

Indradrive API

0.5

Generated by Doxygen 1.8.13

Contents

1	Main Page	1
1.1	Introduction	2
1.1.1	Drive modes	2
1.1.2	API Modules	4
1.2	Installation	5
1.3	Usage	6
1.3.1	API Function Overview	6
1.4	Examples	6
1.4.1	C# Examples	6
1.4.2	Python Examples	6
2	Todo List	7
3	Bug List	7
4	Namespace Documentation	7
4.1	TGM Namespace Reference	7
4.1.1	Detailed Description	8
4.1.2	Typedef Documentation	8
4.1.3	Enumeration Type Documentation	9
4.2	TGM::Bitfields Namespace Reference	12
4.2.1	Detailed Description	13
4.2.2	Typedef Documentation	13
4.3	TGM::Commands Namespace Reference	13
4.3.1	Detailed Description	14
4.3.2	Typedef Documentation	14
4.4	TGM::Reactions Namespace Reference	15
4.4.1	Detailed Description	15
4.4.2	Typedef Documentation	15

5	Class Documentation	16
5.1	TGM::Bytestream Struct Reference	16
5.1.1	Detailed Description	16
5.1.2	Constructor & Destructor Documentation	16
5.1.3	Member Function Documentation	16
5.1.4	Member Data Documentation	17
5.2	TGM::Data Struct Reference	17
5.2.1	Detailed Description	18
5.2.2	Constructor & Destructor Documentation	18
5.2.3	Member Function Documentation	20
5.2.4	Member Data Documentation	22
5.3	SISProtocol::ExceptionGeneric Class Reference	23
5.3.1	Detailed Description	23
5.3.2	Constructor & Destructor Documentation	24
5.3.3	Member Function Documentation	24
5.3.4	Member Data Documentation	24
5.4	SISProtocol::ExceptionSISError Class Reference	25
5.4.1	Detailed Description	25
5.4.2	Constructor & Destructor Documentation	26
5.4.3	Member Function Documentation	26
5.4.4	Member Data Documentation	26
5.5	SISProtocol::ExceptionTransceiveFailed Class Reference	27
5.5.1	Detailed Description	27
5.5.2	Constructor & Destructor Documentation	27
5.5.3	Member Function Documentation	28
5.6	GenericErrHandle Struct Reference	28
5.6.1	Detailed Description	29
5.6.2	Constructor & Destructor Documentation	29
5.6.3	Member Function Documentation	29
5.6.4	Member Data Documentation	30

5.7	TGM::Header Struct Reference	31
5.7.1	Detailed Description	32
5.7.2	Constructor & Destructor Documentation	32
5.7.3	Member Function Documentation	33
5.7.4	Member Data Documentation	34
5.8	TGM::BitFields::HeaderControl Struct Reference	37
5.8.1	Detailed Description	38
5.8.2	Constructor & Destructor Documentation	38
5.8.3	Member Data Documentation	38
5.9	TGM::HeaderExt Struct Reference	40
5.9.1	Detailed Description	40
5.9.2	Member Data Documentation	40
5.10	TGM::Map< THeader, TPayload > Union Template Reference	42
5.10.1	Detailed Description	43
5.10.2	Constructor & Destructor Documentation	43
5.10.3	Member Function Documentation	43
5.10.4	Member Data Documentation	44
5.11	TGM::Map< THeader, TPayload >::Mapping Struct Reference	44
5.11.1	Detailed Description	45
5.11.2	Constructor & Destructor Documentation	45
5.11.3	Member Data Documentation	45
5.12	OPSTATE Struct Reference	46
5.12.1	Detailed Description	46
5.12.2	Constructor & Destructor Documentation	47
5.12.3	Member Data Documentation	47
5.13	TGM::Reactions::SercosList Struct Reference	48
5.13.1	Detailed Description	49
5.13.2	Constructor & Destructor Documentation	49
5.13.3	Member Function Documentation	49
5.13.4	Member Data Documentation	50

5.14	TGM::Commands::SercosList Struct Reference	51
5.14.1	Detailed Description	52
5.14.2	Constructor & Destructor Documentation	52
5.14.3	Member Function Documentation	53
5.14.4	Member Data Documentation	53
5.15	TGM::Reactions::SercosParam Struct Reference	55
5.15.1	Detailed Description	55
5.15.2	Constructor & Destructor Documentation	56
5.15.3	Member Function Documentation	56
5.15.4	Member Data Documentation	57
5.16	TGM::Commands::SercosParam Struct Reference	58
5.16.1	Detailed Description	58
5.16.2	Constructor & Destructor Documentation	58
5.16.3	Member Function Documentation	60
5.16.4	Member Data Documentation	61
5.17	TGM::Bitfields::SercosParamAttribute Struct Reference	62
5.17.1	Detailed Description	62
5.17.2	Constructor & Destructor Documentation	62
5.17.3	Member Data Documentation	63
5.18	TGM::Bitfields::SercosParamControl Struct Reference	65
5.18.1	Detailed Description	66
5.18.2	Constructor & Destructor Documentation	66
5.18.3	Member Data Documentation	67
5.19	TGM::Bitfields::SercosParamIdent Struct Reference	68
5.19.1	Detailed Description	69
5.19.2	Constructor & Destructor Documentation	69
5.19.3	Member Data Documentation	69
5.20	SISProtocol Class Reference	71
5.20.1	Detailed Description	72
5.20.2	Member Typedef Documentation	72

5.20.3	Member Enumeration Documentation	72
5.20.4	Constructor & Destructor Documentation	73
5.20.5	Member Function Documentation	74
5.21	SPEEDUNITS Struct Reference	77
5.21.1	Detailed Description	78
5.21.2	Constructor & Destructor Documentation	78
5.21.3	Member Data Documentation	78
5.22	TGM::Reactions::Subservice Struct Reference	80
5.22.1	Detailed Description	81
5.22.2	Constructor & Destructor Documentation	81
5.22.3	Member Function Documentation	81
5.22.4	Member Data Documentation	82
5.23	TGM::Commands::Subservice Struct Reference	83
5.23.1	Detailed Description	84
5.23.2	Constructor & Destructor Documentation	84
5.23.3	Member Function Documentation	85
5.23.4	Member Data Documentation	85
6	File Documentation	86
6.1	errors.h File Reference	86
6.1.1	Detailed Description	87
6.1.2	Macro Definition Documentation	87
6.1.3	Typedef Documentation	87
6.1.4	Enumeration Type Documentation	88
6.1.5	Function Documentation	89
6.2	errors.h	89
6.3	mainpage.dox File Reference	91
6.4	SISProtocol.cpp File Reference	91
6.5	SISProtocol.cpp	91
6.6	SISProtocol.h File Reference	100
6.6.1	Macro Definition Documentation	101

6.7	SISProtocol.h	102
6.8	Telegrams.h File Reference	105
6.8.1	Detailed Description	106
6.8.2	Macro Definition Documentation	106
6.9	Telegrams.h	107
6.10	Telegrams_Bitfields.h File Reference	113
6.10.1	Detailed Description	115
6.11	Telegrams_Bitfields.h	115
6.12	Wrapper.cpp File Reference	117
6.12.1	Detailed Description	118
6.12.2	Function Documentation	118
6.13	Wrapper.cpp	134
6.14	Wrapper.h File Reference	142
6.14.1	Detailed Description	144
6.14.2	Macro Definition Documentation	144
6.14.3	Typedef Documentation	145
6.14.4	Function Documentation	146
6.15	Wrapper.h	161
7	Example Documentation	163
7.1	apps/PythonApplication1/PythonApplication1.py	163
7.2	apps/WpfApplication1/Indradrive.cs	164
	Index	165

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 transferred 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:

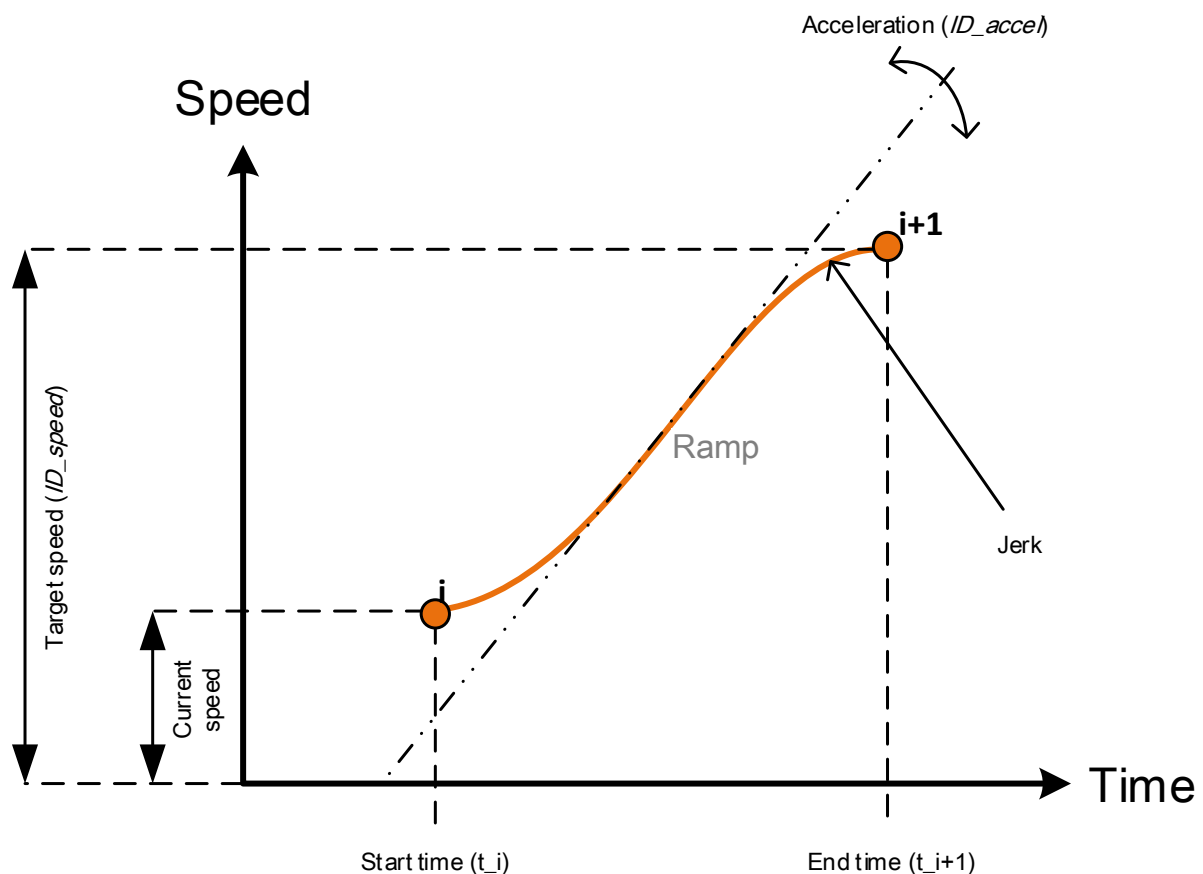


Figure 1 Principle of Speed Control

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:

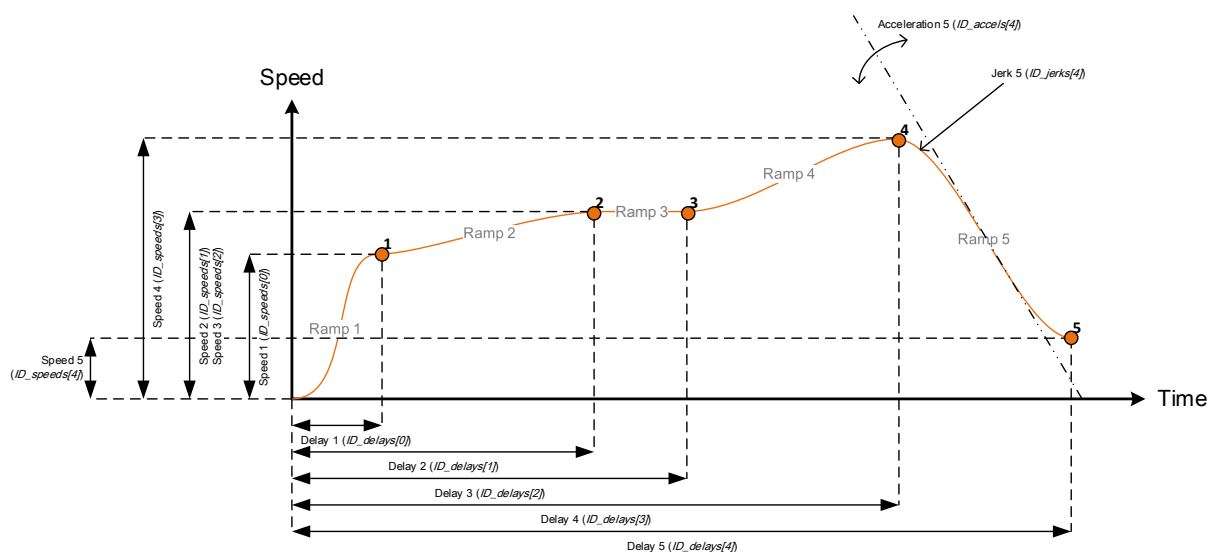


Figure 2 Principle of Sequencer

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 calculating 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:

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

Installation is very easy, if IndrdriveAPI.dll, or IndrdriveAPI-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 IndrdriveAPI.dll, or IndrdriveAPI-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 IndrdriveAPI.vcxproj)
5. Open the Visual Studio solution called Indrdrive.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_SIZE_MAX](#)]
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_SIZE_MAX_PAYLOAD]`
Actual payload Bytes [TGM_SIZE_MAX_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

<code>_PayloadData</code>	(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

<i>_PayloadData</i>	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

<i>_PayloadData</i>	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

<i>_PayloadData</i>	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

<code>_PayloadData</code>	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

<code>_idx</code>	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

<i>rhs</i>	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

<i>_size</i>	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](#).

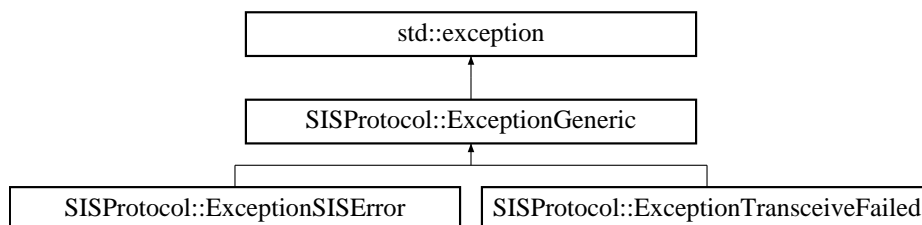
The documentation for this struct was generated from the following file:

- [Telegrams.h](#)

5.3 SISProtocol::ExceptionGeneric Class Reference

Generic exceptions for SIS protocol.

Inheritance diagram for SISProtocol::ExceptionGeneric:



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](#).

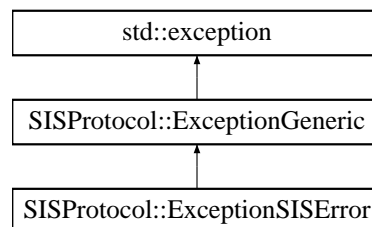
The documentation for this class was generated from the following file:

- [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](#).

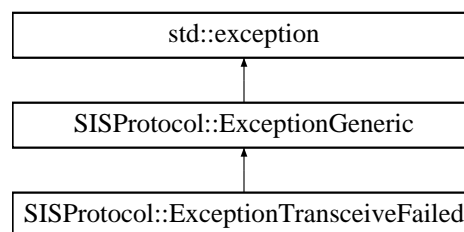
The documentation for this class was generated from the following file:

- [SISProtocol.h](#)

5.5 SISProtocol::ExceptionTransceiveFailed Class Reference

Specific exception handling of SIS Protocol transceiving failed.

Inheritance diagram for SISProtocol::ExceptionTransceiveFailed:



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 (`LVErrCluster_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>_code</code>	(Optional) Error code. Can be later on set also via set() function.
<code>_msg</code>	(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>_code</code>	Error code.
<code>_msg</code>	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

<code>_msg</code>	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>_code</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](#).

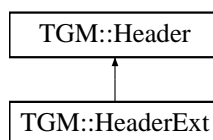
The documentation for this struct was generated from the following file:

- [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

<code>_addr_master</code>	(Optional) The address master id.
<code>_addr_slave</code>	(Optional) The address slave id.
<code>_service</code>	(Optional) The service id.
<code>_cntrl</code>	(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

<i>exclude_cs</i>	(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

<i>_payload_len</i>	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

	<i>_payload_len</i>	Length of the payload.
in	<i>_payload</i>	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 ($255 - 7\{\text{subaddresses}\} - 1\{\text{running telegram number}\}$) 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

<i>type</i>	(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 { ... }
```

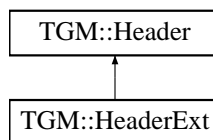
The documentation for this struct was generated from the following file:

- [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	<code>_header</code>	(Optional) The Telegram header.
in	<code>_payload</code>	(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	<code>_header</code>	The Telegram header.
in	<code>_payload</code>	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	<i>_header</i>	The Telegram header.
in	<i>_payload</i>	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 to 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

<code>P_0_0115</code>	(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](#)(), [USHORT](#) _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 handled.
- [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::SercosParamControl(),
    BYTE _unit_addr = 0,
    TGM::Bitfields::SercosParamIdent _ParamIdent = TGM::Bitfields::SercosParamIdent(),
    USHORT _ListOffset = 0,
    USHORT _SegmentSize = 0,
    TGM::Data _PayloadData = Data() ) [inline]
```

Constructor.

