,1)
        typedef struct SercosParam
        {
            BYTE Status;

            BYTE Control;

            BYTE UnitAddr;

            union
            {
                Data    Bytes;
                USHORT  Error;
            };

            SercosParam() :
                Status(1),
                Control(0),
                UnitAddr(0),
                Bytes(TGM::Data())
            {}

            void clear()
            {
                Status = 1;
                Control = UnitAddr = 0;
                Bytes.clear();
            }

            size_t get_head_size() { return 3; }

            size_t get_size() { return get_head_size() + Bytes.get_size(); }

        } SercosParam;
#pragma pack(pop)


#pragma pack(push,1)
        typedef struct SercosList
        {
            BYTE Status;

            BYTE Control;

            BYTE UnitAddr;

            union
            {
                Data Bytes;
                USHORT Error;
            };

            SercosList() :
                Status(1),
                Control(0),
                UnitAddr(0),
                Bytes(TGM::Data())
            {}

            void clear()
            {
                Status = 1;
                Control = UnitAddr = 0;
                Bytes.clear();
            }

            size_t get_head_size() { return 3; }

            size_t get_size() { return get_head_size() + Bytes.get_size(); }

        }  SercosList;
#pragma pack(pop)
    }
}


#endif /* _TELEGRAMS_H_ */
```

6.10 Telegrams_Bitfields.h File Reference

Contains enums, structs and unions to make Telegram creation, transmission and reception as flexible as possible.

Classes

- struct TGM::Bitfields::HeaderControl

  Control byte consisting of several bit fields. Size: 8 bit.
- struct TGM::Bitfields::SercosParamControl

  The control byte specifies how a Bytes block element of a parameter is accessed.
- struct TGM::Bitfields::SercosParamIdent

  Identification of the parameter. Size: 16 bit.
- struct TGM::Bitfields::SercosParamAttribute

  Attribute for a SERCOS parameter that is callable via SercosDatablock.

Namespaces

- TGM

  Grouping structs/enums/unions for a SIS Telegram.
- TGM::Bitfields

  Grouping unions that merge together both raw and structured information.

Typedefs

- typedef struct TGM::Bitfields::HeaderControl TGM::Bitfields::HeaderControl

  Control byte consisting of several bit fields. Size: 8 bit.
- typedef struct TGM::Bitfields::SercosParamControl TGM::Bitfields::SercosParamControl

  The control byte specifies how a Bytes block element of a parameter is accessed.
- typedef struct TGM::Bitfields::SercosParamIdent TGM::Bitfields::SercosParamIdent

  Identification of the parameter. Size: 16 bit.
- typedef struct TGM::Bitfields::SercosParamAttribute TGM::Bitfields::SercosParamAttribute

  Attribute for a SERCOS parameter that is callable via SercosDatablock.

Enumerations

- enum TGM::HeaderType : BYTE { TGM::TypeCommand, TGM::TypeReaction }

  Values that represent Telegram header types.
- enum TGM::SercosParamVar : BYTE { TGM::SercosParamS, TGM::SercosParamP }

  Values that represent SERCOS Parameter variants.
- enum TGM::SercosDatablock : BYTE {
  TGM::Datablock_ChannelNotActive, TGM::Datablock_IdentNumber, TGM::Datablock_Name, TGM::↩
  Datablock_Attribute,
  TGM::Datablock_Unit, TGM::Datablock_Minval, TGM::Datablock_Maxval, TGM::Datablock_OperationData }

  Values that represent SERCOS Parameter Bytes block to be processed.
- enum TGM::SercosCommandrequest : BYTE { TGM::Commandrequest_NotSet = 0x0, TGM::↩
  Commandrequest_Cancel = 0x1, TGM::Commandrequest_Set = 0x3 }

  Values that represent SERCOS command requests value.
- enum TGM::SercosCommandstatus : BYTE {
  TGM::Commandstatus_NotSet = 0x0, TGM::Commandstatus_OK = 0x3, TGM::Commandstatus_Canceled =
  0x5, TGM::Commandstatus_Busy = 0x7,
  TGM::Commandstatus_Error = 0xF }

  Values that represent SERCOS command status.
- enum TGM::SercosTxProgress : BYTE { TGM::TxProgress_InProgress, TGM::TxProgress_Final }

  Values that represent information in the SIS Telegram's Control Byte about the type of the Command Telegram or Reception Telegram.
- enum TGM::SercosDatalen : UINT32 {
  TGM::Datalen_Res1 = 0b000, TGM::Datalen_2ByteParam = 0b001, TGM::Datalen_4ByteParam = 0b010,
  TGM::Datalen_8ByteParam = 0b011,
  TGM::Datalen_1ByteList = 0b100, TGM::Datalen_2ByteList = 0b101, TGM::Datalen_4ByteList = 0b110, T↩
  GM::Datalen_8ByteList = 0b111 }

  Values that represent the information stored in a Parameter attributes (can be retrieved by attribute datablock).

6.10.1 Detailed Description

Contains enums, structs and unions to make Telegram creation, transmission and reception as flexible as possible.

For example, by defining different Telegram structs (such as Telegram Command header , Telegram Reception payload, etc) and using unions for each of these types, Telegrams can be easily created, debugged and later provided to a transmission caller in raw byte format.

Definition in file Telegrams_Bitfields.h.

6.11 Telegrams_Bitfields.h

```

#ifndef _TELEGRAMS_BITFIELDS_H_
#define _TELEGRAMS_BITFIELDS_H_

#include <Windows.h>
#include <vector>


namespace TGM
{
    enum HeaderType : BYTE {
        TypeCommand,
        TypeReaction
    };

    enum SercosParamVar : BYTE {
        SercosParamS,
        SercosParamP
    };

    enum SercosDatablock : BYTE {
        Datablock_ChannelNotActive,
        Datablock_IdentNumber,
        Datablock_Name,
        Datablock_Attribute,
        Datablock_Unit,
        Datablock_Minval,
        Datablock_Maxval,
        Datablock_OperationData
    };

    enum SercosCommandrequest : BYTE {
        Commandrequest_NotSet  = 0x0,
        Commandrequest_Cancel  = 0x1,
        Commandrequest_Set        = 0x3
    };

    enum SercosCommandstatus : BYTE {
        Commandstatus_NotSet    = 0x0,
        Commandstatus_OK        = 0x3,
        Commandstatus_Canceled    = 0x5,
        Commandstatus_Busy        = 0x7,
        Commandstatus_Error      = 0xF
    };

    enum SercosTxProgress : BYTE {
        TxProgress_InProgress,
        TxProgress_Final
    };

    enum SercosDatalen : UINT32 {
        Datalen_Res1        = 0b000,
        Datalen_2ByteParam    = 0b001,
        Datalen_4ByteParam    = 0b010,
        Datalen_8ByteParam    = 0b011,
        Datalen_1ByteList  = 0b100,
        Datalen_2ByteList  = 0b101,
        Datalen_4ByteList  = 0b110,
        Datalen_8ByteList  = 0b111,
    };


    namespace Bitfields
    {
        typedef struct HeaderControl
        {
```

```
      union
      {
        struct Bits
        {
          BYTE NumSubAddresses : 3;

          BYTE NumRunningTgm : 1;

          HeaderType Type : 1;

          BYTE StatusReactionTgm : 3;

          Bits(HeaderType type = TypeCommand) :
            NumSubAddresses(0),
            NumRunningTgm(0),
            Type(type),
            StatusReactionTgm(0)
          {}
        } Bits;

        BYTE Value;
      };

      HeaderControl(HeaderType type = TypeCommand) : Bits(TypeCommand) {}
    } HeaderControl;


    typedef struct SercosParamControl
    {
      union
      {
        struct Bits
        {
          BYTE res1 : 1;
          BYTE res2 : 1;

          SercosTxProgress TxProgress : 1;

          SercosDatablock Datablock : 3;

          BYTE res6 : 1;
          BYTE res7 : 1;

          Bits(SercosDatablock datablock = Datablock_OperationData) :
            res1(0), res2(0), TxProgress(TxProgress_Final), Datablock(datablock), res6(0), res7(0)
          {}
        } Bits;

        BYTE Value;
      };

      SercosParamControl(SercosDatablock datablock = Datablock_OperationData) : Bits(datablock) {}

      SercosParamControl(BYTE value) : Value(value) {}
    } SercosParamControl;


    typedef struct SercosParamIdent
    {
      union
      {
        struct Bits
        {
          USHORT ParamNumber : 12;

          USHORT ParamSet : 3;

          USHORT ParamVariant : 1;

          Bits(SercosParamVar param_variant = TGM::SercosParamS, USHORT param_num = 0) :
            ParamNumber(param_num),
            ParamSet(0),
            ParamVariant(param_variant)
          {}
        } Bits;

        USHORT Value;
      };

      SercosParamIdent(SercosParamVar param_variant = TGM::SercosParamS, USHORT param_num = 0) :
        Bits(param_variant, param_num)
```

```
            {}
        } SercosParamIdent;


        typedef struct SercosParamAttribute
        {
            union
            {
                struct Bits
                {
                    UINT32 ConversionFactor : 16;

