

Stewart Platform

Generated by Doxygen 1.8.17

1 canopen	1
2 lxm32	3
3 stewart_platform	5
4 Hierarchical Index	7
4.1 Class Hierarchy	7
5 Class Index	9
5.1 Class List	9
6 File Index	11
6.1 File List	11
7 Class Documentation	13
7.1 CANopen::Driver Class Reference	13
7.1.1 Detailed Description	16
7.1.2 Member Enumeration Documentation	16
7.1.2.1 Control	16
7.1.2.2 OperationMode	17
7.1.2.3 Register	17
7.1.2.4 State	18
7.1.3 Constructor & Destructor Documentation	20
7.1.3.1 Driver()	20
7.1.4 Member Function Documentation	20
7.1.4.1 activate_PDO()	20
7.1.4.2 ctrl_to_str()	21
7.1.4.3 get()	21
7.1.4.4 get_mode()	22
7.1.4.5 get_position()	22
7.1.4.6 get_state()	23
7.1.4.7 get_torque()	23
7.1.4.8 get_velocity()	23
7.1.4.9 map_PDO()	23
7.1.4.10 send()	24
7.1.4.11 set()	25
7.1.4.12 set_control()	25
7.1.4.13 set_mode()	26
7.1.4.14 set_position()	26
7.1.4.15 set_position_offset()	26
7.1.4.16 set_torque()	27
7.1.4.17 set_velocity()	27
7.1.4.18 update()	27

7.1.4.19 wait_state()	28
7.2 CANopen::EMCYMessage Class Reference	28
7.2.1 Detailed Description	29
7.3 stp::Gazebo_sim Class Reference	30
7.3.1 Detailed Description	31
7.4 stp::Gnuplot_sim Class Reference	31
7.4.1 Detailed Description	32
7.5 CANopen::LXM32 Class Reference	32
7.5.1 Detailed Description	33
7.5.2 Constructor & Destructor Documentation	33
7.5.2.1 LXM32()	33
7.5.3 Member Function Documentation	33
7.5.3.1 get_angle()	34
7.5.3.2 set_angle()	34
7.6 MainWindow Class Reference	35
7.6.1 Detailed Description	35
7.7 CANopen::Message Class Reference	36
7.7.1 Detailed Description	37
7.7.2 Member Enumeration Documentation	37
7.7.2.1 FunctionCode	37
7.7.3 Member Function Documentation	38
7.7.3.1 function_code()	38
7.7.3.2 id()	38
7.7.3.3 node_id()	38
7.7.3.4 payload()	39
7.8 stp::Model Class Reference	39
7.8.1 Detailed Description	40
7.8.2 Member Data Documentation	40
7.8.2.1 _d2	40
7.8.2.2 m_A	40
7.8.2.3 m_a2	41
7.8.2.4 m_alpha	41
7.8.2.5 m_alpha_spd	41
7.8.2.6 m_B	41
7.8.2.7 m_beta	41
7.8.2.8 m_gamma	42
7.8.2.9 m_I2	42
7.8.2.10 m_P	42
7.8.2.11 m_P1	42
7.8.2.12 m_parity	42
7.8.2.13 m_R	43
7.8.2.14 m_T	43

7.8.2.15 m_T_spd	43
7.8.2.16 m_theta	43
7.8.2.17 m_theta_spd	43
7.8.2.18 m_trig	44
7.9 CANopen::NMTMessage Class Reference	44
7.9.1 Detailed Description	45
7.10 CANopen::Parameter Struct Reference	45
7.10.1 Detailed Description	47
7.10.2 Member Typedef Documentation	47
7.10.2.1 param_cb_t	47
7.10.3 Constructor & Destructor Documentation	47
7.10.3.1 Parameter() [1/2]	47
7.10.3.2 Parameter() [2/2]	48
7.10.4 Member Function Documentation	48
7.10.4.1 from_payload()	48
7.10.4.2 get()	49
7.10.4.3 has_been_sent()	49
7.10.4.4 link_to_pdo()	50
7.10.4.5 operator=() [1/6]	50
7.10.4.6 operator=() [2/6]	50
7.10.4.7 operator=() [3/6]	51
7.10.4.8 operator=() [4/6]	51
7.10.4.9 operator=() [5/6]	51
7.10.4.10 operator=() [6/6]	52
7.10.4.11 payload()	52
7.10.4.12 set()	53
7.11 CANopen::Payload Class Reference	53
7.11.1 Detailed Description	54
7.11.2 Constructor & Destructor Documentation	54
7.11.2.1 Payload()	54
7.11.3 Member Function Documentation	55
7.11.3.1 operator<<() [1/2]	55
7.11.3.2 operator<<() [2/2]	55
7.11.3.3 store_at()	56
7.11.3.4 value()	56
7.12 CANopen::PDOMessage Class Reference	57
7.12.1 Detailed Description	58
7.13 stp::Platform Class Reference	59
7.13.1 Detailed Description	60
7.13.2 Member Data Documentation	60
7.13.2.1 m_motors	60
7.14 CANopen::SDOInbound Class Reference	60