Parameters

<code>_ControlByte</code>	(Optional) Control Byte.
<code>_unit_addr</code>	(Optional) Unit address, which is the same as the SIS address of the receiver.
<code>_ParamIdent</code>	(Optional) Parameter Identifier.
<code>_ListOffset</code>	(Optional) List offset.
<code>_SegmentSize</code>	(Optional) Size of a single segment.
<code>_PayloadData</code>	(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 handled -> 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 handled.

For example: The 11th element of a list consisting of 4-byte elements should be handled -> SegmentSize=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::SercosParamIdent(),
    TGM::Data _data = Data() ) [inline]
```

Constructor.

Parameters

<code>_ControlByte</code>	(Optional) Control Byte.
<code>_unit_addr</code>	(Optional) Unit address, which is the same as the SIS address of the receiver.
<code>_ParamIdent</code>	(Optional) Parameter Identifier (e.g. S-0-4000).
<code>_PayloadData</code>	(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 convert the raw data value to the decimal value.
[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 the decimal value.
 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 decimal point.
 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

<code>_value</code>	(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

<i>datablock</i>	(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

<i>value</i>	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

<i>param_variant</i>	(Optional) The parameter variant, represented by SercosParamVar.
<i>param_num</i>	(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 and 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 and 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 to 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

<code>S_0_0044</code>	(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` parameter for [set_error\(\)](#) function.

Functions

- static int32_t [set_error](#) ([ErrHandle](#) errhdl, std::string errstr, int32_t block_code, int32_t issue_code=1)
Sets an error handle to the `errhdl` 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 (`LVErrCluster_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 errhdl,
    std::string errstr,
    int32_t block_code,
    int32_t issue_code = 1 ) [inline], [static]
```

Sets an error handle to the errhdl 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: $0 << 8 \mid \text{block_code} << 4 \mid \text{issue_code}$, whereas "|" indicates an OR- concatenation.

Parameters

out	<i>errhdl</i>	Error handle pointer.
in	<i>errstr</i>	Error message.
in	<i>block_code</i>	Error block code defined by EErrorBlocks enum.
in	<i>issue_code</i>	(Optional) The issue code.

Returns

The final error code.

See also

[EErrorBlocks](#)

Definition at line 183 of file [errors.h](#).

6.2 errors.h

```
00001
00004 #ifndef LABVIEW_ERRORS_H
00005 #define LABVIEW_ERRORS_H
00006
00007 #include <cstring>
00008 #include <string>
00009
00010
00011 #ifdef USE_LABVIEW_ENV
00012 // Including Labviewv.lib in case of project is compiled for LabVIEW
00013 #pragma comment(lib,"labviewv.lib")
00014 // Including extcode.h in case of project is compiled for LabVIEW
00015 #include "extcode.h"
00016 #endif
00017
00027 typedef struct GenericErrHandle {
00028
00030     uint32_t     code;
```

```

00032     char          msg[2048];
00033
00038     GenericErrHandle(uint32_t _code = 0, const char* _msg = "") :
00039         code(_code)
00040     {}
00041
00046     void set(uint32_t _code, const char* _msg)
00047     {
00048         code = _code;
00049
00050         for (size_t i = 0; i < strlen(_msg); i++)
00051             msg[i] = _msg[i];
00052     }
00053
00057     void set_msg(const char* _msg)
00058     {
00059         set(code, _msg);
00060     }
00061
00065     void set_code(uint32_t _code)
00066     {
00067         set(_code, msg);
00068     }
00069 } GenericErrHandle;
00070
00071
00072 #ifdef USE_LABVIEW_ENV
00073 typedef LStrHandle ErrHandle;
00077 #else
00078 typedef GenericErrHandle* ErrHandle;
00082 #endif
00083
00084
00085 #ifdef USE_LABVIEW_ENV
00086 #pragma pack(push,1)
00087 #include "extcode.h"
00088 #pragma pack(pop)
00089 #endif
00090
00091 #ifdef USE_LABVIEW_ENV
00092 #pragma pack(push,1)
00093 typedef struct {
00096     LVBoolean    status;
00097     int32        code;
00098     LStrHandle    msg;
00099 } LVErrCluster_t;
00100 #pragma pack(pop)
00101 #endif
00102
00103
00104 #ifdef USE_LABVIEW_ENV
00105 #define Err_Base (0x08EF)
00109 #else
00110
00114 #define Err_Base (0x0)
00115 #endif
00116
00120 typedef enum EErrorBlocks
00121 {
00123     Err_NoError                = 0,
00125     Err_Block_OpenByCOM        = 1,
00127     Err_Block_Close            = 2,
00129     Err_Block_Test             = 3,
00131     Err_Block_SeqInit          = 6,
00133     Err_Block_SeqWrite         = 7,
00135     Err_Block_VelCInit         = 8,
00137     Err_Block_VelCWrite        = 9,
00139     Err_Block_GetStatus        = 10,
00141     Err_Block_SetControl       = 11,
00143     Err_Invalid_Pointer       = 12
00144 } EErrorBlocks;
00145
00146 #ifdef USE_LABVIEW_ENV
00147 static MgErr write_string(ErrHandle lvhandle, std::string str)
00148 {
00155     //Initializes the buffer
00156     MgErr err = NumericArrayResize(uB, 1, (UHandle*)&lvhandle, str.length());
00157     if (err) return err;
00158
00159     //Informs the LabVIEW string handle about the size of the size
00160     (**lvhandle).cnt = str.length();
00161
00162     //Fills the string buffer with str
00163     strcpy((char*)(**lvhandle).str, str.c_str());
00164
00165     return noErr;
00166 }

```



```

00167 #endif
00168
00183 inline static int32_t set_error(ErrHandle errhdl, std::string errstr, int32_t block_code
, int32_t issue_code = 1)
00184 {
00185     int32_t retcode = (Err_Base << 8) | (block_code << 4) | issue_code;
00186
00187 #ifdef USE_LABVIEW_ENV
00188     write_string(errhdl, errstr);
00189 #else
00190     errhdl->set(retcode, errstr.c_str());
00191 #endif
00192
00193     return retcode;
00194 }
00195
00196
00197 #endif // LABVIEW_ERRORS_H
00198

```

6.3 mainpage.dox File Reference

6.4 SISProtocol.cpp File Reference

6.5 SISProtocol.cpp

```

00001 #include "SISProtocol.h"
00002
00003
00004
00005 SISProtocol::SISProtocol()
00006 {
00007 }
00008
00009
00010 SISProtocol::~SISProtocol()
00011 {
00012 }
00013
00014
00015 void SISProtocol::open(const wchar_t * _port)
00016 {
00017     STACK;
00018
00019     LPCTSTR cport = (LPCTSTR)_port;
00020     CSerial::EBaudrate cbaudrate = CSerial::EBaud19200;
00021     CSerial::EDataBits cdata = CSerial::EData8;
00022     CSerial::EParity cparity = CSerial::EParNone;
00023     CSerial::EStopBits cstopbits = CSerial::EStop1;
00024     CSerial::EHandshake chandshake = CSerial::EHandshakeOff;
00025
00026     try
00027     {
00028         CSerial::CheckPort(cport);
00029
00030         m_serial.Open(cport, RS232_BUFFER, RS232_BUFFER, true /* overlapped */);
00031         m_serial.Setup(cbaudrate, cdata, cparity, cstopbits);
00032         m_serial.SetupHandshaking(chandshake);
00033
00034         m_serial.SetMask(CSerial::EEventBreak |
00035             CSerial::EEventError |
00036             CSerial::EEventRecv);
00037
00038         m_serial.SetupReadTimeouts(CSerial::EReadTimeoutNonblocking);
00039     }
00040     catch (CSerial::ExceptionGeneric &ex)
00041     {
00042         throw;
00043     }
00044 }
00045
00046
00047 void SISProtocol::close()
00048 {
00049     STACK;
00050
00051     try
00052     {
00053         m_serial.Close();
00054     }
00055 }
00056

```



```

00132     // Fetching attributes for length and scale ...
00133     size_t datalen = 1;
00134     UINT8 scalefactor = 0;
00135     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00136
00137     // Communication with Telegrams ...
00138     auto rx_tgm = transceive_param
00139         <TGM::Header, TGM::Commands::SercosParam,
TGM::Header, TGM::Reactions::SercosParam>
00140         (_paramvar, _paramnum, SIS_SERVICE_SERCOS_PARAM_READ);
00141
00142     // Convert responded Bytes ...
00143     INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
00144     _rcvddata = (double)response / std::pow(10, scalefactor);
00145 }
00146
00147
00148 void SISProtocol::read_parameter(TGM::SercosParamVar
_paramvar, USHORT _paramnum, char _rcvddata[TGM_SIZEMAX_PAYLOAD])
00149 {
00150     // Communication with Telegrams ...
00151     auto rx_tgm = transceive_param
00152         <TGM::Header, TGM::Commands::SercosParam,
TGM::Header, TGM::Reactions::SercosParam>
00153         (_paramvar, _paramnum, SIS_SERVICE_SERCOS_PARAM_READ);
00154
00155     // Convert responded Bytes ...
00156     memcpy(_rcvddata, (char*)rx_tgm.Mapping.Payload.Bytes.Bytes, rx_tgm.Mapping.Payload.Bytes.Size);
00157     _rcvddata[rx_tgm.Mapping.Payload.Bytes.Size] = '\0';
00158 }
00159
00160
00161 void SISProtocol::read_listelm(TGM::SercosParamVar _paramvar,
USHORT _paramnum, USHORT _elm_pos, UINT32 & _rcvdelm)
00162 {
00163     STACK;
00164
00165     // Fetching attributes for length and scale ...
00166     size_t datalen = 1;
00167     UINT8 scalefactor = 0;
00168     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00169
00170     // Communication with Telegrams ...
00171     USHORT SegmentSize = (USHORT)datalen;
00172     USHORT ListOffset = _elm_pos * SegmentSize;
00173
00174     auto rx_tgm = transceive_list
00175         <TGM::Header, TGM::Commands::SercosList,
TGM::Header, TGM::Reactions::SercosList>
00176         (_paramvar, _paramnum, SIS_SERVICE_SERCOS_LIST_READ, SegmentSize,
ListOffset);
00177
00178     // Response Bytes ...
00179     INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
00180     _rcvdelm = static_cast<UINT32>(response);
00181 }
00182
00183
00184 void SISProtocol::read_listelm(TGM::SercosParamVar _paramvar,
USHORT _paramnum, USHORT _elm_pos, UINT64& _rcvdelm)
00185 {
00186     STACK;
00187
00188     // Fetching attributes for length and scale ...
00189     size_t datalen = 1;
00190     UINT8 scalefactor = 0;
00191     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00192
00193     // Communication with Telegrams ...
00194     USHORT SegmentSize = (USHORT)datalen;
00195     USHORT ListOffset = _elm_pos * SegmentSize;
00196
00197     auto rx_tgm = transceive_list
00198         <TGM::Header, TGM::Commands::SercosList,
TGM::Header, TGM::Reactions::SercosList>
00199         (_paramvar, _paramnum, SIS_SERVICE_SERCOS_LIST_READ, SegmentSize,
ListOffset);
00200
00201     // Response Bytes ...
00202     INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
00203     _rcvdelm = static_cast<UINT64>(response);
00204 }
00205
00206
00207 void SISProtocol::read_listelm(TGM::SercosParamVar _paramvar,
USHORT _paramnum, USHORT _elm_pos, DOUBLE & _rcvdelm)
00208 {

```

```

00209     STACK;
00210
00211     // Fetching attributes for length and scale ...
00212     size_t datalen = 1;
00213     UINT8 scalefactor = 0;
00214     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00215
00216     // Communication with Telegrams ...
00217     USHORT SegmentSize = (USHORT)datalen;
00218     USHORT ListOffset = _elm_pos * SegmentSize;
00219
00220     auto rx_tgm = transceive_list
00221     <TGM::Header, TGM::Commands::SercosList,
00222     TGM::Header, TGM::Reactions::SercosList>
00222     (_paramvar, _paramnum, SIS_SERVICE_SERCOS_LIST_READ,
00223     SegmentSize, ListOffset);
00223
00224     // Response Bytes ...
00225     INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
00226     _rcvdelm = (double)response / std::pow(10, scalefactor);
00227 }
00228
00229
00230 INT64 SISProtocol::get_sized_data(TGM::Data& rx_data, const size_t &datalen)
00231 {
00232     STACK;
00233
00234     if (datalen == 1)
00235     {
00236         UINT8 Bytes = rx_data.toUINT8();
00237         UINT64 mask = ((Bytes >> 7) & 1) ? 0xFFFFFFFFFFFFFFF0 : 0;
00238         return (INT64)(Bytes | mask);
00239     }
00240     else if (datalen == 2)
00241     {
00242         UINT16 Bytes = rx_data.toUINT16();
00243         UINT64 mask = ((Bytes >> 15) & 1) ? 0xFFFFFFFFFFFF0000 : 0;
00244         return (INT64)(Bytes | mask);
00245     }
00246     else if (datalen == 8)
00247     {
00248         return (INT64)rx_data.toUINT64();
00249     }
00250     else
00251     {
00252         UINT32 Bytes = rx_data.toUINT32();
00253         UINT64 mask = ((Bytes >> 31) & 1) ? 0xFFFFFFFF00000000 : 0;
00254         return (INT64)(Bytes | mask);
00255     }
00256 }
00257
00258
00259 void SISProtocol::write_parameter(
00260     TGM::SercosParamVar _paramvar, USHORT _paramnum, const UINT32 _data)
00260 {
00261     STACK;
00262
00263     write_parameter(_paramvar, _paramnum, static_cast<DOUBLE>(_data));
00264 }
00265
00266
00267 void SISProtocol::write_parameter(
00268     TGM::SercosParamVar _paramvar, USHORT _paramnum, const UINT64 _data)
00268 {
00269     STACK;
00270
00271     write_parameter(_paramvar, _paramnum, static_cast<DOUBLE>(_data));
00272 }
00273
00274
00275 void SISProtocol::write_parameter(
00276     TGM::SercosParamVar _paramvar, USHORT _paramnum, const DOUBLE _data)
00276 {
00277     STACK;
00278
00279     // Fetching attributes for length and scale ...
00280     size_t datalen = 1;
00281     UINT8 scalefactor = 0;
00282     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00283
00284     // Preprocess Bytes ...
00285     UINT64 inval = static_cast<UINT64>(_data * std::pow(10, scalefactor));
00286
00287     // Communication with Telegrams ...
00288     TGM::Data Bytes;
00289     set_sized_data(Bytes, datalen, inval);
00290

```

```

00291     transceive_param
00292     <TGM::Header, TGM::Commands::SercosParam,
00293     TGM::Header, TGM::Reactions::SercosParam>
00294     (_paramvar, _paramnum, SIS_SERVICE_SERCOS_PARAM_WRITE, &Bytes);
00295 }
00296
00297 void SISProtocol::write_listelm(TGM::SercosParamVar _paramvar,
00298     USHORT _paramnum, USHORT _elm_pos, const UINT32 _rcvdelm)
00299 {
00300     STACK;
00301     DOUBLE buf = static_cast<DOUBLE>(_rcvdelm);
00302     write_listelm(_paramvar, _paramnum, _elm_pos, buf);
00303 }
00304
00305 void SISProtocol::write_listelm(TGM::SercosParamVar _paramvar,
00306     USHORT _paramnum, USHORT _elm_pos, const UINT64 _rcvdelm)
00307 {
00308     STACK;
00309     DOUBLE buf = static_cast<DOUBLE>(_rcvdelm);
00310     write_listelm(_paramvar, _paramnum, _elm_pos, buf);
00311 }
00312
00313 void SISProtocol::write_listelm(TGM::SercosParamVar _paramvar,
00314     USHORT _paramnum, USHORT _elm_pos, const DOUBLE _rcvdelm)
00315 {
00316     STACK;
00317     // Fetching attributes for length and scale ...
00318     size_t datalen = 0;
00319     UINT8 scalefactor = 0;
00320     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00321     UINT64 inval = static_cast<UINT64>(_rcvdelm * std::pow(10, scalefactor));
00322     // Re-adjusting list size, if needed
00323     set_parameter_listsize(_paramvar, _paramnum, datalen, _elm_pos);
00324     TGM::Data Bytes;
00325     set_sized_data(Bytes, datalen, inval);
00326     // Communication with Telegrams ...
00327     USHORT SegmentSize = (USHORT)datalen;
00328     USHORT ListOffset = _elm_pos * SegmentSize;
00329     transceive_list
00330     <TGM::Header, TGM::Commands::SercosList,
00331     TGM::Header, TGM::Reactions::SercosList>
00332     (_paramvar, _paramnum, SIS_SERVICE_SERCOS_LIST_WRITE, SegmentSize,
00333     ListOffset, &Bytes);
00334 }
00335
00336 void SISProtocol::execute_command(
00337     TGM::SercosParamVar _paramvar, USHORT _paramnum)
00338 {
00339     TGM::SercosCommandrequest cmd;
00340     TGM::SercosCommandstatus Status =
00341     TGM::Commandstatus_Busy;
00342     int iterations;
00343     // Start command ...
00344     cmd = TGM::Commandrequest_Set;
00345     try
00346     {
00347         write_parameter(_paramvar, _paramnum, static_cast<UINT64>(cmd));
00348     }
00349     catch (SISProtocol::ExceptionSISError &ex)
00350     {
00351         if (ex.get_errorcode() == 0x700C)
00352             throw SISProtocol::ExceptionGeneric(-1, "Command cannot be
00353             executed, because it is write-protected. Release the drive torque (disable drive), or restart the Indradrive
00354             system.");
00355         else
00356             throw;
00357     }
00358     iterations = 0;
00359     do
00360     {
00361         get_parameter_status(_paramvar, _paramnum, Status);
00362         if (iterations > MAX_COMMANDCHECK_ITERATIONS) throw