                    SercosDatalen DataLen : 3;

                    UINT32 DataFunction : 1;

                    UINT32 DataDisplay : 3;

                    UINT32 res5 : 1;

                    UINT32 ScaleFactor : 4;

                    UINT32 is_writeonly_phase2 : 1;

                    UINT32 is_writeonly_phase3 : 1;

                    UINT32 is_writeonly_phase4 : 1;

                    UINT32 res10 : 1;

                    Bits() :
                        ConversionFactor(0),
                        DataLen(Datalen_2ByteParam),
                        DataFunction(0),
                        DataDisplay(0),
                        res5(0),
                        ScaleFactor(0),
                        is_writeonly_phase2(0),
                        is_writeonly_phase3(0),
                        is_writeonly_phase4(0),
                        res10(0)
                    {}
                } Bits;

                UINT32 Value;
            };

            SercosParamAttribute(UINT32 _value = 0) : Value(_value) {}
        } SercosParamAttribute;
    }
}


#endif // !_TELEGRAMS_BITFIELDS_H_
```

6.12 Wrapper.cpp File Reference

Implementation of API functions that are exported to the API DLL.

Functions

- SISProtocol ∗ init ()

    Creates API reference.
- int32_t open (SISProtocol ∗ID_ref, const wchar_t ∗ID_comport, uint32_t ID_combaudrate, ErrHandle ID_err)

    Opens the communication port to the Indradrive device.
- int32_t close (SISProtocol ∗ID_ref, ErrHandle ID_err)

    Closes the communication port at the Indradrive device.
- int32_t sequencer_activate (SISProtocol ∗ID_ref, ErrHandle ID_err)

    Activates the drive mode "Sequencer".
- int32_t sequencer_init (SISProtocol ∗ID_ref, double_t ID_max_accel, double_t ID_max_jerk, ErrHandle ID←_err)

Initializes limits and sets the right scaling/unit factors for operation of "Sequencer" drive mode.
- int32_t sequencer_write (SISProtocol ∗ID_ref, double_t ID_speeds[ ], double_t ID_accels[ ], double_t ID_↩ jerks[ ], uint32_t ID_delays[ ], const uint16_t ID_set_length, ErrHandle ID_err)

  Writes the whole run sequence into the device.
- int32_t sequencer_softtrigger (SISProtocol ∗ID_ref, ErrHandle ID_err)

  Software-Trigger to start operation of the "Sequencer" drive mode.
- int32_t sequencer_hardtrigger (SISProtocol ∗ID_ref, ErrHandle ID_err)

  Hardware-Trigger to start operation of the "Sequencer" drive mode.
- int32_t sequencer_getstatus (SISProtocol ∗ID_ref, uint16_t ∗ID_status, ErrHandle ID_err)

  Get the status of the "Sequencer" drive mode.
- int32_t speedcontrol_activate (SISProtocol ∗ID_ref, ErrHandle ID_err)

  Activates the drive mode "Speed Control".
- int32_t speedcontrol_init (SISProtocol ∗ID_ref, double_t ID_max_accel, double_t ID_max_jerk, ErrHandle ID_err)

  Initializes limits and sets the right scaling/unit factors for operation of "Speed Control" drive mode.
- int32_t speedcontrol_write (SISProtocol ∗ID_ref, double_t ID_speed, double_t ID_accel, ErrHandle ID_err)

  Writes the current kinematic (speed and acceleration) into the device.
- int32_t set_stdenvironment (SISProtocol ∗ID_ref, ErrHandle ID_err)

  Sets the proper unit and language environment.
- int32_t get_drivemode (SISProtocol ∗ID_ref, uint32_t ∗ID_drvmode, ErrHandle ID_err)

  Retrieve information about the drive mode: Speed Control or Sequencer.
- int32_t get_opstate (SISProtocol ∗ID_ref, uint8_t ∗ID_opstate, ErrHandle ID_err)

  Retrieve information about the operation states: bb, Ab, or AF.
- int32_t get_speed (SISProtocol ∗ID_ref, double_t ∗ID_speed, ErrHandle ID_err)

  Gets the actual rotation speed.
- int32_t get_diagnostic_msg (SISProtocol ∗ID_ref, char ∗ID_diagnostic_msg, ErrHandle ID_err)

  Gets diagnostic message string of the current Indradrive status.
- int32_t get_diagnostic_num (SISProtocol ∗ID_ref, uint32_t ∗ID_diagnostic_num, ErrHandle ID_err)

  Gets diagnostic number of the current Indradrive status.
- int32_t clear_error (SISProtocol ∗ID_ref, ErrHandle ID_err)

  Clears a latched error in the Indradrive device.
- void change_opmode (SISProtocol ∗ID_ref, const uint64_t opmode)
- SPEEDUNITS get_units (SISProtocol ∗ID_ref)
- void change_units (SISProtocol ∗ID_ref)
- void change_language (SISProtocol ∗ID_ref, const uint8_t lang_code)

6.12.1 Detailed Description

Implementation of API functions that are exported to the API DLL.

Definition in file Wrapper.cpp.

6.12.2 Function Documentation

6.12.2.1 init()

```
SISProtocol* init ( )
```

Creates API reference.

The API references is a fundamental prerequisite.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int init();
```

.
How to call with Python:

```
indraref = indralib.init()
```

.

Returns

API reference. Pointer can be casted and treated as UINT32 (see examples).

Definition at line 7 of file Wrapper.cpp.

6.12.2.2 open()

```
int32_t open (
            SISProtocol * ID_ref,
            const wchar_t * ID_comport = L"COM1",
            uint32_t ID_combaudrate = 19200,
            ErrHandle ID_err = ErrHandle() )
```

Opens the communication port to the Indradrive device.

Attention

Baudrate selection is not support. Default of 19200 Bits/s is used.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int open(int ID_ref, Byte[] ID_comport, UInt32 ID_combaudrate, ref
      ErrHandle ID_err);
```

.
How to call with Python:

```
result = indralib.open(indraref, b"COM1", 19200, ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
in	*ID_comport*	(Optional) Communication port. Default: L"COM1".
in	*ID_combaudrate*	(Optional) Communication baudrate in [Bits/s]. Default: 19200 Bits/s.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 14 of file Wrapper.cpp.

6.12.2.3 close()

```
int32_t close (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Closes the communication port at the Indradrive device.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int close(int ID_ref, ref ErrHandle ID_err);
```

.
How to call with Python:

```
result = indralib.close(indraref, ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 38 of file Wrapper.cpp.

6.12.2.4 sequencer_activate()

```
int32_t sequencer_activate (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Activates the drive mode "Sequencer".

Attention

Reiterate calls of this functions will harm the Indradrive EEPROM (due to limited write cycles). Use get_↩ drivemode() to check if this function call is really needed.

Remarks

This function is exported to the Indradrive API DLL.
Calling sequencer_∗ functions without calling sequencer_activate() first means that the drive will not operate in this mode.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_activate(int ID_ref, ref
      ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 65 of file Wrapper.cpp.

6.12.2.5 sequencer_init()

```
int32_t sequencer_init (
            SISProtocol * ID_ref,
            double_t ID_max_accel = 10000,
            double_t ID_max_jerk = 1000,
            ErrHandle ID_err = ErrHandle() )
```

Initializes limits and sets the right scaling/unit factors for operation of "Sequencer" drive mode.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_init(int ID_ref, Double ID_max_accel, Double ID_max_jerk,
       ref ErrHandle ID_err);
```

.

Parameters

`in`	*ID_ref*	API reference. Pointer can be casted in from UINT32.
`in`	*ID_max_accel*	(Optional) Maximum allowed acceleration in [rad/s^2]. Default: 10000 rad/s^2.
`in`	*ID_max_jerk*	(Optional) Maximum allowed jerk in [rad/s^3]. Default: 1000 rad/s^3.
`out`	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 91 of file Wrapper.cpp.

6.12.2.6 sequencer_write()

```
int32_t sequencer_write (
            SISProtocol * ID_ref,
            double_t ID_speeds[ ],
            double_t ID_accels[ ],
            double_t ID_jerks[ ],
            uint32_t ID_delays[ ],
            const uint16_t ID_set_length,
            ErrHandle ID_err = ErrHandle() )
```

Writes the whole run sequence into the device.

The run sequence is defined by several kinematic parameters, such as speed, acceleration, or jerk. A proper calculation of the kinetics before writing is assumed.

Bug List length will not be extended automatically. In case of list length is set too short, programming of all values might fail. This may cause an improper operation of the "Sequencer" drive mode.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_write(int ID_ref, Double[] ID_speeds, Double[] ID_accels
      , Double[] ID_jerks, UInt32[] ID_delays, UInt16 ID_set_length, ref ErrHandle ID_err);
```

.

