

Indradrive API

0.5

Generated by Doxygen 1.8.12

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	8
2	Bug List	10
3	Namespace Documentation	10
3.1	TGM Namespace Reference	10
3.1.1	Detailed Description	11
3.1.2	Typedef Documentation	11
3.1.3	Enumeration Type Documentation	12
3.2	TGM::BitFields Namespace Reference	15
3.2.1	Detailed Description	15
3.2.2	Typedef Documentation	15
3.3	TGM::Commands Namespace Reference	16
3.3.1	Detailed Description	16
3.3.2	Typedef Documentation	16
3.4	TGM::Reactions Namespace Reference	17
3.4.1	Detailed Description	17
3.4.2	Typedef Documentation	17

4	Class Documentation	18
4.1	TGM::Bytestream Struct Reference	18
4.1.1	Detailed Description	18
4.1.2	Constructor & Destructor Documentation	18
4.1.3	Member Function Documentation	19
4.1.4	Member Data Documentation	19
4.2	TGM::Data Struct Reference	19
4.2.1	Detailed Description	20
4.2.2	Constructor & Destructor Documentation	20
4.2.3	Member Function Documentation	21
4.2.4	Member Data Documentation	24
4.3	SISProtocol::ExceptionGeneric Class Reference	25
4.3.1	Detailed Description	25
4.3.2	Constructor & Destructor Documentation	25
4.3.3	Member Function Documentation	26
4.3.4	Member Data Documentation	26
4.4	SISProtocol::ExceptionSISError Class Reference	26
4.4.1	Detailed Description	27
4.4.2	Constructor & Destructor Documentation	27
4.4.3	Member Function Documentation	27
4.4.4	Member Data Documentation	28
4.5	SISProtocol::ExceptionTransceiveFailed Class Reference	28
4.5.1	Detailed Description	29
4.5.2	Constructor & Destructor Documentation	29
4.5.3	Member Function Documentation	29
4.6	GenericErrHandle Struct Reference	29
4.6.1	Detailed Description	30
4.6.2	Constructor & Destructor Documentation	30
4.6.3	Member Function Documentation	31
4.6.4	Member Data Documentation	31

4.7	TGM::Header Struct Reference	32
4.7.1	Detailed Description	33
4.7.2	Constructor & Destructor Documentation	33
4.7.3	Member Function Documentation	34
4.7.4	Member Data Documentation	35
4.8	TGM::BitFields::HeaderControl Struct Reference	37
4.8.1	Detailed Description	38
4.8.2	Constructor & Destructor Documentation	38
4.8.3	Member Data Documentation	38
4.9	TGM::HeaderExt Struct Reference	40
4.9.1	Detailed Description	40
4.9.2	Member Data Documentation	41
4.10	TGM::Map< THeader, TPayload > Union Template Reference	42
4.10.1	Detailed Description	43
4.10.2	Constructor & Destructor Documentation	43
4.10.3	Member Function Documentation	43
4.10.4	Member Data Documentation	44
4.11	TGM::Map< THeader, TPayload >::Mapping Struct Reference	44
4.11.1	Detailed Description	44
4.11.2	Constructor & Destructor Documentation	44
4.11.3	Member Data Documentation	45
4.12	OPSTATE Struct Reference	45
4.12.1	Detailed Description	46
4.12.2	Constructor & Destructor Documentation	46
4.12.3	Member Data Documentation	46
4.13	TGM::Reactions::SercosList Struct Reference	48
4.13.1	Detailed Description	48
4.13.2	Constructor & Destructor Documentation	48
4.13.3	Member Function Documentation	49
4.13.4	Member Data Documentation	49

4.14	TGM::Commands::SercosList Struct Reference	50
4.14.1	Detailed Description	51
4.14.2	Constructor & Destructor Documentation	51
4.14.3	Member Function Documentation	52
4.14.4	Member Data Documentation	52
4.15	TGM::Reactions::SercosParam Struct Reference	53
4.15.1	Detailed Description	54
4.15.2	Constructor & Destructor Documentation	54
4.15.3	Member Function Documentation	55
4.15.4	Member Data Documentation	55
4.16	TGM::Commands::SercosParam Struct Reference	56
4.16.1	Detailed Description	57
4.16.2	Constructor & Destructor Documentation	57
4.16.3	Member Function Documentation	57
4.16.4	Member Data Documentation	58
4.17	TGM::Bitfields::SercosParamAttribute Struct Reference	59
4.17.1	Detailed Description	60
4.17.2	Constructor & Destructor Documentation	60
4.17.3	Member Data Documentation	60
4.18	TGM::Bitfields::SercosParamControl Struct Reference	62
4.18.1	Detailed Description	63
4.18.2	Constructor & Destructor Documentation	63
4.18.3	Member Data Documentation	64
4.19	TGM::Bitfields::SercosParamIdent Struct Reference	65
4.19.1	Detailed Description	65
4.19.2	Constructor & Destructor Documentation	65
4.19.3	Member Data Documentation	66
4.20	SISProtocol Class Reference	67
4.20.1	Detailed Description	68
4.20.2	Member Typedef Documentation	68

4.20.3	Member Enumeration Documentation	68
4.20.4	Constructor & Destructor Documentation	69
4.20.5	Member Function Documentation	70
4.21	SPEEDUNITS Struct Reference	73
4.21.1	Detailed Description	73
4.21.2	Constructor & Destructor Documentation	73
4.21.3	Member Data Documentation	74
4.22	TGM::Reactions::Subservice Struct Reference	76
4.22.1	Detailed Description	76
4.22.2	Constructor & Destructor Documentation	76
4.22.3	Member Function Documentation	77
4.22.4	Member Data Documentation	77
4.23	TGM::Commands::Subservice Struct Reference	78
4.23.1	Detailed Description	79
4.23.2	Constructor & Destructor Documentation	79
4.23.3	Member Function Documentation	79
4.23.4	Member Data Documentation	80
5	File Documentation	80
5.1	errors.h File Reference	80
5.1.1	Detailed Description	81
5.1.2	Macro Definition Documentation	81
5.1.3	Typedef Documentation	82
5.1.4	Enumeration Type Documentation	82
5.1.5	Function Documentation	83
5.2	errors.h	84
5.3	mainpage.dox File Reference	85
5.4	SISProtocol.cpp File Reference	85
5.5	SISProtocol.cpp	85
5.6	SISProtocol.h File Reference	94
5.6.1	Macro Definition Documentation	94

5.7	SISProtocol.h	95
5.8	Telegrams.h File Reference	98
5.8.1	Detailed Description	100
5.8.2	Macro Definition Documentation	100
5.9	Telegrams.h	100
5.10	Telegrams_Bitfields.h File Reference	106
5.10.1	Detailed Description	108
5.11	Telegrams_Bitfields.h	108
5.12	Wrapper.cpp File Reference	110
5.12.1	Detailed Description	111
5.12.2	Function Documentation	111
5.13	Wrapper.cpp	126
5.14	Wrapper.h File Reference	133
5.14.1	Detailed Description	135
5.14.2	Macro Definition Documentation	135
5.14.3	Typedef Documentation	136
5.14.4	Function Documentation	137
5.15	Wrapper.h	149
6	Example Documentation	152
6.1	apps/PythonApplication1/PythonApplication1.py	152
6.2	apps/WpfApplication1/Indradrive.cs	153
	Index	157

1 Main Page

Author

Robert Hartmann (ATV PTS SDAE), Infineon Technologies AG
Application Engineering for Alternator Control ICs

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:

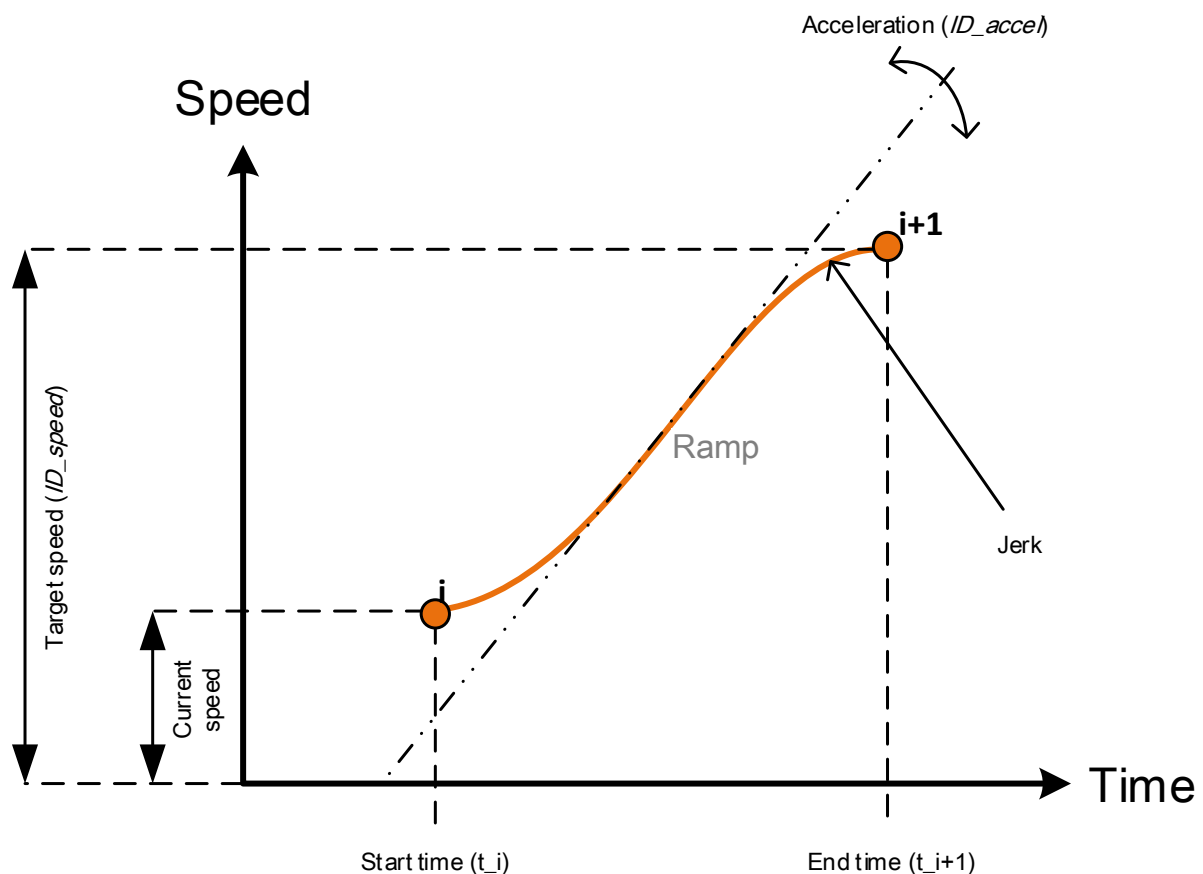


Figure 1 Principle of Speed Control

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:

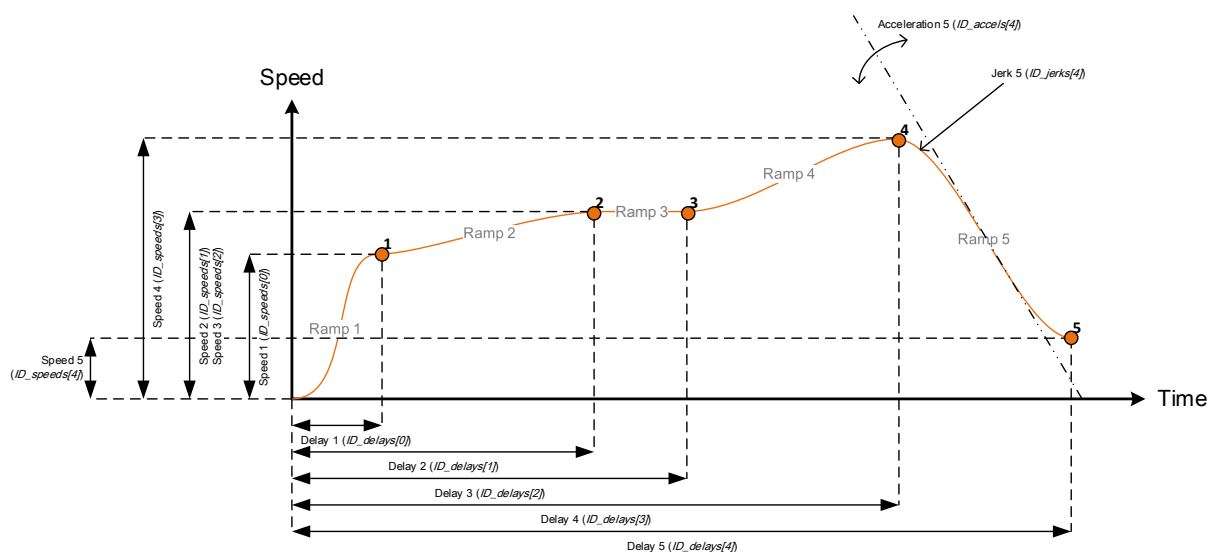


Figure 2 Principle of Sequencer

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.
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).

```
using System;
using System.Runtime.InteropServices;
using System.Text;
using System.Windows.Controls;

namespace WpfApplication1
{
    public class Indradrive
    {
        [StructLayout(LayoutKind.Sequential)]
        public unsafe struct ErrHandle
        {
            [MarshalAs(UnmanagedType.U4)]
            public UInt32 code;
            [MarshalAs(UnmanagedType.ByValArray, SizeConst = 2048)]

```

```

        public byte[] msg;
    }

    private int idref;
    private const string dllpath = "..\\..\\..\\..\\bin\\IndradriveAPI.dll";

    private ErrHandle indraerr;
    private ListBox listboxerr;

    public Indradrive(ref ListBox listbox)
    {
        listboxerr = listbox;
        idref = init();
    }

    // Fundamentals

    [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
    private static extern int init();

    [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);
    public int open(Byte[] ID_comport, UInt32 ID_combaudrate) { return CheckResult(
    open(idref, ID_comport, ID_combaudrate, ref indraerr)); }

    [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
    private static extern int close(int ID_ref, ref ErrHandle ID_err);
    public int close() { return CheckResult(close(idref, ref indraerr)); }

    // Speed Control

    [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
    private static extern int speedcontrol_activate(int ID_ref, ref
    ErrHandle ID_err);
    public int speedcontrol_activate() { return CheckResult(
    speedcontrol_activate(idref, ref indraerr)); }

    [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);
    public int speedcontrol_init(Double ID_max_accel, Double ID_max_jerk) { return
    CheckResult(speedcontrol_init(idref, ID_max_accel, ID_max_jerk, ref indraerr)); }

    [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);
    public int speedcontrol_write(Double ID_speed, Double ID_accel) { return
    CheckResult(speedcontrol_write(idref, ID_speed, ID_accel, ref indraerr)); }

    // Sequencer

    [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
    private static extern int sequencer_activate(int ID_ref, ref
    ErrHandle ID_err);
    public int sequencer_activate() { return CheckResult(
    sequencer_activate(idref, ref indraerr)); }

    [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);
    public int sequencer_init(Double ID_max_accel, Double ID_max_jerk) { return
    CheckResult(sequencer_init(idref, ID_max_accel, ID_max_jerk, ref indraerr)); }

    [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, Byte ID_direction, ref
    ErrHandle ID_err);
    public int sequencer_write(Double[] ID_speeds, Double[] ID_accels, Double[] ID_jerks
    , UInt32[] ID_delays, UInt16 ID_set_length, Byte ID_direction) { return CheckResult(
    sequencer_write(idref, ID_speeds, ID_accels, ID_jerks, ID_delays, ID_set_length,
    ID_direction, ref indraerr)); }

    [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
    private static extern int sequencer_softtrigger(int ID_ref, ref
    ErrHandle ID_err);
    public int sequencer_softtrigger() { return CheckResult(
    sequencer_softtrigger(idref, ref indraerr)); }

    // Status

    [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
    private static extern int get_drivemode(int ID_ref, ref UInt32 mode, ref

```

```

ErrHandle ID_err);
    public int get_drivemode(ref UInt32 mode) { return CheckResult(
get_drivemode(idref, ref mode, ref indraerr)); }

    [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
    private static extern int get_opstate(int ID_ref, ref Byte state, ref
ErrHandle ID_err);
    public int get_opstate(ref Byte state) { return CheckResult(
get_opstate(idref, ref state, ref indraerr)); }

    [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
    private static extern int get_speed(int ID_ref, ref Double speed, ref
ErrHandle ID_err);
    public int get_speed(ref Double speed) { return CheckResult(
get_speed(idref, ref speed, ref indraerr)); }

    [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);
    public int get_diagnostic_msg(Byte[] ID_diagnostic_msg) { return CheckResult(
get_diagnostic_msg(idref, ID_diagnostic_msg, ref indraerr)); }

    [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);
    public int get_diagnostic_num(ref UInt32 ID_diagnostic_num) { return CheckResult(
get_diagnostic_num(idref, ref ID_diagnostic_num, ref indraerr)); }

    [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
    private static extern int clear_error(int ID_ref, ref
ErrHandle ID_err);
    public int clear_error() { return CheckResult(clear_error(idref, ref indraerr
)); }

    // Helpers

    public int CheckResult(int ret)
    {
        if (ret != 0)
        {
            String err = Encoding.ASCII.GetString(indraerr.msg).TrimEnd((Char)0);

            Console.WriteLine(err);
            listBoxerr.Dispatcher.BeginInvoke((System.Windows.Forms.MethodInvoker) (() =>
            {
                listBoxerr.Items.Add(err);
            }));
        }

        return ret;
    }
}

```

1.4.2 Python Examples

```

import sys
import ctypes
from ctypes import cdll
import os

# Minimum Python 3.3 required
assert sys.version_info >= (3,3)

# Load Indradrive API DLL into memory (use absolute or relative path for 'libpath')
libpath = os.path.dirname(__file__) + "\\..\\..\\bin\\IndradriveAPI.dll"
indralib = cdll.LoadLibrary(libpath)

# Error-specific class
class ERR(ctypes.Structure):
    _fields_ = [("code", ctypes.c_int32), ("msg", ctypes.c_char * 2048)]

    def get_msg_str(self):
        return str(self.msg, "UTF-8")

indra_error = ERR(0)

def check_result(result):
    if result:
        print("Error occurred: " + indra_error.get_msg_str())

```

```

        sys.exit(result)

def get_bit(byteval, idx):
    return ((byteval & (1 << idx)) != 0);

# MAIN ENTRY POINT
def main():
    # Getting API reference
    indraref = indralib.init()

    # Opening communication channel
    result = indralib.open(indraref, b"COM1", 19200, ctypes.byref(indra_error))
    check_result(result)

    # Set standard environment
    result = indralib.set_stdenviroment(indraref, ctypes.byref(indra_error))
    check_result(result)

    #
    # Check Drive Mode
    #
    drvmode = ctypes.c_uint32(0)
    result = indralib.get_drivemode(indraref, ctypes.byref(drvmode), ctypes.byref(indra_error))
    check_result(result)

    if drvmode.value != 2: # Drive Mode is not "Speed Control" -> Change it
        input("Please make sure to DISABLE the drive release before continue (stand-by mode)!\n(Press any
            key to continue...)")

    # Activate Speed Control
    result = indralib.speedcontrol_activate(indraref, ctypes.byref(indra_error))
    check_result(result)

    # Diagnostic message
    diagmsg = ctypes.create_string_buffer(256)
    result = indralib.get_diagnostic_msg(indraref, diagmsg, ctypes.byref(indra_error))
    check_result(result)
    print("Current status:\n" + diagmsg.raw.decode('ascii'))

    #
    # Check Operation State
    #
    while True:
        opstate = ctypes.c_uint8(0)
        result = indralib.get_opstate(indraref, ctypes.byref(opstate), ctypes.byref(indra_error))
        check_result(result)

        if (opstate.value & 0b11) != 0b11:
            input("Please make sure to RELEASE before continue (torque-controlled operation mode)!\n(Press
                any key to continue...)")
        else:
            break

    # Set limits
    result = indralib.speedcontrol_init(indraref, ctypes.c_double(10000), ctypes.c_double(1000),
        ctypes.byref(indra_error))
    check_result(result)

    while True:
        speed_str = input("Speed [rpm] = ?")
        if (speed_str == ""): break

        # Set speed
        speed = int(speed_str)
        result = indralib.speedcontrol_write(indraref, ctypes.c_double(speed), ctypes.c_double(10),
            ctypes.byref(indra_error))
        check_result(result)

    # Closing communication channel
    result = indralib.close(indraref, ctypes.byref(indra_error))
    check_result(result)

    return 0

if __name__ == "__main__":
    sys.exit(int(main() or 0))

```

2 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.