```

```

ExceptionGeneric(-1, "Command execution caused a continuous busy loop. Please restart the
Indradrive system.");
00368     } while (Status == TGM::Commandstatus_Busy);
00369
00370     if (Status != TGM::Commandstatus_OK)
00371         throw ExceptionGeneric(static_cast<int>(Status), sformat("Command execution failed
with status code %d. Command execution canceled or not possible due to released operation state of the
drive.", Status));
00372
00373     // Delete command ...
00374     cmd = TGM::Commandrequest_NotSet;
00375     write_parameter(_paramvar, _paramnum, static_cast<UINT64>(cmd));
00376
00377     Status = TGM::Commandstatus_Busy;
00378     iterations = 0;
00379     do
00380     {
00381         get_parameter_status(_paramvar, _paramnum, Status);
00382
00383         if (iterations > MAX_COMMANDCHECK_ITERATIONS) throw
ExceptionGeneric(-1, "Command execution caused a continuous busy loop. Please restart the
Indradrive system.");
00384     } while (Status == TGM::Commandstatus_Busy);
00385
00386     if (Status != TGM::Commandstatus_NotSet)
00387         throw ExceptionGeneric(static_cast<int>(Status), sformat("Command execution failed
with status code %d. Command execution canceled or not possible due to released operation state of the
drive.", Status));
00388 }
00389
00390
00391
00392 void SISProtocol::get_parameter_status(const TGM::SercosParamVar _paramvar, const USHORT
& _paramnum, TGM::SercosCommandstatus& _datastatus)
00393 {
00394     STACK;
00395
00396     // Communication with Telegrams ...
00397     BYTE service = SIS_SERVICE_SERCOS_PARAM_WRITE;
00398
00399     auto rx_tgm = transceive_param
00400         <TGM::Header, TGM::Commands::SercosParam,
TGM::Header, TGM::Reactions::SercosParam>
00401         (_paramvar, _paramnum, service, new TGM::Data(),
TGM::Datablock_IdentNumber);
00402
00403     // Read back Datablock ...
00404     _datastatus = static_cast<TGM::SercosCommandstatus>(rx_tgm.Mapping.Payload.
Bytes.toUINT8());
00405 }
00406
00407
00408 void SISProtocol::set_sized_data(TGM::Data& tx_data, const size_t &datalen, UINT64 & _rcvdelm)
00409 {
00410     STACK;
00411
00412     if (datalen == 1) tx_data = TGM::Data((UINT8)_rcvdelm);
00413     else if (datalen == 2) tx_data = TGM::Data((UINT16)_rcvdelm);
00414     else if (datalen == 4) tx_data = TGM::Data((UINT32)_rcvdelm);
00415     else if (datalen == 8) tx_data = TGM::Data((UINT64)_rcvdelm);
00416     else tx_data = TGM::Data((UINT8&)_rcvdelm);
00417 }
00418
00419
00420 template <class TCHheader, class TCPayload, class TRHeader, class TRPayload>
00421 TGM::Map<TRHeader, TRPayload> SISProtocol::transceive_param(
TGM::SercosParamVar _paramvar, const USHORT &_paramnum, BYTE _service,
TGM::Data const * const _data, TGM::SercosDatablock _attribute)
00422 {
00423     // Build Telegrams ...
00424     TGM::Bitfields::SercosParamControl ParamControl(_attribute);
00425     TGM::Bitfields::SercosParamIdent ParamIdent(_paramvar, _paramnum);
00426
00427     // Mapping for SEND Telegram
00428     TGM::Map<TCHheader, TCPayload>
00429     tx_tgm(
00430         // Init header
00431         TCHheader(SIS_ADDR_MASTER, SIS_ADDR_SLAVE, _service,
TGM::Bitfields::HeaderControl(TGM::TypeCommand)),
00432         // Init payload
00433         TCPayload(ParamControl, SIS_ADDR_SLAVE, ParamIdent, *_data)
00434     );
00435
00436     // Set payload size
00437     tx_tgm.Mapping.Header.set_DatL(tx_tgm.Mapping.Payload.get_size());
00438
00439     // Calculate Checksum

```

```

00440     tx_tgm.Mapping.Header.calc_checksum(&tx_tgm.Raw);
00441
00442     if (!check_boundaries(tx_tgm))
00443         throw SISProtocol::ExceptionGeneric(-1, "Boundaries are out of spec.
Telegram is not ready to be sent.");
00444
00445     // Mapping for RECEPTION Telegram
00446     TGM::Map<TRHeader, TRPayload> rx_tgm;
00447
00448     // Transceive ...
00449     // Send and receive
00450     transceiving< TCHheader, TCPayload,
00451         TRHeader, TRPayload >
00452         (tx_tgm, rx_tgm);
00453
00454     return rx_tgm;
00455 }
00456
00457
00458 void SISProtocol::set_parameter_listsize(TGM::SercosParamVar param_variant, USHORT &
param_number, const size_t &datalen, const USHORT & segment_position, bool retain_following_segments)
00459 {
00460     // Getting Parameter header ...
00461     USHORT size = datalen;
00462     USHORT pos = 0;
00463     auto rx_tgm = transceive_list
00464         <TGM::Header, TGM::Commands::SercosList,
TGM::Header, TGM::Reactions::SercosList>
00465         (param_variant, param_number, SIS_SERVICE_SERCOS_LIST_READ, size, pos);
00466
00467     UINT32 param_header = rx_tgm.Mapping.Payload.Bytes.toUINT32();
00468     // Maximum possible size of parameter list
00469     UINT16 param_size_max = param_header >> 16;
00470     // Actual size of parameter list
00471     UINT16 param_size_cur = param_header & 0xFFFF;
00472
00473     // 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
00474     param_size_cur = retain_following_segments ?
00475         std::max<UINT16>(param_size_cur, segment_position * datalen) :
00476         static_cast<UINT16>(segment_position * datalen);
00477
00478     if (param_size_cur > param_size_max) return;
00479     if (param_size_cur == param_size_max) return;
00480
00481     // Update the Parameter header ...
00482     TGM::Data *new_header = new TGM::Data((UINT32)((param_size_max << 16) |
param_size_cur));
00483
00484     transceive_list
00485         <TGM::Header, TGM::Commands::SercosList,
TGM::Header, TGM::Reactions::SercosList>
00486         (param_variant, param_number, SIS_SERVICE_SERCOS_LIST_WRITE, size, pos
, new_header);
00487 }
00488
00489
00490 template<class TCHheader, class TCPayload, class TRHeader, class TRPayload>
00491 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)
00492 {
00493     // Build Telegrams ...
00494     TGM::Bitfields::SercosParamControl sercos_control(_attribute);
00495     TGM::Bitfields::SercosParamIdent ParamNum(_paramvar, _paramnum);
00496
00497     // Mapping for SEND Telegram
00498     TGM::Map<TCHheader, TCPayload>
00499     tx_tgm(
00500         // Init header
00501         TCHheader(SIS_ADDR_MASTER, SIS_ADDR_SLAVE, _service,
TGM::Bitfields::HeaderControl(TGM::TypeCommand)),
00502         // Init payload
00503         TCPayload(sercos_control, SIS_ADDR_SLAVE, ParamNum, _list_offset, _element_size,
*_data)
00504     );
00505
00506     // Set payload size
00507     tx_tgm.Mapping.Header.set_DatL(tx_tgm.Mapping.Payload.get_size());
00508
00509     // Calculate Checksum
00510     tx_tgm.Mapping.Header.calc_checksum(&tx_tgm.Raw);
00511
00512     if (!check_boundaries(tx_tgm))
00513         throw SISProtocol::ExceptionGeneric(-1, "Boundaries are out of spec.

```

```

Telegram is not ready to be sent.");
00514
00515 // Mapping for RECEPTION Telegram
00516 TGM::Map<TRHeader, TRPayload> rx_tgm;
00517
00518 // Transceive ...
00519 // Send and receive
00520 transceiving< TCHeader, TCPayload,
00521 TRHeader, TRPayload >
00522 (tx_tgm, rx_tgm);
00523
00524 return rx_tgm;
00525 }
00526
00527
00528 void SISProtocol::get_parameter_attributes(TGM::SercosParamVar _paramvar, const USHORT &
_paramnum, UINT8& _scalefactor, size_t& _datalen)
00529 {
00530     STACK;
00531
00532 // Communication with Telegrams ...
00533 BYTE service = SIS_SERVICE_SERCOS_PARAM_READ;
00534
00535 auto rx_tgm = transceive_param
00536 <TGM::Header, TGM::Commands::SercosParam,
TGM::Header, TGM::Reactions::SercosParam>
00537 (_paramvar, _paramnum, service, new TGM::Data(),
TGM::Datablock_Attribute);
00538
00539 // Read back Datablock ...
00540 UINT32 attr = rx_tgm.Mapping.Payload.Bytes.toUINT32();
00541 TGM::BitFields::SercosParamAttribute sercos_attribute(attr);
00542
00543 _datalen = 1;
00544 if (sercos_attribute.Bits.DataLen == TGM::Datalen_2ByteList) _datalen = 2;
00545 else if (sercos_attribute.Bits.DataLen == TGM::Datalen_4ByteList) _datalen =
4;
00546 else if (sercos_attribute.Bits.DataLen == TGM::Datalen_8ByteList) _datalen =
8;
00547 else if (sercos_attribute.Bits.DataLen == TGM::Datalen_2ByteParam) _datalen
= 2;
00548 else if (sercos_attribute.Bits.DataLen == TGM::Datalen_4ByteParam) _datalen
= 4;
00549 else if (sercos_attribute.Bits.DataLen == TGM::Datalen_8ByteParam) _datalen
= 8;
00550
00551 _scalefactor = 0xFF & sercos_attribute.Bits.ScaleFactor;
00552 }
00553
00554
00555 template <class TCHeader, class TCPayload, class TRHeader, class TRPayload>
00556 void SISProtocol::transceiving(TGM::Map<TCHeader, TCPayload>& tx_tgm,
TGM::Map<TRHeader, TRPayload>& rx_tgm)
00557 {
00558     STACK;
00559
00560 // Lock mutex to set the semaphore, so that the SIS access be reentrant
00561 mutex_sis.lock();
00562
00563 // Transceiver lengths
00564 size_t tx_payload_len = tx_tgm.Mapping.Payload.get_size();
00565 size_t tx_header_len = tx_tgm.Mapping.Header.get_size();
00566
00567 // Receiver lengths
00568 size_t rx_header_len = rx_tgm.Mapping.Header.get_size();
00569 size_t rx_payload_len = 0;
00570
00571 // Clear buffers
00572 m_serial.Purge();
00573
00574 // Write ...
00575 m_serial.Write(tx_tgm.Raw.Bytes, tx_header_len + tx_payload_len);
00576
00577 // Read ...
00578 bool bContd = true;
00579 DWORD rcvd_cur = 0;
00580 DWORD rcvd_rcnt = 0;
00581
00582 do
00583 {
00584 // Wait for an event
00585 m_serial.WaitEvent(0, RS232_READ_TIMEOUT);
00586
00587 // Save event
00588 const CSerial::EEvent event = m_serial.GetEventType();
00589
00590 // Handle Break event

```



```

00591         if (event & CSerial::EEventBreak)
00592             throw SISProtocol::ExceptionTransceiveFailed(
CSerial::EEventBreak, "Break event occurred. Transceive has been aborted.", true);
00593
00594         // Handle error event
00595         if (event & CSerial::EEventError)
00596             throw_rs232_error_events(m_serial.GetError());
00597
00598         // Handle Bytes receive event
00599         if (event & CSerial::EEventRecv)
00600         {
00601             // Read header Bytes
00602             m_serial.Read(rx_tgm.Raw.Bytes + rcvd_rcnt, RS232_BUFFER - rcvd_rcnt, &rcvd_cur,
0, RS232_READ_TIMEOUT);
00603
00604             // Loop back if nothing received
00605             if (rcvd_cur == 0) continue;
00606
00607             // Hold back number of already received bytes
00608             rcvd_rcnt += rcvd_cur;
00609
00610             // It is assumed that if the number of received bytes is bigger than 4,
00611             // which is the position of the payload length, the length can be read out.
00612             if (rcvd_rcnt > 4)
00613             {
00614                 rx_payload_len = rx_tgm.Mapping.Header.DatL;
00615                 rx_tgm.Mapping.Payload.Bytes.set_size(rx_payload_len - rx_tgm.
Mapping.Payload.get_head_size());
00616             }
00617
00618             // Length of payload is zero --> No payload received
00619             if (rx_payload_len == 0)
00620             {
00621                 std::string tx_hexstream = hexprint_bytestream(tx_tgm.Raw.Bytes, tx_header_len +
tx_payload_len);
00622                 std::string rx_hexstream = hexprint_bytestream(rx_tgm.Raw.Bytes, rx_header_len);
00623                 throw SISProtocol::ExceptionTransceiveFailed(-1,
sformat("Reception Telegram received without payload, but just the header.\nReception Header bytestream: %s.\n
Command Telegram bytestream was: %s.", rx_hexstream.c_str(), tx_hexstream.c_str()), true);
00624             }
00625
00626
00627             // Complete Telegram received
00628             if (rx_header_len + rx_payload_len <= rcvd_rcnt)
00629             {
00630                 if (rx_tgm.Mapping.Payload.Status)
00631                 {
00632                     std::string tx_hexstream = hexprint_bytestream(tx_tgm.Raw.Bytes, tx_header_len +
tx_payload_len);
00633
00634                     USHORT error = rx_tgm.Mapping.Payload.Error;
00635
00636                     if (error == 0x800C || error == 0x800B || error == 0x8001)
00637                     {
00638                         mutex_sis.unlock();
00639                         transceiving<TCHheader, TCPayload, TRHeader, TRPayload>(tx_tgm, rx_tgm);
00640                     }
00641                     else
00642                         throw SISProtocol::ExceptionSISError(rx_tgm.
Mapping.Payload.Status, rx_tgm.Mapping.Payload.Error, tx_hexstream);
00643                 }
00644
00645                 bContd = false;
00646             }
00647         }
00648     } while (bContd);
00649 } while (bContd);
00650
00651 // Unlock mutex to unset the semaphore
00652 mutex_sis.unlock();
00653 }
00654
00655
00656 template<class THeader, class TPayload>
00657 bool SISProtocol::check_boundaries(TGM::Map<THeader, TPayload>& _tgm)
00658 {
00659     STACK;
00660
00661     size_t tgm_size = _tgm.Mapping.Header.get_size() + _tgm.Mapping.
Payload.get_size();
00662     if (tgm_size <= RS232_BUFFER) return true;
00663
00664     return false;
00665 }
00666
00667
00668 std::string SISProtocol::hexprint_bytestream(const BYTE * _bytestream, const size_t _len)

```

```

00669 {
00670     STACK;
00671
00672     std::string buf;
00673
00674     for (size_t i = 0; i < _len; i++)
00675         buf.append(sformat("%02X ", (BYTE)_bytestream[i]));
00676
00677     return buf;
00678 }
00679
00680 void SISProtocol::throw_rs232_error_events(CSerial::EError _err)
00681 {
00682     STACK;
00683
00684     switch (_err)
00685     {
00686     case CSerial::EErrorBreak:
00687         throw SISProtocol::ExceptionTransceiveFailed(
00688             CSerial::EErrorBreak, "Break condition occurred. Transceive has been aborted.", true);
00689
00690     case CSerial::EErrorFrame:
00691         throw SISProtocol::ExceptionTransceiveFailed(
00692             CSerial::EErrorFrame, "Framing error occurred. Transceive has been aborted.", true);
00693
00694     case CSerial::EErrorIOE:
00695         throw SISProtocol::ExceptionTransceiveFailed(
00696             CSerial::EErrorIOE, "IO device error occurred. Transceive has been aborted.", true);
00697
00698     case CSerial::EErrorMode:
00699         throw SISProtocol::ExceptionTransceiveFailed(
00700             CSerial::EErrorMode, "Unsupported mode detected. Transceive has been aborted.", true);
00701
00702     case CSerial::EErrorOverrun:
00703         throw SISProtocol::ExceptionTransceiveFailed(
00704             CSerial::EErrorOverrun, "Buffer overrun detected. Transceive has been aborted.", true);
00705
00706     case CSerial::EErrorRxOver:
00707         throw SISProtocol::ExceptionTransceiveFailed(
00708             CSerial::EErrorRxOver, "Input buffer overflow detected. Transceive has been aborted.", true);
00709
00710     case CSerial::EErrorParity:
00711         throw SISProtocol::ExceptionTransceiveFailed(
00712             CSerial::EErrorParity, "Input parity occurred. Transceive has been aborted.", true);
00713
00714     case CSerial::EErrorTxFull:
00715         throw SISProtocol::ExceptionTransceiveFailed(
00716             CSerial::EErrorTxFull, "Output buffer full. Transceive has been aborted.", true);
00717
00718     default:
00719         throw SISProtocol::ExceptionTransceiveFailed(
00720             CSerial::EErrorBreak, "Unknown error occurred. Transceive has been aborted.", true);
00721     }
00722 }
00723 }

```

6.6 SISProtocol.h File Reference

Classes

- class [SISProtocol](#)
Class to hold functions and 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,
```