Parameters

`in`	*ID_ref*	API reference. Pointer can be casted in from UINT32.
`in`	*ID_speeds*	Sequencer speed list in [1/min]. Rotation directions are defined by the sign of each element: • Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.
`in`	*ID_accels*	Sequencer acceleration list in [rad/s^2].
`in`	*ID_jerks*	Sequencer jerk list in [rad/s^3].
`in`	*ID_delays*	Delay list representing delay between each kinematic step in [cs].
`in`	*ID_set_length*	Length of the sequence (=number of elements of ID_speeds, ID_accels, etc).

Returns

Error handle return code (ErrHandle()).

Definition at line 129 of file Wrapper.cpp.

6.12.2.7 sequencer_softtrigger()

```
int32_t sequencer_softtrigger (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Software-Trigger to start operation of the "Sequencer" drive mode.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_softtrigger(int ID_ref, ref
      ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 187 of file Wrapper.cpp.

6.12.2.8 sequencer_hardtrigger()

```
int32_t sequencer_hardtrigger (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Hardware-Trigger to start operation of the "Sequencer" drive mode.

By special PLC software (if configured), the hardware trigger is realized through a 24V rising edge input line.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_hardtrigger(int ID_ref, ref
      ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

READ CUSTOM DATA

Definition at line 253 of file Wrapper.cpp.

6.12.2.9 sequencer_getstatus()

```
int32_t sequencer_getstatus (
            SISProtocol * ID_ref,
            uint16_t * ID_status,
            ErrHandle ID_err = ErrHandle() )
```

Get the status of the "Sequencer" drive mode.

The information is derived from the PLC that reports the actual status within an internal register.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_getstatus(int ID_ref, ref UInt16 status, ref
      ErrHandle ID_err);
```

.
How to call with Python:

```
plcstatus = ctypes.c_uint16(0)
result = indralib.sequencer_getstatus(indraref, ctypes.byref(plcstatus), ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_status*	Pointer that provides the respective information: • Bit 0 - bSeqFinished, if true then end of the sequence has been reached. • Bit 1 - bCamTick, if true then cam is currently shifting to the next position. • Bit 2 - bDriveStopped, if true then drive has been stopped by PLC. • Bit 3 - bDriveStarted, if true then drive has been started by PLC. • Bit 4 - ERROR_T_PARAM, if true then error has been occurred while reading input parameters.
		• Bit 5 - RESULT_READ_OK, if true then reading input parameters have been processed correctly. • Bit 6 - RESULT_SEQUENCER_OK, if true then sequencer is/was running well.

Returns

Error handle return code (ErrHandle()).

Definition at line 294 of file Wrapper.cpp.

6.12.2.10 speedcontrol_activate()

```
int32_t speedcontrol_activate (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Activates the drive mode "Speed Control".

Attention

Reiterate calls of this functions will harm the Indradrive EEPROM (due to limited write cycles). Use get_↩
drivemode() to check if this function call is really needed.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int speedcontrol_activate(int ID_ref, ref
      ErrHandle ID_err);
```

.
How to call with Python:

```
result = indralib.speedcontrol_activate(indraref, ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 323 of file Wrapper.cpp.

6.12.2.11 speedcontrol_init()

```
int32_t speedcontrol_init (
            SISProtocol * ID_ref,
            double_t ID_max_accel = 10000,
            double_t ID_max_jerk = 1000,
            ErrHandle ID_err = ErrHandle() )
```

Initializes limits and sets the right scaling/unit factors for operation of "Speed Control" drive mode.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int speedcontrol_init(int ID_ref, Double ID_max_accel, Double
      ID_max_jerk, ref ErrHandle ID_err);
```

.
How to call with Python:

```
result = indralib.speedcontrol_init(indraref, ctypes.c_double(10000), ctypes.c_double(1000), ctypes.byref(
      indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_max_accel*	(Optional) Maximum allowed acceleration in [rad/s^2]. Default: 10000 rad/s^2.
out	*ID_max_jerk*	(Optional) Maximum allowed jerk in [rad/s^3]. Default: 1000 rad/s^3.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 349 of file Wrapper.cpp.

6.12.2.12 speedcontrol_write()

```
int32_t speedcontrol_write (
            SISProtocol * ID_ref,
            double_t ID_speed,
            double_t ID_accel,
            ErrHandle ID_err = ErrHandle() )
```

Writes the current kinematic (speed and acceleration) into the device.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int speedcontrol_write(int ID_ref, Double ID_speed, Double ID_accel
      , ref ErrHandle ID_err);
```

.
How to call with Python:

```
result = indralib.speedcontrol_write(indraref, ctypes.c_double(speed), ctypes.c_double(10), ctypes.byref(
      indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_speed*	Target speed in [1/min]. Sign represents the rotation direction: • Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.
out	*ID_accel*	Target acceleration in [rad/s^2].
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 381 of file Wrapper.cpp.

6.12.2.13 set_stdenvironment()

```
int32_t set_stdenvironment (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Sets the proper unit and language environment.

Proper unit and language environment is:

- for unit setup: Preferred scaling / Rotary scaling / Unit [rpm] / Velocity data scaling
- for language environment: English language.

Attention

Not setting the proper unit and language environment may cause unexpected behavior when programming and setting kinematics.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with Python:

```
result = indralib.set_stdenvironment(indraref, ctypes.byref(indra_error))
```

.

Parameters

`in,out`	*ID_ref*	API reference. Pointer can be casted in from UINT32.
	ID_err	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 415 of file Wrapper.cpp.

6.12.2.14 get_drivemode()

```
int32_t get_drivemode (
            SISProtocol * ID_ref,
            uint32_t * ID_drvmode,
            ErrHandle ID_err = ErrHandle() )
```

Retrieve information about the drive mode: Speed Control or Sequencer.

The drive mode feedback is provided by `ID_drvmode` parameter. The following table depicts the coding:

If	Then
`*ID_drvmode == 0`	Drive Mode not supported
`*ID_drvmode == 1`	"Sequencer" drive mode active
`*ID_drvmode == 2`	"Speed Control" drive mode active.

Remarks

This function is exported to the Indradrive API DLL.
The drive mode can be changed by speedcontrol_activate() or sequencer_activate().
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_drivemode(int ID_ref, ref UInt32 mode, ref
      ErrHandle ID_err);
```

.
How to call with Python:

```
drvmode = ctypes.c_uint32(0)
result = indralib.get_drivemode(indraref, ctypes.byref(drvmode), ctypes.byref(indra_error))
```

.

Parameters

`in`	*ID_ref*	API reference. Pointer can be casted in from UINT32.
`out`	*ID_drvmode*	Pointer that provides the respective information: • 0 - Drive Mode not supported, • 1 - "Sequencer" drive mode active, • 2 - "Speed Control" drive mode active.

Parameters

out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 441 of file Wrapper.cpp.

6.12.2.15 get_opstate()

```
int32_t get_opstate (
            SISProtocol * ID_ref,
            uint8_t * ID_opstate,
            ErrHandle ID_err = ErrHandle() )
```

Retrieve information about the operation states: bb, Ab, or AF.

The operation state feedback is provided by `ID_opstate` parameter. The following table depicts the coding:

If	Then
`(*ID_opstate & 0b11) == 0b00`	Control section / power section not ready for operation(e.g., drive error or phase 2)
`(*ID_opstate & 0b11) == 0b01`	Control section ready for operation "bb"
`(*ID_opstate & 0b11) == 0b10`	Control section and power section ready for op. "Ab"
`(*ID_opstate & 0b11) == 0b11`	Drive with torque "AF".
`((*ID_opstate & 0b100) >> 2) == 1`	Drive Halt is active and axis is in standstill
`((*ID_opstate & 0b1000) >> 3) == 1`	Drive error.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(&lt;path_to_DLL&gt;, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_opstate(int ID_ref, ref Byte state, ref
      ErrHandle ID_err);
```

.
How to call with Python:

```
opstate = ctypes.c_uint8(0)
result = indralib.get_opstate(indraref, ctypes.byref(opstate), ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.

Parameters

out	*ID_opstate*	Pointer that provides the respective information: • Bit 0-1: Operation state – `0b00`: Control section / power section not ready for operation(e.g., drive error or phase 2) – `0b01`: Control section ready for operation "bb" – `0b10`: Control section and power section ready for op. "Ab" – `0b11`: Drive with torque "AF". • Bit 2: Drive Halt acknowledgment – `0`: Drive not halted – `1`: Drive Halt is active and axis is in standstill • Bit 3: Drive error – `0`: No error – `1`: Drive error present.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 481 of file Wrapper.cpp.

6.12.2.16 get_speed()

```
int32_t get_speed (
            SISProtocol * ID_ref,
            double_t * ID_speed,
            ErrHandle ID_err = ErrHandle() )
```

Gets the actual rotation speed.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_speed(int ID_ref, ref Double speed, ref
      ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_speed*	Pointer that provides the speed information as double Value in [1/min]. Sign represents the rotation direction:
		• Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 511 of file Wrapper.cpp.

6.12.2.17 get_diagnostic_msg()

```
int32_t get_diagnostic_msg (
            SISProtocol * ID_ref,
            char * ID_diagnostic_msg,
            ErrHandle ID_err = ErrHandle() )
```

Gets diagnostic message string of the current Indradrive status.

Attention

The API presumes a properly allocated char array for `ID_diagnostic_msg` parameter.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_diagnostic_msg(int ID_ref, Byte[] ID_diagnostic_msg, ref
      ErrHandle ID_err);
```

.
How to call with Python:

```
diagmsg = ctypes.create_string_buffer(256)
result = indralib.get_diagnostic_msg(indraref, diagmsg, ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_diagnostic_msg*	Pointer that provides the diagnostic message string.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 540 of file Wrapper.cpp.

6.12.2.18 get_diagnostic_num()

```
int32_t get_diagnostic_num (
            SISProtocol * ID_ref,
            uint32_t * ID_diagnostic_num,
            ErrHandle ID_err = ErrHandle() )
```

Gets diagnostic number of the current Indradrive status.

Remarks

This function is exported to the Indradrive API DLL.
The coding of the diagnostic number is described in the document "Rexroth IndraDrive Firmware for Drive Controller MPH-04, MPB-04, MPD-04 / Function Description (Chapter 10.3). For example, operation state "AF" is described as 0xA0101.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_diagnostic_num(int ID_ref, ref UInt32 ID_diagnostic_num,
      ref ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_diagnostic_num*	Pointer that provides the diagnostic number.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 569 of file Wrapper.cpp.

6.12.2.19 clear_error()

```
int32_t clear_error (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Clears a latched error in the Indradrive device.