7.14.1 Detailed Description	61
7.15 CANopen::SDOMessage Class Reference	62
7.15.1 Detailed Description	63
7.15.2 Member Function Documentation	63
7.15.2.1 id()	63
7.15.2.2 index()	64
7.15.2.3 index__sub()	64
7.15.2.4 is_confirmation()	64
7.15.2.5 is_error()	64
7.15.2.6 payload()	65
7.15.2.7 size_data()	65
7.15.2.8 subindex()	65
7.16 CANopen::SDOOutbound Class Reference	66
7.16.1 Detailed Description	67
7.17 CANopen::SDOOutboundRead Class Reference	67
7.17.1 Detailed Description	68
7.18 CANopen::SDOOutboundWrite Class Reference	69
7.18.1 Detailed Description	70
7.19 CANopen::Socket Class Reference	70
7.19.1 Detailed Description	71
7.19.2 Constructor & Destructor Documentation	71
7.19.2.1 Socket() [1/2]	71
7.19.2.2 Socket() [2/2]	72
7.19.3 Member Function Documentation	72
7.19.3.1 send()	72
8 File Documentation	73
8.1 CANopen_driver.h File Reference	73
8.1.1 Detailed Description	74
8.2 CANopen_driver.h	75
8.3 CANopen_lxm32.h File Reference	77
8.3.1 Detailed Description	78
8.4 CANopen_lxm32.h	79
8.5 CANopen_socket.h File Reference	79
8.5.1 Detailed Description	80
8.5.2 Macro Definition Documentation	81
8.5.2.1 IF_VERBOSE	81
8.6 CANopen_socket.h	81
8.7 emcy.h File Reference	82
8.7.1 Detailed Description	83
8.8 emcy.h	83
8.9 message.h File Reference	83

8.9.1 Detailed Description	84
8.10 message.h	85
8.11 nmt.h File Reference	85
8.11.1 Detailed Description	86
8.12 nmt.h	87
8.13 parameter.h File Reference	87
8.13.1 Detailed Description	88
8.14 parameter.h	88
8.15 payload.h File Reference	90
8.15.1 Detailed Description	91
8.16 payload.h	92
8.17 sdo.h File Reference	92
8.17.1 Detailed Description	94
8.18 sdo.h	94
Index	97

Chapter 1

canopen

Information

- **Brief:** Canopen object able to send command through a CAN interface using the UNIX socket.
- **Languages:** C++
- **Libraries:**
- **Note:** /
- **Compatibility:**

Ubuntu	Window10	MacOS
:heavy_check_↔ mark:	:grey_↔ question:	:grey_↔ question:

Building

Ubuntu

Steps

- Clone the repository and go inside.
`git clone https://gitlab-dev.isir.upmc.fr/devillard/canopen.git && cd canpen`
- Create a build directory and go inside.
- Configure the project.
- Build the project.
`mkdir build && cd build && cmake .. && cmake --build .`

Testing

Install can tools `sudo apt install can-utils`

Setup a virtual CAN bus `sudo ip link add dev vcan0 type vcan && ip link set up vcan0`

Listen to the CAN bus `candump vcan0`

Canopen program

The executable file canopen enable you to send SDO message to a CAN bus.

usage:

`./canopen ifname 0xindex 0xsub [size base data]`

Arg: ifname : CAN interface name 0xindex : Object register index 0xsub : Object register subindex size : Data size (number of bytes) base : Numerical base of the value passed. data : Value to write.