3 Namespace Documentation

3.1 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).

3.1.1 Detailed Description

Grouping structs/enums/unions for a SIS Telegram.

3.1.2 Typedef Documentation

3.1.2.1 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.

3.1.2.2 Bytestream

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

Container for Telegram in raw Bytes.

3.1.2.3 Header

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

The Telegram [Header](#) contains all information required for conducting orderly telegram traffic..

3.1.2.4 HeaderExt

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

Extended Telegram [Header](#) to be used for Routing and Sequential Telegrams.

See also

[Header](#)

3.1.3 Enumeration Type Documentation

3.1.3.1 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](#).

3.1.3.2 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](#).

3.1.3.3 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)

Enumerator

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](#).

3.1.3.4 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](#).

3.1.3.5 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](#).

3.1.3.6 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](#).

3.1.3.7 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](#).

3.2 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.

3.2.1 Detailed Description

Grouping unions that merge together both raw and structured information.

3.2.2 Typedef Documentation

3.2.2.1 HeaderControl

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

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

3.2.2.2 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.

3.2.2.3 SercosParamIdent

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

Identification of the parameter. Size: 16 bit.

3.2.2.4 SercosParamAttribute

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

Attribute for a SERCOS parameter that is callable via SercosDatablock.

See also

[SercosDatablock](#)

3.3 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.

3.3.1 Detailed Description

Grouping SIS Telegram Payload struct definitions for commands.

3.3.2 Typedef Documentation

3.3.2.1 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).

3.3.2.2 SercosParam

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

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

3.3.2.3 SercosList

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

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

3.4 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..

3.4.1 Detailed Description

Grouping SIS Telegram Payload struct definitions for reception.

3.4.2 Typedef Documentation

3.4.2.1 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.

3.4.2.2 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.

3.4.2.3 SercosList

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

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

4 Class Documentation

4.1 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.

4.1.1 Detailed Description

Container for Telegram in raw Bytes.

Definition at line [210](#) of file [Telegrams.h](#).

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Bytestream()

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

Default constructor.

Definition at line [216](#) of file [Telegrams.h](#).

4.1.3 Member Function Documentation

4.1.3.1 clear()

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

Clears this object to its blank/initial state.

Definition at line 219 of file [Telegrams.h](#).

4.1.4 Member Data Documentation

4.1.4.1 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](#)

4.2 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.

4.2.1 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](#).

4.2.2 Constructor & Destructor Documentation

4.2.2.1 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](#).

4.2.2.2 Data() [2/5]

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

Constructor.

Parameters

<code>_PayloadData</code>	Single data byte.
---------------------------	-------------------

Definition at line 52 of file [Telegrams.h](#).

4.2.2.3 Data() [3/5]

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

Constructor.

Parameters

<code>_PayloadData</code>	Single data word (2 bytes).
---------------------------	-----------------------------

Definition at line 62 of file [Telegrams.h](#).

4.2.2.4 Data() [4/5]

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

Constructor.

Parameters

<code>_PayloadData</code>	Single data integer (4 bytes).
---------------------------	--------------------------------

Definition at line 73 of file [Telegrams.h](#).

4.2.2.5 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](#).

4.2.3 Member Function Documentation

4.2.3.1 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](#).

4.2.3.2 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](#).

4.2.3.3 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](#).

4.2.3.4 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](#).

4.2.3.5 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](#).

4.2.3.6 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](#).

4.2.3.7 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](#).

4.2.3.8 clear()

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

Clears this object to its blank/initial state.

Definition at line 179 of file [Telegrams.h](#).

4.2.3.9 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](#).

4.2.3.10 get_size()

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

Gets the size.

Returns

The size.

Definition at line 199 of file [Telegrams.h](#).

4.2.3.11 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](#).

4.2.4 Member Data Documentation**4.2.4.1 Bytes**

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

Actual payload Bytes [TGM_SIZEMAX_PAYLOAD].

Definition at line 32 of file [Telegrams.h](#).

4.2.4.2 Size

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

Size of the payload Bytes.

Definition at line 34 of file [Telegrams.h](#).

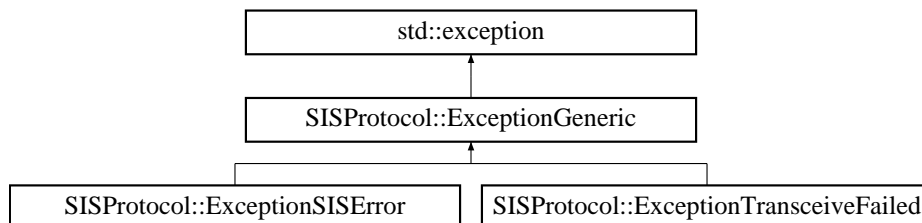
The documentation for this struct was generated from the following file:

- [Telegrams.h](#)

4.3 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](#)

4.3.1 Detailed Description

Generic exceptions for SIS protocol.

See also

[std::exception](#)

Definition at line [151](#) of file [SISProtocol.h](#).

4.3.2 Constructor & Destructor Documentation

4.3.2.1 ExceptionGeneric()

```

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

Definition at line [156](#) of file [SISProtocol.h](#).

4.3.3 Member Function Documentation

4.3.3.1 what()

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

Reimplemented in [SISProtocol::ExceptionSISError](#), and [SISProtocol::ExceptionTransceiveFailed](#).

Definition at line 166 of file [SISProtocol.h](#).

4.3.3.2 get_status()

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

Definition at line 177 of file [SISProtocol.h](#).

4.3.4 Member Data Documentation

4.3.4.1 warning

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

Definition at line 154 of file [SISProtocol.h](#).

4.3.4.2 m_status

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

Definition at line 180 of file [SISProtocol.h](#).

4.3.4.3 m_message

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

Definition at line 182 of file [SISProtocol.h](#).

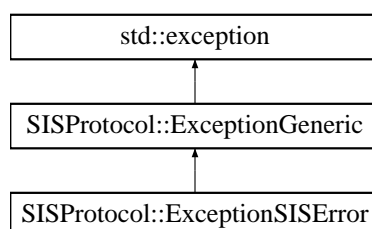
The documentation for this class was generated from the following file:

- [SISProtocol.h](#)

4.4 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

4.4.1 Detailed Description

Specific exception handling of SIS Protocol error codes.

See also

[SISProtocol::ExceptionGeneric](#)

Definition at line 215 of file [SISProtocol.h](#).

4.4.2 Constructor & Destructor Documentation

4.4.2.1 ExceptionSISError()

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

Definition at line 218 of file [SISProtocol.h](#).

4.4.2.2 ~ExceptionSISError()

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

Definition at line 228 of file [SISProtocol.h](#).

4.4.3 Member Function Documentation

4.4.3.1 what()

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

Reimplemented from [SISProtocol::ExceptionGeneric](#).

Definition at line 230 of file [SISProtocol.h](#).

4.4.3.2 `get_errorcode()`

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

Definition at line 241 of file [SISProtocol.h](#).

4.4.4 Member Data Documentation

4.4.4.1 `m_errorcode`

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

Definition at line 244 of file [SISProtocol.h](#).

4.4.4.2 `m_bytestream`

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

Definition at line 245 of file [SISProtocol.h](#).

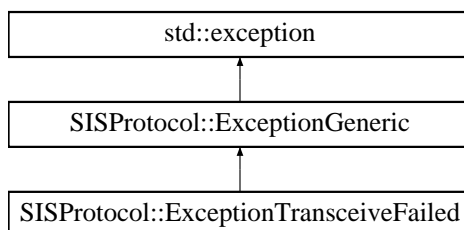
The documentation for this class was generated from the following file:

- [SISProtocol.h](#)

4.5 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

4.5.1 Detailed Description

Specific exception handling of SIS Protocol transceiving failed.

See also

[SISProtocol::ExceptionGeneric](#)

Definition at line 188 of file [SISProtocol.h](#).

4.5.2 Constructor & Destructor Documentation

4.5.2.1 ExceptionTransceiveFailed()

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

Definition at line 191 of file [SISProtocol.h](#).

4.5.2.2 ~ExceptionTransceiveFailed()

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

Definition at line 198 of file [SISProtocol.h](#).

4.5.3 Member Function Documentation

4.5.3.1 what()

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

Reimplemented from [SISProtocol::ExceptionGeneric](#).

Definition at line 200 of file [SISProtocol.h](#).

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

- [SISProtocol.h](#)

4.6 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].

4.6.1 Detailed Description

Generic error handle that is returned from each API function.

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

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

Remarks

Depending on the USE_LABVIEW_ENV switch, the [GenericErrHandle](#) can be replaced by LStrHandle.

Examples:

[apps/WpfApplication1/Indradrive.cs.](#)

Definition at line 27 of file [errors.h](#).

4.6.2 Constructor & Destructor Documentation

4.6.2.1 GenericErrHandle()

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

Constructor.

Parameters

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

Definition at line 38 of file [errors.h](#).

4.6.3 Member Function Documentation

4.6.3.1 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](#).

4.6.3.2 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](#).

4.6.3.3 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](#).

4.6.4 Member Data Documentation

4.6.4.1 code

```
uint32_t GenericErrHandle::code
```

Error code.

Definition at line 30 of file [errors.h](#).

4.6.4.2 msg

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

Error message [2048].

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 32 of file [errors.h](#).

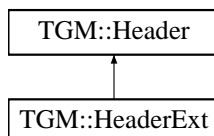
The documentation for this struct was generated from the following file:

- [errors.h](#)

4.7 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:

4.7.1 Detailed Description

The Telegram [Header](#) contains all information required for conducting orderly telegram traffic..

Definition at line 277 of file [Telegrams.h](#).

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Header()

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

Default constructor.

Parameters

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

See also

[TGM::BitFields::HeaderControl](#)

Definition at line 343 of file [Telegrams.h](#).

4.7.3 Member Function Documentation

4.7.3.1 `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

<code><i>exclude_cs</i></code>	(Optional) true to exclude checksum from calculation.
--------------------------------	---

Returns

The sum.

Definition at line 359 of file [Telegrams.h](#).

4.7.3.2 `get_size()`

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

Gets the size.

Returns

The size.

Definition at line 371 of file [Telegrams.h](#).

4.7.3.3 `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

<code><i>_payload_len</i></code>	Length of the payload.
----------------------------------	------------------------

Definition at line 377 of file [Telegrams.h](#).

4.7.3.4 `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](#).

4.7.3.5 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

	<code>_payload_len</code>	Length of the payload.
in	<code>_payload</code>	Bytestream of payload (head + Bytes) with the raw Bytes.

Definition at line 390 of file [Telegrams.h](#).

4.7.4 Member Data Documentation

4.7.4.1 StZ

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

Start symbol: STX (0x02).

Definition at line 280 of file [Telegrams.h](#).

4.7.4.2 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](#).

4.7.4.3 DatL

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

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

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

Definition at line 289 of file [Telegrams.h](#).

4.7.4.4 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](#).

4.7.4.5 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](#).

4.7.4.6 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](#).

4.7.4.7 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](#).

4.7.4.8 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](#)

4.8 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.

4.8.1 Detailed Description

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

Definition at line 106 of file [Telegrams_Bitfields.h](#).

4.8.2 Constructor & Destructor Documentation

4.8.2.1 HeaderControl()

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

Constructor.

Parameters

type	(Optional) Header type, represented by HeaderType.
----------------------	--

See also

[HeaderType](#)

Definition at line 153 of file [Telegrams_Bitfields.h](#).

4.8.3 Member Data Documentation

4.8.3.1 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](#).

4.8.3.2 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](#).

4.8.3.3 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](#).

4.8.3.4 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](#).

4.8.3.5 Bits

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

4.8.3.6 Value

```
BYTE TGM::BitFields::HeaderControl::Value
```

Representation of the raw value.

Definition at line 145 of file [Telegrams_Bitfields.h](#).

4.8.3.7 "@7

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

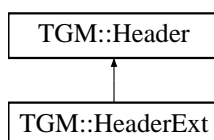
The documentation for this struct was generated from the following file:

- [Telegrams_Bitfields.h](#)

4.9 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

4.9.1 Detailed Description

Extended Telegram [Header](#) to be used for Routing and Sequential Telegrams.

See also

[Header](#)

Definition at line 411 of file [Telegrams.h](#).

4.9.2 Member Data Documentation

4.9.2.1 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](#).

4.9.2.2 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](#).

4.9.2.3 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](#).

4.9.2.4 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](#).

4.9.2.5 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](#).

4.9.2.6 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](#).

4.9.2.7 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](#).

4.9.2.8 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](#)

4.10 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](#)

4.10.1 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](#).

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Map()

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

Default constructor.

Parameters

in	<i>_header</i>	(Optional) The Telegram header.
in	<i>_payload</i>	(Optional) The Telegram payload.

Definition at line [258](#) of file [Telegrams.h](#).

4.10.2.2 ~Map()

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

Destructor.

Definition at line [262](#) of file [Telegrams.h](#).

4.10.3 Member Function Documentation

4.10.3.1 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	<i>_header</i>	The Telegram header.
in	<i>_payload</i>	The Telegram payload.

Definition at line 268 of file [Telegrams.h](#).

4.10.4 Member Data Documentation

4.10.4.1 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](#).

4.10.4.2 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](#)

4.11 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.

4.11.1 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](#).

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Mapping()

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

Constructor.

Parameters

in	<code>_header</code>	The Telegram header.
in	<code>_payload</code>	The Telegram payload.

Definition at line 247 of file [Telegrams.h](#).

4.11.3 Member Data Documentation

4.11.3.1 Header

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

The Telegram header.

Definition at line 239 of file [Telegrams.h](#).

4.11.3.2 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](#)

4.12 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.
};

4.12.1 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](#).

4.12.2 Constructor & Destructor Documentation

4.12.2.1 OPSTATE()

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

Constructor.

Parameters

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

Definition at line 107 of file [Wrapper.h](#).

4.12.3 Member Data Documentation

4.12.3.1 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](#).

4.12.3.2 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](#).

4.12.3.3 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](#).

4.12.3.4 Bits

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

4.12.3.5 Value

```
uint8_t OPSTATE::Value
```

Raw and unstructured data value.

Definition at line 101 of file [Wrapper.h](#).

4.12.3.6 "@15

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

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

- [Wrapper.h](#)

4.13 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.

4.13.1 Detailed Description

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

Definition at line 740 of file [Telegrams.h](#).

4.13.2 Constructor & Destructor Documentation

4.13.2.1 SercosList()

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

Default constructor.

Definition at line 761 of file [Telegrams.h](#).

4.13.3 Member Function Documentation

4.13.3.1 clear()

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

Clears this object to its blank/initial state.

Definition at line 769 of file [Telegrams.h](#).

4.13.3.2 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](#).

4.13.3.3 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](#).

4.13.4 Member Data Documentation

4.13.4.1 Status

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

Recipient status.

Definition at line 743 of file [Telegrams.h](#).

4.13.4.2 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](#).

4.13.4.3 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](#).

4.13.4.4 Bytes

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

Definition at line 756 of file [Telegrams.h](#).

4.13.4.5 Error

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

Definition at line 757 of file [Telegrams.h](#).

4.13.4.6 "@5

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

Payload Bytes, or error byte.

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

- [Telegrams.h](#)

4.14 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.

4.14.1 Detailed Description

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

Definition at line 554 of file [Telegrams.h](#).

4.14.2 Constructor & Destructor Documentation

4.14.2.1 SercosList()

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

Constructor.

Parameters

<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](#).

4.14.3 Member Function Documentation

4.14.3.1 clear()

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

Clears this object to its blank/initial state.

Definition at line 607 of file [Telegrams.h](#).

4.14.3.2 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](#).

4.14.3.3 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](#).

4.14.4 Member Data Documentation

4.14.4.1 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](#).

4.14.4.2 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](#).

4.14.4.3 ParamType

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

Definition at line 564 of file [Telegrams.h](#).

4.14.4.4 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](#).

4.14.4.5 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](#).

4.14.4.6 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](#).

4.14.4.7 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](#)

4.15 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.

4.15.1 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](#).

4.15.2 Constructor & Destructor Documentation

4.15.2.1 SercosParam()

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

Default constructor.

Definition at line [709](#) of file [Telegrams.h](#).

4.15.3 Member Function Documentation

4.15.3.1 clear()

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

Clears this object to its blank/initial state.

Definition at line 717 of file [Telegrams.h](#).

4.15.3.2 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](#).

4.15.3.3 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](#).

4.15.4 Member Data Documentation

4.15.4.1 Status

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

Recipient status.

Definition at line 691 of file [Telegrams.h](#).

4.15.4.2 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](#).

4.15.4.3 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](#).

4.15.4.4 Bytes

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

Definition at line 704 of file [Telegrams.h](#).

4.15.4.5 Error

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

Definition at line 705 of file [Telegrams.h](#).

4.15.4.6 "@3

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

Payload Bytes, or error byte.

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

- [Telegrams.h](#)

4.16 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.

4.16.1 Detailed Description

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

Definition at line 491 of file [Telegrams.h](#).

4.16.2 Constructor & Destructor Documentation

4.16.2.1 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](#).

4.16.3 Member Function Documentation

4.16.3.1 clear()

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

Clears this object to its blank/initial state.

Definition at line 529 of file [Telegrams.h](#).

4.16.3.2 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](#).

4.16.3.3 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](#).

4.16.4 Member Data Documentation

4.16.4.1 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](#).

4.16.4.2 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](#).

4.16.4.3 ParamType

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

Definition at line [501](#) of file [Telegrams.h](#).

4.16.4.4 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](#).

4.16.4.5 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](#)

4.17 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 SercosDataLen to Bytes.
 - [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 operation Bytes.
 - 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.

4.17.1 Detailed Description

Attribute for a SERCOS parameter that is callable via SercosDatablock.

See also

[SercosDatablock](#)

Definition at line 250 of file [Telegrams_Bitfields.h](#).

4.17.2 Constructor & Destructor Documentation

4.17.2.1 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](#).

4.17.3 Member Data Documentation

4.17.3.1 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](#).

4.17.3.2 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](#).

4.17.3.3 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](#).

4.17.3.4 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](#).

4.17.3.5 res5

UINT32 TGM::Bitfields::SercosParamAttribute::res5

Bit 23 of Reception Telegram's payload.

Definition at line 274 of file [Telegrams_Bitfields.h](#).

4.17.3.6 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](#).

4.17.3.7 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](#).

4.17.3.8 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](#).

4.17.3.9 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](#).

4.17.3.10 res10

UINT32 TGM::BitFields::SercosParamAttribute::res10

Bit 31 of Reception Telegram's payload.

Definition at line 291 of file [Telegrams_Bitfields.h](#).

4.17.3.11 Bits

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

4.17.3.12 Value

UINT32 TGM::BitFields::SercosParamAttribute::Value

Raw data value.

Definition at line 309 of file [Telegrams_Bitfields.h](#).

4.17.3.13 "@13

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

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

- [Telegrams_Bitfields.h](#)

4.18 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.*

4.18.1 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](#).

4.18.2 Constructor & Destructor Documentation

4.18.2.1 SercosParamControl() [1/2]