In case of error that has been occurred on the Indradrive, the error information is latched until cleared with this call.

Remarks

This function is exported to the Indradrive API DLL.
Use get_diagnostic_message() and/or get_diagnostic_num() for retrieving the error information.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int clear_error(int ID_ref, ref ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 598 of file Wrapper.cpp.

6.12.2.20 change_opmode()

```
void change_opmode (
            SISProtocol * ID_ref,
            const uint64_t opmode )
```

Definition at line 624 of file Wrapper.cpp.

6.12.2.21 get_units()

```
SPEEDUNITS get_units (
            SISProtocol * ID_ref )  [inline]
```

Definition at line 646 of file Wrapper.cpp.

6.12.2.22 change_units()

```
void change_units (
            SISProtocol * ID_ref )
```

Definition at line 656 of file Wrapper.cpp.

6.12.2.23 change_language()

```
void change_language (
            SISProtocol * ID_ref,
            const uint8_t lang_code )  [inline]
```

Definition at line 668 of file Wrapper.cpp.

6.13 Wrapper.cpp

```

#include "Wrapper.h"


DLLEXPORT SISProtocol * DLLCALLCONV init()
{
    SISProtocol * protocol = new SISProtocol();
    return protocol;
}


DLLEXPORT int32_t DLLCALLCONV open(SISProtocol* ID_ref, const wchar_t*
ID_comport, uint32_t ID_combaudrate, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        ID_ref->open(ID_comport);
        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_OpenByCOM);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_OpenByCOM);
    }
}


DLLEXPORT int32_t DLLCALLCONV close(SISProtocol* ID_ref,
ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        ID_ref->close();

        delete ID_ref;
        ID_ref = NULL;
        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_Close);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_Close);
    }
}


DLLEXPORT int32_t DLLCALLCONV sequencer_activate(
SISProtocol * ID_ref, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        // Change mode
        change_opmode(ID_ref, DRIVEMODE_SEQUENCER);

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqInit);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_SeqInit);
    }
}


DLLEXPORT int32_t DLLCALLCONV sequencer_init(
SISProtocol * ID_ref, double_t ID_max_accel, double_t ID_max_jerk,
ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        // Set required units (preferred scaling, rotary scaling, [rpm])
        change_units(ID_ref);

        // Max Acceleration (S-0-0138)
        ID_ref->write_parameter(TGM::SercosParamS, 138, ID_max_accel);

        // Max Jerk (S-0-0349)
        ID_ref->write_parameter(TGM::SercosParamS, 349, ID_max_jerk);

        // SPS Global Register G1 (P-0-1371) - Reset Read Trigger
        ID_ref->write_parameter(TGM::SercosParamP, 1371,
static_cast<uint32_t>(0));

        // SPS Global Register G2 (P-0-1372) - Reset Sequencer Trigger
        ID_ref->write_parameter(TGM::SercosParamP, 1372,
static_cast<uint32_t>(0));

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqInit);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_SeqInit);
    }
}


DLLEXPORT int32_t DLLCALLCONV sequencer_write(
SISProtocol * ID_ref, double_t ID_speeds[], double_t ID_accels[], double_t ID_jerks[], uint32_t
ID_delays[], const uint16_t ID_set_length, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        ID_ref->write_listelm(TGM::SercosParamP, 4019, 1,
static_cast<uint32_t>(0b10000100));

        for (uint16_t i = 0; i < ID_set_length; i++)
        {
            // Speed in min^-1 (P-0-4007)
            ID_ref->write_listelm(TGM::SercosParamP, 4007, i + 1, abs(
ID_speeds[i]));

            // Acceleration in rad/s^2 (P-0-4008)
            ID_ref->write_listelm(TGM::SercosParamP, 4008, i + 1, ID_accels[i
]);

            // Deceleration in rad/s^2 (P-0-4063)
            ID_ref->write_listelm(TGM::SercosParamP, 4063, i + 1, ID_accels[i
]);

            // Jerk in rad/s^3 (P-0-4009)
            ID_ref->write_listelm(TGM::SercosParamP, 4009, i + 1, ID_jerks[i]
);

```

```
            // Mode (P-0-4019)
            ID_ref->write_listelm(TGM::SercosParamP, 4019, i + 2,
static_cast<uint32_t>(stde::sgn<double_t>(ID_speeds[i]) == 1 ? 0b10000100 : 0b10001000));

            // Pos (P-0-4006)
            ID_ref->write_listelm(TGM::SercosParamP, 4006, i + 1,
static_cast<uint64_t>(0));

            // Wait (P-0-4018)
            ID_ref->write_listelm(TGM::SercosParamP, 4018, i + 1,
static_cast<uint64_t>(0));

            // Delay (P-0-4063)
            ID_ref->write_listelm(TGM::SercosParamP, 4063, i + 1,
static_cast<uint64_t>(0));

            // Timers in cs (P-0-1389)
            ID_ref->write_listelm(TGM::SercosParamP, 1389, i + 1, ID_delays[i
]);
        }

        // Time triggers for cam (P-0-1370)
        ID_ref->write_parameter(TGM::SercosParamP, 1370,
static_cast<uint32_t>(ID_set_length));

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
    }
}


DLLEXPORT int32_t DLLCALLCONV sequencer_softtrigger(
SISProtocol * ID_ref, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        uint32_t qb0stat;
        int iterations;

        // FEED DATA:

        // SPS Global Register G1 (P-0-1371) - Reset Read Trigger
        ID_ref->write_parameter(TGM::SercosParamP, 1371,
static_cast<uint64_t>(0));

        // SPS Global Register G2 (P-0-1372) - Reset Sequencer Trigger
        ID_ref->write_parameter(TGM::SercosParamP, 1372,
static_cast<uint64_t>(0));

        // READ CUSTOM DATA ...

        iterations = 0;
        do
        {
            // Check status (P-0-1410)
            ID_ref->read_parameter(TGM::SercosParamP, 1410, qb0stat);

            if (iterations > 300) return set_error(ID_err, "RESULT_READ_OK was not set. Input
 parameters cannot be accepted.", Err_Block_SeqWrite);
        } while (qb0stat & 0b100000 >> 5);

        // SPS Global Register G1 (P-0-1371) - Set Read Trigger
        ID_ref->write_parameter(TGM::SercosParamP, 1371,
static_cast<uint64_t>(1));


        // TRIGGER:

        // SPS Global Register G2 (P-0-1372) - Reset Sequencer Trigger
        ID_ref->write_parameter(TGM::SercosParamP, 1372,
static_cast<uint64_t>(0));