Ex: To read register 0x1000:2 of node 4 on "can0": `./canopen can0 4 1000 2`

To write in register 0x2000:F of node 3 the value 0x1234 on "can0": `./canopen can0 3 2000 F 2 x 1234`

Chapter 2

lxm32

Information

- **Brief:** Library to control a Lexium32A driver from Schneider.
- **Languages:** C++
- **Libraries:**
- **Note:** /
- **Compatibility:**

Ubuntu	Window10	MacOS
:heavy_check_↔ mark:	:grey_↔ question:	:grey_↔ question:

Building

Ubuntu

Steps

- Clone the repository and go inside.
`git clone https://gitlab-dev.isir.upmc.fr/devillard/lxm32.git && cd lxm32`
- Create a build directory and go inside.
- Configure the project.
- Build the project.
`mkdir build && cd build && cmake .. && cmake --build .`

Schematics

Chapter 3

stewart_platform

Project to control the Stewart platform. The platform use to be control via an OLIMEX board doing the interface IP/CAN to communicate with the servo's drivers. This project aims to bypass this hardware to communicate directly on the CANopen bus.

Electrical Design

Stucture and Parameters used

Globale

Base

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

can_frame	
CANopen::Message	36
CANopen::EMCYMessage	28
CANopen::NMTMessage	44
CANopen::PDOMessage	57
CANopen::SDOMessage	62
CANopen::SDOInbound	60
CANopen::SDOOutbound	66
CANopen::SDOOutboundRead	67
CANopen::SDOOutboundWrite	69
CANopen::Driver	13
CANopen::LXM32	32
stp::Model	39
stp::Gazebo_sim	30
stp::Gnuplot_sim	31
stp::Platform	59
CANopen::Parameter	45
QMainWindow	
MainWindow	35
CANopen::Socket	70
vector	
CANopen::Payload	53

Chapter 5

Class Index

5.1 Class List

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

CANopen::Driver	
Device Profile Drives and Motion Control	13
CANopen::EMCYMessage	
EMCY Message (Emergency Object)	28
stp::Gazebo_sim	30
stp::Gnuplot_sim	31
CANopen::LXM32	
Implementation of the Driver Class for a LXM32 driver	32
MainWindow	35
CANopen::Message	
Can_frame object sent and received throught CANopen socket	36
stp::Model	39
CANopen::NMTMessage	
NMT Message (Network management)	44
CANopen::Parameter	
Object from the object dictionary of a remote CANopen device	45
CANopen::Payload	
Payload of CANopen message: array of 1 to 8 bytes of data	53
CANopen::PDOMessage	
PDO Message (Process Data Object)	57
stp::Platform	59
CANopen::SDOInbound	
SDO received Message	60
CANopen::SDOMessage	
SDO Message (Service Data Object)	62
CANopen::SDOOutbound	
SDO Message to be sent	66
CANopen::SDOOutboundRead	
SDO Message to be sent to read the value from the object dictionary of a remote device	67
CANopen::SDOOutboundWrite	
SDO Message to be sent to write the value of the object dictionary of a remote device	69
CANopen::Socket	
CANopen object able to send Message through a CAN interface using UNIX sockets	70

Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

CANdriver.h	??
CANopen_driver.cpp	??
CANopen_driver.h	
Device Profile Drives and Motion Control	73
CANopen_lxm32.cpp	??
CANopen_lxm32.h	
Implementation of the Driver Class for a LXM32 driver	77
CANopen_socket.cpp	??
CANopen_socket.h	
Canopen socket able to send/receive messages through a CAN interface using the UNIX socket	79
emcy.cpp	??
emcy.h	
PDO message sent and received throught CANopen socket	82
gazebo_sim.cpp	??
gazebo_sim.hpp	??
gnuplot_sim.cpp	??
gnuplot_sim.hpp	??
lib/lxm32/lib/canopen/src/main.cpp	??
lib/lxm32/src/main.cpp	??
src/main.cpp	??
mainwindow.cpp	??
mainwindow.h	??
message.cpp	??
message.h	
CAN_frame message sent and received throught CANopen socket	83
model.cpp	??
model.hpp	??
nmt.cpp	??
nmt.h	
NMT message sent and received throught CANopen socket	85
parameter.cpp	??
parameter.h	
Device Object from the object dictionary of a remote CANopen device	87
payload.cpp	??
payload.h	
Payload of CANopen message: array of 1 to 8 bytes of data	90

pdo.cpp	??
pdo.h	??
platform.cpp	??
platform.hpp	??
sdo.cpp	??
sdo.h	
PDO message sent and received through CANopen socket	92