```
TGM::Bitfields::SercosParamControl::SercosParamControl (
    SercosDatablock datablock = Datablock\_OperationData ) [inline]
```

Constructor.

Parameters

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

Definition at line 196 of file [Telegrams_Bitfields.h](#).

4.18.2.2 SercosParamControl() [2/2]

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

Constructor.

Parameters

value	Raw byte data of the Control Byte.
-----------------------	------------------------------------

Definition at line 201 of file [Telegrams_Bitfields.h](#).

4.18.3 Member Data Documentation

4.18.3.1 res1

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

Definition at line 165 of file [Telegrams_Bitfields.h](#).

4.18.3.2 res2

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

Definition at line 166 of file [Telegrams_Bitfields.h](#).

4.18.3.3 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](#).

4.18.3.4 Datablock

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

SERCOS parameter datablock, represented by SercosDatablock.

Definition at line 174 of file [Telegrams_Bitfields.h](#).

4.18.3.5 res6

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

Definition at line 176 of file [Telegrams_Bitfields.h](#).

4.18.3.6 res7

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

Definition at line 177 of file [Telegrams_Bitfields.h](#).

4.18.3.7 Bits

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

4.18.3.8 Value

BYTE TGM::Bitfields::SercosParamControl::Value

Representation of the raw value.

Definition at line 190 of file [Telegrams_Bitfields.h](#).

4.18.3.9 "@9

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

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

- [Telegrams_Bitfields.h](#)

4.19 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](#)
};

4.19.1 Detailed Description

Identification of the parameter. Size: 16 bit.

Definition at line 206 of file [Telegrams_Bitfields.h](#).

4.19.2 Constructor & Destructor Documentation

4.19.2.1 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](#).

4.19.3 Member Data Documentation**4.19.3.1 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](#).

4.19.3.2 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](#).

4.19.3.3 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](#).

4.19.3.4 Bits

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

4.19.3.5 Value

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

Definition at line 234 of file [Telegrams_Bitfields.h](#).

4.19.3.6 "@11

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

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

- [Telegrams_Bitfields.h](#)

4.20 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_SIS_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<DOUBLE _rcvdelm)
- void [execute_command](#) ([TGM::SercosParamVar](#) _paramvar, USHORT _paramnum)

4.20.1 Detailed Description

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

Definition at line 29 of file [SISProtocol.h](#).

4.20.2 Member Typedef Documentation

4.20.2.1 SIS_SERVICES

```
typedef enum SISProtocol::SIS\_SERVICES SISProtocol::SIS\_SERVICES
```

Values that represent identifiers to be used for SIS services.

4.20.2.2 BAUDRATE

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

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

4.20.3 Member Enumeration Documentation

4.20.3.1 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 40 of file [SISProtocol.h](#).

4.20.3.2 BAUDRATE

enum [SISProtocol::BAUDRATE](#)

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

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 54 of file [SISProtocol.h](#).

4.20.4 Constructor & Destructor Documentation

4.20.4.1 SISProtocol()

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

Default constructor.

Definition at line 5 of file [SISProtocol.cpp](#).

4.20.4.2 ~SISProtocol()

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

Destructor.

Definition at line 10 of file [SISProtocol.cpp](#).

4.20.5 Member Function Documentation

4.20.5.1 open()

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

Definition at line 15 of file [SISProtocol.cpp](#).

4.20.5.2 close()

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

Definition at line 46 of file [SISProtocol.cpp](#).

4.20.5.3 set_baudrate()

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

Definition at line 61 of file [SISProtocol.cpp](#).

4.20.5.4 read_parameter() [1/4]

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

Definition at line 87 of file [SISProtocol.cpp](#).

4.20.5.5 read_parameter() [2/4]

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

Definition at line 108 of file [SISProtocol.cpp](#).

4.20.5.6 read_parameter() [3/4]

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

Definition at line 129 of file [SISProtocol.cpp](#).

4.20.5.7 read_parameter() [4/4]

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

Definition at line 150 of file [SISProtocol.cpp](#).

4.20.5.8 read_listelm() [1/3]

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

Definition at line 164 of file [SISProtocol.cpp](#).

4.20.5.9 read_listelm() [2/3]

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

Definition at line 188 of file [SISProtocol.cpp](#).

4.20.5.10 read_listelm() [3/3]

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

Definition at line 212 of file [SISProtocol.cpp](#).

4.20.5.11 write_parameter() [1/3]

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

Definition at line 267 of file [SISProtocol.cpp](#).

4.20.5.12 write_parameter() [2/3]

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

Definition at line 274 of file [SISProtocol.cpp](#).

4.20.5.13 write_parameter() [3/3]

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

Definition at line 281 of file [SISProtocol.cpp](#).

4.20.5.14 write_listelm() [1/3]

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

Definition at line 305 of file [SISProtocol.cpp](#).

4.20.5.15 write_listelm() [2/3]

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

Definition at line 313 of file [SISProtocol.cpp](#).

4.20.5.16 write_listelm() [3/3]

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

Definition at line 321 of file [SISProtocol.cpp](#).

4.20.5.17 execute_command()

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

Definition at line 347 of file [SISProtocol.cpp](#).

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

- [SISProtocol.h](#)
- [SISProtocol.cpp](#)

4.21 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.
};

4.21.1 Detailed Description

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

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

Definition at line 116 of file [Wrapper.h](#).

4.21.2 Constructor & Destructor Documentation

4.21.2.1 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](#).

4.21.3 Member Data Documentation

4.21.3.1 `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](#).

4.21.3.2 `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](#).

4.21.3.3 `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](#).

4.21.3.4 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](#).

4.21.3.5 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](#).

4.21.3.6 res7

```
uint16_t SPEEDUNITS::res7
```

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

Definition at line 148 of file [Wrapper.h](#).

4.21.3.7 Bits

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

4.21.3.8 Value

```
uint16_t SPEEDUNITS::Value
```

Raw and unstructured data value.

Definition at line 170 of file [Wrapper.h](#).

4.21.3.9 "@17

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

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

- [Wrapper.h](#)

4.22 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.

4.22.1 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](#).

4.22.2 Constructor & Destructor Documentation

4.22.2.1 Subservice()

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

Default constructor.

Definition at line [655](#) of file [Telegrams.h](#).

4.22.3 Member Function Documentation

4.22.3.1 clear()

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

Clears this object to its blank/initial state.

Definition at line 663 of file [Telegrams.h](#).

4.22.3.2 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](#).

4.22.3.3 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](#).

4.22.4 Member Data Documentation

4.22.4.1 Status

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

Recipient status.

Definition at line 639 of file [Telegrams.h](#).

4.22.4.2 RecipientAddr

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

Address of the recipient.

Definition at line 642 of file [Telegrams.h](#).

4.22.4.3 ServiceNumber

BYTE TGM::Reactions::Subservice::ServiceNumber

SIS service number.

Definition at line 645 of file [Telegrams.h](#).

4.22.4.4 Bytes

Data TGM::Reactions::Subservice::Bytes

Definition at line 650 of file [Telegrams.h](#).

4.22.4.5 Error

BYTE TGM::Reactions::Subservice::Error

Definition at line 651 of file [Telegrams.h](#).

4.22.4.6 "@1

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

Payload Bytes, or error byte.

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

- [Telegrams.h](#)

4.23 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.

4.23.1 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](#).

4.23.2 Constructor & Destructor Documentation

4.23.2.1 Subservice()

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

Constructor.

Parameters

<code>_addr</code>	(Optional) The recipient address.
<code>_subservice</code>	(Optional) The subservice number.
<code>_PayloadData</code>	(Optional) The data.

Definition at line 463 of file [Telegrams.h](#).

4.23.3 Member Function Documentation

4.23.3.1 clear()

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

Clears this object to its blank/initial state.

Definition at line 473 of file [Telegrams.h](#).

4.23.3.2 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](#).

4.23.3.3 `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](#).

4.23.4 Member Data Documentation

4.23.4.1 `RecipientAddr`

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

The recipient address.

Definition at line [452](#) of file [Telegrams.h](#).

4.23.4.2 `ServiceNumber`

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

The subservice number.

Definition at line [454](#) of file [Telegrams.h](#).

4.23.4.3 `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](#)

5 File Documentation

5.1 `errors.h` File Reference

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

Classes

- struct [GenericErrHandle](#)
Generic error handle that is returned from each API function.

Macros

- `#define Err_Base (0x0)`
A macro that defines Error base that is used for coding the final error code.

Typedefs

- typedef struct [GenericErrHandle](#) [GenericErrHandle](#)
Generic error handle that is returned from each API function.
- typedef [GenericErrHandle](#) * [ErrHandle](#)
Defines an alias representing a pointer to [GenericErrHandle](#).
- typedef enum [EErrorBlocks](#) [EErrorBlocks](#)
Values that represent error blocks to be used as `block_code` paramater for [set_error\(\)](#) function.

Enumerations

- enum [EErrorBlocks](#) {
[Err_NoError](#) = 0, [Err_Block_OpenByCOM](#) = 1, [Err_Block_Close](#) = 2, [Err_Block_Test](#) = 3,
[Err_Block_SeqInit](#) = 6, [Err_Block_SeqWrite](#) = 7, [Err_Block_VelCInit](#) = 8, [Err_Block_VelCWrite](#) = 9,
[Err_Block_GetStatus](#) = 10, [Err_Block_SetControl](#) = 11, [Err_Invalid_Pointer](#) = 12 }
Values that represent error blocks to be used as `block_code` paramater for [set_error\(\)](#) function.

Functions

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

5.1.1 Detailed Description

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

Definition in file [errors.h](#).

5.1.2 Macro Definition Documentation

5.1.2.1 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](#).

5.1.3 Typedef Documentation

5.1.3.1 GenericErrHandle

```
typedef struct GenericErrHandle GenericErrHandle
```

Generic error handle that is returned from each API function.

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

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

Remarks

Depending on the USE_LABVIEW_ENV switch, the [GenericErrHandle](#) can be replaced by LStrHandle.

5.1.3.2 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](#).

5.1.3.3 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\(\)](#)

5.1.4 Enumeration Type Documentation

5.1.4.1 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](#).

5.1.5 Function Documentation

5.1.5.1 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: $0 \ll 8 \mid \text{block_code} \ll 4 \mid \text{issue_code}$, 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](#).

5.2 errors.h

```

00001
00004 #ifndef LABVIEW_ERRORS_H
00005 #define LABVIEW_ERRORS_H
00006
00007 #include <cstring>
00008 #include <string>
00009
00010
00011 #ifndef 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 #ifndef USE_LABVIEW_ENV
00073 typedef LStrHandle ErrHandle;
00077 #else
00078 typedef GenericErrHandle* ErrHandle;
00082 #endif
00083
00084
00085 #ifndef USE_LABVIEW_ENV
00086 #pragma pack(push,1)
00087 #include "extcode.h"
00088 #pragma pack(pop)
00089 #endif
00090
00091 #ifndef 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 #ifndef USE_LABVIEW_ENV
00105 #define Err_Base (0x08EF)
00109 #else
00110 #define Err_Base (0x0)
00114 #endif
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 {
00149     //Initializes the buffer
00150     MgErr err = NumericArrayResize(uB, 1, (UHandle*)&lvhandle, str.length());
00151     if (err) return err;
00152
00153     //Informs the LabVIEW string handle about the size of the size
00154     (**lvhandle).cnt = str.length();
00155
00156     //Fills the string buffer with str
00157     strcpy((char*)(**lvhandle).str, str.c_str());
00158
00159     return noErr;
00160 }
00161 #endif
00162
00163 inline static int32_t set_error(ErrHandle errhdl, std::string errstr, int32_t block_code
00164 , int32_t issue_code = 1)
00165 {
00166     int32_t retcode = (Err_Base << 8) | (block_code << 4) | issue_code;
00167
00168     #ifdef USE_LABVIEW_ENV
00169         write_string(errhdl, errstr);
00170     #else
00171         errhdl->set(retcode, errstr.c_str());
00172     #endif
00173
00174     return retcode;
00175 }
00176
00177 #endif // LABVIEW_ERRORS_H
00178

```

5.3 mainpage.dox File Reference

5.4 SISProtocol.cpp File Reference

5.5 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

```

```

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 void SISProtocol::close()
00047 {
00048     STACK;
00049
00050     try
00051     {
00052         m_serial.Close();
00053     }
00054     catch (CSerial::ExceptionGeneric &ex)
00055     {
00056         throw;
00057     }
00058 }
00059
00060
00061 void SISProtocol::set_baudrate(BAUDRATE baudrate)
00062 {
00063     STACK;
00064
00065     // Build Telegrams ...
00066
00067     // Mapping for SEND Telegram
00068     TGM::Map<TGM::Header, TGM::Commands::Subservice>
00069         tx_tgm(
00070             // Init header
00071             TGM::Header(SIS_ADDR_MASTER,
00072                 SIS_ADDR_SLAVE, SIS_SERVICE_INIT_COMM,
00073                 TGM::Bitfields::HeaderControl(TGM::TypeCommand)),
00074             // Init payload
00075             TGM::Commands::Subservice(SIS_ADDR_UNIT, 0x07,
00076                 TGM::Data({ (BYTE)baudrate })))
00077 );
00078
00079     // Mapping for RECEPTION Telegram
00080     TGM::Map<TGM::Header, TGM::Reactions::Subservice>
00081         rx_tgm;
00082
00083     // Set payload size
00084     tx_tgm.Mapping.Header.set_DatL(tx_tgm.Mapping.Payload.get_size());
00085
00086     // Transceive ...
00087     transceiving(tx_tgm, rx_tgm);
00088 }
00089
00090 void SISProtocol::read_parameter(TGM::SercosParamVar
00091     _paramvar, USHORT _paramnum, UINT32 & _rcvddata)
00092 {
00093     STACK;
00094
00095     // Fetching attributes for length and scale ...
00096     size_t datalen = 1;
00097     UINT8 scalefactor = 0;
00098     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00099
00100     // Communication with Telegrams ...
00101     BYTE service = SIS_SERVICE_SERCOS_PARAM_READ;
00102
00103     auto rx_tgm = transceive_param
00104         <TGM::Header, TGM::Commands::SercosParam,
00105         TGM::Header, TGM::Reactions::SercosParam>
00106         (_paramvar, _paramnum, service);
00107
00108     // Convert responded Bytes ...
00109     INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
00110     _rcvddata = static_cast<UINT32>(response);
00111 }
00112
00113 void SISProtocol::read_parameter(TGM::SercosParamVar
00114     _paramvar, USHORT _paramnum, UINT64 & _rcvddata)
00115 {
00116     STACK;
00117
00118     // Fetching attributes for length and scale ...

```

```

00113     size_t datalen = 1;
00114     UINT8 scalefactor = 0;
00115     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00116
00117     // Communication with Telegrams ...
00118     BYTE service = SIS_SERVICE_SERCOS_PARAM_READ;
00119
00120     auto rx_tgm = transceive_param
00121         <TGM::Header, TGM::Commands::SercosParam,
00122         TGM::Header, TGM::Reactions::SercosParam>
00123         (_paramvar, _paramnum, service);
00124
00125     // Convert responded Bytes ...
00126     INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
00127     _rcvddata = static_cast<UINT64>(response);
00128 }
00129 void SISProtocol::read_parameter(TGM::SercosParamVar
00130     _paramvar, USHORT _paramnum, DOUBLE & _rcvddata)
00131 {
00132     STACK;
00133
00134     // Fetching attributes for length and scale ...
00135     size_t datalen = 1;
00136     UINT8 scalefactor = 0;
00137     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00138
00139     // Communication with Telegrams ...
00140     BYTE service = SIS_SERVICE_SERCOS_PARAM_READ;
00141
00142     auto rx_tgm = transceive_param
00143         <TGM::Header, TGM::Commands::SercosParam,
00144         TGM::Header, TGM::Reactions::SercosParam>
00145         (_paramvar, _paramnum, service);
00146
00147     // Convert responded Bytes ...
00148     INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
00149     _rcvddata = (double)response / std::pow(10, scalefactor);
00150 }
00151 void SISProtocol::read_parameter(TGM::SercosParamVar
00152     _paramvar, USHORT _paramnum, char _rcvddata[TGM_SIZEMAX_PAYLOAD])
00153 {
00154     // Communication with Telegrams ...
00155     BYTE service = SIS_SERVICE_SERCOS_PARAM_READ;
00156
00157     auto rx_tgm = transceive_param
00158         <TGM::Header, TGM::Commands::SercosParam,
00159         TGM::Header, TGM::Reactions::SercosParam>
00160         (_paramvar, _paramnum, service);
00161
00162     // Convert responded Bytes ...
00163     memcpy(_rcvddata, (char*)rx_tgm.Mapping.Payload.Bytes, rx_tgm.Mapping.Payload.Bytes.Size);
00164     _rcvddata[rx_tgm.Mapping.Payload.Bytes.Size] = '\0';
00165 }
00166 void SISProtocol::read_listelm(TGM::SercosParamVar _paramvar,
00167     USHORT _paramnum, USHORT _elm_pos, UINT32 & _rcvdelm)
00168 {
00169     STACK;
00170
00171     // Fetching attributes for length and scale ...
00172     size_t datalen = 1;
00173     UINT8 scalefactor = 0;
00174     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00175
00176     USHORT SegmentSize = (USHORT)datalen;
00177     USHORT ListOffset = _elm_pos * SegmentSize;
00178
00179     // Communication with Telegrams ...
00180     BYTE service = SIS_SERVICE_SERCOS_LIST_WRITE;
00181
00182     auto rx_tgm = transceive_list
00183         <TGM::Header, TGM::Commands::SercosList,
00184         TGM::Header, TGM::Reactions::SercosList>
00185         (_paramvar, _paramnum, service, SegmentSize, ListOffset);
00186
00187     // Response Bytes ...
00188     INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
00189     _rcvdelm = static_cast<UINT32>(response);
00190 }
00191 void SISProtocol::read_listelm(TGM::SercosParamVar _paramvar,
00192     USHORT _paramnum, USHORT _elm_pos, UINT64 & _rcvdelm)
00193 {
00194     STACK;

```

```

00192 // Fetching attributes for length and scale ...
00193 size_t datalen = 1;
00194 UINT8 scalefactor = 0;
00195 get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00196
00197 USHORT SegmentSize = (USHORT)datalen;
00198 USHORT ListOffset = _elm_pos * SegmentSize;
00199
00200 // Communication with Telegrams ...
00201 BYTE service = SIS_SERVICE_SERCOS_LIST_WRITE;
00202
00203 auto rx_tgm = transceive_list
00204     <TGM::Header, TGM::Commands::SercosList,
TGM::Header, TGM::Reactions::SercosList>
00205     (_paramvar, _paramnum, service, SegmentSize, ListOffset);
00206
00207 // Response Bytes ...
00208 INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
00209 _rcvdelm = static_cast<UINT64>(response);
00210 }
00211
00212 void SISProtocol::read_listelm(TGM::SercosParamVar _paramvar,
    USHORT _paramnum, USHORT _elm_pos, DOUBLE & _rcvdelm)
00213 {
00214     STACK;
00215
00216     // Fetching attributes for length and scale ...
00217     size_t datalen = 1;
00218     UINT8 scalefactor = 0;
00219     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00220
00221     USHORT SegmentSize = (USHORT)datalen;
00222     USHORT ListOffset = _elm_pos * SegmentSize;
00223
00224     // Communication with Telegrams ...
00225     BYTE service = SIS_SERVICE_SERCOS_LIST_WRITE;
00226
00227     auto rx_tgm = transceive_list
00228         <TGM::Header, TGM::Commands::SercosList,
TGM::Header, TGM::Reactions::SercosList>
00229         (_paramvar, _paramnum, service, SegmentSize, ListOffset);
00230
00231     // Response Bytes ...
00232     INT64 response = get_sized_data(rx_tgm.Mapping.Payload.Bytes, datalen);
00233     _rcvdelm = (double)response / std::pow(10, scalefactor);
00234 }
00235
00236
00237 INT64 SISProtocol::get_sized_data(TGM::Data& rx_data, const size_t &datalen)
00238 {
00239     STACK;
00240
00241     if (datalen == 1)
00242     {
00243         UINT8 Bytes = rx_data.toUINT8();
00244         UINT64 mask = ((Bytes >> 7) & 1) ? 0xFFFFFFFFFFFFFFF0 : 0;
00245         return (INT64)(Bytes | mask);
00246     }
00247     else if (datalen == 2)
00248     {
00249         UINT16 Bytes = rx_data.toUINT16();
00250         UINT64 mask = ((Bytes >> 15) & 1) ? 0xFFFFFFFFFFFF0000 : 0;
00251         return (INT64)(Bytes | mask);
00252     }
00253     else if (datalen == 8)
00254     {
00255         return (INT64)rx_data.toUINT64();
00256     }
00257     else
00258     {
00259         UINT32 Bytes = rx_data.toUINT32();
00260         UINT64 mask = ((Bytes >> 31) & 1) ? 0xFFFFFFFF00000000 : 0;
00261         return (INT64)(Bytes | mask);
00262     }
00263 }
00264
00265
00266
00267 void SISProtocol::write_parameter(
    TGM::SercosParamVar _paramvar, USHORT _paramnum, const UINT32 _data)
00268 {
00269     STACK;
00270
00271     write_parameter(_paramvar, _paramnum, static_cast<DOUBLE>(_data));
00272 }
00273
00274 void SISProtocol::write_parameter(