```

```
        // SPS Global Register G2 (P-0-1372) - Set Sequencer Trigger
        ID_ref->write_parameter(TGM::SercosParamP, 1372, static_cast<uint64_t>(1));


        iterations = 0;
        do
        {
            // Check status (P-0-1410)
            ID_ref->read_parameter(TGM::SercosParamP, 1410, qb0stat);

            if (iterations > 300) return set_error(ID_err, "bDriveStarted was not set. Input parameters cannot be accepted.", Err_Block_SeqWrite);
        } while (qb0stat & 0b1000 >> 3);

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_SeqWrite);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_SeqWrite);
    }
}

DLLEXPORT int32_t DLLCALLCONV sequencer_hardtrigger(SISProtocol * ID_ref, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        uint32_t qb0stat;

        // FEED DATA:

        // SPS Global Register G1 (P-0-1371) - Reset Read Trigger
        ID_ref->write_parameter(TGM::SercosParamP, 1371, static_cast<uint64_t>(0));

        // SPS Global Register G2 (P-0-1372) - Reset Sequencer Trigger
        ID_ref->write_parameter(TGM::SercosParamP, 1372, static_cast<uint64_t>(0));


        // SPS Global Register G1 (P-0-1371) - Set Read Trigger
        ID_ref->write_parameter(TGM::SercosParamP, 1371, static_cast<uint64_t>(1));

        // Check status (P-0-1410)
        ID_ref->read_parameter(TGM::SercosParamP, 1410, qb0stat); // TODO: Check RESULT_READ_OK bit (0b100000)

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_SeqWrite);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_SeqWrite);
    }
}


DLLEXPORT int32_t DLLCALLCONV sequencer_getstatus(SISProtocol * ID_ref, uint16_t * ID_status, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        uint32_t plc_status;
        // PLC register / Status value (P-0-1410)
        ID_ref->read_parameter(TGM::SercosParamP, 1410, plc_status);

        *ID_status = static_cast<uint16_t>(plc_status & 0xFFFF);

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
    }
}


DLLEXPORT int32_t DLLCALLCONV speedcontrol_activate(
SISProtocol * ID_ref, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        // Change mode
        change_opmode(ID_ref, DRIVEMODE_SPEEDCONTROL);

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_VelCInit);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_VelCInit);
    }
}


DLLEXPORT int32_t DLLCALLCONV speedcontrol_init(
SISProtocol * ID_ref, double_t ID_max_accel, double_t ID_max_jerk,
ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        // Set required units (preferred scaling, rotary scaling, [rpm])
        change_units(ID_ref);

        // Max Acceleration (S-0-0138)
        ID_ref->write_parameter(TGM::SercosParamS, 138, ID_max_accel);

        // Max Jerk (S-0-0349)
        ID_ref->write_parameter(TGM::SercosParamS, 349, ID_max_jerk);

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_VelCInit);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_VelCInit);
    }
}


DLLEXPORT int32_t DLLCALLCONV speedcontrol_write(
SISProtocol * ID_ref, double_t ID_speed, double_t ID_accel, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        // Rotation direction - Positive ID_speed: Clockwise rotation, Negative ID_speed: Counter-clockwise rotation
        uint32_t rotmode = static_cast<uint32_t>((stde::sgn<double_t>(ID_speed) == 1 ? 0 : 1) << 10);
        // Control Mode (P-0-1200)
        ID_ref->write_parameter(TGM::SercosParamP, 1200, rotmode);

        // Acceleration in rad/s^2 (P-0-1203)
        ID_ref->write_parameter(TGM::SercosParamP, 1203, ID_accel);

        // Speed in rpm (S-0-0036)
        ID_ref->write_parameter(TGM::SercosParamS, 36, abs(ID_speed));

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_VelCWrite);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_VelCWrite);
    }
}


DLLEXPORT int32_t DLLCALLCONV set_stdenvironment(
    SISProtocol * ID_ref, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        change_units(ID_ref);
        change_language(ID_ref);

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_GetStatus);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_GetStatus);
    }
}


DLLEXPORT int32_t DLLCALLCONV get_drivemode(
    SISProtocol * ID_ref, uint32_t * ID_drvmode, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        uint64_t curdrvmode;
        // Primary Operation Mode (S-0-0032)
        ID_ref->read_parameter(TGM::SercosParamS, 32, curdrvmode);

        switch (curdrvmode)
        {
        case DRIVEMODE_SEQUENCER: // Drive Mode: Sequencer
            *ID_drvmode = 1;
            break;
        case DRIVEMODE_SPEEDCONTROL: // Drive Mode: Speed Control
```

```
            *ID_drvmode = 2;
            break;
        default: // Drive Mode not supported
            *ID_drvmode = 0;
            break;
        }

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_GetStatus);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_GetStatus);
    }
}


DLLEXPORT int32_t DLLCALLCONV get_opstate(SISProtocol * ID_ref, uint8_t * ID_opstate, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        uint64_t curopstate;
        // Device control: Status word (P-0-0115)
        ID_ref->read_parameter(TGM::SercosParamP, 115, curopstate);

        OPSTATE opstate(static_cast<uint16_t>(curopstate));
        *ID_opstate = opstate.Value;

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_GetStatus);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_GetStatus);
    }
}


DLLEXPORT int32_t DLLCALLCONV get_speed(SISProtocol * ID_ref, double_t * ID_speed, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        double_t speed;
        // Velocity feedback Value (S-0-0040)
        ID_ref->read_parameter(TGM::SercosParamS, 40, speed);

        *ID_speed = speed;

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_GetStatus);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()), Err_Block_GetStatus);
    }
}


```

```
DLLEXPORT int32_t DLLCALLCONV get_diagnostic_msg(
SISProtocol * ID_ref, char * ID_diagnostic_msg, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        char msg[TGM_SIZEMAX_PAYLOAD];
        // Diagnostic message (S-0-0095)
        ID_ref->read_parameter(TGM::SercosParamS, 95, msg);

        strncpy(ID_diagnostic_msg, msg+4, TGM_SIZEMAX_PAYLOAD-4);

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
    }
}


DLLEXPORT int32_t DLLCALLCONV get_diagnostic_num(
SISProtocol * ID_ref, uint32_t * ID_diagnostic_num, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        UINT32 num;
        // Diagnostic number (S-0-0390)
        ID_ref->read_parameter(TGM::SercosParamS, 390, num);

        *ID_diagnostic_num = num;

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
    }
}


DLLEXPORT int32_t DLLCALLCONV clear_error(
SISProtocol * ID_ref, ErrHandle ID_err)
{
    if (!dynamic_cast<SISProtocol*>(ID_ref))
        // Return error for wrong reference
        return set_error(
            ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
            Err_Invalid_Pointer);

    try
    {
        // Clear error (S-0-0099) // Command C0500
        ID_ref->execute_command(TGM::SercosParamS, 99);

        return Err_NoError;
    }
    catch (SISProtocol::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
    }
    catch (CSerial::ExceptionGeneric &ex)
    {
        return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
    }
}


void change_opmode(SISProtocol * ID_ref, const uint64_t opmode)
{
    uint64_t curopmode;
    // Primary Operation Mode (S-0-0032)
    ID_ref->read_parameter(TGM::SercosParamS, 32, curopmode);

    // Operation change will trigger flash operations that may cause limited life time
    // Thus, operation change should be mainly triggered if required only
    if (curopmode != opmode)
    {
        // Enter parameterization level 1 (S-0-0420) // Command C0400
        ID_ref->execute_command(TGM::SercosParamS, 420);

        // Primary Operation Mode (S-0-0032)
        ID_ref->write_parameter(TGM::SercosParamS, 32, opmode);

        // Leave parameterization level 1 (S-0-0422) // Command C0200
        ID_ref->execute_command(TGM::SercosParamS, 422);
    }
}


inline SPEEDUNITS get_units(SISProtocol * ID_ref)
{
    uint64_t curunits;
    // Scaling of speed units (S-0-0044)
    ID_ref->read_parameter(TGM::SercosParamS, 44, curunits);

    return SPEEDUNITS(static_cast<uint16_t>(curunits));
}


void change_units(SISProtocol * ID_ref)
{
    SPEEDUNITS units = get_units(ID_ref);
    if (units.Bits.type_of_scaling == 0b010 && !units.Bits.automode && !units.
Bits.scale_units && !units.Bits.time_units && !units.Bits.data_rel) return;

    // Set required units (preferred scaling, rotary scaling, [rpm])
    uint64_t scalingtype = 0b0000000000000010;
    // Velocity data scaling Type (S-0-0044)
    ID_ref->write_parameter(TGM::SercosParamS, 44, scalingtype);
}


inline void change_language(SISProtocol * ID_ref, const uint8_t lang_code)
{
    // Language selection (S-0-0265):
    // * 0: German
    // * 1: English
    // * 2: French
    // * 3: Spanish
    // * 4: Italian
    ID_ref->write_parameter(TGM::SercosParamS, 265, (UINT32)lang_code);
}
```

6.14 Wrapper.h File Reference

Definition of API functions that are exported to the API DLL.

Classes

- struct OPSTATE

  Structure is used for loading the payload of the Reception Telegram from the Indradrive SERCOS parameter P-0-0115.
- struct SPEEDUNITS

  Structure is used for loading the payload of the Reception Telegram from the Indradrive SERCOS parameter S-0-0044.

Macros

- #define DLLEXPORT __declspec(dllexport)

    Doxygen's mainpage documentation.
- #define DLLCALLCONV __cdecl
- #define DRIVEMODE_SEQUENCER 0b111011

    Positioning mode lagless, encoder 1.
- #define DRIVEMODE_SPEEDCONTROL 0b10

    Velocity Control.

Typedefs

- typedef struct OPSTATE OPSTATE

    Structure is used for loading the payload of the Reception Telegram from the Indradrive SERCOS parameter P-0-0115.
- typedef struct SPEEDUNITS SPEEDUNITS

    Structure is used for loading the payload of the Reception Telegram from the Indradrive SERCOS parameter S-0-0044.
- typedef struct SISProtocol SISProtocol

    Faking the actual SISProtocol class to a struct so that the C compiler can handle compilation of this file.