Chapter 7

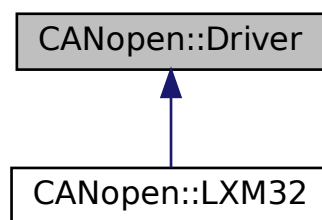
Class Documentation

7.1 CANopen::Driver Class Reference

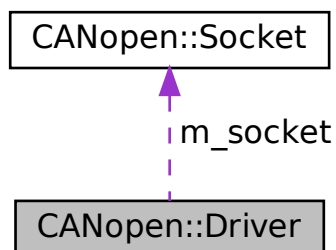
Device Profile Drives and Motion Control.

```
#include <CANopen_driver.h>
```

Inheritance diagram for CANopen::Driver:



Collaboration diagram for CANopen::Driver:



Public Types

- enum **Register** : uint32_t {
_DCOMstatus = 0x60410000, **DCOMcontrol** = 0x60400000, **DCOMopmode** = 0x60600000, **_DCOMopmd_act** = 0x60610000,
PPp_target = 0x607A0000, **PPv_target** = 0x60810000, **PVv_target** = 0x60FF0000, **PTtq_target** = 0x60710000,
RAMP_v_acc = 0x60830000, **RAMP_v_dec** = 0x60840000, **_p_act** = 0x60640000, **_v_act** = 0x606C0000,
_tq_act = 0x60770000, **HMmethod** = 0x60980000, **HMv** = 0x60990001, **HMv_out** = 0x60990002 }
- enum **OperationMode** : int8_t {
ProfilePosition = 1, **Velocity** = 2, **ProfileVelocity** = 3, **ProfileTorque** = 4,
Homing = 6, **InterpolatedPosition** = 7 }
- enum **State** : uint16_t {
mask = 0x006f, **NotReadyToSwitchON** = 0x0000, **SwitchONDisabled** = 0x0040, **ReadyToSwitchON** = 0x0021,
SwitchedON = 0x0023, **OperationEnabled** = 0x0037, **Fault** = 0x000f, **FaultReactionActive** = 0x000f,
QuickStopActive = 0x0007 }
- enum **StatusBits** : uint16_t {
ReadyToSwitchOn_bit = 0x0001, **SwitchedOn_bit** = 0x0002, **OperationEnabled_bit** = 0x0004, **Fault_bit** = 0x0008,
VoltageEnabled_bit = 0x0010, **QuickStop_bit** = 0x0020, **SwitchONDisabled_bit** = 0x0040, **Error0_bit** = 0x0080,
HaltRequest_bit = 0x0100, **Remote_bit** = 0x0200, **TargetReached_bit** = 0x0400, **InternalLimitReached_bit** = 0x0800,
OperationMode_bit = 0x1000, **BlockingError_bit** = 0x2000, **OperationModeStart_bit** = 0x4000, **ValidRef_bit** = 0x8000 }
- enum **Control** : uint16_t {
Shutdown = 0x0006, **SwitchON** = 0x0007, **DisableVoltage** = 0x0000, **QuickStop** = 0x0002,
DisableOperation = 0x0007, **EnableOperation** = 0x000f, **FaultResest** = 0x0080 }
- enum **PDOFunctionCode** : uint32_t {
PDO1Transmit = Message::PDO1Transmit, **PDO1Receive** = Message::PDO1Receive, **PDO2Transmit** = Message::PDO2Transmit, **PDO2Receive** = Message::PDO2Receive,
PDO3Transmit = Message::PDO3Transmit, **PDO3Receive** = Message::PDO3Receive, **PDO4Transmit** = Message::PDO4Transmit, **PDO4Receive** = Message::PDO4Receive }