```

```

    TGM::SercosParamVar _paramvar, USHORT _paramnum, const UINT64 _data)
00275 {
00276     STACK;
00277
00278     write_parameter(_paramvar, _paramnum, static_cast<DOUBLE>(_data));
00279 }
00280
00281 void SISProtocol::write_parameter(
    TGM::SercosParamVar _paramvar, USHORT _paramnum, const DOUBLE _data)
00282 {
00283     STACK;
00284
00285     // Fetching attributes for length and scale ...
00286     size_t datalen = 1;
00287     UINT8 scalefactor = 0;
00288     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00289
00290     // Preprocess Bytes ...
00291     UINT64 inval = static_cast<UINT64>(_data * std::pow(10, scalefactor));
00292
00293     TGM::Data Bytes;
00294     set_sized_data(Bytes, datalen, inval);
00295
00296     // Communication with Telegrams ...
00297     BYTE service = SIS_SERVICE_SERCOS_PARAM_WRITE;
00298
00299     transceive_param
00300         <TGM::Header, TGM::Commands::SercosParam,
    TGM::Header, TGM::Reactions::SercosParam>
    (_paramvar, _paramnum, service, &Bytes);
00301 }
00302
00303
00304
00305 void SISProtocol::write_listelm(TGM::SercosParamVar _paramvar,
    USHORT _paramnum, USHORT _elm_pos, const UINT32 _rcvdelm)
00306 {
00307     STACK;
00308
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,
    USHORT _paramnum, USHORT _elm_pos, const UINT64 _rcvdelm)
00314 {
00315     STACK;
00316
00317     DOUBLE buf = static_cast<DOUBLE>(_rcvdelm);
00318     write_listelm(_paramvar, _paramnum, _elm_pos, buf);
00319 }
00320
00321 void SISProtocol::write_listelm(TGM::SercosParamVar _paramvar,
    USHORT _paramnum, USHORT _elm_pos, const DOUBLE _rcvdelm)
00322 {
00323     STACK;
00324
00325     // Fetching attributes for length and scale ...
00326     size_t datalen = 1;
00327     UINT8 scalefactor = 0;
00328     get_parameter_attributes(_paramvar, _paramnum, scalefactor, datalen);
00329
00330     UINT64 inval = static_cast<UINT64>(_rcvdelm * std::pow(10, scalefactor));
00331
00332     TGM::Data Bytes;
00333     set_sized_data(Bytes, datalen, inval);
00334
00335     USHORT SegmentSize = (USHORT)datalen;
00336     USHORT ListOffset = _elm_pos * SegmentSize;
00337
00338     // Communication with Telegrams ...
00339     BYTE service = SIS_SERVICE_SERCOS_LIST_WRITE;
00340
00341     transceive_list
00342         <TGM::Header, TGM::Commands::SercosList,
    TGM::Header, TGM::Reactions::SercosList>
    (_paramvar, _paramnum, service, SegmentSize, ListOffset, &Bytes);
00343 }
00344
00345
00346
00347 void SISProtocol::execute_command(
    TGM::SercosParamVar _paramvar, USHORT _paramnum)
00348 {
00349     TGM::SercosCommandrequest cmd;
00350     TGM::SercosCommandstatus Status =
    TGM::Commandstatus_Busy;
00351     int iterations;
00352

```

```

00353 // Start command ...
00354 cmd = TGM::Commandrequest_Set;
00355 try
00356 {
00357     write_parameter(_paramvar, _paramnum, static_cast<UINT64>(cmd));
00358 }
00359 catch (SISProtocol::ExceptionSISError &ex)
00360 {
00361     if (ex.get_errorcode() == 0x700C)
00362     {
00363         throw SISProtocol::ExceptionGeneric(-1, "Command cannot be
executed, because it is write-protected. Release the drive torque (disable drive), or restart the Indradrive
system.");
00364     }
00365     else
00366     {
00367         throw;
00368     }
00369     iterations = 0;
00370     do
00371     {
00372         get_parameter_status(_paramvar, _paramnum, Status);
00373         if (iterations > 300) throw ExceptionGeneric(-1, "Command execution caused a
continuous busy loop. Please restart the Indradrive system.");
00374     } while (Status == TGM::Commandstatus_Busy);
00375     if (Status != TGM::Commandstatus_OK)
00376     {
00377         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));
00378     }
00379 // Delete command ...
00380 cmd = TGM::Commandrequest_NotSet;
00381 write_parameter(_paramvar, _paramnum, static_cast<UINT64>(cmd));
00382 Status = TGM::Commandstatus_Busy;
00383 iterations = 0;
00384 do
00385 {
00386     get_parameter_status(_paramvar, _paramnum, Status);
00387     if (iterations > 300) throw ExceptionGeneric(-1, "Command execution caused a
continuous busy loop. Please restart the Indradrive system.");
00388 } while (Status == TGM::Commandstatus_Busy);
00389 if (Status != TGM::Commandstatus_NotSet)
00390 {
00391     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));
00392 }
00393 }
00394 void SISProtocol::get_parameter_status(const TGM::SercosParamVar _paramvar, const USHORT
& _paramnum, TGM::SercosCommandstatus& _datastatus)
00395 {
00396     STACK;
00397     // Communication with Telegrams ...
00398     BYTE service = SIS_SERVICE_SERCOS_PARAM_WRITE;
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     // Read back Datablock ...
00403     _datastatus = static_cast<TGM::SercosCommandstatus>(rx_tgm.Mapping.Payload.
Bytes.toUINT8());
00404 }
00405 void SISProtocol::set_sized_data(TGM::Data& tx_data, const size_t &datalen, UINT64 & _rcvdelm)
00406 {
00407     STACK;
00408     if (datalen == 1) tx_data = TGM::Data((UINT8)_rcvdelm);
00409     else if (datalen == 2) tx_data = TGM::Data((UINT16)_rcvdelm);
00410     else if (datalen == 4) tx_data = TGM::Data((UINT32)_rcvdelm);
00411     else if (datalen == 8) tx_data = TGM::Data((UINT64)_rcvdelm);
00412     else tx_data = TGM::Data((UINT8&)_rcvdelm);
00413 }
00414 template <class TCHeader, class TCPayload, class TRHeader, class TRPayload>
00415 TGM::Map<TRHeader, TRPayload> SISProtocol::transceive_param(
TGM::SercosParamVar _paramvar, const USHORT &_paramnum, BYTE _service,

```



```

TGM::Data const * const _data, TGM::SercosDatablock _attribute)
00427 {
00428     // Build Telegrams ...
00429     TGM::Bitfields::SercosParamControl    ParamControl(_attribute);
00430     TGM::Bitfields::SercosParamIdent      ParamIdent(_paramvar, _paramnum);
00431
00432     // Mapping for SEND Telegram
00433     TGM::Map<TCHheader, TCPayload>
00434         tx_tgm(
00435             // Init header
00436             TCHheader(SIS_ADDR_MASTER, SIS_ADDR_SLAVE, _service,
00437 TGM::Bitfields::HeaderControl(TGM::TypeCommand)),
00438             // Init payload
00439             TCPayload(ParamControl, SIS_ADDR_SLAVE, ParamIdent, *_data)
00440         );
00441
00442     // Set payload size
00443     tx_tgm.Mapping.Header.set_DatL(tx_tgm.Mapping.Payload.get_size());
00444
00445     // Calculate Checksum
00446     tx_tgm.Mapping.Header.calc_checksum(&tx_tgm.raw);
00447
00448     if (!check_boundaries(tx_tgm))
00449         throw SISProtocol::ExceptionGeneric(-1, "Boundaries are out of spec.
Telegram is not ready to be sent.");
00450
00451     // Mapping for RECEPTION Telegram
00452     TGM::Map<TRHeader, TRPayload> rx_tgm;
00453
00454     // Transceive ...
00455     // Send and receive
00456     transceiving< TCHheader, TCPayload,
00457 TRHeader, TRPayload >
00458         (tx_tgm, rx_tgm);
00459     return rx_tgm;
00460 }
00461
00462 template<class TCHheader, class TCPayload, class TRHeader, class TRPayload>
00463 TGM::Map<TRHeader, TRPayload> SISProtocol::transceive_list(
00464     TGM::SercosParamVar _paramvar, const USHORT & _paramnum, BYTE _service, USHORT &
00465     _element_size, USHORT & _list_offset, TGM::Data const * const _data,
00466     TGM::SercosDatablock _attribute)
00467 {
00468     // Build Telegrams ...
00469     TGM::Bitfields::SercosParamControl    sercos_control(_attribute);
00470     TGM::Bitfields::SercosParamIdent      ParamNum(_paramvar, _paramnum);
00471
00472     // Mapping for SEND Telegram
00473     TGM::Map<TCHheader, TCPayload>
00474         tx_tgm(
00475             // Init header
00476             TCHheader(SIS_ADDR_MASTER, SIS_ADDR_SLAVE, _service,
00477 TGM::Bitfields::HeaderControl(TGM::TypeCommand)),
00478             // Init payload
00479             TCPayload(sercos_control, SIS_ADDR_SLAVE, ParamNum, _list_offset, _element_size,
*_data)
00480         );
00481
00482     // Set payload size
00483     tx_tgm.Mapping.Header.set_DatL(tx_tgm.Mapping.Payload.get_size());
00484
00485     // Calculate Checksum
00486     tx_tgm.Mapping.Header.calc_checksum(&tx_tgm.raw);
00487
00488     if (!check_boundaries(tx_tgm))
00489         throw SISProtocol::ExceptionGeneric(-1, "Boundaries are out of spec.
Telegram is not ready to be sent.");
00490
00491     // Mapping for RECEPTION Telegram
00492     TGM::Map<TRHeader, TRPayload> rx_tgm;
00493
00494     // Transceive ...
00495     // Send and receive
00496     transceiving< TCHheader, TCPayload,
00497 TRHeader, TRPayload >
00498         (tx_tgm, rx_tgm);
00499     return rx_tgm;
00500 }
00501
00502 void SISProtocol::get_parameter_attributes(TGM::SercosParamVar _paramvar, const USHORT &
00503     _paramnum, UINT8& _scalefactor, size_t& _datalen)
00504 {
00505     STACK;

```

```

00503
00504 // Communication with Telegrams ...
00505 BYTE service = SIS_SERVICE_SERCOS_PARAM_READ;
00506
00507 auto rx_tgm = transceive_param
00508     <TGM::Header, TGM::Commands::SercosParam,
TGM::Header, TGM::Reactions::SercosParam>
00509     (_paramvar, _paramnum, service, new TGM::Data(),
TGM::Datablock_Attribute);
00510
00511 // Read back Datablock ...
00512 UINT32 attr = rx_tgm.Mapping.Payload.Bytes.toUINT32();
00513 TGM::Bitfields::SercosParamAttribute sercos_attribute(attr);
00514
00515 _datalen = 1;
00516 if (sercos_attribute.Bits.DataLen == TGM::Datalen_2ByteList) _datalen = 2;
00517 if (sercos_attribute.Bits.DataLen == TGM::Datalen_4ByteList) _datalen = 4;
00518 if (sercos_attribute.Bits.DataLen == TGM::Datalen_8ByteList) _datalen = 8;
00519 if (sercos_attribute.Bits.DataLen == TGM::Datalen_2ByteParam) _datalen = 2;
00520 if (sercos_attribute.Bits.DataLen == TGM::Datalen_4ByteParam) _datalen = 4;
00521 if (sercos_attribute.Bits.DataLen == TGM::Datalen_8ByteParam) _datalen = 8;
00522
00523 _scalefactor = 0xFF & sercos_attribute.Bits.ScaleFactor;
00524 }
00525
00526
00527 template <class TCHeader, class TCPayload, class TRHeader, class TRPayload>
00528 void SISProtocol::transceiving(TGM::Map<TCHeader, TCPayload>& tx_tgm,
TGM::Map<TRHeader, TRPayload>& rx_tgm)
00529 {
00530     STACK;
00531
00532     // Lock mutex to set the semaphore, so that the SIS access be reentrant
00533     mutex_sis.lock();
00534
00535     // Transceiver lengths
00536     size_t tx_payload_len = tx_tgm.Mapping.Payload.get_size();
00537     size_t tx_header_len = tx_tgm.Mapping.Header.get_size();
00538
00539     // Receiver lengths
00540     size_t rx_header_len = rx_tgm.Mapping.Header.get_size();
00541     size_t rx_payload_len = 0;
00542
00543     // Clear buffers
00544     m_serial.Purge();
00545
00546     // Write ...
00547     m_serial.Write(tx_tgm.raw.Bytes, tx_header_len + tx_payload_len);
00548
00549     // Read ...
00550     bool bContd = true;
00551     DWORD rcvd_cur = 0;
00552     DWORD rcvd_rcnt = 0;
00553
00554     do
00555     {
00556         // Wait for an event
00557         m_serial.WaitEvent(0, RS232_READ_TIMEOUT);
00558
00559         // Save event
00560         const CSerial::EEvent event = m_serial.GetEventType();
00561
00562         // Handle Break event
00563         if (event & CSerial::EEventBreak)
00564             throw SISProtocol::ExceptionTransceiveFailed(
CSerial::EEventBreak, "Break event occurred. Transceive has been aborted.", true);
00565
00566         // Handle error event
00567         if (event & CSerial::EEventError)
00568             throw_rs232_error_events(m_serial.GetError());
00569
00570         // Handle Bytes receive event
00571         if (event & CSerial::EEventRecv)
00572         {
00573             // Read header Bytes
00574             m_serial.Read(rx_tgm.raw.Bytes + rcvd_rcnt, RS232_BUFFER - rcvd_rcnt, &rcvd_cur,
0, RS232_READ_TIMEOUT);
00575
00576             // Loop back if nothing received
00577             if (rcvd_cur == 0) continue;
00578
00579             // Hold back number of already received bytes
00580             rcvd_rcnt += rcvd_cur;
00581
00582             // It is assumed that if the number of received bytes is bigger than 4,
00583             // which is the position of the payload length, the length can be read out.
00584             if (rcvd_rcnt > 4)

```

```

00585         {
00586             rx_payload_len = rx_tgm.Mapping.Header.DatL;
00587             rx_tgm.Mapping.Payload.Bytes.set_size(rx_payload_len - rx_tgm.
Mapping.Payload.get_head_size());
00588         }
00589
00590         // Length of payload is zero --> No payload received
00591         if (rx_payload_len == 0)
00592         {
00593             std::string tx_hexstream = hexprint_bytestream(tx_tgm.raw.Bytes, tx_header_len +
tx_payload_len);
00594             std::string rx_hexstream = hexprint_bytestream(rx_tgm.raw.Bytes, rx_header_len);
00595             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);
00596         }
00597
00598
00599         // Complete Telegram received
00600         if (rx_header_len + rx_payload_len <= rcvd_rcnt)
00601         {
00602             if (rx_tgm.Mapping.Payload.Status)
00603             {
00604                 std::string tx_hexstream = hexprint_bytestream(tx_tgm.raw.Bytes, tx_header_len +
tx_payload_len);
00605                 //std::string rx_hexstream = hexprint_bytestream(rx_tgm.raw.bytes, rx_header_len +
rx_payload_len);
00606                 throw SISProtocol::ExceptionSISError(rx_tgm.
Mapping.Payload.Status, rx_tgm.Mapping.Payload.Error, tx_hexstream);
00607             }
00608
00609             bContd = false;
00610         }
00611     }
00612     } while (bContd);
00613
00614     // Unlock mutex to unset the semaphore
00615     mutex_sis.unlock();
00616 }
00617
00618
00619
00620 template<class THeader, class TPayload>
00621 bool SISProtocol::check_boundaries(TGM::Map<THeader, TPayload> & _tgm)
00622 {
00623     STACK;
00624
00625     size_t tgm_size = _tgm.Mapping.Header.get_size() + _tgm.Mapping.
Payload.get_size();
00626     if (tgm_size <= RS232_BUFFER) return true;
00627
00628     return false;
00629 }
00630
00631
00632 std::string SISProtocol::hexprint_bytestream(const BYTE * _bytestream, const size_t _len)
00633 {
00634     STACK;
00635
00636     std::string buf;
00637
00638     for (size_t i = 0; i < _len; i++)
00639         buf.append(sformat("%02X ", (BYTE)_bytestream[i]));
00640
00641     return buf;
00642 }
00643
00644 void SISProtocol::throw_rs232_error_events(CSerial::EError _err)
00645 {
00646     STACK;
00647
00648     switch (_err)
00649     {
00650     case CSerial::EErrorBreak:
00651         throw SISProtocol::ExceptionTransceiveFailed(
CSerial::EErrorBreak, "Break condition occurred. Transceive has been aborted.", true);
00652
00653     case CSerial::EErrorFrame:
00654         throw SISProtocol::ExceptionTransceiveFailed(
CSerial::EErrorFrame, "Framing error occurred. Transceive has been aborted.", true);
00655
00656     case CSerial::EErrorIOE:
00657         throw SISProtocol::ExceptionTransceiveFailed(
CSerial::EErrorIOE, "IO device error occurred. Transceive has been aborted.", true);
00658
00659     case CSerial::EErrorMode:
00660         throw SISProtocol::ExceptionTransceiveFailed(

```

```

        CSerial::EErrorMode, "Unsupported mode detected. Transceive has been aborted.", true);
00661
00662     case CSerial::EErrorOverrun:
00663         throw SISProtocol::ExceptionTransceiveFailed(
        CSerial::EErrorOverrun, "Buffer overrun detected. Transceive has been aborted.", true);
00664
00665     case CSerial::EErrorRxOver:
00666         throw SISProtocol::ExceptionTransceiveFailed(
        CSerial::EErrorRxOver, "Input buffer overflow detected. Transceive has been aborted.", true);
00667
00668     case CSerial::EErrorParity:
00669         throw SISProtocol::ExceptionTransceiveFailed(
        CSerial::EErrorParity, "Input parity occurred. Transceive has been aborted.", true);
00670
00671     case CSerial::EErrorTxFull:
00672         throw SISProtocol::ExceptionTransceiveFailed(
        CSerial::EErrorTxFull, "Output buffer full. Transceive has been aborted.", true);
00673
00674     default:
00675         throw SISProtocol::ExceptionTransceiveFailed(
        CSerial::EErrorBreak, "Unknown error occurred. Transceive has been aborted.", true);
00676     }
00677 }
00678
00679

```

5.6 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.