```

00048
00049     SIS_SERVICE_SERCOS_PARAM_READ = 0x10,
00050     SIS_SERVICE_SERCOS_LIST_READ = 0x11,
00051     SIS_SERVICE_SERCOS_READ_PHASE = 0x12,
00052     SIS_SERVICE_SERCOS_SWITCH_PHASE = 0x1D,
00053     SIS_SERVICE_SERCOS_LIST_WRITE = 0x1E,
00054     SIS_SERVICE_SERCOS_PARAM_WRITE = 0x1F
00055 } SIS_SERVICES;
00056
00059 typedef enum BAUDRATE
00060 {
00062     Baud_9600 = 0b00000000,
00064     Baud_19200 = 0b00000001,
00066     Baud_38400 = 0b00000010,
00068     Baud_57600 = 0b00000100,
00070     Baud_115200 = 0b00001000
00071 } BAUDRATE;
00072
00074 SISProtocol();
00076 virtual ~SISProtocol();
00077
00078
00079
00080 void open(const wchar_t * _port = L"COM1");
00081 void close();
00082
00083 void set_baudrate(BAUDRATE baudrate);
00084
00085 void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00086     UINT32& _rcvddata);
00087 void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00088     UINT64& _rcvddata);
00089 void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00090     DOUBLE& _rcvddata);
00091 void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00092     char _rcvddata[TGM::SIZEMAX_PAYLOAD]);
00093
00094 void read_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00095     USHORT _elm_pos, UINT32& _rcvdelm);
00096 void read_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00097     USHORT _elm_pos, UINT64& _rcvdelm);
00098 void read_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00099     USHORT _elm_pos, DOUBLE& _rcvdelm);
00100
00101 void write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00102     const UINT32 _data);
00103 void write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00104     const UINT64 _data);
00105 void write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00106     const DOUBLE _data);
00107
00108 void write_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00109     USHORT _elm_pos, const UINT32 _rcvdelm);
00110 void write_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00111     USHORT _elm_pos, const UINT64 _rcvdelm);
00112 void write_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00113     USHORT _elm_pos, const DOUBLE _rcvdelm);
00114
00115 void execute_command(TGM::SercosParamVar _paramvar, USHORT _paramnum)
00116 ;
00117
00118 private:
00119
00120 inline void get_parameter_attributes(TGM::SercosParamVar _paramvar, const USHORT &
00121     _paramnum, UINT8& _scalefactor, size_t& _datalen);
00122 inline void get_parameter_status(const TGM::SercosParamVar _paramvar, const USHORT &
00123     _paramnum, TGM::SercosCommandStatus& _datastatus);
00124
00125 template <class TCHheader, class TCPayload, class TRHeader, class TRPayload>
00126 TGM::Map<TRHeader, TRPayload> transceive_param(
00127     TGM::SercosParamVar _paramvar, const USHORT &_paramnum, BYTE _service,
00128     TGM::Data const * const _data = new TGM::Data(),
00129     TGM::SercosDatablock _attribute =
00130     TGM::Datablock_OperationData);
00131
00132 template <class TCHheader, class TCPayload, class TRHeader, class TRPayload>
00133 TGM::Map<TRHeader, TRPayload> transceive_list(
00134     TGM::SercosParamVar _paramvar, const USHORT &_paramnum, BYTE _service, USHORT &
00135     _element_size, USHORT & _list_offset, TGM::Data const * const _data = new
00136     TGM::Data(), TGM::SercosDatablock _attribute =
00137     TGM::Datablock_OperationData);
00138
00139 template <class THeader, class TPayload>
00140 inline bool check_boundaries(TGM::Map<THeader, TPayload>& _tgm);
00141
00142 static std::string hexprint_bytestream(const BYTE * _bytestream, const size_t _len);

```

```

00136
00137     inline INT64 get_sized_data(TGM::Data& rx_data, const size_t &datalen);
00138     inline void set_sized_data(TGM::Data& tx_data, const size_t &datalen, UINT64& _rcvdelm);
00139     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);
00140
00141 private:
00142
00143     template <class TCHheader, class TCPayload, class TRHeader, class TRPayload>
00144     void transceiving(TGM::Map<TCHheader, TCPayload>& tx_tgm,
TGM::Map<TRHeader, TRPayload>& rx_tgm);
00145
00146     static void throw_rs232_error_events(CSerial::EError _err);
00147
00148 private:
00149     CSerial m_serial;
00150
00151     std::mutex mutex_sis;
00152 };
00153
00157 class SISProtocol::ExceptionGeneric : public std::exception
00158 {
00159 public:
00160     bool warning;
00161
00162     ExceptionGeneric(
00163         int _status,
00164         const std::string _trace_log,
00165         bool _warning = false) :
00166
00167         m_status(_status),
00168         m_message(_trace_log),
00169         warning(_warning)
00170     {}
00171
00172     virtual const char* what() const throw ()
00173     {
00174 #ifdef NDEBBUG
00175         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()));
00176 #else
00177         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()));
00178         OutputDebugStringA((LPCSTR)ex);
00179         return ex;
00180 #endif
00181     }
00182
00183     int get_status() { return m_status; }
00184
00185 protected:
00186     int m_status;
00187
00188     std::string m_message;
00189 };
00190
00194 class SISProtocol::ExceptionTransceiveFailed : public
SISProtocol::ExceptionGeneric
00195 {
00196 public:
00197     ExceptionTransceiveFailed(
00198         int _status,
00199         const std::string _message,
00200         bool _warning = false) :
00201
00202         ExceptionGeneric(_status, _message, _warning)
00203     {}
00204     ~ExceptionTransceiveFailed() throw() {}
00205
00206     virtual const char* what() const throw ()
00207     {
00208 #ifdef NDEBBUG
00209         return str2char(sformat("SIS Protocol reception fail caused: STATUS=0x%04x (%d) ### MESSAGE='%s'",
m_status, m_status, m_message.c_str()));
00210 #else
00211         const char* ex = str2char(sformat("SIS Protocol reception fail caused: STATUS=0x%04x (%d) ###
MESSAGE='%s'", m_status, m_status, m_message.c_str()));
00212         OutputDebugStringA((LPCSTR)ex);
00213         return ex;
00214 #endif
00215     }
00216 };
00217
00221 class SISProtocol::ExceptionSISError : public
SISProtocol::ExceptionGeneric
00222 {

```

```

00223 public:
00224     ExceptionSISError(
00225         int _status,
00226         int _code,
00227         const std::string _bytestream,
00228         bool _warning = false) :
00229
00230         ExceptionGeneric(_status, std::string(), _warning),
00231         m_errorcode(_code),
00232         m_bytestream(_bytestream)
00233     {}
00234     ~ExceptionSISError() throw() {}
00235
00236     virtual const char* what() const throw ()
00237     {
00238 #ifdef NDEBUG
00239         return str2char(sformat("(Return code: %d) SIS Protocol Error code returned has been received:
00240                                0x%04X.\nOriginal Telegram bytestream: %s", m_status, m_errorcode,
00241                                m_bytestream.c_str()));
00242 #else
00243         const char* ex = str2char(sformat("(Return code: %d) SIS Protocol Error code returned has been
00244         received: 0x%04X.\nOriginal Telegram bytestream: %s", m_status, m_errorcode,
00245         m_bytestream.c_str()));
00246         OutputDebugStringA(LPCSTR)ex);
00247         return ex;
00248 #endif
00249     }
00250     int get_errorcode() { return m_errorcode; }
00251
00252 protected:
00253     int m_errorcode;
00254     std::string m_bytestream;
00255 };
00256 #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);
```

```

00045         Size = _data.size();
00046     }
00047
00048     Data(UINT8 _data)
00052     {
00053         clear();
00054
00055         operator<<(_data);
00056     }
00057
00058     Data(UINT16 _data)
00062     {
00063         clear();
00064
00065         operator<<(_data & 0xFF);
00066         operator<<((_data & 0xFF00) >> 8);
00067     }
00068
00069     Data(UINT32 _data)
00073     {
00074         clear();
00075
00076         operator<<(_data & 0xFF);
00077         operator<<((_data & 0xFF00) >> 8);
00078         operator<<((_data & 0xFF0000) >> 16);
00079         operator<<((_data & 0xFF000000) >> 24);
00080     }
00081
00082     Data(UINT64 _data)
00086     {
00087         clear();
00088
00089         operator<<(_data & 0xFF);
00090         operator<<((_data & 0xFF00) >> 8);
00091         operator<<((_data & 0xFF0000) >> 16);
00092         operator<<((_data & 0xFF000000) >> 24);
00093         operator<<((_data & 0xFF00000000) >> 32);
00094         operator<<((_data & 0xFF0000000000) >> 40);
00095         operator<<((_data & 0xFF000000000000) >> 48);
00096         operator<<((_data & 0xFF00000000000000) >> 54);
00097     }
00098
00099     BYTE at(UINT32 _idx)
00105     {
00106         return Bytes[_idx];
00107     }
00108
00109     std::vector<BYTE> toVector()
00113     {
00114         std::vector<BYTE> out;
00115
00116         for (int i = 0; i < Size; i++)
00117             out.push_back(Bytes[i]);
00118
00119         return out;
00120     }
00121
00122     UINT64 toUINT64()
00126     {
00127         UINT64 out = 0;
00128
00129         for (int i = 0; i < std::min<size_t>(Size, 8); i++)
00130             out |= Bytes[i] << (i * 8);
00131
00132         return out;
00133     }
00134
00135     UINT32 toUINT32()
00139     {
00140         UINT32 out = 0;
00141
00142         for (int i = 0; i < std::min<size_t>(Size, 4); i++)
00143             out |= Bytes[i] << (i * 8);
00144
00145         return out;
00146     }
00147
00148     UINT16 toUINT16()
00152     {
00153         UINT16 out = 0;
00154
00155         for (int i = 0; i < std::min<size_t>(Size, 2); i++)
00156             out |= Bytes[i] << (i * 8);
00157
00158         return out;
00159     }
00160

```

```

00161
00165     UINT8 toUINT8()
00166     {
00167         return toBYTE();
00168     }
00169
00173     BYTE toBYTE()
00174     {
00175         return Size > 0 ? (BYTE)Bytes[0] : (BYTE)0;
00176     }
00177
00179     void clear()
00180     {
00181         memset(Bytes, 0, sizeof(Bytes));
00182         Size = 0;
00183     }
00184
00190     Data& operator<<(const BYTE& rhs)
00191     {
00192         Bytes[Size++] = rhs;
00193         return *this;
00194     }
00195
00199     size_t get_size() { return Size; }
00200
00204     void set_size(size_t _size) { Size = _size; }
00205
00206 } Data;
00207
00208
00210 typedef struct Bytestream
00211 {
00213     BYTE Bytes[TGM_SIZEMAX];
00214
00216     Bytestream() { clear(); }
00217
00219     void clear()
00220     {
00221         memset(Bytes, 0, sizeof(Bytes));
00222     }
00223 } Bytestream;
00224
00225
00227 template <class THeader, class TPayload>
00228 union Map
00229 {
00230 public:
00232     Bytestream Raw;
00233
00235 #pragma pack(push,1)
00236     struct Mapping
00237     {
00239         THeader    Header;
00241         TPayload    Payload;
00242
00247         Mapping(THeader& _header, TPayload _payload) :
00248             Header(_header),
00249             Payload(_payload)
00250         {};
00251     } Mapping;
00252 #pragma pack(pop)
00253
00258     Map(THeader& _header = THeader(), TPayload& _payload = TPayload()) :
00259         Mapping(_header, _payload)
00260     {};
00262     ~Map() {};
00263
00268     void set(THeader& _header, TPayload& _payload)
00269     {
00270         Mapping = Mapping(_header, _payload);
00271     }
00272 };
00273
00274
00275 #pragma pack(push,1)
00276 typedef struct Header
00277 {
00280     BYTE StZ = 0x02;
00281
00285     BYTE CS;
00286
00289     BYTE DatL;
00290
00293     BYTE DatLW;
00294
00296     BYTE Cntrl;
00297

```

```

00315         BYTE Service;
00316
00321         BYTE AdrS;
00322
00333         BYTE AdrE;
00334
00343         Header(BYTE _addr_master = 0, BYTE _addr_slave = 0, BYTE _service = 0,
TGM::Bitfields::HeaderControl _cntrl =
TGM::Bitfields::HeaderControl()) :
00344             StZ(0x02),
00345             CS(0),
00346             DatL(get_size()),
00347             DatLW(get_size()),
00348             Cntrl(_cntrl.Value),
00349             Service(_service),
00350             AdrS(_addr_master),
00351             AdrE(_addr_slave)
00352     {}
00353
00359     BYTE get_sum(bool exclude_cs = true)
00360     {
00361         BYTE res = StZ + DatL + DatLW + Cntrl + Service + AdrS + AdrE;
00362
00363         if (!exclude_cs) res += CS;
00364
00365         return res;
00366     }
00367
00371     size_t get_size() { return sizeof(*this); }
00372
00377     inline void set_DatL(size_t _payload_len) { DatL = DatLW = (BYTE)_payload_len; }
00378
00382     inline size_t get_DatL() { return DatL; }
00383
00390     void calc_checksum(TGM::Bytestream * _payload)
00391     {
00392         // Sum of payload
00393         BYTE sum_of_payload = 0;
00394         for (int i = TGM_SIZE_HEADER; i < TGM_SIZE_HEADER + get_DatL(); i
00395 ++)
00396             sum_of_payload += (BYTE)_payload->Bytes[i];
00397
00398         // Calc difference
00399         BYTE diff_cs = get_sum() + sum_of_payload;
00400
00401         // Calc negation and assign to checksum (Byte 1)
00402         CS = (BYTE)0 - diff_cs;
00403     } Header;
00404 #pragma pack(pop)
00405
00406
00407 #pragma pack(push,1)
00408     typedef struct HeaderExt : Header
00409     {
00410     {
00414         BYTE AdrES1;
00415
00417         BYTE AdrES2;
00418
00420         BYTE AdrES3;
00421
00423         BYTE AdrES4;
00424
00426         BYTE AdrES5;
00427
00429         BYTE AdrES6;
00430
00432         BYTE AdrES7;
00433
00436         BYTE PaketN;
00437
00438     } HeaderExt;
00439 #pragma pack(pop)
00440
00441
00443     namespace Commands
00444     {
00445
00446 #pragma pack(push,1)
00447         typedef struct Subservice
00448         {
00452             BYTE RecipientAddr;
00454             BYTE ServiceNumber;
00456             Data Bytes;
00457
00463             Subservice(
00464                 BYTE _addr = 0,

```

```

00465         BYTE _subservice = 0,
00466         Data _data = Data() ) :
00467     RecipientAddr(_addr),
00468     ServiceNumber(_subservice),
00469     Bytes(_data)
00470 {}
00471
00473     void clear() { RecipientAddr = ServiceNumber = 0; }
00474
00478     size_t get_head_size() { return 2; }
00479
00483     size_t get_size() { return get_head_size() + Bytes.get_size(); }
00484
00485 } Subservice;
00486 #pragma pack(pop)
00487
00488 #pragma pack(push,1)
00489 typedef struct SercosParam
00490 {
00494     BYTE Control;
00495
00499     BYTE UnitAddr;
00500
00501     BYTE ParamType;
00502
00505     USHORT ParamNum;
00506
00508     Data Bytes;
00509
00516     SercosParam(
00517         TGM::Bitfields::SercosParamControl _control =
00518         TGM::Bitfields::SercosParamControl(),
00519         BYTE _unit_addr = 0,
00520         TGM::Bitfields::SercosParamIdent _param_ident =
00521         TGM::Bitfields::SercosParamIdent(),
00522         TGM::Data _data = Data() ) :
00523         Control(_control.Value),
00524         UnitAddr(_unit_addr),
00525         ParamType(0),
00526         ParamNum(_param_ident.Value),
00527         Bytes(_data)
00528     {}
00529
00530     void clear()
00531     {
00532         Control = 0;
00533         UnitAddr = 0;
00534         ParamType = 0;
00535         ParamNum = 0;
00536         Bytes.clear();
00537     }
00541
00542     size_t get_head_size() { return 5; }
00543
00546     size_t get_size() { return get_head_size() + Bytes.get_size(); }
00547
00548 } SercosParam;
00549 #pragma pack(pop)
00550
00551 #pragma pack(push,1)
00552 typedef struct SercosList
00553 {
00557     BYTE Control;
00558
00562     BYTE UnitAddr;
00563
00564     BYTE ParamType;
00565
00568     USHORT ParamNum;
00569
00572     USHORT ListOffset;
00573
00576     USHORT SegmentSize;
00577
00579     Data Bytes;
00580
00589     SercosList(
00590         TGM::Bitfields::SercosParamControl _ControlByte =
00591         TGM::Bitfields::SercosParamControl(),
00592         BYTE _unit_addr = 0,
00593         TGM::Bitfields::SercosParamIdent _ParamIdent =
00594         TGM::Bitfields::SercosParamIdent(),
00595         USHORT _ListOffset = 0,
00596         USHORT _SegmentSize = 0,
00597         TGM::Data _PayloadData = Data() ) :

```

```

00596
00597         Control(_ControlByte.Value),
00598         UnitAddr(_unit_addr),
00599         ParamType(0),
00600         ParamNum(_ParamIdent.Value),
00601         ListOffset(_ListOffset),
00602         SegmentSize(_SegmentSize),
00603         Bytes(_PayloadData)
00604     {}
00605
00606     void clear()
00607     {
00608         Control = UnitAddr = ParamNum = ListOffset = SegmentSize = 0;
00609         Bytes.clear();
00610     }
00611
00612     size_t get_head_size() { return 9; }
00613
00614     size_t get_size() { return get_head_size() + Bytes.get_size(); }
00615 } SercosList;
00616 #pragma pack(pop)
00617
00618 namespace Reactions
00619 {
00620     #pragma pack(push,1)
00621     typedef struct Subservice
00622     {
00623         BYTE    Status;
00624
00625         BYTE    RecipientAddr;
00626
00627         BYTE    ServiceNumber;
00628
00629         union
00630         {
00631             Data    Bytes;
00632             BYTE    Error;
00633         };
00634
00635         Subservice() :
00636             Status(1),
00637             RecipientAddr(0),
00638             ServiceNumber(0),
00639             Error(0)
00640         {}
00641
00642         void clear()
00643         {
00644             Status = 1;
00645             RecipientAddr = ServiceNumber = 0;
00646             Bytes.clear();
00647         }
00648
00649         size_t get_head_size() { return 3; }
00650
00651         size_t get_size() { return get_head_size() + Bytes.get_size(); }
00652     } Subservice;
00653 #pragma pack(pop)
00654
00655     #pragma pack(push,1)
00656     typedef struct SercosParam
00657     {
00658         BYTE Status;
00659
00660         BYTE Control;
00661
00662         BYTE UnitAddr;
00663
00664         union
00665         {
00666             Data    Bytes;
00667             USHORT  Error;
00668         };
00669
00670         SercosParam() :
00671             Status(1),
00672             Control(0),
00673             UnitAddr(0),
00674             Bytes(TGM::Data())
00675         {}
00676     }
00677 }

```