Functions

- SISProtocol ∗ init ()

    Creates API reference.
- int32_t open (SISProtocol ∗ID_ref, const wchar_t ∗ID_comport=L"COM1", uint32_t ID_combaudrate=19200, ErrHandle ID_err=ErrHandle())

    Opens the communication port to the Indradrive device.
- int32_t close (SISProtocol ∗ID_ref, ErrHandle ID_err=ErrHandle())

    Closes the communication port at the Indradrive device.
- int32_t sequencer_activate (SISProtocol ∗ID_ref, ErrHandle ID_err=ErrHandle())

    Activates the drive mode "Sequencer".
- int32_t sequencer_init (SISProtocol ∗ID_ref, double_t ID_max_accel=10000, double_t ID_max_jerk=1000, ErrHandle ID_err=ErrHandle())

    Initializes limits and sets the right scaling/unit factors for operation of "Sequencer" drive mode.
- int32_t sequencer_write (SISProtocol ∗ID_ref, double_t ID_speeds[ ], double_t ID_accels[ ], double_t ID_↩ jerks[ ], uint32_t ID_delays[ ], const uint16_t ID_set_length, ErrHandle ID_err=ErrHandle())

    Writes the whole run sequence into the device.
- int32_t sequencer_softtrigger (SISProtocol ∗ID_ref, ErrHandle ID_err=ErrHandle())

    Software-Trigger to start operation of the "Sequencer" drive mode.
- int32_t sequencer_hardtrigger (SISProtocol ∗ID_ref, ErrHandle ID_err=ErrHandle())

    Hardware-Trigger to start operation of the "Sequencer" drive mode.
- int32_t sequencer_getstatus (SISProtocol ∗ID_ref, uint16_t ∗ID_status, ErrHandle ID_err=ErrHandle())

    Get the status of the "Sequencer" drive mode.
- int32_t speedcontrol_activate (SISProtocol ∗ID_ref, ErrHandle ID_err=ErrHandle())

    Activates the drive mode "Speed Control".
- int32_t speedcontrol_init (SISProtocol ∗ID_ref, double_t ID_max_accel=10000, double_t ID_max_jerk=1000, ErrHandle ID_err=ErrHandle())

    Initializes limits and sets the right scaling/unit factors for operation of "Speed Control" drive mode.
- int32_t speedcontrol_write (SISProtocol ∗ID_ref, double_t ID_speed, double_t ID_accel, ErrHandle ID_↩ err=ErrHandle())

Writes the current kinematic (speed and acceleration) into the device.

- int32_t set_stdenvironment (SISProtocol ∗ID_ref, ErrHandle ID_err=ErrHandle())

  Sets the proper unit and language environment.
- int32_t get_drivemode (SISProtocol ∗ID_ref, uint32_t ∗ID_drvmode, ErrHandle ID_err=ErrHandle())

  Retrieve information about the drive mode: Speed Control or Sequencer.
- int32_t get_opstate (SISProtocol ∗ID_ref, uint8_t ∗ID_opstate, ErrHandle ID_err=ErrHandle())

  Retrieve information about the operation states: bb, Ab, or AF.
- int32_t get_speed (SISProtocol ∗ID_ref, double_t ∗ID_speed, ErrHandle ID_err=ErrHandle())

  Gets the actual rotation speed.
- int32_t get_diagnostic_msg (SISProtocol ∗ID_ref, char ∗ID_diagnostic_msg, ErrHandle ID_err=ErrHandle())

  Gets diagnostic message string of the current Indradrive status.
- int32_t get_diagnostic_num (SISProtocol ∗ID_ref, uint32_t ∗ID_diagnostic_num, ErrHandle ID_err=Err↩ Handle())

  Gets diagnostic number of the current Indradrive status.
- int32_t clear_error (SISProtocol ∗ID_ref, ErrHandle ID_err=ErrHandle())

  Clears a latched error in the Indradrive device.

6.14.1 Detailed Description

Definition of API functions that are exported to the API DLL.

Definition in file Wrapper.h.

6.14.2 Macro Definition Documentation

6.14.2.1 DLLEXPORT

```
#define DLLEXPORT __declspec(dllexport)
```

Doxygen's mainpage documentation.

Macro to indicate that a static function shall be exported for the target DLL

Definition at line 19 of file Wrapper.h.

6.14.2.2 DLLCALLCONV

```
#define DLLCALLCONV __cdecl
```

Definition at line 20 of file Wrapper.h.

6.14.2.3 DRIVEMODE_SEQUENCER

```
#define DRIVEMODE_SEQUENCER 0b111011
```

Positioning mode lagless, encoder 1.

Definition at line 43 of file Wrapper.h.

6.14.2.4 DRIVEMODE_SPEEDCONTROL

```
#define DRIVEMODE_SPEEDCONTROL 0b10
```

Velocity Control.

Definition at line 45 of file Wrapper.h.

6.14.3 Typedef Documentation

6.14.3.1 OPSTATE

```
typedef struct OPSTATE OPSTATE
```

Structure is used for loading the payload of the Reception Telegram from the Indradrive SERCOS parameter P-0-0115.

The structure is designed to be loaded with an integer, but automatically structured into its components. Thus, it is possible extract the exact information that are requested (e.g. Operate State of Indradrive M device).

The following code demonstrates a possible usage of this struct:

```
uint64_t curopstate;
SISProtocol_ref->read_parameter(TGM::SercosParamP, 115, curopstate);

OPSTATE opstate(static_cast<uint16_t>(curopstate));
int foo = opstate.Value;
```

.

See also

SISProtocol
SISProtocol::read_parameter

6.14.3.2 SPEEDUNITS

```
typedef struct SPEEDUNITS SPEEDUNITS
```

Structure is used for loading the payload of the Reception Telegram from the Indradrive SERCOS parameter S-0-0044.

The structure is designed to be loaded with an integer, but automatically structured into its components. Thus, it is possible extract the exact information that are requested (e.g. Operate State of Indradrive M device).

6.14.3.3 SISProtocol

```
typedef struct SISProtocol SISProtocol
```

Faking the actual SISProtocol class to a struct so that the C compiler can handle compilation of this file.

The SISProtocol files itself should be automically compiled using the C++ compilation process. This is automatically handled using extern "C".

Definition at line 183 of file Wrapper.h.

6.14.4 Function Documentation

6.14.4.1 init()

```
SISProtocol* init ( )
```

Creates API reference.

The API references is a fundamental prerequisite.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int init();
```

.
How to call with Python:

```
indraref = indralib.init()
```

.

Returns

API reference. Pointer can be casted and treated as UINT32 (see examples).

Definition at line 7 of file Wrapper.cpp.

6.14.4.2 open()

```
int32_t open (
            SISProtocol * ID_ref,
            const wchar_t * ID_comport = L"COM1",
            uint32_t ID_combaudrate = 19200,
            ErrHandle ID_err = ErrHandle() )
```

Opens the communication port to the Indradrive device.

Attention

Baudrate selection is not support. Default of 19200 Bits/s is used.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int open(int ID_ref, Byte[] ID_comport, UInt32 ID_combaudrate, ref
     ErrHandle ID_err);
```

.
How to call with Python:

```
result = indralib.open(indraref, b"COM1", 19200, ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
in	*ID_comport*	(Optional) Communication port. Default: L"COM1".
in	*ID_combaudrate*	(Optional) Communication baudrate in [Bits/s]. Default: 19200 Bits/s.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 14 of file Wrapper.cpp.

6.14.4.3 close()

```
int32_t close (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Closes the communication port at the Indradrive device.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int close(int ID_ref, ref ErrHandle ID_err);
```

.
How to call with Python:

```
result = indralib.close(indraref, ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 38 of file Wrapper.cpp.

6.14.4.4 sequencer_activate()

```
int32_t sequencer_activate (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Activates the drive mode "Sequencer".

Attention

Reiterate calls of this functions will harm the Indradrive EEPROM (due to limited write cycles). Use get_↩
drivemode() to check if this function call is really needed.

Remarks

This function is exported to the Indradrive API DLL.
Calling sequencer_∗ functions without calling sequencer_activate() first means that the drive will not operate in this mode.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_activate(int ID_ref, ref
      ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 65 of file Wrapper.cpp.

6.14.4.5 sequencer_init()

```
int32_t sequencer_init (
            SISProtocol * ID_ref,
            double_t ID_max_accel = 10000,
            double_t ID_max_jerk = 1000,
            ErrHandle ID_err = ErrHandle() )
```

Initializes limits and sets the right scaling/unit factors for operation of "Sequencer" drive mode.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_init(int ID_ref, Double ID_max_accel, Double ID_max_jerk,
       ref ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
in	*ID_max_accel*	(Optional) Maximum allowed acceleration in [rad/s^2]. Default: 10000 rad/s^2.
in	*ID_max_jerk*	(Optional) Maximum allowed jerk in [rad/s^3]. Default: 1000 rad/s^3.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 91 of file Wrapper.cpp.

6.14.4.6 sequencer_write()