5.6.1 Macro Definition Documentation

5.6.1.1 RS232_BUFFER

```
#define RS232_BUFFER 254
```

Definition at line 15 of file [SISProtocol.h](#).

5.6.1.2 RS232_READ_LOOPS_MAX

```
#define RS232_READ_LOOPS_MAX 100
```

Definition at line 16 of file [SISProtocol.h](#).

5.6.1.3 RS232_READ_TIMEOUT

```
#define RS232_READ_TIMEOUT 1000
```

Definition at line 17 of file [SISProtocol.h](#).

5.6.1.4 SIS_ADDR_MASTER

```
#define SIS_ADDR_MASTER 0x00
```

Defines address master.

Definition at line 21 of file [SISProtocol.h](#).

5.6.1.5 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](#).

5.6.1.6 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](#).

5.7 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 class SISProtocol
00030 {
00031 public:
00032     class ExceptionGeneric;
00033     class ExceptionTransceiveFailed;
00034     class ExceptionSISError;
00035
00036     typedef enum SIS_SERVICES
00037     {
00038         SIS_SERVICE_INIT_COMM = 0x03,
00039         SIS_SERVICE_SEQUENTIALOP = 0x04,
00040
00041         SIS_SERVICE_SERCOS_PARAM_READ = 0x10,
00042         SIS_SERVICE_SERCOS_LIST_READ = 0x11,
00043         SIS_SERVICE_SERCOS_READ_PHASE = 0x12,
00044         SIS_SERVICE_SERCOS_SWITCH_PHASE = 0x1D,
00045         SIS_SERVICE_SERCOS_LIST_WRITE = 0x1E,
00046         SIS_SERVICE_SERCOS_PARAM_WRITE = 0x1F
00047     } SIS_SERVICES;
00048
00049     typedef enum BAUDRATE
00050     {
00051         Baud_9600 = 0b00000000,
00052         Baud_19200 = 0b00000001,
00053         Baud_38400 = 0b00000010,
00054         Baud_57600 = 0b00000100,
00055         Baud_115200 = 0b00001000
00056     } BAUDRATE;
00057
00058     SISProtocol();
00059     virtual ~SISProtocol();
00060
00061     void open(const wchar_t * _port = L"COM1");
00062     void close();
00063
00064     void set_baudrate(BAUDRATE baudrate);
00065
00066     void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00067         UINT32& _rcvddata);
00068     void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00069         UINT64& _rcvddata);
00070     void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00071         DOUBLE& _rcvddata);
00072     void read_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00073         char _rcvddata[TGM::SIZEMAX_PAYLOAD]);
00074
00075     void read_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00076         USHORT _elm_pos, UINT32& _rcvdelm);
00077     void read_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00078         USHORT _elm_pos, UINT64& _rcvdelm);
00079     void read_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00080         USHORT _elm_pos, DOUBLE& _rcvdelm);
00081
00082     void write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00083         const UINT32 _data);
00084     void write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00085         const UINT64 _data);
00086     void write_parameter(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00087         const DOUBLE _data);
00088
00089     void write_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00090         USHORT _elm_pos, const UINT32 _rcvdelm);
00091     void write_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00092         USHORT _elm_pos, const UINT64 _rcvdelm);
00093     void write_listelm(TGM::SercosParamVar _paramvar, USHORT _paramnum,
00094         USHORT _elm_pos, const DOUBLE _rcvdelm);
00095
00096     void execute_command(TGM::SercosParamVar _paramvar, USHORT _paramnum)
00097     ;
00098
00099 private:
00100
00101     inline void get_parameter_attributes(TGM::SercosParamVar _paramvar, const USHORT &
00102         _paramnum, UINT8& _scalefactor, size_t& _datalen);
00103     inline void get_parameter_status(const TGM::SercosParamVar _paramvar, const USHORT &
00104         _paramnum, TGM::SercosCommandstatus& _datastatus);
00105
00106     template <class TCHheader, class TCPayload, class TRHeader, class TRPayload>
00107     TGM::Map<TRHeader, TRPayload> transceive_param(
00108         TGM::SercosParamVar _paramvar, const USHORT & _paramnum, BYTE _service,
00109         TGM::Data const * const _data = new TGM::Data(),
00110         TGM::SercosDatablock _attribute =
00111         TGM::Datablock_OperationData);
00112
00113

```

```

00124     template <class TCHheader, class TCPayload, class TRHeader, class TRPayload>
00125     TGM::Map<TRHeader, TRPayload> transceive_list(
TGM::SercosParamVar _paramvar, const USHORT &_paramnum, BYTE _service, USHORT &
_element_size, USHORT & _list_offset, TGM::Data const * const _data = new
TGM::Data(), TGM::SercosDatablock _attribute =
TGM::Datablock_OperationData);
00126
00127     template <class THeader, class TPayload>
00128     inline bool check_boundaries(TGM::Map<THeader, TPayload>& _tgm);
00129
00130     static std::string hexprint_bytestream(const BYTE * _bytestream, const size_t _len);
00131
00132     inline INT64 get_sized_data(TGM::Data& rx_data, const size_t &datalen);
00133     inline void set_sized_data(TGM::Data& tx_data, const size_t &datalen, UINT64 & _rcvdelm);
00134
00135 private:
00136
00137     template <class TCHheader, class TCPayload, class TRHeader, class TRPayload>
00138     void transceiving(TGM::Map<TCHheader, TCPayload>& tx_tgm,
TGM::Map<TRHeader, TRPayload>& rx_tgm);
00139
00140     static void throw_rs232_error_events(CSerial::EError _err);
00141
00142 private:
00143     CSerial m_serial;
00144
00145     std::mutex mutex_sis;
00146 };
00147
00151 class SISProtocol::ExceptionGeneric : public std::exception
00152 {
00153 public:
00154     bool warning;
00155
00156     ExceptionGeneric(
00157         int _status,
00158         const std::string _trace_log,
00159         bool _warning = false) :
00160
00161         m_status(_status),
00162         m_message(_trace_log),
00163         warning(_warning)
00164     {}
00165
00166     virtual const char* what() const throw ()
00167     {
00168 #ifdef NDEBBUG
00169         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()));
00170 #else
00171         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()));
00172         OutputDebugStringA((LPCSTR)ex);
00173         return ex;
00174 #endif
00175     }
00176
00177     int get_status() { return m_status; }
00178
00179 protected:
00180     int m_status;
00181
00182     std::string m_message;
00183 };
00184
00188 class SISProtocol::ExceptionTransceiveFailed : public
SISProtocol::ExceptionGeneric
00189 {
00190 public:
00191     ExceptionTransceiveFailed(
00192         int _status,
00193         const std::string _message,
00194         bool _warning = false) :
00195
00196         ExceptionGeneric(_status, _message, _warning)
00197     {}
00198     ~ExceptionTransceiveFailed() throw() {}
00199
00200     virtual const char* what() const throw ()
00201     {
00202 #ifdef NDEBBUG
00203         return str2char(sformat("SIS Protocol reception fail caused: STATUS=0x%04x (%d) ### MESSAGE='%s'",
m_status, m_status, m_message.c_str()));
00204 #else
00205         const char* ex = str2char(sformat("SIS Protocol reception fail caused: STATUS=0x%04x (%d) ###
MESSAGE='%s'", m_status, m_status, m_message.c_str()));

```

```

00206         OutputDebugStringA( (LPCSTR)ex);
00207         return ex;
00208 #endif
00209     }
00210 };
00211
00215 class SISProtocol::ExceptionSISError : public
    SISProtocol::ExceptionGeneric
00216 {
00217 public:
00218     ExceptionSISError(
00219         int _status,
00220         int _code,
00221         const std::string _bytestream,
00222         bool _warning = false) :
00223
00224         ExceptionGeneric(_status, std::string(), _warning),
00225         m_errorcode(_code),
00226         m_bytestream(_bytestream)
00227     {}
00228     ~ExceptionSISError() throw() {}
00229
00230     virtual const char* what() const throw ()
00231     {
00232 #ifdef NDEBUG
00233         return str2char(sformat("(Return code: %d) SIS Protocol Error code returned has been received:
00234         0x%04X.\nOriginal Telegram bytestream: %s", m_status, m_errorcode,
00235         m_bytestream.c_str()));
00236 #else
00237         const char* ex = str2char(sformat("(Return code: %d) SIS Protocol Error code returned has been
00238         received: 0x%04X.\nOriginal Telegram bytestream: %s", m_status, m_errorcode,
00239         m_bytestream.c_str()));
00240         OutputDebugStringA( (LPCSTR)ex);
00241         return ex;
00242 #endif
00243     }
00244     int get_errorcode() { return m_errorcode; }
00245
00246 protected:
00247     int m_errorcode;
00248     std::string m_bytestream;
00249 };
00250 #endif /* _SISPROTOCOL_H_ */

```

5.8 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..

5.8.1 Detailed Description

Contains struct definitions for different types of Telegrams.

Definition in file [Telegrams.h](#).

5.8.2 Macro Definition Documentation

5.8.2.1 TGM_SIZE_HEADER

```
#define TGM_SIZE_HEADER 8
```

Definition at line 17 of file [Telegrams.h](#).

5.8.2.2 TGM_SIZE_HEADER_EXT

```
#define TGM_SIZE_HEADER_EXT 16
```

Definition at line 18 of file [Telegrams.h](#).

5.8.2.3 TGM_SIZEMAX_PAYLOAD

```
#define TGM_SIZEMAX_PAYLOAD 246
```

Definition at line 19 of file [Telegrams.h](#).

5.8.2.4 TGM_SIZEMAX

```
#define TGM_SIZEMAX 254
```

Definition at line 20 of file [Telegrams.h](#).

5.9 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
00046         Size = _data.size();
00047     }
00048
00052     Data(UINT8 _data)
00053     {
00054         clear();
00055
00056         operator<<(_data);
00057     }
00058
00062     Data(UINT16 _data)
00063     {
00064         clear();
00065
00066         operator<<(_data & 0xFF);
00067         operator<<((_data & 0xFF00) >> 8);
00068     }
00069
00073     Data(UINT32 _data)
00074     {
00075         clear();
00076
00077         operator<<(_data & 0xFF);
00078         operator<<((_data & 0xFF00) >> 8);
00079         operator<<((_data & 0xFF0000) >> 16);
00080         operator<<((_data & 0xFF000000) >> 24);
00081     }
00082
00086     Data(UINT64 _data)
00087     {
00088         clear();
00089
00090         operator<<(_data & 0xFF);
00091         operator<<((_data & 0xFF00) >> 8);
00092         operator<<((_data & 0xFF0000) >> 16);
00093         operator<<((_data & 0xFF000000) >> 24);
00094         operator<<((_data & 0xFF00000000) >> 32);
00095         operator<<((_data & 0xFF0000000000) >> 40);
00096         operator<<((_data & 0xFF000000000000) >> 48);
00097         operator<<((_data & 0xFF00000000000000) >> 54);
00098     }
00099
00105     BYTE at(UINT32 _idx)
00106     {
00107         return Bytes[_idx];
00108     }
00109
00113     std::vector<BYTE> toVector()
00114     {
00115         std::vector<BYTE> out;
00116
00117         for (int i = 0; i < Size; i++)
00118             out.push_back(Bytes[i]);
00119
00120         return out;
00121     }
00122
00126     UINT64 toUINT64()
00127     {
00128         UINT64 out = 0;
00129
00130         for (int i = 0; i < std::min<size_t>(Size, 8); i++)
00131             out |= Bytes[i] << (i * 8);
00132
00133         return out;
00134     }
00135
00139     UINT32 toUINT32()
00140     {
00141         UINT32 out = 0;
00142
00143         for (int i = 0; i < std::min<size_t>(Size, 4); i++)
00144             out |= Bytes[i] << (i * 8);
00145
00146         return out;
00147     }
00148
00152     UINT16 toUINT16()
00153     {

```

```

00154         UINT16 out = 0;
00155
00156         for (int i = 0; i < std::min<size_t>(Size, 2); i++)
00157             out |= Bytes[i] << (i * 8);
00158
00159         return out;
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_SIZE_MAX];
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     }
00403 } Header;
00404 #pragma pack(pop)
00405
00406 #pragma pack(push,1)
00407 typedef struct HeaderExt : Header
00408 {
00412     {
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     } HeaderExt;
00438 #pragma pack(pop)
00439
00440 namespace Commands
00441 {
00443 #pragma pack(push,1)
00446 typedef struct Subservice

```

```

00450     {
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)
00490 typedef struct SercosParam
00492 {
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,
00519         TGM::Bitfields::SercosParamIdent _param_ident =
00520         TGM::Bitfields::SercosParamIdent(),
00520         TGM::Data _data = Data() ) :
00521         Control(_control.Value),
00522         UnitAddr(_unit_addr),
00523         ParamType(0),
00524         ParamNum(_param_ident.Value),
00525         Bytes(_data)
00526     {}
00527
00529     void clear()
00530     {
00531         Control = 0;
00532         UnitAddr = 0;
00533         ParamType = 0;
00534         ParamNum = 0;
00535         Bytes.clear();
00536     }
00537
00541     size_t get_head_size() { return 5; }
00542
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)
00553 typedef struct SercosList
00555 {
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 =

```

```

TGM::Bitfields::SercosParamControl(),
00591     BYTE _unit_addr = 0,
00592     TGM::Bitfields::SercosParamIdent _ParamIdent =
TGM::Bitfields::SercosParamIdent(),
00593     USHORT _ListOffset = 0,
00594     USHORT _SegmentSize = 0,
00595     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
00607 void clear()
00608 {
00609     Control = UnitAddr = ParamNum = ListOffset = SegmentSize = 0;
00610     Bytes.clear();
00611 }
00612
00616 size_t get_head_size() { return 9; }
00617
00621 size_t get_size() { return get_head_size() + Bytes.get_size(); }
00622
00623 } SercosList;
00624 #pragma pack(pop)
00625 }
00626
00627
00628
00630 namespace Reactions
00631 {
00632 #pragma pack(push,1)
00633 typedef struct Subservice
00634 {
00639     BYTE    Status;
00640
00642     BYTE    RecipientAddr;
00643
00645     BYTE    ServiceNumber;
00646
00648     union
00649     {
00650         Data    Bytes;
00651         BYTE    Error;
00652     };
00653
00655     Subservice() :
00656         Status(1),
00657         RecipientAddr(0),
00658         ServiceNumber(0),
00659         Error(0)
00660 {}
00661
00663 void clear()
00664 {
00665     Status = 1;
00666     RecipientAddr = ServiceNumber = 0;
00667     Bytes.clear();
00668 }
00669
00673 size_t get_head_size() { return 3; }
00674
00678 size_t get_size() { return get_head_size() + Bytes.get_size(); }
00679
00680 } Subservice;
00681 #pragma pack(pop)
00682
00683
00684 #pragma pack(push,1)
00685 typedef struct SercosParam
00686 {
00691     BYTE Status;
00692
00694     BYTE Control;
00695
00699     BYTE UnitAddr;
00700
00702     union
00703     {
00704         Data    Bytes;
00705         USHORT  Error;
00706     };
00707

```

```

00709         SercosParam() :
00710             Status(1),
00711             Control(0),
00712             UnitAddr(0),
00713             Bytes(TGM::Data())
00714         {}
00715
00716         void clear()
00717         {
00718             Status = 1;
00719             Control = UnitAddr = 0;
00720             Bytes.clear();
00721         }
00722
00723         size_t get_head_size() { return 3; }
00724
00725         size_t get_size() { return get_head_size() + Bytes.get_size(); }
00726
00727     } SercosParam;
00728 #pragma pack(pop)
00729
00730 #pragma pack(push,1)
00731     typedef struct SercosList
00732     {
00733         BYTE Status;
00734
00735         BYTE Control;
00736
00737         BYTE UnitAddr;
00738
00739         union
00740         {
00741             Data Bytes;
00742             USHORT Error;
00743         };
00744
00745         SercosList() :
00746             Status(1),
00747             Control(0),
00748             UnitAddr(0),
00749             Bytes(TGM::Data())
00750         {}
00751
00752         void clear()
00753         {
00754             Status = 1;
00755             Control = UnitAddr = 0;
00756             Bytes.clear();
00757         }
00758
00759         size_t get_head_size() { return 3; }
00760
00761         size_t get_size() { return get_head_size() + Bytes.get_size(); }
00762
00763     } SercosList;
00764 #pragma pack(pop)
00765 }
00766
00767 #endif /* _TELEGRAMS_H */

```

5.10 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).

5.10.1 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](#).

5.11 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_

```

5.12 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 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 Indrdrive status.*
- `int32_t get_diagnostic_num (SISProtocol *ID_ref, uint32_t *ID_diagnostic_num, ErrHandle ID_err)`
- Gets diagnostic number of the current Indrdrive status.*
- `int32_t clear_error (SISProtocol *ID_ref, ErrHandle ID_err)`
- Clears a latched error in the Indrdrive 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)`

5.12.1 Detailed Description

Implementation of API functions that are exported to the API DLL.

Definition in file [Wrapper.cpp](#).

5.12.2 Function Documentation

5.12.2.1 init()

```
SISProtocol* init ( )
```

Creates API reference.

The API references is a fundamental prerequisite.

Remarks

This function is exported to the Indradrive API DLL.

Refer to [Examples](#) for detailed code examples.

How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int init();
```

.

How to call with Python:

```
indraref = indralib.init()
```

.

Returns

API reference. Pointer can be casted and treated as UINT32 (see examples).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 7 of file [Wrapper.cpp](#).

5.12.2.2 open()

```
int32_t open (
    SISProtocol * ID_ref,
    const wchar_t * ID_comport = L"COM1",
    uint32_t ID_combaudrate = 19200,
    ErrHandle ID_err = ErrHandle() )
```

Opens the communication port to the Indradrive device.

Attention

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

Remarks

This function is exported to the Indradrive API DLL.

Refer to [Examples](#) for detailed code examples.