```

00717         void clear()
00718         {
00719             Status = 1;
00720             Control = UnitAddr = 0;
00721             Bytes.clear();
00722         }
00723
00727         size_t get_head_size() { return 3; }
00728
00732         size_t get_size() { return get_head_size() + Bytes.get_size(); }
00733
00734     } SercosParam;
00735 #pragma pack(pop)
00736
00737 #pragma pack(push,1)
00738 typedef struct SercosList
00739 {
00741     BYTE Status;
00743
00744     BYTE Control;
00746
00747     BYTE UnitAddr;
00751
00752     union
00753     {
00756         Data Bytes;
00757         USHORT Error;
00758     };
00759
00761     SercosList() :
00762         Status(1),
00763         Control(0),
00764         UnitAddr(0),
00765         Bytes(TGM::Data())
00766     {}
00767
00769     void clear()
00770     {
00771         Status = 1;
00772         Control = UnitAddr = 0;
00773         Bytes.clear();
00774     }
00775
00779     size_t get_head_size() { return 3; }
00780
00784     size_t get_size() { return get_head_size() + Bytes.get_size(); }
00785
00786 } SercosList;
00787 #pragma pack(pop)
00788 }
00789 }
00790
00791 #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, [TGM::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

```

00001
00005 #ifndef _TELEGRAMS_BITFIELDS_H_
00006 #define _TELEGRAMS_BITFIELDS_H_
00007
00008 #include <Windows.h>
00009 #include <vector>
00010
00011
00013 namespace TGM
00014 {
00016     enum HeaderType : BYTE {
00018         TypeCommand,
00020         TypeReaction
00021     };
00022
00024     enum SercosParamVar : BYTE {
00026         SercosParamS,
00028         SercosParamP
00029     };
00030
00033     enum SercosDatablock : BYTE {
00035         Datablock_ChannelNotActive,
00037         Datablock_IdentNumber,
00039         Datablock_Name,
00042         Datablock_Attribute,
00044         Datablock_Unit,
00046         Datablock_Minval,
00048         Datablock_Maxval,
00050         Datablock_OperationData
00051     };
00052
00058     enum SercosCommandrequest : BYTE {
00059         Commandrequest_NotSet = 0x0,
00060         Commandrequest_Cancel = 0x1,
00061         Commandrequest_Set = 0x3
00062     };
00063
00069     enum SercosCommandstatus : BYTE {
00070         Commandstatus_NotSet = 0x0,
00071         Commandstatus_OK = 0x3,
00072         Commandstatus_Canceled = 0x5,
00073         Commandstatus_Busy = 0x7,
00074         Commandstatus_Error = 0xF
00075     };
00076
00081     enum SercosTxProgress : BYTE {
00083         TxProgress_InProgress,
00085         TxProgress_Final
00086     };
00087
00090     enum SercosDatalen : UINT32 {
00091         Datalen_Res1 = 0b000,
00092         Datalen_2ByteParam = 0b001,
00093         Datalen_4ByteParam = 0b010,
00094         Datalen_8ByteParam = 0b011,
00095         Datalen_1ByteList = 0b100,
00096         Datalen_2ByteList = 0b101,
00097         Datalen_4ByteList = 0b110,
00098         Datalen_8ByteList = 0b111,
00099     };
00100
00101
00103 namespace Bitfields
00104 {
00106     typedef struct HeaderControl
00107     {

```

```

00108         union
00109         {
00110             struct Bits
00111             {
00113                 BYTE NumSubAddresses : 3;
00114
00118                 BYTE NumRunningTgm : 1;
00119
00121                 HeaderType Type : 1;
00122
00129                 BYTE StatusReactionTgm : 3;
00130
00136                 Bits(HeaderType type = TypeCommand) :
00137                     NumSubAddresses(0),
00138                     NumRunningTgm(0),
00139                     Type(type),
00140                     StatusReactionTgm(0)
00141             {}
00142             } Bits;
00143
00145             BYTE Value;
00146         };
00147
00153         HeaderControl(HeaderType type = TypeCommand) :
00154         Bits(TypeCommand) {}
00155     } HeaderControl;
00156
00159     typedef struct SercosParamControl
00160     {
00161         union
00162         {
00163             struct Bits
00164             {
00165                 BYTE res1 : 1;
00166                 BYTE res2 : 1;
00167
00171                 SercosTxProgress TxProgress : 1;
00172
00174                 SercosDatablock Datablock : 3;
00175
00176                 BYTE res6 : 1;
00177                 BYTE res7 : 1;
00178
00184                 Bits(SercosDatablock datablock =
00185                 Datablock_OperationData) :
00186                     res1(0), res2(0), TxProgress(TxProgress_Final), Datablock(datablock
00187                     ), res6(0), res7(0)
00188             {}
00189             } Bits;
00190
00191             BYTE Value;
00192         };
00193
00196         SercosParamControl(SercosDatablock datablock =
00197         Datablock_OperationData) : Bits(datablock) {}
00198
00201         SercosParamControl(BYTE value) : Value(value) {}
00202     } SercosParamControl;
00203
00204
00206     typedef struct SercosParamIdent
00207     {
00208         union
00209         {
00210             struct Bits
00211             {
00213                 USHORT ParamNumber : 12;
00214
00216                 USHORT ParamSet : 3;
00217
00221                 USHORT ParamVariant : 1;
00222
00227                 Bits(SercosParamVar param_variant =
00228                 TGM::SercosParamS, USHORT param_num = 0) :
00229                     ParamNumber(param_num),
00230                     ParamSet(0),
00231                     ParamVariant(param_variant)
00232             {}
00233             } Bits;
00234
00235             USHORT Value;
00236         };
00237
00241         SercosParamIdent(SercosParamVar param_variant =
00242         TGM::SercosParamS, USHORT param_num = 0) :
00243             Bits(param_variant, param_num)

```

```

00243     {}
00244     } SercosParamIdent;
00245
00246
00250     typedef struct SercosParamAttribute
00251     {
00252         union
00253         {
00254             struct Bits
00255             {
00259                 UINT32 ConversionFactor : 16;
00260
00263                 SercosDatalen DataLen : 3;
00264
00268                 UINT32 DataFunction : 1;
00269
00271                 UINT32 DataDisplay : 3;
00272
00274                 UINT32 res5 : 1;
00275
00279                 UINT32 ScaleFactor : 4;
00280
00282                 UINT32 is_writeonly_phase2 : 1;
00283
00285                 UINT32 is_writeonly_phase3 : 1;
00286
00288                 UINT32 is_writeonly_phase4 : 1;
00289
00291                 UINT32 res10 : 1;
00292
00294                 Bits() :
00295                     ConversionFactor(0),
00296                     DataLen(DataLen_2ByteParam),
00297                     DataFunction(0),
00298                     DataDisplay(0),
00299                     res5(0),
00300                     ScaleFactor(0),
00301                     is_writeonly_phase2(0),
00302                     is_writeonly_phase3(0),
00303                     is_writeonly_phase4(0),
00304                     res10(0)
00305                 {}
00306             } Bits;
00307
00309             UINT32 Value;
00310         };
00311
00315         SercosParamAttribute(UINT32 _value = 0) : Value(_value) {}
00316     } SercosParamAttribute;
00317 }
00318 }
00319
00320
00321 #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 Indrdrive 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 Indrdrive device.

Attention

Baudrate selection is not support. Default of 19200 Bits/s is used.

Remarks

This function is exported to the Indrdrive 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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
in	<i>ID_comport</i>	(Optional) Communication port. Default: L"COM1".
in	<i>ID_combaudrate</i>	(Optional) Communication baudrate in [Bits/s]. Default: 19200 Bits/s.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
in	<i>ID_max_accel</i>	(Optional) Maximum allowed acceleration in [rad/s ²]. Default: 10000 rad/s ² .
in	<i>ID_max_jerk</i>	(Optional) Maximum allowed jerk in [rad/s ³]. Default: 1000 rad/s ³ .
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
in	<i>ID_speeds</i>	Sequencer speed list in [1/min]. Rotation directions are defined by the sign of each element: <ul style="list-style-type: none"> • Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.
in	<i>ID_accels</i>	Sequencer acceleration list in [rad/s ²]. Generated by Doxygen
in	<i>ID_jerks</i>	Sequencer jerk list in [rad/s ³].
in	<i>ID_delays</i>	Delay list representing delay between each kinematic step in [cs].
in	<i>ID_set_length</i>	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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_status</i>	Pointer that provides the respective information: <ul style="list-style-type: none"> • 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.
		<ul style="list-style-type: none"> • 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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_max_accel</i>	(Optional) Maximum allowed acceleration in [rad/s ²]. Default: 10000 rad/s ² .
out	<i>ID_max_jerk</i>	(Optional) Maximum allowed jerk in [rad/s ³]. Default: 1000 rad/s ³ .
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_speed</i>	Target speed in [1/min]. Sign represents the rotation direction: <ul style="list-style-type: none"> • Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.
out	<i>ID_accel</i>	Target acceleration in [rad/s ²].
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_drvmode</i>	Pointer that provides the respective information: <ul style="list-style-type: none"> • 0 - Drive Mode not supported, • 1 - "Sequencer" drive mode active, • 2 - "Speed Control" drive mode active.

Parameters

out	<i>ID_err</i>	(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(<path_to_DLL>, 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(indreref, ctypes.byref(opstate), ctypes.byref(indra_error))
```

.

Parameters

in	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
----	---------------	--

Parameters

out	<i>ID_opstate</i>	Pointer that provides the respective information: <ul style="list-style-type: none"> • Bit 0-1: Operation state <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – 0: Drive not halted – 1: Drive Halt is active and axis is in standstill • Bit 3: Drive error <ul style="list-style-type: none"> – 0: No error – 1: Drive error present.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_speed</i>	Pointer that provides the speed information as double Value in [1/min]. Sign represents the rotation direction:
		<ul style="list-style-type: none"> • Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_diagnostic_msg</i>	Pointer that provides the diagnostic message string.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_diagnostic_num</i>	Pointer that provides the diagnostic number.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_err</i>	(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

```

00001
00004 #include "Wrapper.h"
00005
00006
00007 DLLEXPORT SISProtocol * DLLCALLCONV init()
00008 {
00009     SISProtocol * protocol = new SISProtocol();
00010     return protocol;
00011 }
00012
00013
00014 DLLEXPORT int32_t DLLCALLCONV open(SISProtocol* ID_ref, const wchar_t*
ID_comport, uint32_t ID_combaudrate, ErrHandle ID_err)
00015 {
00016     if (!dynamic_cast<SISProtocol*>(ID_ref))
00017         // Return error for wrong reference
00018         return set_error(
00019             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00020             Err_Invalid_Pointer);
00021
00022     try
00023     {
00024         ID_ref->open(ID_comport);
00025         return Err_NoError;
00026     }
00027     catch (SISProtocol::ExceptionGeneric &ex)
00028     {
00029         return set_error(ID_err, char2str(ex.what()),
Err_Block_OpenByCOM);
00030     }
00031     catch (CSerial::ExceptionGeneric &ex)
00032     {
00033         return set_error(ID_err, char2str(ex.what()),
Err_Block_OpenByCOM);
00034     }
00035 }
00036
00037
00038 DLLEXPORT int32_t DLLCALLCONV close(SISProtocol* ID_ref,
ErrHandle ID_err)
00039 {
00040     if (!dynamic_cast<SISProtocol*>(ID_ref))
00041         // Return error for wrong reference
00042         return set_error(
00043             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00044             Err_Invalid_Pointer);
00045
00046     try
00047     {
00048         ID_ref->close();
00049
00050         delete ID_ref;
00051         ID_ref = NULL;
00052         return Err_NoError;
00053     }
00054     catch (SISProtocol::ExceptionGeneric &ex)
00055     {
00056         return set_error(ID_err, char2str(ex.what()),
Err_Block_Close);
00057     }
00058     catch (CSerial::ExceptionGeneric &ex)
00059     {
00060         return set_error(ID_err, char2str(ex.what()), Err_Block_Close);
00061     }
00062 }
00063
00064
00065 DLLEXPORT int32_t DLLCALLCONV sequencer_activate(
SISProtocol * ID_ref, ErrHandle ID_err)
00066 {
00067     if (!dynamic_cast<SISProtocol*>(ID_ref))
00068         // Return error for wrong reference
00069         return set_error(
00070             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00071             Err_Invalid_Pointer);
00072
00073     try
00074     {
00075         // Change mode
00076         change_opmode(ID_ref, DRIVEMODE_SEQUENCER);
00077
00078         return Err_NoError;
00079     }
00080     catch (SISProtocol::ExceptionGeneric &ex)

```

```

00081     {
00082         return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqInit);
00083     }
00084     catch (CSerial::ExceptionGeneric &ex)
00085     {
00086         return set_error(ID_err, char2str(ex.what()), Err_Block_SeqInit);
00087     }
00088 }
00089
00090
00091 DLLEXPORT int32_t DLLCALLCONV sequencer_init(
SISProtocol * ID_ref, double_t ID_max_accel, double_t ID_max_jerk,
ErrHandle ID_err)
00092 {
00093     if (!dynamic_cast<SISProtocol*>(ID_ref))
00094         // Return error for wrong reference
00095         return set_error(
ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00096
00097     try
00098     {
00099         // Set required units (preferred scaling, rotary scaling, [rpm])
00100         change_units(ID_ref);
00101
00102         // Max Acceleration (S-0-0138)
00103         ID_ref->write_parameter(TGM::SercosParamS, 138, ID_max_accel);
00104
00105         // Max Jerk (S-0-0349)
00106         ID_ref->write_parameter(TGM::SercosParamS, 349, ID_max_jerk);
00107
00108         // SPS Global Register G1 (P-0-1371) - Reset Read Trigger
00109         ID_ref->write_parameter(TGM::SercosParamP, 1371,
static_cast<uint32_t>(0));
00110
00111         // SPS Global Register G2 (P-0-1372) - Reset Sequencer Trigger
00112         ID_ref->write_parameter(TGM::SercosParamP, 1372,
static_cast<uint32_t>(0));
00113
00114         return Err_NoError;
00115     }
00116     catch (SISProtocol::ExceptionGeneric &ex)
00117     {
00118         return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqInit);
00119     }
00120     catch (CSerial::ExceptionGeneric &ex)
00121     {
00122         return set_error(ID_err, char2str(ex.what()), Err_Block_SeqInit);
00123     }
00124 }
00125
00126
00127
00128
00129 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)
00130 {
00131     if (!dynamic_cast<SISProtocol*>(ID_ref))
00132         // Return error for wrong reference
00133         return set_error(
ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00134
00135     try
00136     {
00137         ID_ref->write_listelm(TGM::SercosParamP, 4019, 1,
static_cast<uint32_t>(0b10000100));
00138
00139         for (uint16_t i = 0; i < ID_set_length; i++)
00140         {
00141             // Speed in min^-1 (P-0-4007)
00142             ID_ref->write_listelm(TGM::SercosParamP, 4007, i + 1, abs(
ID_speeds[i]));
00143
00144             // Acceleration in rad/s^2 (P-0-4008)
00145             ID_ref->write_listelm(TGM::SercosParamP, 4008, i + 1, ID_accels[i
]);
00146
00147             // Deceleration in rad/s^2 (P-0-4063)
00148             ID_ref->write_listelm(TGM::SercosParamP, 4063, i + 1, ID_accels[i
]);
00149
00150             // Jerk in rad/s^3 (P-0-4009)
00151             ID_ref->write_listelm(TGM::SercosParamP, 4009, i + 1, ID_jerks[i
]);
00152
00153         }
00154     }

```

```

00155         // Mode (P-0-4019)
00156         ID_ref->write_listelm(TGM::SercosParamP, 4019, i + 2,
static_cast<uint32_t>(std::sgn<double_t>(ID_speeds[i]) == 1 ? 0b10000100 : 0b10001000));
00157
00158         // Pos (P-0-4006)
00159         ID_ref->write_listelm(TGM::SercosParamP, 4006, i + 1,
static_cast<uint64_t>(0));
00160
00161         // Wait (P-0-4018)
00162         ID_ref->write_listelm(TGM::SercosParamP, 4018, i + 1,
static_cast<uint64_t>(0));
00163
00164         // Delay (P-0-4063)
00165         ID_ref->write_listelm(TGM::SercosParamP, 4063, i + 1,
static_cast<uint64_t>(0));
00166
00167         // Timers in cs (P-0-1389)
00168         ID_ref->write_listelm(TGM::SercosParamP, 1389, i + 1, ID_delays[i
]);
00169     }
00170
00171     // Time triggers for cam (P-0-1370)
00172     ID_ref->write_parameter(TGM::SercosParamP, 1370,
static_cast<uint32_t>(ID_set_length));
00173
00174     return Err_NoError;
00175 }
00176 catch (SISProtocol::ExceptionGeneric &ex)
00177 {
00178     return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
00179 }
00180 catch (CSerial::ExceptionGeneric &ex)
00181 {
00182     return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
00183 }
00184 }
00185
00186
00187 DLLEXPORT int32_t DLLCALLCONV sequencer_softtrigger(
SISProtocol * ID_ref, ErrHandle ID_err)
00188 {
00189     if (!dynamic_cast<SISProtocol*>(ID_ref))
00190         // Return error for wrong reference
00191         return set_error(
00192             ID_err, sprintf("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00193
00194     try
00195     {
00196         uint32_t qb0stat;
00197         int iterations;
00198
00199         // FEED DATA:
00200
00201         // SPS Global Register G1 (P-0-1371) - Reset Read Trigger
00202         ID_ref->write_parameter(TGM::SercosParamP, 1371,
static_cast<uint64_t>(0));
00203
00204         // SPS Global Register G2 (P-0-1372) - Reset Sequencer Trigger
00205         ID_ref->write_parameter(TGM::SercosParamP, 1372,
static_cast<uint64_t>(0));
00206
00207         // READ CUSTOM DATA ...
00208
00209         iterations = 0;
00210         do
00211         {
00212             // Check status (P-0-1410)
00213             ID_ref->read_parameter(TGM::SercosParamP, 1410, qb0stat);
00214
00215             if (iterations > 300) return set_error(ID_err, "RESULT_READ_OK was not set. Input
parameters cannot be accepted.", Err_Block_SeqWrite);
00216             } while (qb0stat & 0b100000 >> 5);
00217
00218         // SPS Global Register G1 (P-0-1371) - Set Read Trigger
00219         ID_ref->write_parameter(TGM::SercosParamP, 1371,
static_cast<uint64_t>(1));
00220
00221
00222         // TRIGGER:
00223
00224         // SPS Global Register G2 (P-0-1372) - Reset Sequencer Trigger
00225         ID_ref->write_parameter(TGM::SercosParamP, 1372,
static_cast<uint64_t>(0));
00226
00227