Public Member Functions

- **Driver** (const char *ifname, uint16_t can_id, int verbose_lvl=0)
Constructor.
- template<typename T >
void **set** (**Register** reg, T val, bool force_sdo=false, bool wait=false)
Enables to store a value in a specified registers.
- template<typename T >
T **get** (**Register** reg, bool force_sdo=false)
get Gets the value of a specified registers.
- void **set_control** (**Control** ctrl)
set_control : Send transition states order.
- **State** **get_state** ()
Returns the current state of the driver by reading the status word.
- void **wait_state** (**State** state, uint16_t _mask=**mask**)
wait_state loop until the driver state is different from the one passed while(actual_state()&mask) != (state&mask))
- void **set_mode** (**OperationMode** mode, bool wait=false)
set_mode Set the desired operation mode.
- **OperationMode** **get_mode** (bool force_sdo=true)

- get_mode* Returns the actual operation mode of the driver.
- bool **set_position** (int32_t target, bool absolute=true)
 - set_position* Send the new position to reach. Has to be in ProfilPositon mode to have some effect.
- bool **set_velocity** (int32_t target)
 - set_velocity* Send the new velocity to reach. Has to be in ProfilPositon or ProfilVelocity mode to have some effect.
- bool **set_torque** (int16_t target)
 - set_torque* Send the new torque to reach. Has to be in ProfilTorque mode to have some effect.
- int32_t **get_position** ()
 - get_position* Returns the actual postion of the motor.
- int32_t **get_velocity** ()
 - get_velocity* Returns the actual velocity of the motor.
- int32_t **get_torque** ()
 - get_torque* Returns the actual torque of the motor.
- void **set_position_offset** (int32_t offset_pos)
 - set_position_offset* Set the postion offset of the motor. Rq better to have a large offset to avoid letting the motor switch off in negativ position: it would result in a wrong position when restarting
- void **start** ()
 - start*
- void **pause** ()
 - pause*
- void **stop** ()
 - stop*
- void **profilePosition_mode** ()
 - profilePosition_mode*
- void **profileVelocity_mode** ()
 - profileVelocity_mode*
- void **profileTorque_mode** ()
 - profileTorque_mode*
- void **homing** ()
 - homing*
- **Parameter** * **get_param** (Register reg)
- virtual void **print_manufacturer_status** ()=0
- std::string **ctrl_to_str** (Control control)
- bool **is_available** ()
 - return true if the can interface is available*

Protected Member Functions

- void **send** (Parameter *param)
 - send the parameter via a Writting SDO message to the driver*
- void **update** (Parameter *param)
 - Request an update of the parameter via a Reading SDO message. (the parameter has been updated when param->sdo_flag is down.*
- void **map_PDO** (PDOFunctionCode fn, Parameter *param, int slot)
 - Enables to map the different parameters of the driver to the Transmit PDO. When a PDO is received in the T_PDO↔_socket() thread, the value of the pdo will be stored in the mapped parameter.*
- void **activate_PDO** (PDOFunctionCode fn, bool set=true)
 - Sends a SDO message to activate the specified PDO.*
- void **T_socket** ()
- void **RPDO_socket** ()

Protected Attributes

- `std::thread * m_rpdo_socket_thread`
- `std::atomic_flag rpdo_socket_flag`
- `std::mutex rpdo_mutex`
- `std::thread * m_t_socket_thread`
- `std::atomic_flag t_socket_flag`
- `const char * m_ifname`
- `int m_verbose_level`
- `bool m_available`
- `CANopen::Socket m_socket`
- `std::map< PDOFunctionCode, std::vector< Parameter * > > m_PDO_map`
- `std::map< Register, Parameter * > m_parameters`
- `uint8_t m_node_id`
- `uint16_t m_can_baud`
- `int32_t m_offset_pos = 0`

7.1.1 Detailed Description

Device Profile Drives and Motion Control.

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

7.1.2 Member Enumeration Documentation

7.1.2.1 Control

```
enum CANopen::Driver::Control : uint16_t
```

Possible Control commands

Enumerator

Shutdown	goto ReadySwitchON
SwitchON	goto SwitchedON
DisableVoltage	goto SwitchONDisabled
QuickStop	goto QuickStopActiv
DisableOperation	goto SwitchedON
EnableOperation	goto OperationEnabled
FaultResest	goto SwitchONDisabled

Definition at line 160 of file [CANopen_driver.h](#).