```
int32_t sequencer_write (
            SISProtocol * ID_ref,
            double_t ID_speeds[ ],
            double_t ID_accels[ ],
            double_t ID_jerks[ ],
            uint32_t ID_delays[ ],
            const uint16_t ID_set_length,
            ErrHandle ID_err = ErrHandle() )
```

Writes the whole run sequence into the device.

The run sequence is defined by several kinematic parameters, such as speed, acceleration, or jerk. A proper calculation of the kinetics before writing is assumed.

Bug List length will not be extended automatically. In case of list length is set too short, programming of all values might fail. This may cause an improper operation of the "Sequencer" drive mode.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_write(int ID_ref, Double[] ID_speeds, Double[] ID_accels
      , Double[] ID_jerks, UInt32[] ID_delays, UInt16 ID_set_length, ref ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
in	*ID_speeds*	Sequencer speed list in [1/min]. Rotation directions are defined by the sign of each element: • Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.
in	*ID_accels*	Sequencer acceleration list in [rad/s^2].
in	*ID_jerks*	Sequencer jerk list in [rad/s^3].
in	*ID_delays*	Delay list representing delay between each kinematic step in [cs].
in	*ID_set_length*	Length of the sequence (=number of elements of ID_speeds, ID_accels, etc).
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 129 of file Wrapper.cpp.

6.14.4.7 sequencer_softtrigger()

```
int32_t sequencer_softtrigger (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Software-Trigger to start operation of the "Sequencer" drive mode.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_softtrigger(int ID_ref, ref
      ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 187 of file Wrapper.cpp.

6.14.4.8 sequencer_hardtrigger()

```
int32_t sequencer_hardtrigger (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Hardware-Trigger to start operation of the "Sequencer" drive mode.

By special PLC software (if configured), the hardware trigger is realized through a 24V rising edge input line.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_hardtrigger(int ID_ref, ref
      ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

READ CUSTOM DATA

Definition at line 253 of file Wrapper.cpp.

6.14.4.9 sequencer_getstatus()

```
int32_t sequencer_getstatus (
            SISProtocol * ID_ref,
            uint16_t * ID_status,
            ErrHandle ID_err = ErrHandle() )
```

Get the status of the "Sequencer" drive mode.

The information is derived from the PLC that reports the actual status within an internal register.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_getstatus(int ID_ref, ref UInt16 status, ref
      ErrHandle ID_err);
```

.
How to call with Python:

```
plcstatus = ctypes.c_uint16(0)
result = indralib.sequencer_getstatus(indraref, ctypes.byref(plcstatus), ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_status*	Pointer that provides the respective information: • Bit 0 - bSeqFinished, if true then end of the sequence has been reached. • Bit 1 - bCamTick, if true then cam is currently shifting to the next position. • Bit 2 - bDriveStopped, if true then drive has been stopped by PLC. • Bit 3 - bDriveStarted, if true then drive has been started by PLC. • Bit 4 - ERROR_T_PARAM, if true then error has been occurred while reading input parameters.
		• Bit 5 - RESULT_READ_OK, if true then reading input parameters have been processed correctly. • Bit 6 - RESULT_SEQUENCER_OK, if true then sequencer is/was running well.

Returns

Error handle return code (ErrHandle()).

Definition at line 294 of file Wrapper.cpp.

6.14.4.10 speedcontrol_activate()

```
int32_t speedcontrol_activate (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Activates the drive mode "Speed Control".

Attention

Reiterate calls of this functions will harm the Indradrive EEPROM (due to limited write cycles). Use get_↩
drivemode() to check if this function call is really needed.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int speedcontrol_activate(int ID_ref, ref
      ErrHandle ID_err);
```

.
How to call with Python:

```
result = indralib.speedcontrol_activate(indraref, ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 323 of file Wrapper.cpp.

6.14.4.11 speedcontrol_init()

```
int32_t speedcontrol_init (
            SISProtocol * ID_ref,
            double_t ID_max_accel = 10000,
            double_t ID_max_jerk = 1000,
            ErrHandle ID_err = ErrHandle() )
```

Initializes limits and sets the right scaling/unit factors for operation of "Speed Control" drive mode.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int speedcontrol_init(int ID_ref, Double ID_max_accel, Double
      ID_max_jerk, ref ErrHandle ID_err);
```

.
How to call with Python:

```
result = indralib.speedcontrol_init(indraref, ctypes.c_double(10000), ctypes.c_double(1000), ctypes.byref(
      indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_max_accel*	(Optional) Maximum allowed acceleration in [rad/s^2]. Default: 10000 rad/s^2.
out	*ID_max_jerk*	(Optional) Maximum allowed jerk in [rad/s^3]. Default: 1000 rad/s^3.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 349 of file Wrapper.cpp.

6.14.4.12 speedcontrol_write()

```
int32_t speedcontrol_write (
            SISProtocol * ID_ref,
            double_t ID_speed,
            double_t ID_accel,
            ErrHandle ID_err = ErrHandle() )
```

Writes the current kinematic (speed and acceleration) into the device.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int speedcontrol_write(int ID_ref, Double ID_speed, Double ID_accel
      , ref ErrHandle ID_err);
```

.
How to call with Python:

```
result = indralib.speedcontrol_write(indraref, ctypes.c_double(speed), ctypes.c_double(10), ctypes.byref(
      indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_speed*	Target speed in [1/min]. Sign represents the rotation direction: • Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.
out	*ID_accel*	Target acceleration in [rad/s^2].
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 381 of file Wrapper.cpp.

6.14.4.13 set_stdenvironment()

```
int32_t set_stdenvironment (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Sets the proper unit and language environment.

Proper unit and language environment is:

- for unit setup: Preferred scaling / Rotary scaling / Unit [rpm] / Velocity data scaling
- for language environment: English language.

Attention

Not setting the proper unit and language environment may cause unexpected behavior when programming and setting kinematics.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with Python:

```
result = indralib.set_stdenvironment(indraref, ctypes.byref(indra_error))
```

.

Parameters

`in,out`	*ID_ref*	API reference. Pointer can be casted in from UINT32.
	ID_err	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 415 of file Wrapper.cpp.

6.14.4.14 get_drivemode()

```
int32_t get_drivemode (
            SISProtocol * ID_ref,
            uint32_t * ID_drvmode,
            ErrHandle ID_err = ErrHandle() )
```

Retrieve information about the drive mode: Speed Control or Sequencer.

The drive mode feedback is provided by `ID_drvmode` parameter. The following table depicts the coding:

If	Then
`*ID_drvmode == 0`	Drive Mode not supported
`*ID_drvmode == 1`	"Sequencer" drive mode active
`*ID_drvmode == 2`	"Speed Control" drive mode active.

Remarks

This function is exported to the Indradrive API DLL.
The drive mode can be changed by speedcontrol_activate() or sequencer_activate().
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_drivemode(int ID_ref, ref UInt32 mode, ref
      ErrHandle ID_err);
```

.
How to call with Python:

```
drvmode = ctypes.c_uint32(0)
result = indralib.get_drivemode(indraref, ctypes.byref(drvmode), ctypes.byref(indra_error))
```

.

Parameters

`in`	*ID_ref*	API reference. Pointer can be casted in from UINT32.
`out`	*ID_drvmode*	Pointer that provides the respective information: • 0 - Drive Mode not supported, • 1 - "Sequencer" drive mode active, • 2 - "Speed Control" drive mode active.

Parameters

out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 441 of file Wrapper.cpp.

6.14.4.15 get_opstate()

```
int32_t get_opstate (
            SISProtocol * ID_ref,
            uint8_t * ID_opstate,
            ErrHandle ID_err = ErrHandle() )
```

Retrieve information about the operation states: bb, Ab, or AF.

The operation state feedback is provided by `ID_opstate` parameter. The following table depicts the coding:

If	Then
`(*ID_opstate & 0b11) == 0b00`	Control section / power section not ready for operation(e.g., drive error or phase 2)
`(*ID_opstate & 0b11) == 0b01`	Control section ready for operation "bb"
`(*ID_opstate & 0b11) == 0b10`	Control section and power section ready for op. "Ab"
`(*ID_opstate & 0b11) == 0b11`	Drive with torque "AF".
`((*ID_opstate & 0b100) >> 2) == 1`	Drive Halt is active and axis is in standstill
`((*ID_opstate & 0b1000) >> 3) == 1`	Drive error.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(&lt;path_to_DLL&gt;, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_opstate(int ID_ref, ref Byte state, ref
      ErrHandle ID_err);
```

.
How to call with Python:

```
opstate = ctypes.c_uint8(0)
result = indralib.get_opstate(indraref, ctypes.byref(opstate), ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.

Parameters

out	*ID_opstate*	Pointer that provides the respective information: • Bit 0-1: Operation state – `0b00`: Control section / power section not ready for operation(e.g., drive error or phase 2) – `0b01`: Control section ready for operation "bb" – `0b10`: Control section and power section ready for op. "Ab" – `0b11`: Drive with torque "AF". • Bit 2: Drive Halt acknowledgment – `0`: Drive not halted – `1`: Drive Halt is active and axis is in standstill • Bit 3: Drive error – `0`: No error – `1`: Drive error present.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 481 of file Wrapper.cpp.