```

```

00228         // SPS Global Register G2 (P-0-1372) - Set Sequencer Trigger
00229         ID_ref->write_parameter(TGM::SercosParamP, 1372,
static_cast<uint64_t>(1));
00230
00231
00232         iterations = 0;
00233         do
00234         {
00235             // Check status (P-0-1410)
00236             ID_ref->read_parameter(TGM::SercosParamP, 1410, qb0stat);
00237
00238             if (iterations > 300) return set_error(ID_err, "bDriveStarted was not set. Input
parameters cannot be accepted.", Err_Block_SeqWrite);
00239             } while (qb0stat & 0b1000 >> 3);
00240
00241             return Err_NoError;
00242         }
00243         catch (SISProtocol::ExceptionGeneric &ex)
00244         {
00245             return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
00246         }
00247         catch (CSerial::ExceptionGeneric &ex)
00248         {
00249             return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
00250         }
00251     }
00252 }
00253 DLLEXPORT int32_t DLLCALLCONV sequencer_hardtrigger(
SISProtocol * ID_ref, ErrHandle ID_err)
00254 {
00255     if (!dynamic_cast<SISProtocol*>(ID_ref))
00256         // Return error for wrong reference
00257         return set_error(
ID_err, sprintf("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00258
00259     try
00260     {
00261         {
00262             uint32_t qb0stat;
00263
00264             // FEED DATA:
00265
00266             // SPS Global Register G1 (P-0-1371) - Reset Read Trigger
00267             ID_ref->write_parameter(TGM::SercosParamP, 1371,
static_cast<uint64_t>(0));
00268
00269             // SPS Global Register G2 (P-0-1372) - Reset Sequencer Trigger
00270             ID_ref->write_parameter(TGM::SercosParamP, 1372,
static_cast<uint64_t>(0));
00271
00272             // SPS Global Register G1 (P-0-1371) - Set Read Trigger
00273             ID_ref->write_parameter(TGM::SercosParamP, 1371,
static_cast<uint64_t>(1));
00274
00275             // Check status (P-0-1410)
00276             ID_ref->read_parameter(TGM::SercosParamP, 1410, qb0stat); // TODO:
Check RESULT_READ_OK bit (0b100000)
00277
00278             return Err_NoError;
00279         }
00280         catch (SISProtocol::ExceptionGeneric &ex)
00281         {
00282             return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
00283         }
00284         catch (CSerial::ExceptionGeneric &ex)
00285         {
00286             return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
00287         }
00288     }
00289 }
00290 }
00291 }
00292
00293
00294 DLLEXPORT int32_t DLLCALLCONV sequencer_getstatus(
SISProtocol * ID_ref, uint16_t * ID_status, ErrHandle ID_err)
00295 {
00296     if (!dynamic_cast<SISProtocol*>(ID_ref))
00297         // Return error for wrong reference
00298         return set_error(
ID_err, sprintf("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00299
00300     try
00301     {
00302         {
00303

```

```

00304         uint32_t plc_status;
00305         // PLC register / Status value (P-0-1410)
00306         ID_ref->read_parameter(TGM::SercosParamP, 1410, plc_status);
00307
00308         *ID_status = static_cast<uint16_t>(plc_status & 0xFFFF);
00309
00310         return Err_NoError;
00311     }
00312     catch (SISProtocol::ExceptionGeneric &ex)
00313     {
00314         return set_error(ID_err, char2str(ex.what()),
00315 Err_Block_SeqWrite);
00316     }
00317     catch (CSerial::ExceptionGeneric &ex)
00318     {
00319         return set_error(ID_err, char2str(ex.what()),
00320 Err_Block_SeqWrite);
00321     }
00322 }
00323
00324 DLLEXPORT int32_t DLLCALLCONV speedcontrol_activate(
00325     SISProtocol * ID_ref, ErrHandle ID_err)
00326 {
00327     if (!dynamic_cast<SISProtocol*>(ID_ref))
00328         // Return error for wrong reference
00329         return set_error(
00330             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00331             Err_Invalid_Pointer);
00332
00333     try
00334     {
00335         // Change mode
00336         change_opmode(ID_ref, DRIVEMODE_SPEEDCONTROL);
00337
00338         return Err_NoError;
00339     }
00340     catch (SISProtocol::ExceptionGeneric &ex)
00341     {
00342         return set_error(ID_err, char2str(ex.what()),
00343 Err_Block_VelCInit);
00344     }
00345     catch (CSerial::ExceptionGeneric &ex)
00346     {
00347         return set_error(ID_err, char2str(ex.what()),
00348 Err_Block_VelCInit);
00349     }
00350 }
00351
00352 DLLEXPORT int32_t DLLCALLCONV speedcontrol_init(
00353     SISProtocol * ID_ref, double_t ID_max_accel, double_t ID_max_jerk,
00354     ErrHandle ID_err)
00355 {
00356     if (!dynamic_cast<SISProtocol*>(ID_ref))
00357         // Return error for wrong reference
00358         return set_error(
00359             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00360             Err_Invalid_Pointer);
00361
00362     try
00363     {
00364         // Set required units (preferred scaling, rotary scaling, [rpm])
00365         change_units(ID_ref);
00366
00367         // Max Acceleration (S-0-0138)
00368         ID_ref->write_parameter(TGM::SercosParamS, 138, ID_max_accel);
00369
00370         // Max Jerk (S-0-0349)
00371         ID_ref->write_parameter(TGM::SercosParamS, 349, ID_max_jerk);
00372
00373         return Err_NoError;
00374     }
00375     catch (SISProtocol::ExceptionGeneric &ex)
00376     {
00377         return set_error(ID_err, char2str(ex.what()),
00378 Err_Block_VelCInit);
00379     }
00380     catch (CSerial::ExceptionGeneric &ex)
00381     {
00382         return set_error(ID_err, char2str(ex.what()),
00383 Err_Block_VelCInit);
00384     }
00385 }
00386
00387 DLLEXPORT int32_t DLLCALLCONV speedcontrol_write(

```



```

        SISProtocol * ID_ref, double_t ID_speed, double_t ID_accel, ErrHandle ID_err)
00382 {
00383     if (!dynamic_cast<SISProtocol*>(ID_ref))
00384         // Return error for wrong reference
00385         return set_error(
00386             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00387             Err_Invalid_Pointer);
00388
00389     try
00390     {
00391         // Rotation direction - Positive ID_speed: Clockwise rotation, Negative ID_speed: Counter-clockwise
        rotation
00392         uint32_t rotmode = static_cast<uint32_t>((std::sgn<double_t>(ID_speed) == 1 ? 0 : 1) << 10);
00393         // Control Mode (P-0-1200)
00394         ID_ref->write_parameter(TGM::SercosParamP, 1200, rotmode);
00395
00396         // Acceleration in rad/s^2 (P-0-1203)
00397         ID_ref->write_parameter(TGM::SercosParamP, 1203, ID_accel);
00398
00399         // Speed in rpm (S-0-0036)
00400         ID_ref->write_parameter(TGM::SercosParamS, 36, abs(ID_speed));
00401
00402         return Err_NoError;
00403     }
00404     catch (SISProtocol::ExceptionGeneric &ex)
00405     {
00406         return set_error(ID_err, char2str(ex.what()),
        Err_Block_VelCWrite);
00407     }
00408     catch (CSerial::ExceptionGeneric &ex)
00409     {
00410         return set_error(ID_err, char2str(ex.what()),
        Err_Block_VelCWrite);
00411     }
00412 }
00413
00414
00415 DLLEXPORT int32_t DLLCALLCONV set_stdenvironment(
        SISProtocol * ID_ref, ErrHandle ID_err)
00416 {
00417     if (!dynamic_cast<SISProtocol*>(ID_ref))
00418         // Return error for wrong reference
00419         return set_error(
00420             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00421             Err_Invalid_Pointer);
00422
00423     try
00424     {
00425         change_units(ID_ref);
00426         change_language(ID_ref);
00427
00428         return Err_NoError;
00429     }
00430     catch (SISProtocol::ExceptionGeneric &ex)
00431     {
00432         return set_error(ID_err, char2str(ex.what()),
        Err_Block_GetStatus);
00433     }
00434     catch (CSerial::ExceptionGeneric &ex)
00435     {
00436         return set_error(ID_err, char2str(ex.what()),
        Err_Block_GetStatus);
00437     }
00438 }
00439
00440
00441 DLLEXPORT int32_t DLLCALLCONV get_drivemode(
        SISProtocol * ID_ref, uint32_t * ID_drvmode, ErrHandle ID_err)
00442 {
00443     if (!dynamic_cast<SISProtocol*>(ID_ref))
00444         // Return error for wrong reference
00445         return set_error(
00446             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00447             Err_Invalid_Pointer);
00448
00449     try
00450     {
00451         uint64_t curdrvmode;
00452         // Primary Operation Mode (S-0-0032)
00453         ID_ref->read_parameter(TGM::SercosParamS, 32, curdrvmode);
00454
00455         switch (curdrvmode)
00456         {
00457             case DRIVEMODE_SEQUENCER: // Drive Mode: Sequencer
00458                 *ID_drvmode = 1;
00459                 break;
00460             case DRIVEMODE_SPEEDCONTROL: // Drive Mode: Speed Control

```

```

00461         *ID_drvmode = 2;
00462         break;
00463     default: // Drive Mode not supported
00464         *ID_drvmode = 0;
00465         break;
00466     }
00467
00468     return Err_NoError;
00469 }
00470 catch (SISProtocol::ExceptionGeneric &ex)
00471 {
00472     return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00473 }
00474 catch (CSerial::ExceptionGeneric &ex)
00475 {
00476     return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00477 }
00478 }
00479
00480
00481 DLLEXPORT int32_t DLLCALLCONV get_opstate(
SISProtocol * ID_ref, uint8_t * ID_opstate, ErrHandle ID_err)
00482 {
00483     if (!dynamic_cast<SISProtocol*>(ID_ref))
00484         // Return error for wrong reference
00485         return set_error(
00486             ID_err, sprintf("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00487
00488     try
00489     {
00490         uint64_t curopstate;
00491         // Device control: Status word (P-0-0115)
00492         ID_ref->read_parameter(TGM::SercosParamP, 115, curopstate);
00493
00494         OPSTATE opstate(static_cast<uint16_t>(curopstate));
00495         *ID_opstate = opstate.Value;
00496
00497         return Err_NoError;
00498     }
00499     catch (SISProtocol::ExceptionGeneric &ex)
00500     {
00501         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00502     }
00503     catch (CSerial::ExceptionGeneric &ex)
00504     {
00505         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00506     }
00507 }
00508 }
00509
00510
00511 DLLEXPORT int32_t DLLCALLCONV get_speed(SISProtocol * ID_ref,
double_t * ID_speed, ErrHandle ID_err)
00512 {
00513     if (!dynamic_cast<SISProtocol*>(ID_ref))
00514         // Return error for wrong reference
00515         return set_error(
00516             ID_err, sprintf("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00517
00518     try
00519     {
00520         double_t speed;
00521         // Velocity feedback Value (S-0-0040)
00522         ID_ref->read_parameter(TGM::SercosParamS, 40, speed);
00523
00524         *ID_speed = speed;
00525
00526         return Err_NoError;
00527     }
00528     catch (SISProtocol::ExceptionGeneric &ex)
00529     {
00530         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00531     }
00532     catch (CSerial::ExceptionGeneric &ex)
00533     {
00534         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00535     }
00536 }
00537 }
00538
00539

```

```

00540 DLLEXPORT int32_t DLLCALLCONV get_diagnostic_msg(
    SISProtocol * ID_ref, char * ID_diagnostic_msg, ErrHandle ID_err)
00541 {
00542     if (!dynamic_cast<SISProtocol*>(ID_ref))
00543         // Return error for wrong reference
00544         return set_error(
00545             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00546             Err_Invalid_Pointer);
00547
00548     try
00549     {
00550         char msg[TGM_SIZE_MAX_PAYLOAD];
00551         // Diagnostic message (S-0-0095)
00552         ID_ref->read_parameter(TGM::SercosParamS, 95, msg);
00553
00554         strncpy(ID_diagnostic_msg, msg+4, TGM_SIZE_MAX_PAYLOAD-4);
00555
00556         return Err_NoError;
00557     }
00558     catch (SISProtocol::ExceptionGeneric &ex)
00559     {
00560         return set_error(ID_err, char2str(ex.what()),
00561             Err_Block_GetStatus);
00562     }
00563     catch (CSerial::ExceptionGeneric &ex)
00564     {
00565         return set_error(ID_err, char2str(ex.what()),
00566             Err_Block_GetStatus);
00567     }
00568 }
00569 DLLEXPORT int32_t DLLCALLCONV get_diagnostic_num(
    SISProtocol * ID_ref, uint32_t * ID_diagnostic_num, ErrHandle ID_err)
00570 {
00571     if (!dynamic_cast<SISProtocol*>(ID_ref))
00572         // Return error for wrong reference
00573         return set_error(
00574             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00575             Err_Invalid_Pointer);
00576
00577     try
00578     {
00579         UINT32 num;
00580         // Diagnostic number (S-0-0390)
00581         ID_ref->read_parameter(TGM::SercosParamS, 390, num);
00582
00583         *ID_diagnostic_num = num;
00584
00585         return Err_NoError;
00586     }
00587     catch (SISProtocol::ExceptionGeneric &ex)
00588     {
00589         return set_error(ID_err, char2str(ex.what()),
00590             Err_Block_GetStatus);
00591     }
00592     catch (CSerial::ExceptionGeneric &ex)
00593     {
00594         return set_error(ID_err, char2str(ex.what()),
00595             Err_Block_GetStatus);
00596     }
00597 }
00598 DLLEXPORT int32_t DLLCALLCONV clear_error(
    SISProtocol * ID_ref, ErrHandle ID_err)
00599 {
00600     if (!dynamic_cast<SISProtocol*>(ID_ref))
00601         // Return error for wrong reference
00602         return set_error(
00603             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00604             Err_Invalid_Pointer);
00605
00606     try
00607     {
00608         // Clear error (S-0-0099) // Command C0500
00609         ID_ref->execute_command(TGM::SercosParamS, 99);
00610
00611         return Err_NoError;
00612     }
00613     catch (SISProtocol::ExceptionGeneric &ex)
00614     {
00615         return set_error(ID_err, char2str(ex.what()),
00616             Err_Block_GetStatus);
00617     }
00618     catch (CSerial::ExceptionGeneric &ex)
00619     {

```

```

00619         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00620     }
00621 }
00622
00623
00624 void change_opmode(SISProtocol * ID_ref, const uint64_t opmode)
00625 {
00626     uint64_t curopmode;
00627     // Primary Operation Mode (S-0-0032)
00628     ID_ref->read_parameter(TGM::SercosParamS, 32, curopmode);
00629
00630     // Operation change will trigger flash operations that may cause limited life time
00631     // Thus, operation change should be mainly triggered if required only
00632     if (curopmode != opmode)
00633     {
00634         // Enter parameterization level 1 (S-0-0420) // Command C0400
00635         ID_ref->execute_command(TGM::SercosParamS, 420);
00636
00637         // Primary Operation Mode (S-0-0032)
00638         ID_ref->write_parameter(TGM::SercosParamS, 32, opmode);
00639
00640         // Leave parameterization level 1 (S-0-0422) // Command C0200
00641         ID_ref->execute_command(TGM::SercosParamS, 422);
00642     }
00643 }
00644
00645
00646 inline SPEEDUNITS get_units(SISProtocol * ID_ref)
00647 {
00648     uint64_t curunits;
00649     // Scaling of speed units (S-0-0044)
00650     ID_ref->read_parameter(TGM::SercosParamS, 44, curunits);
00651
00652     return SPEEDUNITS(static_cast<uint16_t>(curunits));
00653 }
00654
00655
00656 void change_units(SISProtocol * ID_ref)
00657 {
00658     SPEEDUNITS units = get_units(ID_ref);
00659     if (units.Bits.type_of_scaling == 0b010 && !units.Bits.automode && !units.
Bits.scale_units && !units.Bits.time_units && !units.Bits.data_rel) return;
00660
00661     // Set required units (preferred scaling, rotary scaling, [rpm])
00662     uint64_t scalingtype = 0b0000000000000010;
00663     // Velocity data scaling Type (S-0-0044)
00664     ID_ref->write_parameter(TGM::SercosParamS, 44, scalingtype);
00665 }
00666
00667
00668 inline void change_language(SISProtocol * ID_ref, const uint8_t lang_code)
00669 {
00670     // Language selection (S-0-0265):
00671     // * 0: German
00672     // * 1: English
00673     // * 2: French
00674     // * 3: Spanish
00675     // * 4: Italian
00676     ID_ref->write_parameter(TGM::SercosParamS, 265, (UINT32)lang_code);
00677 }

```

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=ErrHandle())`

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 to 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 to 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](#) file itself should be automatically 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 reference 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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
in	<i>ID_comport</i>	(Optional) Communication port. Default: L"COM1".
in	<i>ID_combaudrate</i>	(Optional) Communication baudrate in [Bits/s]. Default: 19200 Bits/s.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
in	<i>ID_max_accel</i>	(Optional) Maximum allowed acceleration in [rad/s ²]. Default: 10000 rad/s ² .
in	<i>ID_max_jerk</i>	(Optional) Maximum allowed jerk in [rad/s ³]. Default: 1000 rad/s ³ .
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
in	<i>ID_speeds</i>	Sequencer speed list in [1/min]. Rotation directions are defined by the sign of each element: <ul style="list-style-type: none"> • Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.
in	<i>ID_accels</i>	Sequencer acceleration list in [rad/s ²].
in	<i>ID_jerks</i>	Sequencer jerk list in [rad/s ³].
in	<i>ID_delays</i>	Delay list representing delay between each kinematic step in [cs].
in	<i>ID_set_length</i>	Length of the sequence (=number of elements of ID_speeds, ID_accels, etc).
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_status</i>	<p>Pointer that provides the respective information:</p> <ul style="list-style-type: none"> • 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.
		<ul style="list-style-type: none"> • 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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_max_accel</i>	(Optional) Maximum allowed acceleration in [rad/s ²]. Default: 10000 rad/s ² .
out	<i>ID_max_jerk</i>	(Optional) Maximum allowed jerk in [rad/s ³]. Default: 1000 rad/s ³ .
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_speed</i>	Target speed in [1/min]. Sign represents the rotation direction: <ul style="list-style-type: none"> • Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.
out	<i>ID_accel</i>	Target acceleration in [rad/s ²].
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
	<i>ID_err</i>	(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 Indrdrive 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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_drvmode</i>	Pointer that provides the respective information: <ul style="list-style-type: none"> • 0 - Drive Mode not supported, • 1 - "Sequencer" drive mode active, • 2 - "Speed Control" drive mode active.