How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int open(int ID_ref, Byte[] ID_comport, UInt32 ID_combaudrate, ref
    ErrHandle ID_err);
```

.

How to call with Python:

```
result = indralib.open(indraref, b"COM1", 19200, ctypes.byref(indra_error))
```

.

Parameters

in	<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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 14 of file [Wrapper.cpp](#).

5.12.2.3 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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 38 of file [Wrapper.cpp](#).

5.12.2.4 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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 65 of file [Wrapper.cpp](#).

5.12.2.5 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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 91 of file [Wrapper.cpp](#).

5.12.2.6 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].
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 <i>ID_speeds</i> , <i>ID_accels</i> , etc).
out	<i>ID_err</i>	(Optional) Error handle.

Returns

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

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 129 of file [Wrapper.cpp](#).

5.12.2.7 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.

By special PLC software (if configured), the Indradrive can be triggered by both software trigger and hardware trigger. The hardware trigger is realized through a 24V rising edge input line.

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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 186 of file [Wrapper.cpp](#).

5.12.2.8 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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 233 of file [Wrapper.cpp](#).

5.12.2.9 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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 259 of file [Wrapper.cpp](#).

5.12.2.10 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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 291 of file [Wrapper.cpp](#).

5.12.2.11 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 [325](#) of file [Wrapper.cpp](#).

5.12.2.12 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.
out	<i>ID_err</i>	(Optional) Error handle.

Returns

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

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 351 of file [Wrapper.cpp](#).

5.12.2.13 get_opstate()

```
int32_t get_opstate (
    SISProtocol * ID_ref,
    uint8_t * ID_opstate,
    ErrHandle ID_err = ErrHandle() )
```

Retrieve information about the operation states: bb, Ab, or AF.

The operation state feedback is provided by *ID_opstate* parameter. The following table depicts the coding:

If	Then
(*ID_opstate & 0b11) == 0b00	Control section / power section not ready for operation(e.g., drive error or phase 2)
(*ID_opstate & 0b11) == 0b01	Control section ready for operation "bb"
(*ID_opstate & 0b11) == 0b10	Control section and power section ready for op. "Ab"
(*ID_opstate & 0b11) == 0b11	Drive with torque "AF".
((*ID_opstate & 0b100) >> 2) == 1	Drive Halt is active and axis is in standstill
((*ID_opstate & 0b1000) >> 3) == 1	Drive error.

Remarks

This function is exported to the Indradrive API DLL.
Refer to [Examples](#) for detailed code examples.
How to call with C#:

```
[DllImport(&lt;path_to_DLL&gt;;, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_opstate(int ID_ref, ref Byte state, ref
    ErrHandle ID_err);
```

How to call with Python:

```
opstate = ctypes.c_uint8(0)
result = indralib.get_opstate(indraref, ctypes.byref(opstate), ctypes.byref(indra_error))
```

Parameters

in	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 391 of file [Wrapper.cpp](#).

5.12.2.14 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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 421 of file [Wrapper.cpp](#).

5.12.2.15 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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 450 of file [Wrapper.cpp](#).

5.12.2.16 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\(\)](#)).

Examples:

[apps/WpfApplication1/Indradrive.cs](#).

Definition at line 479 of file [Wrapper.cpp](#).

5.12.2.17 clear_error()

```
int32_t clear_error (
    SISProtocol * ID_ref,
    ErrHandle ID_err = ErrHandle() )
```

Clears a latched error in the Indrdrive device.

In case of error that has been occurred on the Indrdrive, the error information is latched until cleared with this call.

Remarks

This function is exported to the Indrdrive 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()`).

Examples:

[apps/WpfApplication1/Indrdrive.cs](#).

Definition at line 508 of file [Wrapper.cpp](#).

5.12.2.18 change_opmode()

```
void change_opmode (
    SISProtocol * ID_ref,
    const uint64_t opmode )
```

Definition at line 534 of file [Wrapper.cpp](#).

5.12.2.19 get_units()

```
SPEEDUNITS get_units (
    SISProtocol * ID_ref ) [inline]
```

Definition at line 556 of file [Wrapper.cpp](#).

5.12.2.20 change_units()

```
void change_units (
    SISProtocol * ID_ref )
```

Definition at line 566 of file [Wrapper.cpp](#).

5.12.2.21 change_language()

```
void change_language (
    SISProtocol * ID_ref,
    const uint8_t lang_code ) [inline]
```

Definition at line 578 of file [Wrapper.cpp](#).

5.13 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, sprintf("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()),
00083             Err_Block_SeqInit);
00084     }
00085     catch (CSerial::ExceptionGeneric &ex)
00086     {
00087         return set_error(ID_err, char2str(ex.what()), Err_Block_SeqInit);
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(
00096             ID_err, sprintf("Reference pointing to invalid location '%p'.", ID_ref),
00097             Err_Invalid_Pointer);
00098
00099     try
00100     {
00101         // Set required units (preferred scaling, rotary scaling, [rpm])
00102         change_units(ID_ref);
00103
00104         // Max Acceleration (S-0-0138)
00105         ID_ref->write_parameter(TGM::SercosParamS, 138, ID_max_accel);
00106
00107         // Max Jerk (S-0-0349)
00108         ID_ref->write_parameter(TGM::SercosParamS, 349, ID_max_jerk);
00109
00110         // SPS Global Register G1 (P-0-1371) - Reset Read Trigger
00111         ID_ref->write_parameter(TGM::SercosParamP, 1371,
00112             static_cast<uint32_t>(0));
00113
00114         // SPS Global Register G2 (P-0-1372) - Reset Sequencer Trigger
00115         ID_ref->write_parameter(TGM::SercosParamP, 1372,
00116             static_cast<uint32_t>(0));
00117
00118         return Err_NoError;
00119     }
00120     catch (SISProtocol::ExceptionGeneric &ex)
00121     {
00122         return set_error(ID_err, char2str(ex.what()),
00123             Err_Block_SeqInit);
00124     }
00125     catch (CSerial::ExceptionGeneric &ex)
00126     {
00127         return set_error(ID_err, char2str(ex.what()), Err_Block_SeqInit);
00128     }
00129 }
00130
00131 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)
00132 {
00133     if (!dynamic_cast<SISProtocol*>(ID_ref))
00134         // Return error for wrong reference
00135         return set_error(

```

```

00134         ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00135         Err_Invalid_Pointer);
00136
00137     try
00138     {
00139
00140         for (uint16_t i = 0; i < ID_set_length; i++)
00141         {
00142             // Speed in min^-1 (P-0-4007)
00143             ID_ref->write_listelm(TGM::SercosParamP, 4007, i + 1, abs(
00144                 ID_speeds[i]));
00145             // Acceleration in rad/s^2 (P-0-4008)
00146             ID_ref->write_listelm(TGM::SercosParamP, 4008, i + 1, ID_accels[i]
00147         });
00148             // Deceleration in rad/s^2 (P-0-4063)
00149             ID_ref->write_listelm(TGM::SercosParamP, 4063, i + 1, ID_accels[i]
00150         });
00151             // Jerk in rad/s^3 (P-0-4009)
00152             ID_ref->write_listelm(TGM::SercosParamP, 4009, i + 1, ID_jerks[i]
00153         );
00154             // Mode (P-0-4019)
00155             ID_ref->write_listelm(TGM::SercosParamP, 4019, i + 1,
00156                 static_cast<uint32_t>(0b10000000 | ((stde::sgn<double_t>(ID_speeds[i]) == 1 ? 0b10 : 0b01) << 2)));
00157             // Pos (P-0-4006)
00158             ID_ref->write_listelm(TGM::SercosParamP, 4006, i + 1,
00159                 static_cast<uint64_t>(0));
00160             // Wait (P-0-4018)
00161             ID_ref->write_listelm(TGM::SercosParamP, 4018, i + 1,
00162                 static_cast<uint64_t>(0));
00163             // Delay (P-0-4063)
00164             ID_ref->write_listelm(TGM::SercosParamP, 4063, i + 1,
00165                 static_cast<uint64_t>(0));
00166             // Timers in cs (P-0-1389)
00167             ID_ref->write_listelm(TGM::SercosParamP, 1389, i + 1, ID_delays[i]
00168         });
00169         }
00170         // Time triggers for cam (P-0-1370)
00171         ID_ref->write_parameter(TGM::SercosParamP, 1370,
00172             static_cast<uint32_t>(ID_set_length));
00173         return Err_NoError;
00174     }
00175     catch (SISProtocol::ExceptionGeneric &ex)
00176     {
00177         return set_error(ID_err, char2str(ex.what()),
00178             Err_Block_SeqWrite);
00179     }
00180     catch (CSerial::ExceptionGeneric &ex)
00181     {
00182         return set_error(ID_err, char2str(ex.what()),
00183             Err_Block_SeqWrite);
00184     }
00185 }
00186 DLLEXPORT int32_t DLLCALLCONV sequencer_softtrigger(
00187     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, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00193             Err_Invalid_Pointer);
00194     try
00195     {
00196         uint32_t qb0stat;
00197
00198         // FEED DATA:
00199
00200         // SPS Global Register G1 (P-0-1371) - Reset Read Trigger
00201         ID_ref->write_parameter(TGM::SercosParamP, 1371,
00202             static_cast<uint64_t>(0));
00203         // SPS Global Register G1 (P-0-1371) - Set Read Trigger
00204         ID_ref->write_parameter(TGM::SercosParamP, 1371,
00205             static_cast<uint64_t>(1));

```

```

00206         // Check status (P-0-1410)
00207         ID_ref->read_parameter(TGM::SercosParamP, 1410, qb0stat); // TODO:
        Check RESULT_READ_OK bit (0b100000)
00208
00209         // TRIGGER:
00210
00211         // SPS Global Register G2 (P-0-1372) - Reset Sequencer Trigger
00212         ID_ref->write_parameter(TGM::SercosParamP, 1372,
static_cast<uint64_t>(0));
00213
00214         // SPS Global Register G2 (P-0-1372) - Set Sequencer Trigger
00215         ID_ref->write_parameter(TGM::SercosParamP, 1372,
static_cast<uint64_t>(1));
00216
00217         // Check status (P-0-1410)
00218         ID_ref->read_parameter(TGM::SercosParamP, 1410, qb0stat); // TODO:
        Check Drive started bit (0b1000)
00219
00220         return Err_NoError;
00221     }
00222     catch (SISProtocol::ExceptionGeneric &ex)
00223     {
00224         return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
00225     }
00226     catch (CSerial::ExceptionGeneric &ex)
00227     {
00228         return set_error(ID_err, char2str(ex.what()),
Err_Block_SeqWrite);
00229     }
00230 }
00231
00232
00233 DLLEXPORT int32_t DLLCALLCONV speedcontrol_activate(
SISProtocol * ID_ref, ErrHandle ID_err)
00234 {
00235     if (!dynamic_cast<SISProtocol*>(ID_ref))
00236         // Return error for wrong reference
00237         return set_error(
00238             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00239
00240
00241     try
00242     {
00243         // Change mode
00244         change_opmode(ID_ref, DRIVEMODE_SPEEDCONTROL);
00245
00246         return Err_NoError;
00247     }
00248     catch (SISProtocol::ExceptionGeneric &ex)
00249     {
00250         return set_error(ID_err, char2str(ex.what()),
Err_Block_VelCInit);
00251     }
00252     catch (CSerial::ExceptionGeneric &ex)
00253     {
00254         return set_error(ID_err, char2str(ex.what()),
Err_Block_VelCInit);
00255     }
00256 }
00257
00258
00259 DLLEXPORT int32_t DLLCALLCONV speedcontrol_init(
SISProtocol * ID_ref, double_t ID_max_accel, double_t ID_max_jerk,
ErrHandle ID_err)
00260 {
00261     if (!dynamic_cast<SISProtocol*>(ID_ref))
00262         // Return error for wrong reference
00263         return set_error(
00264             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00265
00266
00267     try
00268     {
00269         // Set required units (preferred scaling, rotary scaling, [rpm])
00270         change_units(ID_ref);
00271
00272         // Max Acceleration (S-0-0138)
00273         ID_ref->write_parameter(TGM::SercosParamS, 138, ID_max_accel);
00274
00275         // Max Jerk (S-0-0349)
00276         ID_ref->write_parameter(TGM::SercosParamS, 349, ID_max_jerk);
00277
00278         return Err_NoError;
00279     }
00280     catch (SISProtocol::ExceptionGeneric &ex)
00281     {

```

```

00282         return set_error(ID_err, char2str(ex.what()),
Err_Block_VelCInit);
00283     }
00284     catch (CSerial::ExceptionGeneric &ex)
00285     {
00286         return set_error(ID_err, char2str(ex.what()),
Err_Block_VelCInit);
00287     }
00288 }
00289
00290
00291 DLLEXPORT int32_t DLLCALLCONV speedcontrol_write(
SISProtocol * ID_ref, double_t ID_speed, double_t ID_accel, ErrHandle ID_err)
00292 {
00293     if (!dynamic_cast<SISProtocol*>(ID_ref))
00294         // Return error for wrong reference
00295         return set_error(
00296             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00297             Err_Invalid_Pointer);
00298
00299     try
00300     {
00301         // Rotation direction - Positive ID_speed: Clockwise rotation, Negative ID_speed: Counter-clockwise
rotation
00302         uint32_t rotmode = static_cast<uint32_t>((stde::sgn<double_t>(ID_speed) == 1 ? 0 : 1) << 10);
00303         // Control Mode (P-0-1200)
00304         ID_ref->write_parameter(TGM::SercosParamP, 1200, rotmode);
00305
00306         // Acceleration in rad/s^2 (P-0-1203)
00307         ID_ref->write_parameter(TGM::SercosParamP, 1203, ID_accel);
00308
00309         // Speed in rpm (S-0-0036)
00310         ID_ref->write_parameter(TGM::SercosParamS, 36, abs(ID_speed));
00311
00312         return Err_NoError;
00313     }
00314     catch (SISProtocol::ExceptionGeneric &ex)
00315     {
00316         return set_error(ID_err, char2str(ex.what()),
Err_Block_VelCWrite);
00317     }
00318     catch (CSerial::ExceptionGeneric &ex)
00319     {
00320         return set_error(ID_err, char2str(ex.what()),
Err_Block_VelCWrite);
00321     }
00322 }
00323
00324
00325 DLLEXPORT int32_t DLLCALLCONV set_stdenvironment(
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_units(ID_ref);
00336         change_language(ID_ref);
00337
00338         return Err_NoError;
00339     }
00340     catch (SISProtocol::ExceptionGeneric &ex)
00341     {
00342         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00343     }
00344     catch (CSerial::ExceptionGeneric &ex)
00345     {
00346         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00347     }
00348 }
00349
00350
00351 DLLEXPORT int32_t DLLCALLCONV get_drivemode(
SISProtocol * ID_ref, uint32_t * ID_drvmode, ErrHandle ID_err)
00352 {
00353     if (!dynamic_cast<SISProtocol*>(ID_ref))
00354         // Return error for wrong reference
00355         return set_error(
00356             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00357             Err_Invalid_Pointer);
00358

```



```

00359     try
00360     {
00361         uint64_t curdrvmode;
00362         // Primary Operation Mode (S-0-0032)
00363         ID_ref->read_parameter(TGM::SercosParams, 32, curdrvmode);
00364
00365         switch (curdrvmode)
00366         {
00367             case DRIVEMODE_SEQUENCER: // Drive Mode: Sequencer
00368                 *ID_drvmode = 1;
00369                 break;
00370             case DRIVEMODE_SPEEDCONTROL: // Drive Mode: Speed Control
00371                 *ID_drvmode = 2;
00372                 break;
00373             default: // Drive Mode not supported
00374                 *ID_drvmode = 0;
00375                 break;
00376         }
00377
00378         return Err_NoError;
00379     }
00380     catch (SISProtocol::ExceptionGeneric &ex)
00381     {
00382         return set_error(ID_err, char2str(ex.what()),
00383             Err_Block_GetStatus);
00384     }
00385     catch (CSerial::ExceptionGeneric &ex)
00386     {
00387         return set_error(ID_err, char2str(ex.what()),
00388             Err_Block_GetStatus);
00389     }
00390 }
00391
00392 DLLEXPORT int32_t DLLCALLCONV get_opstate(
00393     SISProtocol * ID_ref, uint8_t * ID_opstate, ErrHandle ID_err)
00394 {
00395     if (!dynamic_cast<SISProtocol*>(ID_ref))
00396         // Return error for wrong reference
00397         return set_error(
00398             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00399             Err_Invalid_Pointer);
00400
00401     try
00402     {
00403         uint64_t curopstate;
00404         // Device control: Status word (P-0-0115)
00405         ID_ref->read_parameter(TGM::SercosParamP, 115, curopstate);
00406
00407         OPSTATE opstate(static_cast<uint16_t>(curopstate));
00408         *ID_opstate = opstate.Value;
00409
00410         return Err_NoError;
00411     }
00412     catch (SISProtocol::ExceptionGeneric &ex)
00413     {
00414         return set_error(ID_err, char2str(ex.what()),
00415             Err_Block_GetStatus);
00416     }
00417     catch (CSerial::ExceptionGeneric &ex)
00418     {
00419         return set_error(ID_err, char2str(ex.what()),
00420             Err_Block_GetStatus);
00421     }
00422 }
00423
00424 DLLEXPORT int32_t DLLCALLCONV get_speed(SISProtocol * ID_ref,
00425     double_t * ID_speed, ErrHandle ID_err)
00426 {
00427     if (!dynamic_cast<SISProtocol*>(ID_ref))
00428         // Return error for wrong reference
00429         return set_error(
00430             ID_err, sformat("Reference pointing to invalid location '%p'.", ID_ref),
00431             Err_Invalid_Pointer);
00432
00433     try
00434     {
00435         double_t speed;
00436         // Velocity feedback Value (S-0-0040)
00437         ID_ref->read_parameter(TGM::SercosParams, 40, speed);
00438
00439         *ID_speed = speed;
00440
00441         return Err_NoError;
00442     }
00443     catch (SISProtocol::ExceptionGeneric &ex)

```

```

00440     {
00441         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00442     }
00443     catch (CSerial::ExceptionGeneric &ex)
00444     {
00445         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00446     }
00447 }
00448
00449
00450 DLLEXPORT int32_t DLLCALLCONV get_diagnostic_msg(
SISProtocol * ID_ref, char * ID_diagnostic_msg, ErrHandle ID_err)
00451 {
00452     if (!dynamic_cast<SISProtocol*>(ID_ref))
00453         // Return error for wrong reference
00454         return set_error(
00455             ID_err, sprintf("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00456
00457     try
00458     {
00459         char msg[TGM_SIZE_MAX_PAYLOAD];
00460         // Diagnostic message (S-0-0095)
00461         ID_ref->read_parameter(TGM::SercosParamS, 95, msg);
00462
00463         strncpy(ID_diagnostic_msg, msg+4, TGM_SIZE_MAX_PAYLOAD-4);
00464
00465         return Err_NoError;
00466     }
00467     catch (SISProtocol::ExceptionGeneric &ex)
00468     {
00469         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00470     }
00471     catch (CSerial::ExceptionGeneric &ex)
00472     {
00473         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00474     }
00475 }
00476 }
00477
00478
00479 DLLEXPORT int32_t DLLCALLCONV get_diagnostic_num(
SISProtocol * ID_ref, uint32_t * ID_diagnostic_num, ErrHandle ID_err)
00480 {
00481     if (!dynamic_cast<SISProtocol*>(ID_ref))
00482         // Return error for wrong reference
00483         return set_error(
00484             ID_err, sprintf("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00485
00486     try
00487     {
00488         UINT32 num;
00489         // Diagnostic number (S-0-0390)
00490         ID_ref->read_parameter(TGM::SercosParamS, 390, num);
00491
00492         *ID_diagnostic_num = num;
00493
00494         return Err_NoError;
00495     }
00496     catch (SISProtocol::ExceptionGeneric &ex)
00497     {
00498         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00499     }
00500     catch (CSerial::ExceptionGeneric &ex)
00501     {
00502         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00503     }
00504 }
00505 }
00506
00507
00508 DLLEXPORT int32_t DLLCALLCONV clear_error(
SISProtocol * ID_ref, ErrHandle ID_err)
00509 {
00510     if (!dynamic_cast<SISProtocol*>(ID_ref))
00511         // Return error for wrong reference
00512         return set_error(
00513             ID_err, sprintf("Reference pointing to invalid location '%p'.", ID_ref),
Err_Invalid_Pointer);
00514
00515     try
00516     {
00517

```

```

00518         // Clear error (S-0-0099) // Command C0500
00519         ID_ref->execute_command(TGM::SercosParamS, 99);
00520
00521         return Err_NoError;
00522     }
00523     catch (SISProtocol::ExceptionGeneric &ex)
00524     {
00525         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00526     }
00527     catch (CSerial::ExceptionGeneric &ex)
00528     {
00529         return set_error(ID_err, char2str(ex.what()),
Err_Block_GetStatus);
00530     }
00531 }
00532
00533
00534 void change_opmode(SISProtocol * ID_ref, const uint64_t opmode)
00535 {
00536     uint64_t curopmode;
00537     // Primary Operation Mode (S-0-0032)
00538     ID_ref->read_parameter(TGM::SercosParamS, 32, curopmode);
00539
00540     // Operation change will trigger flash operations that may cause limited life time
00541     // Thus, operation change should be mainly triggered if required only
00542     if (curopmode != opmode)
00543     {
00544         // Enter parameterization level 1 (S-0-0420) // Command C0400
00545         ID_ref->execute_command(TGM::SercosParamS, 420);
00546
00547         // Primary Operation Mode (S-0-0032)
00548         ID_ref->write_parameter(TGM::SercosParamS, 32, opmode);
00549
00550         // Leave parameterization level 1 (S-0-0422) // Command C0200
00551         ID_ref->execute_command(TGM::SercosParamS, 422);
00552     }
00553 }
00554
00555
00556 inline SPEEDUNITS get_units(SISProtocol * ID_ref)
00557 {
00558     uint64_t curunits;
00559     // Scaling of speed units (S-0-0044)
00560     ID_ref->read_parameter(TGM::SercosParamS, 44, curunits);
00561
00562     return SPEEDUNITS(static_cast<uint16_t>(curunits));
00563 }
00564
00565
00566 void change_units(SISProtocol * ID_ref)
00567 {
00568     SPEEDUNITS units = get_units(ID_ref);
00569     if (units.Bits.type_of_scaling == 0b010 && !units.Bits.automode && !units.
Bits.scale_units && !units.Bits.time_units && !units.Bits.data_rel) return;
00570
00571     // Set required units (preferred scaling, rotary scaling, [rpm])
00572     uint64_t scalingtype = 0b0000000000000010;
00573     // Velocity data scaling Type (S-0-0044)
00574     ID_ref->write_parameter(TGM::SercosParamS, 44, scalingtype);
00575 }
00576
00577
00578 inline void change_language(SISProtocol * ID_ref, const uint8_t lang_code)
00579 {
00580     // Language selection (S-0-0265):
00581     // * 0: German
00582     // * 1: English
00583     // * 2: French
00584     // * 3: Spanish
00585     // * 4: Italian
00586     ID_ref->write_parameter(TGM::SercosParamS, 265, (UINT32)lang_code);
00587 }

```

5.14 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 [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.