6.14.4.16 get_speed()

```
int32_t get_speed (
            SISProtocol * ID_ref,
            double_t * ID_speed,
            ErrHandle ID_err = ErrHandle() )
```

Gets the actual rotation speed.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_speed(int ID_ref, ref Double speed, ref
      ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_speed*	Pointer that provides the speed information as double Value in [1/min]. Sign represents the rotation direction:
		• Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.

Returns

Error handle return code (ErrHandle()).

Definition at line 511 of file Wrapper.cpp.

6.14.4.17 get_diagnostic_msg()

```
int32_t get_diagnostic_msg (
            SISProtocol * ID_ref,
            char * ID_diagnostic_msg,
            ErrHandle ID_err = ErrHandle() )
```

Gets diagnostic message string of the current Indradrive status.

Attention

The API presumes a properly allocated char array for `ID_diagnostic_msg` parameter.

Remarks

This function is exported to the Indradrive API DLL.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_diagnostic_msg(int ID_ref, Byte[] ID_diagnostic_msg, ref
      ErrHandle ID_err);
```

.
How to call with Python:

```
diagmsg = ctypes.create_string_buffer(256)
result = indralib.get_diagnostic_msg(indraref, diagmsg, ctypes.byref(indra_error))
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_diagnostic_msg*	Pointer that provides the diagnostic message string.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 540 of file Wrapper.cpp.

6.14.4.18 get_diagnostic_num()

```
int32_t get_diagnostic_num (
            SISProtocol * ID_ref,
            uint32_t * ID_diagnostic_num,
            ErrHandle ID_err = ErrHandle() )
```

Gets diagnostic number of the current Indradrive status.

Remarks

This function is exported to the Indradrive API DLL.
The coding of the diagnostic number is described in the document "Rexroth IndraDrive Firmware for Drive Controller MPH-04, MPB-04, MPD-04 / Function Description (Chapter 10.3). For example, operation state "AF" is described as 0xA0101.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_diagnostic_num(int ID_ref, ref UInt32 ID_diagnostic_num,
      ref ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_diagnostic_num*	Pointer that provides the diagnostic number.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 569 of file Wrapper.cpp.

6.14.4.19 clear_error()

```
int32_t clear_error (
            SISProtocol * ID_ref,
            ErrHandle ID_err = ErrHandle() )
```

Clears a latched error in the Indradrive device.

In case of error that has been occurred on the Indradrive, the error information is latched until cleared with this call.

Remarks

This function is exported to the Indradrive API DLL.
Use get_diagnostic_message() and/or get_diagnostic_num() for retrieving the error information.
Refer to Examples for detailed code examples.
How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int clear_error(int ID_ref, ref ErrHandle ID_err);
```

.

Parameters

in	*ID_ref*	API reference. Pointer can be casted in from UINT32.
out	*ID_err*	(Optional) Error handle.

Returns

Error handle return code (ErrHandle()).

Definition at line 598 of file Wrapper.cpp.

6.15 Wrapper.h

```
#ifndef _WRAPPER_H_
#define _WRAPPER_H_

#include "mainpage.dox"

#include <Windows.h>

#include "SISProtocol.h"
#include "RS232.h"
#include "errors.h"
#include "debug.h"


#define DLLEXPORT __declspec(dllexport)
#define DLLCALLCONV __cdecl

#ifndef _DLL
#error Project output has to be a DLL file
#endif

#if __cplusplus <= 199711L
#if _MSC_VER < 1900
#error This library needs at least Microsoft Visual Studio 2015 or a C++11 compliant compiler
#endif

#ifndef _MSC_VER
#error This library needs at least a C++11 compliant compiler
#endif
#endif

#ifndef __cplusplus
#error C++ compiler required
#else
extern "C" {  /*  using a C++ compiler  */
#endif

    #define DRIVEMODE_SEQUENCER     0b111011
    #define DRIVEMODE_SPEEDCONTROL  0b10

    typedef struct OPSTATE
    {
        union
        {
            struct Bits
            {
                uint8_t OperateState : 2;

                uint8_t DriveHalted : 1;

                uint8_t DriveError : 1;

                Bits(uint16_t P_0_0115 = 0) :
                    OperateState((P_0_0115 >> 14) & 0b11),
                    DriveHalted((P_0_0115 >> 4) & 0b1),
                    DriveError((P_0_0115 >> 13) & 0b1)
                {}
            } Bits;

            uint8_t Value;
        };

        OPSTATE(uint16_t P_0_0115 = 0) : Bits(P_0_0115) {}
    } OPSTATE;


    typedef struct SPEEDUNITS
    {
        union
        {
            struct Bits
            {
                uint16_t type_of_scaling : 3;

                uint16_t automode : 1;

                uint16_t scale_units : 1;

                uint16_t time_units : 1;

                uint16_t data_rel : 1;

                uint16_t res7 : 9;

                Bits(uint16_t S_0_0044 = 0) :
                    // Bit 0-2 @ S-0-0044
                    type_of_scaling((S_0_0044) & 0b111),
                    // Bit 3 @ S-0-0044
                    automode((S_0_0044 >> 3) & 0b1),
                    // Bit 4 @ S-0-0044
                    scale_units((S_0_0044 >> 4) & 0b1),
                    // Bit 5 @ S-0-0044
                    time_units((S_0_0044 >> 5) & 0b1),
                    // Bit 6 @ S-0-0044
                    data_rel((S_0_0044 >> 6) & 0b1),
                    // Bit 7-15 @ S-0-0044
                    res7((S_0_0044 >> 7) & 0b111111111)
                {}
            } Bits;

            uint16_t Value;
        };

        SPEEDUNITS(uint16_t S_0_0044 = 0) : Bits(S_0_0044) {}
    } SPEEDUNITS;


    typedef struct SISProtocol SISProtocol;


#pragma region API Fundamentals

    DLLEXPORT SISProtocol* DLLCALLCONV init();

    DLLEXPORT int32_t DLLCALLCONV open(SISProtocol* ID_ref, const
wchar_t* ID_comport = L"COM1", uint32_t ID_combaudrate = 19200, ErrHandle ID_err =
ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV close(SISProtocol* ID_ref,
ErrHandle ID_err = ErrHandle());

#pragma endregion API Fundamentals


#pragma region API Sequencer

    DLLEXPORT int32_t DLLCALLCONV sequencer_activate(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV sequencer_init(
SISProtocol* ID_ref, double_t ID_max_accel = 10000, double_t ID_max_jerk = 1000,
ErrHandle ID_err = ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV sequencer_write(
SISProtocol* ID_ref, double_t ID_speeds[], double_t ID_accels[], double_t ID_jerks[], uint32_t
ID_delays[], const uint16_t ID_set_length, ErrHandle ID_err = ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV sequencer_softtrigger(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV sequencer_hardtrigger(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV sequencer_getstatus(
SISProtocol* ID_ref, uint16_t * ID_status, ErrHandle ID_err =
ErrHandle());

#pragma endregion API Sequencer

```

```

#pragma region API Speed Control

    DLLEXPORT int32_t DLLCALLCONV speedcontrol_activate(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV speedcontrol_init(
SISProtocol* ID_ref, double_t ID_max_accel = 10000, double_t ID_max_jerk = 1000,
ErrHandle ID_err = ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV speedcontrol_write(
SISProtocol* ID_ref, double_t ID_speed, double_t ID_accel, ErrHandle ID_err =
ErrHandle());

#pragma endregion API Speed Control


#pragma region API Configuration

    DLLEXPORT int32_t DLLCALLCONV set_stdenvironment(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());

#pragma endregion API Configuration


#pragma region API Status

    DLLEXPORT int32_t DLLCALLCONV get_drivemode(
SISProtocol* ID_ref, uint32_t * ID_drvmode, ErrHandle ID_err =
ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV get_opstate(
SISProtocol* ID_ref, uint8_t * ID_opstate, ErrHandle ID_err =
ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV get_speed(
SISProtocol * ID_ref, double_t * ID_speed, ErrHandle ID_err =
ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV get_diagnostic_msg(
SISProtocol* ID_ref, char * ID_diagnostic_msg, ErrHandle ID_err =
ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV get_diagnostic_num(
SISProtocol* ID_ref, uint32_t * ID_diagnostic_num, ErrHandle ID_err =
ErrHandle());

    DLLEXPORT int32_t DLLCALLCONV clear_error(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());

#pragma endregion API Status

    /* \cond Do not document this */

#pragma region Internal helper functions

    inline void change_opmode(SISProtocol * ID_ref, const uint64_t opmode);

    inline SPEEDUNITS get_units(SISProtocol * ID_ref);

    inline void change_units(SISProtocol * ID_ref);

    inline void change_language(SISProtocol * ID_ref, const uint8_t lang_code = 1
);

#pragma endregion Internal helper functions

    /* \endcond Do not document this */

#ifdef __cplusplus
}
#endif

#endif /* _WRAPPER_H_ */
```

7 Example Documentation

7.1 apps/PythonApplication1/PythonApplication1.py

7.2 apps/WpfApplication1/Indradrive.cs

Index