Parameters

out	<i>ID_err</i>	(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
<code>(*ID_opstate & 0b11) == 0b00</code>	Control section / power section not ready for operation(e.g., drive error or phase 2)
<code>(*ID_opstate & 0b11) == 0b01</code>	Control section ready for operation "bb"
<code>(*ID_opstate & 0b11) == 0b10</code>	Control section and power section ready for op. "Ab"
<code>(*ID_opstate & 0b11) == 0b11</code>	Drive with torque "AF".
<code>((*ID_opstate & 0b100) >> 2) == 1</code>	Drive Halt is active and axis is in standstill
<code>((*ID_opstate & 0b1000) >> 3) == 1</code>	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(<path_to_DLL>, 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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
----	---------------	--

Parameters

out	<i>ID_opstate</i>	Pointer that provides the respective information: <ul style="list-style-type: none"> • Bit 0-1: Operation state <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – 0: Drive not halted – 1: Drive Halt is active and axis is in standstill • Bit 3: Drive error <ul style="list-style-type: none"> – 0: No error – 1: Drive error present.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_speed</i>	Pointer that provides the speed information as double Value in [1/min]. Sign represents the rotation direction:
		<ul style="list-style-type: none"> • Positive sign: Clockwise direction • Negative sign: Counter-clockwise direction.
out	<i>ID_err</i>	(Optional) Error handle.

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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_diagnostic_msg</i>	Pointer that provides the diagnostic message string.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_diagnostic_num</i>	Pointer that provides the diagnostic number.
out	<i>ID_err</i>	(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	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_err</i>	(Optional) Error handle.

Returns

Error handle return code ([ErrHandle\(\)](#)).

Definition at line 598 of file [Wrapper.cpp](#).

6.15 Wrapper.h

```

00001
00004 #ifndef _WRAPPER_H_
00005 #define _WRAPPER_H_
00006
00008 #include "mainpage.dox"
00009
00010 #include <Windows.h>
00011
00012 #include "SISProtocol.h"
00013 #include "RS232.h"
00014 #include "errors.h"
00015 #include "debug.h"
00016
00017
00019 #define DLLEXPORT __declspec(dllexport)
00020 #define DLLCALLCONV __cdecl
00021
00022 #ifndef _DLL
00023 #error Project output has to be a DLL file
00024 #endif
00025
00026 #if __cplusplus <= 199711L
00027 #if _MSC_VER < 1900
00028 #error This library needs at least Microsoft Visual Studio 2015 or a C++11 compliant compiler
00029 #endif
00030
00031 #ifndef _MSC_VER
00032 #error This library needs at least a C++11 compliant compiler
00033 #endif
00034 #endif
00035
00036 #ifndef __cplusplus
00037 #error C++ compiler required
00038 #else
00039 extern "C" { /* using a C++ compiler */
00040 #endif
00041
00043     #define DRIVEMODE_SEQUENCER    0b111011
00044     #define DRIVEMODE_SPEEDCONTROL 0b10
00046
00064     typedef struct OPSTATE
00065     {
00066         union
00067         {
00068             struct Bits
00069             {
00075                 uint8_t OperateState : 2;
00076
00080                 uint8_t DriveHalted : 1;
00081
00085                 uint8_t DriveError : 1;
00086
00090                 Bits(uint16_t P_0_0115 = 0) :
00092                     OperateState((P_0_0115 >> 14) & 0b11),
00094                     DriveHalted((P_0_0115 >> 4) & 0b1),
00096                     DriveError((P_0_0115 >> 13) & 0b1)
00097             {}
00098             } Bits;
00099
00101             uint8_t Value;
00102         };
00103

```

```

00107         OPSTATE(uint16_t P_0_0115 = 0) : Bits(P_0_0115) {}
00108     } OPSTATE;
00109
00110
00116     typedef struct SPEEDUNITS
00117     {
00118         union
00119         {
00120             struct Bits
00121             {
00125                 uint16_t type_of_scaling : 3;
00126
00130                 uint16_t automode : 1;
00131
00135                 uint16_t scale_units : 1;
00136
00140                 uint16_t time_units : 1;
00141
00145                 uint16_t data_rel : 1;
00146
00148                 uint16_t res7 : 9;
00149
00153                 Bits(uint16_t S_0_0044 = 0) :
00154                     // Bit 0-2 @ S-0-0044
00155                     type_of_scaling((S_0_0044) & 0b111),
00156                     // Bit 3 @ S-0-0044
00157                     automode((S_0_0044 >> 3) & 0b1),
00158                     // Bit 4 @ S-0-0044
00159                     scale_units((S_0_0044 >> 4) & 0b1),
00160                     // Bit 5 @ S-0-0044
00161                     time_units((S_0_0044 >> 5) & 0b1),
00162                     // Bit 6 @ S-0-0044
00163                     data_rel((S_0_0044 >> 6) & 0b1),
00164                     // Bit 7-15 @ S-0-0044
00165                     res7((S_0_0044 >> 7) & 0b11111111)
00166             {}
00167             } Bits;
00168
00170             uint16_t Value;
00171         };
00172
00176         SPEEDUNITS(uint16_t S_0_0044 = 0) : Bits(S_0_0044) {}
00177     } SPEEDUNITS;
00178
00179
00183     typedef struct SISProtocol SISProtocol;
00184
00185
00186 #pragma region API Fundamentals
00187
00208     DLLEXPORT SISProtocol* DLLCALLCONV init();
00209
00235     DLLEXPORT int32_t DLLCALLCONV open(SISProtocol* ID_ref, const
wchar_t* ID_comport = L"COM1", uint32_t ID_combaudrate = 19200, ErrHandle ID_err =
ErrHandle());
00236
00258     DLLEXPORT int32_t DLLCALLCONV close(SISProtocol* ID_ref,
ErrHandle ID_err = ErrHandle());
00259
00260 #pragma endregion API Fundamentals
00261
00262 #pragma region API Sequencer
00263
00287     DLLEXPORT int32_t DLLCALLCONV sequencer_activate(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());
00288
00307     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());
00308
00339     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());
00340
00357     DLLEXPORT int32_t DLLCALLCONV sequencer_softtrigger(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());
00358
00376     DLLEXPORT int32_t DLLCALLCONV sequencer_hardtrigger(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());
00377
00410     DLLEXPORT int32_t DLLCALLCONV sequencer_getstatus(
SISProtocol* ID_ref, uint16_t * ID_status, ErrHandle ID_err =
ErrHandle());
00411
00412 #pragma endregion API Sequencer
00413

```



```

00414
00415 #pragma region API Speed Control
00416
00441     DLLEXPORT int32_t DLLCALLCONV speedcontrol_activate(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());
00442
00466     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());
00467
00493     DLLEXPORT int32_t DLLCALLCONV speedcontrol_write(
SISProtocol* ID_ref, double_t ID_speed, double_t ID_accel, ErrHandle ID_err =
ErrHandle());
00494
00495 #pragma endregion API Speed Control
00496
00497 #pragma region API Configuration
00498
00522     DLLEXPORT int32_t DLLCALLCONV set_stdenviroment(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());
00523
00524 #pragma endregion API Configuration
00525
00526
00527 #pragma region API Status
00528
00565     DLLEXPORT int32_t DLLCALLCONV get_drivemode(
SISProtocol* ID_ref, uint32_t * ID_drvmode, ErrHandle ID_err =
ErrHandle());
00566
00613     DLLEXPORT int32_t DLLCALLCONV get_opstate(
SISProtocol* ID_ref, uint8_t * ID_opstate, ErrHandle ID_err =
ErrHandle());
00614
00635     DLLEXPORT int32_t DLLCALLCONV get_speed(
SISProtocol * ID_ref, double_t * ID_speed, ErrHandle ID_err =
ErrHandle());
00636
00662     DLLEXPORT int32_t DLLCALLCONV get_diagnostic_msg(
SISProtocol* ID_ref, char * ID_diagnostic_msg, ErrHandle ID_err =
ErrHandle());
00663
00685     DLLEXPORT int32_t DLLCALLCONV get_diagnostic_num(
SISProtocol* ID_ref, uint32_t * ID_diagnostic_num, ErrHandle ID_err =
ErrHandle());
00686
00708     DLLEXPORT int32_t DLLCALLCONV clear_error(
SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());
00709
00710 #pragma endregion API Status
00711
00712     /* \cond Do not document this */
00713
00714 #pragma region Internal helper functions
00715
00721     inline void change_opmode(SISProtocol * ID_ref, const uint64_t opmode);
00722
00728     inline SPEEDUNITS get_units(SISProtocol * ID_ref);
00729
00733     inline void change_units(SISProtocol * ID_ref);
00734
00744     inline void change_language(SISProtocol * ID_ref, const uint8_t lang_code = 1
);
00745
00746 #pragma endregion Internal helper functions
00747
00748     /* \endcond Do not document this */
00749
00750 #ifdef __cplusplus
00751 }
00752 #endif
00753
00754 #endif /* _WRAPPER_H_ */

```

7 Example Documentation

7.1 apps/PythonApplication1/PythonApplication1.py

7.2 apps/WpfApplication1/Indradrive.cs

Index

- ~ExceptionSISError
 - SISProtocol::ExceptionSISError, [26](#)
- ~ExceptionTransceiveFailed
 - SISProtocol::ExceptionTransceiveFailed, [28](#)
- ~Map
 - TGM::Map, [43](#)
- ~SISProtocol
 - SISProtocol, [73](#)
- AdrES1
 - TGM::HeaderExt, [40](#)
- AdrES2
 - TGM::HeaderExt, [41](#)
- AdrES3
 - TGM::HeaderExt, [41](#)
- AdrES4
 - TGM::HeaderExt, [41](#)
- AdrES5
 - TGM::HeaderExt, [41](#)
- AdrES6
 - TGM::HeaderExt, [41](#)
- AdrES7
 - TGM::HeaderExt, [42](#)
- AdrE
 - TGM::Header, [36](#)
- AdrS
 - TGM::Header, [36](#)
- at
 - TGM::Data, [20](#)
- automode
 - SPEEDUNITS, [79](#)
- BAUDRATE
 - SISProtocol, [72, 73](#)
- Bits
 - OPSTATE, [48](#)
 - SPEEDUNITS, [80](#)
 - TGM::Bitfields::HeaderControl, [39](#)
 - TGM::Bitfields::SercosParamAttribute, [65](#)
 - TGM::Bitfields::SercosParamControl, [68](#)
 - TGM::Bitfields::SercosParamIdent, [70](#)
- Bytes
 - TGM::Bytestream, [17](#)
 - TGM::Commands::SercosList, [54](#)
 - TGM::Commands::SercosParam, [61](#)
 - TGM::Commands::Subservice, [86](#)
 - TGM::Data, [22](#)
 - TGM::Reactions::SercosList, [51](#)
 - TGM::Reactions::SercosParam, [57](#)
 - TGM::Reactions::Subservice, [83](#)
- Bytestream
 - TGM::Bytestream, [16](#)
 - TGM, [9](#)
- calc_checksum
 - TGM::Header, [34](#)
- change_language
 - Wrapper.cpp, [133](#)
- change_opmode
 - Wrapper.cpp, [133](#)
- change_units
 - Wrapper.cpp, [133](#)
- clear
 - TGM::Bytestream, [16](#)
 - TGM::Commands::SercosList, [53](#)
 - TGM::Commands::SercosParam, [60](#)
 - TGM::Commands::Subservice, [85](#)
 - TGM::Data, [21](#)
 - TGM::Reactions::SercosList, [49](#)
 - TGM::Reactions::SercosParam, [56](#)
 - TGM::Reactions::Subservice, [81](#)
- clear_error
 - Wrapper.cpp, [132](#)
 - Wrapper.h, [160](#)
- close
 - SISProtocol, [74](#)
 - Wrapper.cpp, [120](#)
 - Wrapper.h, [147](#)
- Cntrl
 - TGM::Header, [35](#)
- code
 - GenericErrHandle, [30](#)
- Control
 - TGM::Commands::SercosList, [53](#)
 - TGM::Commands::SercosParam, [61](#)
 - TGM::Reactions::SercosList, [50](#)
 - TGM::Reactions::SercosParam, [57](#)
- ConversionFactor
 - TGM::Bitfields::SercosParamAttribute, [63](#)
- CS
 - TGM::Header, [34](#)
- DLLCALLCONV
 - Wrapper.h, [144](#)
- DLLEXPORT
 - Wrapper.h, [144](#)
- DRIVEMODE_SEQUENCER
 - Wrapper.h, [144](#)
- DRIVEMODE_SPEEDCONTROL
 - Wrapper.h, [145](#)
- DatLW
 - TGM::Header, [35](#)
- Data
 - TGM::Data, [18, 19](#)
 - TGM, [8](#)
- data_rel
 - SPEEDUNITS, [79](#)
- DataDisplay
 - TGM::Bitfields::SercosParamAttribute, [63](#)
- DataFunction
 - TGM::Bitfields::SercosParamAttribute, [63](#)

- DataLen
 - TGM::BitFields::SercosParamAttribute, 63
- Datablock
 - TGM::BitFields::SercosParamControl, 67
- DatL
 - TGM::Header, 35
- DriveError
 - OPSTATE, 47
- DriveHalted
 - OPSTATE, 47
- EErrorBlocks
 - errors.h, 88
- Err_Base
 - errors.h, 87
- ErrHandle
 - errors.h, 87
- Error
 - TGM::Reactions::SercosList, 51
 - TGM::Reactions::SercosParam, 57
 - TGM::Reactions::Subservice, 83
- errors.h, 86
 - EErrorBlocks, 88
 - Err_Base, 87
 - ErrHandle, 87
 - GenericErrHandle, 87
 - set_error, 89
- ExceptionGeneric
 - SISProtocol::ExceptionGeneric, 24
- ExceptionSISError
 - SISProtocol::ExceptionSISError, 26
- ExceptionTransceiveFailed
 - SISProtocol::ExceptionTransceiveFailed, 27
- execute_command
 - SISProtocol, 77
- GenericErrHandle, 28
 - code, 30
 - errors.h, 87
 - GenericErrHandle, 29
 - msg, 31
 - set, 29
 - set_code, 30
 - set_msg, 30
- get_DatL
 - TGM::Header, 34
- get_diagnostic_msg
 - Wrapper.cpp, 131
 - Wrapper.h, 159
- get_diagnostic_num
 - Wrapper.cpp, 131
 - Wrapper.h, 159
- get_drivemode
 - Wrapper.cpp, 128
 - Wrapper.h, 156
- get_errorcode
 - SISProtocol::ExceptionSISError, 26
- get_head_size
 - TGM::Commands::SercosList, 53
- TGM::Commands::SercosParam, 60
- TGM::Commands::Subservice, 85
- TGM::Reactions::SercosList, 49
- TGM::Reactions::SercosParam, 56
- TGM::Reactions::Subservice, 82
- get_opstate
 - Wrapper.cpp, 129
 - Wrapper.h, 157
- get_size
 - TGM::Commands::SercosList, 53
 - TGM::Commands::SercosParam, 60
 - TGM::Commands::Subservice, 85
 - TGM::Data, 22
 - TGM::Header, 33
 - TGM::Reactions::SercosList, 50
 - TGM::Reactions::SercosParam, 56
 - TGM::Reactions::Subservice, 82
- get_speed
 - Wrapper.cpp, 130
 - Wrapper.h, 158
- get_status
 - SISProtocol::ExceptionGeneric, 24
- get_sum
 - TGM::Header, 33
- get_units
 - Wrapper.cpp, 133
- Header
 - TGM::Header, 32
 - TGM::Map::Mapping, 45
 - TGM, 9
- HeaderControl
 - TGM::BitFields, 13
 - TGM::BitFields::HeaderControl, 38
- HeaderExt
 - TGM, 9
- HeaderType
 - TGM, 9
- init
 - Wrapper.cpp, 118
 - Wrapper.h, 146
- is_writeonly_phase2
 - TGM::BitFields::SercosParamAttribute, 64
- is_writeonly_phase3
 - TGM::BitFields::SercosParamAttribute, 64
- is_writeonly_phase4
 - TGM::BitFields::SercosParamAttribute, 64
- ListOffset
 - TGM::Commands::SercosList, 54
- m_bytestream
 - SISProtocol::ExceptionSISError, 26
- m_errorcode
 - SISProtocol::ExceptionSISError, 26
- m_message
 - SISProtocol::ExceptionGeneric, 24
- m_status