5.14.1 Detailed Description

Definition of API functions that are exported to the API DLL.

Definition in file [Wrapper.h](#).

5.14.2 Macro Definition Documentation

5.14.2.1 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](#).

5.14.2.2 DLLCALLCONV

```
#define DLLCALLCONV __cdecl
```

Definition at line 20 of file [Wrapper.h](#).

5.14.2.3 DRIVEMODE_SEQUENCER

```
#define DRIVEMODE_SEQUENCER 0b111011
```

Positioning mode lagless, encoder 1.

Definition at line 43 of file [Wrapper.h](#).

5.14.2.4 DRIVEMODE_SPEEDCONTROL

```
#define DRIVEMODE_SPEEDCONTROL 0b10
```

Velocity Control.

Definition at line 45 of file [Wrapper.h](#).

5.14.3 Typedef Documentation

5.14.3.1 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](#)

5.14.3.2 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).

5.14.3.3 SISProtocol

```
typedef struct SISProtocol SISProtocol
```

Faking the actual [SISProtocol](#) class to a struct so that the C compiler can handle compilation of this file.

The [SISProtocol](#) files itself should be automatically compiled using the C++ compilation process. This is automatically handled using extern "C".

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

5.14.4 Function Documentation

5.14.4.1 init()

```
SISProtocol* init ( )
```

Creates API reference.

The API references is a fundamental prerequisite.

Remarks

This function is exported to the Indradrive API DLL.

Refer to [Examples](#) for detailed code examples.

How to call with C#:

```
[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int init();
```

.

How to call with Python:

```
indraref = indralib.init()
```

.

Returns

API reference. Pointer can be casted and treated as UINT32 (see examples).

Definition at line 7 of file [Wrapper.cpp](#).

5.14.4.2 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](#).

5.14.4.3 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](#).

5.14.4.4 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](#).

5.14.4.5 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](#).

5.14.4.6 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 <i>ID_speeds</i> , <i>ID_accels</i> , etc).
out	<i>ID_err</i>	(Optional) Error handle.

Returns

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

Definition at line 129 of file [Wrapper.cpp](#).

5.14.4.7 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.

By special PLC software (if configured), the Indradrive can be triggered by both software trigger and hardware trigger. The hardware trigger is realized through a 24V rising edge input line.

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 186 of file [Wrapper.cpp](#).

5.14.4.8 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 233 of file [Wrapper.cpp](#).

5.14.4.9 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 259 of file [Wrapper.cpp](#).

5.14.4.10 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 291 of file [Wrapper.cpp](#).

5.14.4.11 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 325 of file [Wrapper.cpp](#).

5.14.4.12 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
<code>*ID_drvmode == 0</code>	Drive Mode not supported
<code>*ID_drvmode == 1</code>	"Sequencer" drive mode active
<code>*ID_drvmode == 2</code>	"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.
out	<i>ID_err</i>	(Optional) Error handle.

Returns

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

Definition at line 351 of file [Wrapper.cpp](#).

5.14.4.13 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(&lt;path_to_DLL&gt;, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_opstate(int ID_ref, ref Byte state, ref
    ErrHandle ID_err);
```

.

How to call with Python:

```
opstate = ctypes.c_uint8(0)
result = indralib.get_opstate(indraref, ctypes.byref(opstate), ctypes.byref(indra_error))
```

.

Parameters

in	<i>ID_ref</i>	API reference. Pointer can be casted in from UINT32.
out	<i>ID_opstate</i>	<p>Pointer that provides the respective information:</p> <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 391 of file [Wrapper.cpp](#).

5.14.4.14 `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 [421](#) of file [Wrapper.cpp](#).

5.14.4.15 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 450 of file [Wrapper.cpp](#).

5.14.4.16 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 479 of file [Wrapper.cpp](#).

5.14.4.17 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 508 of file [Wrapper.cpp](#).

5.15 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) & 0b111111111)
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
00236     wchar_t* ID_comport = L"COM1", uint32_t ID_combaudrate = 19200, ErrHandle ID_err =
00237     ErrHandle());
00238
00258     DLLEXPORT int32_t DLLCALLCONV close(SISProtocol* ID_ref,
00259     ErrHandle ID_err = ErrHandle());
00260 #pragma endregion API Fundamentals
00261
00262 #pragma region API Sequencer
00263
00287     DLLEXPORT int32_t DLLCALLCONV sequencer_activate(
00288     SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());
00289
00307     DLLEXPORT int32_t DLLCALLCONV sequencer_init(
00308     SISProtocol* ID_ref, double_t ID_max_accel = 10000, double_t ID_max_jerk = 1000,
00309     ErrHandle ID_err = ErrHandle());
00310

```

```

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
00360     DLLEXPORT int32_t DLLCALLCONV sequencer_softtrigger(
        SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());
00361
00362 #pragma endregion API Sequencer
00363
00364 #pragma region API Speed Control
00365
00391     DLLEXPORT int32_t DLLCALLCONV speedcontrol_activate(
        SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());
00392
00416     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());
00417
00443     DLLEXPORT int32_t DLLCALLCONV speedcontrol_write(
        SISProtocol* ID_ref, double_t ID_speed, double_t ID_accel, ErrHandle ID_err =
        ErrHandle());
00444
00445 #pragma endregion API Speed Control
00446
00447 #pragma region API Configuration
00448
00472     DLLEXPORT int32_t DLLCALLCONV set_stdenviroment(
        SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());
00473
00474 #pragma endregion API Configuration
00475
00476 #pragma region API Status
00477
00515     DLLEXPORT int32_t DLLCALLCONV get_drivemode(
        SISProtocol* ID_ref, uint32_t * ID_drvmode, ErrHandle ID_err =
        ErrHandle());
00516
00563     DLLEXPORT int32_t DLLCALLCONV get_opstate(
        SISProtocol* ID_ref, uint8_t * ID_opstate, ErrHandle ID_err =
        ErrHandle());
00564
00585     DLLEXPORT int32_t DLLCALLCONV get_speed(
        SISProtocol * ID_ref, double_t * ID_speed, ErrHandle ID_err =
        ErrHandle());
00586
00612     DLLEXPORT int32_t DLLCALLCONV get_diagnostic_msg(
        SISProtocol* ID_ref, char * ID_diagnostic_msg, ErrHandle ID_err =
        ErrHandle());
00613
00635     DLLEXPORT int32_t DLLCALLCONV get_diagnostic_num(
        SISProtocol* ID_ref, uint32_t * ID_diagnostic_num, ErrHandle ID_err =
        ErrHandle());
00636
00658     DLLEXPORT int32_t DLLCALLCONV clear_error(
        SISProtocol* ID_ref, ErrHandle ID_err = ErrHandle());
00659
00660 #pragma ends
00661
00662 /* \cond Do not document this */
00663
00664 #pragma region Internal helper functions
00665
00671     inline void change_opmode(SISProtocol * ID_ref, const uint64_t opmode);
00672
00678     inline SPEEDUNITS get_units(SISProtocol * ID_ref);
00679
00683     inline void change_units(SISProtocol * ID_ref);
00684
00694     inline void change_language(SISProtocol * ID_ref, const uint8_t lang_code = 1
    );
00695
00696 #pragma endregion Internal helper functions
00697
00698 /* \endcond Do not document this */
00699
00700 #ifdef __cplusplus
00701 }
00702 #endif
00703
00704 #endif /* _WRAPPER_H_ */

```

6 Example Documentation

6.1 apps/PythonApplication1/PythonApplication1.py

```

00001 import sys
00002 import ctypes
00003 from ctypes import cdll
00004 import os
00005
00006 # Minimum Python 3.3 required
00007 assert sys.version_info >= (3,3)
00008
00009
00010 # Load Indradrive API DLL into memory (use absolute or relative path for 'libpath')
00011 libpath = os.path.dirname(__file__) + "\\..\\..\\bin\\IndradriveAPI.dll"
00012 indralib = cdll.LoadLibrary(libpath)
00013
00014 # Error-specific class
00015 class ERR(ctypes.Structure):
00016     _fields_ = [("code", ctypes.c_int32), ("msg", ctypes.c_char * 2048)]
00017
00018     def get_msg_str(self):
00019         return str(self.msg, "UTF-8")
00020
00021 indra_error = ERR(0)
00022
00023
00024 def check_result(result):
00025     if result:
00026         print("Error occurred: " + indra_error.get_msg_str())
00027         sys.exit(result)
00028
00029 def get_bit(byteval, idx):
00030     return ((byteval & (1 << idx)) != 0);
00031
00032
00033 # MAIN ENTRY POINT
00034 def main():
00035     # Getting API reference
00036     indraref = indralib.init()
00037
00038     # Opening communication channel
00039     result = indralib.open(indraref, b"COM1", 19200, ctypes.byref(indra_error))
00040     check_result(result)
00041
00042     # Set standard environment
00043     result = indralib.set_stdenvironment(indraref, ctypes.byref(indra_error))
00044     check_result(result)
00045
00046
00047     #
00048     # Check Drive Mode
00049     #
00050     drvmode = ctypes.c_uint32(0)
00051     result = indralib.get_drivemode(indraref, ctypes.byref(drvmode), ctypes.byref(indra_error))
00052     check_result(result)
00053
00054     if drvmode.value != 2: # Drive Mode is not "Speed Control" -> Change it
00055         input("Please make sure to DISABLE the drive release before continue (stand-by mode)!\n(Press any
key to continue...)")
00056
00057     # Activate Speed Control
00058     result = indralib.speedcontrol_activate(indraref, ctypes.byref(indra_error))
00059     check_result(result)
00060
00061     # Diagnostic message
00062     diagmsg = ctypes.create_string_buffer(256)
00063     result = indralib.get_diagnostic_msg(indraref, diagmsg, ctypes.byref(indra_error))
00064     check_result(result)
00065     print("Current status:\n" + diagmsg.raw.decode('ascii'))
00066
00067
00068     #
00069     # Check Operation State
00070     #
00071     while True:
00072         opstate = ctypes.c_uint8(0)
00073         result = indralib.get_opstate(indraref, ctypes.byref(opstate), ctypes.byref(indra_error))
00074         check_result(result)
00075
00076         if (opstate.value & 0b11) != 0b11:
00077             input("Please make sure to RELEASE before continue (torque-controlled operation mode)!\n(Press
any key to continue...)")

```

```

00078         else:
00079             break
00080
00081     # Set limits
00082     result = indralib.speedcontrol_init(indraref, ctypes.c_double(10000), ctypes.c_double(1000),
00083 ctypes.byref(indra_error))
00084     check_result(result)
00085
00086     while True:
00087         speed_str = input("Speed [rpm] = ?")
00088         if (speed_str == ""): break
00089
00090         # Set speed
00091         speed = int(speed_str)
00092         result = indralib.speedcontrol_write(indraref, ctypes.c_double(speed), ctypes.c_double(10),
00093 ctypes.byref(indra_error))
00094         check_result(result)
00095
00096
00097     # Closing communication channel
00098     result = indralib.close(indraref, ctypes.byref(indra_error))
00099     check_result(result)
00100
00101     return 0
00102
00103
00104 if __name__ == "__main__":
00105     sys.exit(int(main() or 0))

```

6.2 apps/WpfApplication1/Indradrive.cs

```

using System;
using System.Runtime.InteropServices;
using System.Text;
using System.Windows.Controls;

namespace WpfApplication1
{
    public class Indradrive
    {
        [StructLayout(LayoutKind.Sequential)]
        public unsafe struct ErrHandle
        {
            [MarshalAs(UnmanagedType.U4)]
            public UInt32 code;
            [MarshalAs(UnmanagedType.ByValArray, SizeConst = 2048)]
            public byte[] msg;
        }

        private int idref;
        private const string dllpath = "..\\..\\..\\..\\bin\\IndradriveAPI.dll";

        private ErrHandle indraerr;
        private ListBox listboxerr;

        public Indradrive(ref ListBox listbox)
        {
            listboxerr = listbox;
            idref = init();
        }

        // Fundamentals

        [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
        private static extern int init();

        [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);
        public int open(Byte[] ID_comport, UInt32 ID_combaudrate) { return CheckResult(
open(idref, ID_comport, ID_combaudrate, ref indraerr)); }

        [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
        private static extern int close(int ID_ref, ref ErrHandle ID_err);
        public int close() { return CheckResult(close(idref, ref indraerr)); }

        // Speed Control

        [DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]

```

```

private static extern int speedcontrol_activate(int ID_ref, ref
ErrHandle ID_err);
public int speedcontrol_activate() { return CheckResult(
speedcontrol_activate(idref, ref indraerr)); }

[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);
public int speedcontrol_init(Double ID_max_accel, Double ID_max_jerk) { return
CheckResult(speedcontrol_init(idref, ID_max_accel, ID_max_jerk, ref indraerr)); }

[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);
public int speedcontrol_write(Double ID_speed, Double ID_accel) { return
CheckResult(speedcontrol_write(idref, ID_speed, ID_accel, ref indraerr)); }

// Sequencer

[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_activate(int ID_ref, ref
ErrHandle ID_err);
public int sequencer_activate() { return CheckResult(
sequencer_activate(idref, ref indraerr)); }

[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);
public int sequencer_init(Double ID_max_accel, Double ID_max_jerk) { return
CheckResult(sequencer_init(idref, ID_max_accel, ID_max_jerk, ref indraerr)); }

[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, Byte ID_direction, ref
ErrHandle ID_err);
public int sequencer_write(Double[] ID_speeds, Double[] ID_accels, Double[] ID_jerks
, UInt32[] ID_delays, UInt16 ID_set_length, Byte ID_direction) { return CheckResult(
sequencer_write(idref, ID_speeds, ID_accels, ID_jerks, ID_delays, ID_set_length,
ID_direction, ref indraerr)); }

[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int sequencer_softtrigger(int ID_ref, ref
ErrHandle ID_err);
public int sequencer_softtrigger() { return CheckResult(
sequencer_softtrigger(idref, ref indraerr)); }

// Status

[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_drivemode(int ID_ref, ref UInt32 mode, ref
ErrHandle ID_err);
public int get_drivemode(ref UInt32 mode) { return CheckResult(
get_drivemode(idref, ref mode, ref indraerr)); }

[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_opstate(int ID_ref, ref Byte state, ref
ErrHandle ID_err);
public int get_opstate(ref Byte state) { return CheckResult(
get_opstate(idref, ref state, ref indraerr)); }

[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int get_speed(int ID_ref, ref Double speed, ref
ErrHandle ID_err);
public int get_speed(ref Double speed) { return CheckResult(
get_speed(idref, ref speed, ref indraerr)); }

[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);
public int get_diagnostic_msg(Byte[] ID_diagnostic_msg) { return CheckResult(
get_diagnostic_msg(idref, ID_diagnostic_msg, ref indraerr)); }

[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);
public int get_diagnostic_num(ref UInt32 ID_diagnostic_num) { return CheckResult(
get_diagnostic_num(idref, ref ID_diagnostic_num, ref indraerr)); }

[DllImport(dllpath, CharSet = CharSet.Unicode, CallingConvention = CallingConvention.Cdecl)]
private static extern int clear_error(int ID_ref, ref
ErrHandle ID_err);
public int clear_error() { return CheckResult(clear_error(idref, ref indraerr
)); }

```



```
// Helpers

public int CheckResult(int ret)
{
    if (ret != 0)
    {
        String err = Encoding.ASCII.GetString(indraerr.msg).TrimEnd((Char)0);

        Console.WriteLine(err);
        listBoxerr.Dispatcher.BeginInvoke((System.Windows.Forms.MethodInvoker) (() =>
        {
            listBoxerr.Items.Add(err);
        }));
    }

    return ret;
}
}
```


Index

- ~ExceptionSISError
 - SISProtocol::ExceptionSISError, [27](#)
- ~ExceptionTransceiveFailed
 - SISProtocol::ExceptionTransceiveFailed, [29](#)
- ~Map
 - TGM::Map, [43](#)
- ~SISProtocol
 - SISProtocol, [69](#)
- AdrES1
 - TGM::HeaderExt, [41](#)
- AdrES2
 - TGM::HeaderExt, [41](#)
- AdrES3
 - TGM::HeaderExt, [41](#)
- AdrES4
 - TGM::HeaderExt, [41](#)
- AdrES5
 - TGM::HeaderExt, [41](#)
- AdrES6
 - TGM::HeaderExt, [41](#)
- AdrES7
 - TGM::HeaderExt, [41](#)
- AdrE
 - TGM::Header, [37](#)
- AdrS
 - TGM::Header, [36](#)
- at
 - TGM::Data, [21](#)
- automode
 - SPEEDUNITS, [74](#)
- BAUDRATE
 - SISProtocol, [68](#), [69](#)
- Bits
 - OPSTATE, [47](#)
 - SPEEDUNITS, [75](#)
 - TGM::Bitfields::HeaderControl, [39](#)
 - TGM::Bitfields::SercosParamAttribute, [62](#)
 - TGM::Bitfields::SercosParamControl, [64](#)
 - TGM::Bitfields::SercosParamIdent, [66](#)
- Bytes
 - TGM::Bytestream, [19](#)
 - TGM::Commands::SercosList, [53](#)
 - TGM::Commands::SercosParam, [59](#)
 - TGM::Commands::Subservice, [80](#)
 - TGM::Data, [24](#)
 - TGM::Reactions::SercosList, [50](#)
 - TGM::Reactions::SercosParam, [56](#)
 - TGM::Reactions::Subservice, [78](#)
- Bytestream
 - TGM::Bytestream, [18](#)
 - TGM, [11](#)
- calc_checksum
 - TGM::Header, [35](#)
- change_language
 - Wrapper.cpp, [126](#)
- change_opmode
 - Wrapper.cpp, [125](#)
- change_units
 - Wrapper.cpp, [125](#)
- clear
 - TGM::Bytestream, [19](#)
 - TGM::Commands::SercosList, [52](#)
 - TGM::Commands::SercosParam, [57](#)
 - TGM::Commands::Subservice, [79](#)
 - TGM::Data, [23](#)
 - TGM::Reactions::SercosList, [49](#)
 - TGM::Reactions::SercosParam, [55](#)
 - TGM::Reactions::Subservice, [77](#)
- clear_error
 - Wrapper.cpp, [124](#)
 - Wrapper.h, [148](#)
- close
 - SISProtocol, [70](#)
 - Wrapper.cpp, [113](#)
 - Wrapper.h, [138](#)
- Cntrl
 - TGM::Header, [36](#)
- code
 - GenericErrHandle, [31](#)
- Control
 - TGM::Commands::SercosList, [52](#)
 - TGM::Commands::SercosParam, [58](#)
 - TGM::Reactions::SercosList, [49](#)
 - TGM::Reactions::SercosParam, [55](#)
- ConversionFactor
 - TGM::Bitfields::SercosParamAttribute, [60](#)
- CS
 - TGM::Header, [35](#)
- DLLCALLCONV
 - Wrapper.h, [135](#)
- DLLEXPORT
 - Wrapper.h, [135](#)
- DRIVEMODE_SEQUENCER
 - Wrapper.h, [135](#)
- DRIVEMODE_SPEEDCONTROL
 - Wrapper.h, [135](#)
- DatLW
 - TGM::Header, [35](#)
- Data
 - TGM::Data, [20](#), [21](#)
 - TGM, [11](#)
- data_rel
 - SPEEDUNITS, [75](#)
- DataDisplay
 - TGM::Bitfields::SercosParamAttribute, [61](#)
- DataFunction
 - TGM::Bitfields::SercosParamAttribute, [60](#)