- SISProtocol::ExceptionGeneric, [24](#)
- MAX_COMMANDCHECK_ITERATIONS
 - SISProtocol.h, [102](#)
- mainpage.dox, [91](#)
- Map
 - TGM::Map, [43](#)
- Mapping
 - TGM::Map, [44](#)
 - TGM::Map::Mapping, [45](#)
- msg
 - GenericErrHandle, [31](#)
- NumRunningTgm
 - TGM::Bitfields::HeaderControl, [38](#)
- NumSubAddresses
 - TGM::Bitfields::HeaderControl, [38](#)
- OPSTATE, [46](#)
 - Bits, [48](#)
 - DriveError, [47](#)
 - DriveHalted, [47](#)
 - OPSTATE, [47](#)
 - OperateState, [47](#)
 - Value, [48](#)
 - Wrapper.h, [145](#)
- open
 - SISProtocol, [74](#)
 - Wrapper.cpp, [119](#)
 - Wrapper.h, [146](#)
- OperateState
 - OPSTATE, [47](#)
- operator<<
 - TGM::Data, [21](#)
- PaketN
 - TGM::HeaderExt, [42](#)
- ParamNum
 - TGM::Commands::SercosList, [54](#)
 - TGM::Commands::SercosParam, [61](#)
- ParamNumber
 - TGM::Bitfields::SercosParamIdent, [69](#)
- ParamSet
 - TGM::Bitfields::SercosParamIdent, [70](#)
- ParamType
 - TGM::Commands::SercosList, [54](#)
 - TGM::Commands::SercosParam, [61](#)
- ParamVariant
 - TGM::Bitfields::SercosParamIdent, [70](#)
- Payload
 - TGM::Map::Mapping, [45](#)
- RS232_BUFFER
 - SISProtocol.h, [101](#)
- RS232_READ_LOOPS_MAX
 - SISProtocol.h, [101](#)
- RS232_READ_TIMEOUT
 - SISProtocol.h, [101](#)
- Raw
 - TGM::Map, [44](#)
- read_listelm
 - SISProtocol, [75](#)
- read_parameter
 - SISProtocol, [74](#), [75](#)
- RecipientAddr
 - TGM::Commands::Subservice, [85](#)
 - TGM::Reactions::Subservice, [82](#)
- res1
 - TGM::Bitfields::SercosParamControl, [67](#)
- res10
 - TGM::Bitfields::SercosParamAttribute, [65](#)
- res2
 - TGM::Bitfields::SercosParamControl, [67](#)
- res5
 - TGM::Bitfields::SercosParamAttribute, [64](#)
- res6
 - TGM::Bitfields::SercosParamControl, [68](#)
- res7
 - SPEEDUNITS, [80](#)
 - TGM::Bitfields::SercosParamControl, [68](#)
- SIS_ADDR_MASTER
 - SISProtocol.h, [101](#)
- SIS_ADDR_SLAVE
 - SISProtocol.h, [101](#)
- SIS_ADDR_UNIT
 - SISProtocol.h, [102](#)
- SIS_SERVICES
 - SISProtocol, [72](#)
- SISProtocol, [71](#)
 - ~SISProtocol, [73](#)
 - BAUDRATE, [72](#), [73](#)
 - close, [74](#)
 - execute_command, [77](#)
 - open, [74](#)
 - read_listelm, [75](#)
 - read_parameter, [74](#), [75](#)
 - SIS_SERVICES, [72](#)
 - SISProtocol, [73](#)
 - set_baudrate, [74](#)
 - Wrapper.h, [146](#)
 - write_listelm, [76](#), [77](#)
 - write_parameter, [76](#)
- SISProtocol.cpp, [91](#)
- SISProtocol.h, [100](#), [102](#)
 - MAX_COMMANDCHECK_ITERATIONS, [102](#)
 - RS232_BUFFER, [101](#)
 - RS232_READ_LOOPS_MAX, [101](#)
 - RS232_READ_TIMEOUT, [101](#)
 - SIS_ADDR_MASTER, [101](#)
 - SIS_ADDR_SLAVE, [101](#)
 - SIS_ADDR_UNIT, [102](#)
- SISProtocol::ExceptionGeneric, [23](#)
 - ExceptionGeneric, [24](#)
 - get_status, [24](#)
 - m_message, [24](#)
 - m_status, [24](#)
 - warning, [24](#)
 - what, [24](#)

- SISProtocol::ExceptionSISError, 25
 - ~ExceptionSISError, 26
 - ExceptionSISError, 26
 - get_errorcode, 26
 - m_bytestream, 26
 - m_errorcode, 26
 - what, 26
- SISProtocol::ExceptionTransceiveFailed, 27
 - ~ExceptionTransceiveFailed, 28
 - ExceptionTransceiveFailed, 27
 - what, 28
- SPEEDUNITS, 77
 - automode, 79
 - Bits, 80
 - data_rel, 79
 - res7, 80
 - SPEEDUNITS, 78
 - scale_units, 79
 - time_units, 79
 - type_of_scaling, 78
 - Value, 80
 - Wrapper.h, 145
- scale_units
 - SPEEDUNITS, 79
- ScaleFactor
 - TGM::Bitfields::SercosParamAttribute, 64
- SegmentSize
 - TGM::Commands::SercosList, 54
- sequencer_activate
 - Wrapper.cpp, 120
 - Wrapper.h, 148
- sequencer_getstatus
 - Wrapper.cpp, 124
 - Wrapper.h, 152
- sequencer_hardtrigger
 - Wrapper.cpp, 123
 - Wrapper.h, 151
- sequencer_init
 - Wrapper.cpp, 121
 - Wrapper.h, 149
- sequencer_softtrigger
 - Wrapper.cpp, 123
 - Wrapper.h, 150
- sequencer_write
 - Wrapper.cpp, 122
 - Wrapper.h, 149
- SercosCommandrequest
 - TGM, 10
- SercosCommandstatus
 - TGM, 11
- SercosDatablock
 - TGM, 10
- SercosDatalen
 - TGM, 12
- SercosList
 - TGM::Commands, 14
 - TGM::Commands::SercosList, 52
 - TGM::Reactions, 15
 - TGM::Reactions::SercosList, 49
- SercosParam
 - TGM::Commands, 14
 - TGM::Commands::SercosParam, 58
 - TGM::Reactions, 15
 - TGM::Reactions::SercosParam, 56
- SercosParamAttribute
 - TGM::Bitfields, 13
 - TGM::Bitfields::SercosParamAttribute, 62
- SercosParamControl
 - TGM::Bitfields, 13
 - TGM::Bitfields::SercosParamControl, 66
- SercosParamIdent
 - TGM::Bitfields, 13
 - TGM::Bitfields::SercosParamIdent, 69
- SercosParamVar
 - TGM, 10
- SercosTxProgress
 - TGM, 11
- Service
 - TGM::Header, 35
- ServiceNumber
 - TGM::Commands::Subservice, 85
 - TGM::Reactions::Subservice, 83
- set
 - GenericErrHandle, 29
 - TGM::Map, 43
- set_DatL
 - TGM::Header, 33
- set_baudrate
 - SISProtocol, 74
- set_code
 - GenericErrHandle, 30
- set_error
 - errors.h, 89
- set_msg
 - GenericErrHandle, 30
- set_size
 - TGM::Data, 22
- set_stdenvironment
 - Wrapper.cpp, 127
 - Wrapper.h, 155
- Size
 - TGM::Data, 22
- speedcontrol_activate
 - Wrapper.cpp, 125
 - Wrapper.h, 153
- speedcontrol_init
 - Wrapper.cpp, 125
 - Wrapper.h, 153
- speedcontrol_write
 - Wrapper.cpp, 126
 - Wrapper.h, 154
- Status
 - TGM::Reactions::SercosList, 50
 - TGM::Reactions::SercosParam, 57
 - TGM::Reactions::Subservice, 82
- StatusReactionTgm

- TGM::Bitfields::HeaderControl, 39
- StZ
 - TGM::Header, 34
- Subservice
 - TGM::Commands, 14
 - TGM::Commands::Subservice, 84
 - TGM::Reactions, 15
 - TGM::Reactions::Subservice, 81
- TGM::Bitfields, 12
 - HeaderControl, 13
 - SercosParamAttribute, 13
 - SercosParamControl, 13
 - SercosParamIdent, 13
- TGM::Bitfields::HeaderControl, 37
 - Bits, 39
 - HeaderControl, 38
 - NumRunningTgm, 38
 - NumSubAddresses, 38
 - StatusReactionTgm, 39
 - Type, 38
 - Value, 39
- TGM::Bitfields::SercosParamAttribute, 62
 - Bits, 65
 - ConversionFactor, 63
 - DataDisplay, 63
 - DataFunction, 63
 - DataLen, 63
 - is_writeonly_phase2, 64
 - is_writeonly_phase3, 64
 - is_writeonly_phase4, 64
 - res10, 65
 - res5, 64
 - ScaleFactor, 64
 - SercosParamAttribute, 62
 - Value, 65
- TGM::Bitfields::SercosParamControl, 65
 - Bits, 68
 - Datablock, 67
 - res1, 67
 - res2, 67
 - res6, 68
 - res7, 68
 - SercosParamControl, 66
 - TxProgress, 67
 - Value, 68
- TGM::Bitfields::SercosParamIdent, 68
 - Bits, 70
 - ParamNumber, 69
 - ParamSet, 70
 - ParamVariant, 70
 - SercosParamIdent, 69
 - Value, 70
- TGM::Bytestream, 16
 - Bytes, 17
 - Bytestream, 16
 - clear, 16
- TGM::Commands, 13
 - SercosList, 14
 - SercosParam, 14
 - Subservice, 14
- TGM::Commands::SercosList, 51
 - Bytes, 54
 - clear, 53
 - Control, 53
 - get_head_size, 53
 - get_size, 53
 - ListOffset, 54
 - ParamNum, 54
 - ParamType, 54
 - SegmentSize, 54
 - SercosList, 52
 - UnitAddr, 53
- TGM::Commands::SercosParam, 58
 - Bytes, 61
 - clear, 60
 - Control, 61
 - get_head_size, 60
 - get_size, 60
 - ParamNum, 61
 - ParamType, 61
 - SercosParam, 58
 - UnitAddr, 61
- TGM::Commands::Subservice, 83
 - Bytes, 86
 - clear, 85
 - get_head_size, 85
 - get_size, 85
 - RecipientAddr, 85
 - ServiceNumber, 85
 - Subservice, 84
- TGM::Data, 17
 - at, 20
 - Bytes, 22
 - clear, 21
 - Data, 18, 19
 - get_size, 22
 - operator<<, 21
 - set_size, 22
 - Size, 22
 - toBYTE, 21
 - toUINT16, 21
 - toUINT32, 20
 - toUINT64, 20
 - toUINT8, 21
 - toVector, 20
- TGM::Header, 31
 - AdrE, 36
 - AdrS, 36
 - calc_checksum, 34
 - Cntrl, 35
 - CS, 34
 - DatLW, 35
 - DatL, 35
 - get_DatL, 34
 - get_size, 33
 - get_sum, 33

- Header, 32
- Service, 35
- set_DatL, 33
- StZ, 34
- TGM::HeaderExt, 40
 - AdrES1, 40
 - AdrES2, 41
 - AdrES3, 41
 - AdrES4, 41
 - AdrES5, 41
 - AdrES6, 41
 - AdrES7, 42
 - PaketN, 42
- TGM::Map
 - ~Map, 43
 - Map, 43
 - Mapping, 44
 - Raw, 44
 - set, 43
- TGM::Map< THeader, TPayload >, 42
- TGM::Map< THeader, TPayload >::Mapping, 44
- TGM::Map::Mapping
 - Header, 45
 - Mapping, 45
 - Payload, 45
- TGM::Reactions, 15
 - SercosList, 15
 - SercosParam, 15
 - Subservice, 15
- TGM::Reactions::SercosList, 48
 - Bytes, 51
 - clear, 49
 - Control, 50
 - Error, 51
 - get_head_size, 49
 - get_size, 50
 - SercosList, 49
 - Status, 50
 - UnitAddr, 50
- TGM::Reactions::SercosParam, 55
 - Bytes, 57
 - clear, 56
 - Control, 57
 - Error, 57
 - get_head_size, 56
 - get_size, 56
 - SercosParam, 56
 - Status, 57
 - UnitAddr, 57
- TGM::Reactions::Subservice, 80
 - Bytes, 83
 - clear, 81
 - Error, 83
 - get_head_size, 82
 - get_size, 82
 - RecipientAddr, 82
 - ServiceNumber, 83
 - Status, 82
 - Subservice, 81
- TGM_SIZE_HEADER_EXT
 - Telegrams.h, 107
- TGM_SIZE_HEADER
 - Telegrams.h, 106
- TGM_SIZEMAX_PAYLOAD
 - Telegrams.h, 107
- TGM_SIZEMAX
 - Telegrams.h, 107
- TGM, 7
 - Bytestream, 9
 - Data, 8
 - Header, 9
 - HeaderExt, 9
 - HeaderType, 9
 - SercosCommandrequest, 10
 - SercosCommandstatus, 11
 - SercosDatablock, 10
 - SercosDatalen, 12
 - SercosParamVar, 10
 - SercosTxProgress, 11
- Telegrams.h, 105, 107
 - TGM_SIZE_HEADER_EXT, 107
 - TGM_SIZE_HEADER, 106
 - TGM_SIZEMAX_PAYLOAD, 107
 - TGM_SIZEMAX, 107
- Telegrams_Bitfields.h, 113, 115
- time_units
 - SPEEDUNITS, 79
- toBYTE
 - TGM::Data, 21
- toUINT16
 - TGM::Data, 21
- toUINT32
 - TGM::Data, 20
- toUINT64
 - TGM::Data, 20
- toUINT8
 - TGM::Data, 21
- toVector
 - TGM::Data, 20
- TxProgress
 - TGM::Bitfields::SercosParamControl, 67
- Type
 - TGM::Bitfields::HeaderControl, 38
- type_of_scaling
 - SPEEDUNITS, 78
- UnitAddr
 - TGM::Commands::SercosList, 53
 - TGM::Commands::SercosParam, 61
 - TGM::Reactions::SercosList, 50
 - TGM::Reactions::SercosParam, 57
- Value
 - OPSTATE, 48
 - SPEEDUNITS, 80
 - TGM::Bitfields::HeaderControl, 39
 - TGM::Bitfields::SercosParamAttribute, 65

- TGM::Bitfields::SercosParamControl, [68](#)
- TGM::Bitfields::SercosParamIdent, [70](#)
- warning
 - SISProtocol::ExceptionGeneric, [24](#)
- what
 - SISProtocol::ExceptionGeneric, [24](#)
 - SISProtocol::ExceptionSISError, [26](#)
 - SISProtocol::ExceptionTransceiveFailed, [28](#)
- Wrapper.cpp, [117](#)
 - change_language, [133](#)
 - change_opmode, [133](#)
 - change_units, [133](#)
 - clear_error, [132](#)
 - close, [120](#)
 - get_diagnostic_msg, [131](#)
 - get_diagnostic_num, [131](#)
 - get_drivemode, [128](#)
 - get_opstate, [129](#)
 - get_speed, [130](#)
 - get_units, [133](#)
 - init, [118](#)
 - open, [119](#)
 - sequencer_activate, [120](#)
 - sequencer_getstatus, [124](#)
 - sequencer_hardtrigger, [123](#)
 - sequencer_init, [121](#)
 - sequencer_softtrigger, [123](#)
 - sequencer_write, [122](#)
 - set_stdenvironment, [127](#)
 - speedcontrol_activate, [125](#)
 - speedcontrol_init, [125](#)
 - speedcontrol_write, [126](#)
- Wrapper.h, [142](#)
 - clear_error, [160](#)
 - close, [147](#)
 - DLLCALLCONV, [144](#)
 - DLLEXPORT, [144](#)
 - DRIVEMODE_SEQUENCER, [144](#)
 - DRIVEMODE_SPEEDCONTROL, [145](#)
 - get_diagnostic_msg, [159](#)
 - get_diagnostic_num, [159](#)
 - get_drivemode, [156](#)
 - get_opstate, [157](#)
 - get_speed, [158](#)
 - init, [146](#)
 - OPSTATE, [145](#)
 - open, [146](#)
 - SISProtocol, [146](#)
 - SPEEDUNITS, [145](#)
 - sequencer_activate, [148](#)
 - sequencer_getstatus, [152](#)
 - sequencer_hardtrigger, [151](#)
 - sequencer_init, [149](#)
 - sequencer_softtrigger, [150](#)
 - sequencer_write, [149](#)
 - set_stdenvironment, [155](#)
 - speedcontrol_activate, [153](#)
 - speedcontrol_init, [153](#)
 - speedcontrol_write, [154](#)
 - write_listelm
 - SISProtocol, [76](#), [77](#)
 - write_parameter
 - SISProtocol, [76](#)