- DataLen
 - TGM::BitFields::SercosParamAttribute, 60
- Datablock
 - TGM::BitFields::SercosParamControl, 64
- DatL
 - TGM::Header, 35
- DriveError
 - OPSTATE, 47
- DriveHalted
 - OPSTATE, 47
- EErrorBlocks
 - errors.h, 82
- Err_Base
 - errors.h, 81
- ErrHandle
 - errors.h, 82
- Error
 - TGM::Reactions::SercosList, 50
 - TGM::Reactions::SercosParam, 56
 - TGM::Reactions::Subservice, 78
- errors.h, 80
 - EErrorBlocks, 82
 - Err_Base, 81
 - ErrHandle, 82
 - GenericErrHandle, 82
 - set_error, 83
- ExceptionGeneric
 - SISProtocol::ExceptionGeneric, 25
- ExceptionSISError
 - SISProtocol::ExceptionSISError, 27
- ExceptionTransceiveFailed
 - SISProtocol::ExceptionTransceiveFailed, 29
- execute_command
 - SISProtocol, 72
- GenericErrHandle, 29
 - code, 31
 - errors.h, 82
 - GenericErrHandle, 30
 - msg, 32
 - set, 31
 - set_code, 31
 - set_msg, 31
- get_DatL
 - TGM::Header, 34
- get_diagnostic_msg
 - Wrapper.cpp, 123
 - Wrapper.h, 147
- get_diagnostic_num
 - Wrapper.cpp, 124
 - Wrapper.h, 148
- get_drivemode
 - Wrapper.cpp, 120
 - Wrapper.h, 144
- get_errorcode
 - SISProtocol::ExceptionSISError, 27
- get_head_size
 - TGM::Commands::SercosList, 52
- TGM::Commands::SercosParam, 57
- TGM::Commands::Subservice, 79
- TGM::Reactions::SercosList, 49
- TGM::Reactions::SercosParam, 55
- TGM::Reactions::Subservice, 77
- get_opstate
 - Wrapper.cpp, 121
 - Wrapper.h, 145
- get_size
 - TGM::Commands::SercosList, 52
 - TGM::Commands::SercosParam, 58
 - TGM::Commands::Subservice, 79
 - TGM::Data, 24
 - TGM::Header, 34
 - TGM::Reactions::SercosList, 49
 - TGM::Reactions::SercosParam, 55
 - TGM::Reactions::Subservice, 77
- get_speed
 - Wrapper.cpp, 122
 - Wrapper.h, 146
- get_status
 - SISProtocol::ExceptionGeneric, 26
- get_sum
 - TGM::Header, 34
- get_units
 - Wrapper.cpp, 125
- Header
 - TGM::Header, 33
 - TGM::Map::Mapping, 45
 - TGM, 11
- HeaderControl
 - TGM::BitFields, 15
 - TGM::BitFields::HeaderControl, 38
- HeaderExt
 - TGM, 11
- HeaderType
 - TGM, 12
- init
 - Wrapper.cpp, 111
 - Wrapper.h, 137
- is_writeonly_phase2
 - TGM::BitFields::SercosParamAttribute, 61
- is_writeonly_phase3
 - TGM::BitFields::SercosParamAttribute, 61
- is_writeonly_phase4
 - TGM::BitFields::SercosParamAttribute, 61
- ListOffset
 - TGM::Commands::SercosList, 53
- m_bytestream
 - SISProtocol::ExceptionSISError, 28
- m_errorcode
 - SISProtocol::ExceptionSISError, 28
- m_message
 - SISProtocol::ExceptionGeneric, 26
- m_status

- SISProtocol::ExceptionGeneric, 26
- mainpage.dox, 85
- Map
 - TGM::Map, 43
- Mapping
 - TGM::Map, 44
 - TGM::Map::Mapping, 44
- msg
 - GenericErrHandle, 32
- NumRunningTgm
 - TGM::Bitfields::HeaderControl, 38
- NumSubAddresses
 - TGM::Bitfields::HeaderControl, 38
- OPSTATE, 45
 - Bits, 47
 - DriveError, 47
 - DriveHalted, 47
 - OPSTATE, 46
 - OperateState, 46
 - Value, 47
 - Wrapper.h, 136
- open
 - SISProtocol, 70
 - Wrapper.cpp, 112
 - Wrapper.h, 137
- OperateState
 - OPSTATE, 46
- operator<<
 - TGM::Data, 23
- PaketN
 - TGM::HeaderExt, 42
- ParamNum
 - TGM::Commands::SercosList, 53
 - TGM::Commands::SercosParam, 58
- ParamNumber
 - TGM::Bitfields::SercosParamIdent, 66
- ParamSet
 - TGM::Bitfields::SercosParamIdent, 66
- ParamType
 - TGM::Commands::SercosList, 52
 - TGM::Commands::SercosParam, 58
- ParamVariant
 - TGM::Bitfields::SercosParamIdent, 66
- Payload
 - TGM::Map::Mapping, 45
- RS232_BUFFER
 - SISProtocol.h, 94
- RS232_READ_LOOPS_MAX
 - SISProtocol.h, 94
- RS232_READ_TIMEOUT
 - SISProtocol.h, 95
- raw
 - TGM::Map, 44
- read_listelm
 - SISProtocol, 71
- read_parameter
 - SISProtocol, 70
- RecipientAddr
 - TGM::Commands::Subservice, 80
 - TGM::Reactions::Subservice, 77
- res1
 - TGM::Bitfields::SercosParamControl, 64
- res10
 - TGM::Bitfields::SercosParamAttribute, 62
- res2
 - TGM::Bitfields::SercosParamControl, 64
- res5
 - TGM::Bitfields::SercosParamAttribute, 61
- res6
 - TGM::Bitfields::SercosParamControl, 64
- res7
 - SPEEDUNITS, 75
 - TGM::Bitfields::SercosParamControl, 64
- SIS_ADDR_MASTER
 - SISProtocol.h, 95
- SIS_ADDR_SLAVE
 - SISProtocol.h, 95
- SIS_ADDR_UNIT
 - SISProtocol.h, 95
- SIS_SERVICES
 - SISProtocol, 68
- SISProtocol, 67
 - ~SISProtocol, 69
 - BAUDRATE, 68, 69
 - close, 70
 - execute_command, 72
 - open, 70
 - read_listelm, 71
 - read_parameter, 70
 - SIS_SERVICES, 68
 - SISProtocol, 69
 - set_baudrate, 70
 - Wrapper.h, 136
 - write_listelm, 72
 - write_parameter, 71
- SISProtocol.cpp, 85
- SISProtocol.h, 94, 95
 - RS232_BUFFER, 94
 - RS232_READ_LOOPS_MAX, 94
 - RS232_READ_TIMEOUT, 95
 - SIS_ADDR_MASTER, 95
 - SIS_ADDR_SLAVE, 95
 - SIS_ADDR_UNIT, 95
- SISProtocol::ExceptionGeneric, 25
 - ExceptionGeneric, 25
 - get_status, 26
 - m_message, 26
 - m_status, 26
 - warning, 26
 - what, 26
- SISProtocol::ExceptionSISError, 26
 - ~ExceptionSISError, 27
 - ExceptionSISError, 27

- get_errorcode, 27
 - m_bytestream, 28
 - m_errorcode, 28
 - what, 27
- SISProtocol::ExceptionTransceiveFailed, 28
 - ~ExceptionTransceiveFailed, 29
 - ExceptionTransceiveFailed, 29
 - what, 29
- SPEEDUNITS, 73
 - automode, 74
 - Bits, 75
 - data_rel, 75
 - res7, 75
 - SPEEDUNITS, 73
 - scale_units, 74
 - time_units, 74
 - type_of_scaling, 74
 - Value, 75
 - Wrapper.h, 136
- scale_units
 - SPEEDUNITS, 74
- ScaleFactor
 - TGM::Bitfields::SercosParamAttribute, 61
- SegmentSize
 - TGM::Commands::SercosList, 53
- sequencer_activate
 - Wrapper.cpp, 113
 - Wrapper.h, 138
- sequencer_init
 - Wrapper.cpp, 114
 - Wrapper.h, 139
- sequencer_softtrigger
 - Wrapper.cpp, 116
 - Wrapper.h, 141
- sequencer_write
 - Wrapper.cpp, 115
 - Wrapper.h, 140
- SercosCommandrequest
 - TGM, 13
- SercosCommandstatus
 - TGM, 13
- SercosDatablock
 - TGM, 12
- SercosDataalen
 - TGM, 14
- SercosList
 - TGM::Commands, 17
 - TGM::Commands::SercosList, 51
 - TGM::Reactions, 18
 - TGM::Reactions::SercosList, 48
- SercosParam
 - TGM::Commands, 16
 - TGM::Commands::SercosParam, 57
 - TGM::Reactions, 17
 - TGM::Reactions::SercosParam, 54
- SercosParamAttribute
 - TGM::Bitfields, 16
 - TGM::Bitfields::SercosParamAttribute, 60
- SercosParamControl
 - TGM::Bitfields, 15
 - TGM::Bitfields::SercosParamControl, 63
- SercosParamIdent
 - TGM::Bitfields, 15
 - TGM::Bitfields::SercosParamIdent, 65
- SercosParamVar
 - TGM, 12
- SercosTxProgress
 - TGM, 14
- Service
 - TGM::Header, 36
- ServiceNumber
 - TGM::Commands::Subservice, 80
 - TGM::Reactions::Subservice, 77
- set
 - GenericErrHandle, 31
 - TGM::Map, 43
- set_DatL
 - TGM::Header, 34
- set_baudrate
 - SISProtocol, 70
- set_code
 - GenericErrHandle, 31
- set_error
 - errors.h, 83
- set_msg
 - GenericErrHandle, 31
- set_size
 - TGM::Data, 24
- set_stdenvironment
 - Wrapper.cpp, 119
 - Wrapper.h, 143
- Size
 - TGM::Data, 24
- speedcontrol_activate
 - Wrapper.cpp, 117
 - Wrapper.h, 141
- speedcontrol_init
 - Wrapper.cpp, 117
 - Wrapper.h, 142
- speedcontrol_write
 - Wrapper.cpp, 118
 - Wrapper.h, 143
- Status
 - TGM::Reactions::SercosList, 49
 - TGM::Reactions::SercosParam, 55
 - TGM::Reactions::Subservice, 77
- StatusReactionTgm
 - TGM::Bitfields::HeaderControl, 39
- StZ
 - TGM::Header, 35
- Subservice
 - TGM::Commands, 16
 - TGM::Commands::Subservice, 79
 - TGM::Reactions, 17
 - TGM::Reactions::Subservice, 76
- TGM::Bitfields, 15

- HeaderControl, 15
- SercosParamAttribute, 16
- SercosParamControl, 15
- SercosParamIdent, 15
- TGM::Bitfields::HeaderControl, 37
 - Bits, 39
 - HeaderControl, 38
 - NumRunningTgm, 38
 - NumSubAddresses, 38
 - StatusReactionTgm, 39
 - Type, 39
 - Value, 39
- TGM::Bitfields::SercosParamAttribute, 59
 - Bits, 62
 - ConversionFactor, 60
 - DataDisplay, 61
 - DataFunction, 60
 - DataLen, 60
 - is_writeonly_phase2, 61
 - is_writeonly_phase3, 61
 - is_writeonly_phase4, 61
 - res10, 62
 - res5, 61
 - ScaleFactor, 61
 - SercosParamAttribute, 60
 - Value, 62
- TGM::Bitfields::SercosParamControl, 62
 - Bits, 64
 - Datablock, 64
 - res1, 64
 - res2, 64
 - res6, 64
 - res7, 64
 - SercosParamControl, 63
 - TxProgress, 64
 - Value, 64
- TGM::Bitfields::SercosParamIdent, 65
 - Bits, 66
 - ParamNumber, 66
 - ParamSet, 66
 - ParamVariant, 66
 - SercosParamIdent, 65
 - Value, 66
- TGM::Bytestream, 18
 - Bytes, 19
 - Bytestream, 18
 - clear, 19
- TGM::Commands, 16
 - SercosList, 17
 - SercosParam, 16
 - Subservice, 16
- TGM::Commands::SercosList, 50
 - Bytes, 53
 - clear, 52
 - Control, 52
 - get_head_size, 52
 - get_size, 52
 - ListOffset, 53
 - ParamNum, 53
 - ParamType, 52
 - SegmentSize, 53
 - SercosList, 51
 - UnitAddr, 52
- TGM::Commands::SercosParam, 56
 - Bytes, 59
 - clear, 57
 - Control, 58
 - get_head_size, 57
 - get_size, 58
 - ParamNum, 58
 - ParamType, 58
 - SercosParam, 57
 - UnitAddr, 58
- TGM::Commands::Subservice, 78
 - Bytes, 80
 - clear, 79
 - get_head_size, 79
 - get_size, 79
 - RecipientAddr, 80
 - ServiceNumber, 80
 - Subservice, 79
- TGM::Data, 19
 - at, 21
 - Bytes, 24
 - clear, 23
 - Data, 20, 21
 - get_size, 24
 - operator<<, 23
 - set_size, 24
 - Size, 24
 - toBYTE, 23
 - toUINT16, 22
 - toUINT32, 22
 - toUINT64, 22
 - toUINT8, 23
 - toVector, 22
- TGM::Header, 32
 - AdrE, 37
 - AdrS, 36
 - calc_checksum, 35
 - Cntrl, 36
 - CS, 35
 - DatLW, 35
 - DatL, 35
 - get_DatL, 34
 - get_size, 34
 - get_sum, 34
 - Header, 33
 - Service, 36
 - set_DatL, 34
 - StZ, 35
- TGM::HeaderExt, 40
 - AdrES1, 41
 - AdrES2, 41
 - AdrES3, 41
 - AdrES4, 41

- AdrES5, [41](#)
 - AdrES6, [41](#)
 - AdrES7, [41](#)
 - 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, [44](#)
 - Payload, [45](#)
- TGM::Reactions, [17](#)
 - SercosList, [18](#)
 - SercosParam, [17](#)
 - Subservice, [17](#)
- TGM::Reactions::SercosList, [48](#)
 - Bytes, [50](#)
 - clear, [49](#)
 - Control, [49](#)
 - Error, [50](#)
 - get_head_size, [49](#)
 - get_size, [49](#)
 - SercosList, [48](#)
 - Status, [49](#)
 - UnitAddr, [49](#)
- TGM::Reactions::SercosParam, [53](#)
 - Bytes, [56](#)
 - clear, [55](#)
 - Control, [55](#)
 - Error, [56](#)
 - get_head_size, [55](#)
 - get_size, [55](#)
 - SercosParam, [54](#)
 - Status, [55](#)
 - UnitAddr, [55](#)
- TGM::Reactions::Subservice, [76](#)
 - Bytes, [78](#)
 - clear, [77](#)
 - Error, [78](#)
 - get_head_size, [77](#)
 - get_size, [77](#)
 - RecipientAddr, [77](#)
 - ServiceNumber, [77](#)
 - Status, [77](#)
 - Subservice, [76](#)
- TGM_SIZE_HEADER_EXT
 - Telegrams.h, [100](#)
- TGM_SIZE_HEADER
 - Telegrams.h, [100](#)
- TGM_SIZEMAX_PAYLOAD
 - Telegrams.h, [100](#)
- TGM_SIZEMAX
 - Telegrams.h, [100](#)
- TGM, [10](#)
 - Bytestream, [11](#)
 - Data, [11](#)
 - Header, [11](#)
 - HeaderExt, [11](#)
 - HeaderType, [12](#)
 - SercosCommandrequest, [13](#)
 - SercosCommandstatus, [13](#)
 - SercosDatablock, [12](#)
 - SercosDatalen, [14](#)
 - SercosParamVar, [12](#)
 - SercosTxProgress, [14](#)
- Telegrams.h, [98](#), [100](#)
 - TGM_SIZE_HEADER_EXT, [100](#)
 - TGM_SIZE_HEADER, [100](#)
 - TGM_SIZEMAX_PAYLOAD, [100](#)
 - TGM_SIZEMAX, [100](#)
- Telegrams_Bitfields.h, [106](#), [108](#)
- time_units
 - SPEEDUNITS, [74](#)
- toBYTE
 - TGM::Data, [23](#)
- toUINT16
 - TGM::Data, [22](#)
- toUINT32
 - TGM::Data, [22](#)
- toUINT64
 - TGM::Data, [22](#)
- toUINT8
 - TGM::Data, [23](#)
- toVector
 - TGM::Data, [22](#)
- TxProgress
 - TGM::Bitfields::SercosParamControl, [64](#)
- Type
 - TGM::Bitfields::HeaderControl, [39](#)
- type_of_scaling
 - SPEEDUNITS, [74](#)
- UnitAddr
 - TGM::Commands::SercosList, [52](#)
 - TGM::Commands::SercosParam, [58](#)
 - TGM::Reactions::SercosList, [49](#)
 - TGM::Reactions::SercosParam, [55](#)
- Value
 - OPSTATE, [47](#)
 - SPEEDUNITS, [75](#)
 - TGM::Bitfields::HeaderControl, [39](#)
 - TGM::Bitfields::SercosParamAttribute, [62](#)
 - TGM::Bitfields::SercosParamControl, [64](#)
 - TGM::Bitfields::SercosParamIdent, [66](#)
- warning
 - SISProtocol::ExceptionGeneric, [26](#)
- what
 - SISProtocol::ExceptionGeneric, [26](#)
 - SISProtocol::ExceptionSISError, [27](#)
 - SISProtocol::ExceptionTransceiveFailed, [29](#)

Wrapper.cpp, 110
 change_language, 126
 change_opmode, 125
 change_units, 125
 clear_error, 124
 close, 113
 get_diagnostic_msg, 123
 get_diagnostic_num, 124
 get_drivemode, 120
 get_opstate, 121
 get_speed, 122
 get_units, 125
 init, 111
 open, 112
 sequencer_activate, 113
 sequencer_init, 114
 sequencer_softtrigger, 116
 sequencer_write, 115
 set_stdenvironment, 119
 speedcontrol_activate, 117
 speedcontrol_init, 117
 speedcontrol_write, 118
Wrapper.h, 133
 clear_error, 148
 close, 138
 DLLCALLCONV, 135
 DLLEXPORT, 135
 DRIVEMODE_SEQUENCER, 135
 DRIVEMODE_SPEEDCONTROL, 135
 get_diagnostic_msg, 147
 get_diagnostic_num, 148
 get_drivemode, 144
 get_opstate, 145
 get_speed, 146
 init, 137
 OPSTATE, 136
 open, 137
 SISProtocol, 136
 SPEEDUNITS, 136
 sequencer_activate, 138
 sequencer_init, 139
 sequencer_softtrigger, 141
 sequencer_write, 140
 set_stdenvironment, 143
 speedcontrol_activate, 141
 speedcontrol_init, 142
 speedcontrol_write, 143
write_listelm
 SISProtocol, 72
write_parameter
 SISProtocol, 71