7.1.2.2 OperationMode

```
enum CANopen::Driver::OperationMode : int8_t
```

Operational modes

Enumerator

ProfilePosition	The positioning of the drive is defined in this mode. Speed, position and acceleration can be limited and profiled moves using a Trajectory Generator are possible as well.
Velocity	Many frequency inverters use this simple mode to control the velocity of the drive with limits and ramp functions.
ProfileVelocity	The Profile Velocity Mode is used to control the velocity of the drive with no special regard of the position. It supplies limit functions and Trajectory Generation.
ProfileTorque	The profile torque mode allows a host (external) control system (i.e. closed-loop speed controller, open-loop transmission force controller) to transmit the target torque value, which is processed via the trajectory generator. The torque slope and torque profile type parameters are required.
Homing	Homming mode.
InterpolatedPosition	The interpolated position mode is used to control multiple coordinated axes or a single axle with the need for time-interpolation of set-point data. The interpolated position mode normally uses time synchronization mechanisms like the sync object defined in /3/ for a time coordination of the related drive units.

Definition at line 72 of file [CANopen_driver.h](#).

7.1.2.3 Register

```
enum CANopen::Driver::Register : uint32_t
```

CIA 402 CANopen [Driver](#)

Enumerator

_DCOMstatus	6041_h - The statusword indicates the current state of the drive. No bits are latched. The statusword consist of bits for: <ul style="list-style-type: none"> the current state of the drive, the operating state of the mode and manufacturer specific options.
DCOMcontrol	6040_h - The controlword consist of bits for: <ul style="list-style-type: none"> the controlling of the state, the controlling of operating modes and manufacturer specific options.
DCOMopmode	6060_h - The parameter modes of operation switches the actually choosen operation mode.

Enumerator

_DCOMopmd_act	6061_h - The modes of operation display shows the current mode of operation. The meaning of the returned value corresponds to that of the modes of operation option code
PPp_target	607A_h - The target position is the position that the drive should move to in position profile mode using the current settings of motion control parameters such as velocity, acceleration, deceleration, motion profile type etc. The target position is given in user defined position units. It is converted to position increments using the position factor. The target position will be interpreted as absolute or relative depending on the 'abs / rel' flag in the controlword.
PPv_target	6081_h - The profile velocity is the velocity normally attained at the end of the acceleration ramp during a profiled move and is valid for both directions of motion. The profile velocity is given in user defined speed units. It is converted to position increments per second using the velocity encoder factor.
PVv_target	60FF_h - The target velocity is the input for the trajectory generator and the value is given in velocity units.
PTtq_target	6071_h - This parameter is the input value for the torque controller in profile torque mode and the value is given per thousand of rated torque.
RAMP_v_acc	6083_h - The profile acceleration is given in user defined acceleration units. It is converted to position increments per second 2 using the normalizing factors.
RAMP_v_dec	6084_h - The profile deceleration is given in the same units as profile acceleration. If this parameter is not supported, then the profile acceleration value is also used for deceleration.
_p_act	6064_h - This object represents the actual value of the position measurement device in user defined units.
_v_act	606C_h - The velocity actual value is also represented in velocity units and is coupled to the velocity used as input to the velocity controller.
_tq_act	6077_h - The torque actual value corresponds to the instantaneous torque in the drive motor. The value is given per thousand of rated torque.
HMmethod	6098_h - The homing method object determines the method that will be used during homing.
HMv	6099_h01 - Speed during search for switch.
HMv_out	6099_h02 - Speed during search for zero

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

7.1.2.4 State

```
enum CANopen::Driver::State : uint16_t
```

Possible States

Enumerator

mask	Keeping only main state bytes.
NotReadyToSwitchtON	Not Ready to Switch ON: <ul style="list-style-type: none"> • Low level Power (e.g. 15V, 5V) has been applied to the drive. • The drive is being initialized or is running self test. • A brake, if present, has to be applied in this state. • The drive function is disabled.
	Generated by Doxygen

Enumerator

SwitchONDisabled	<p>Switch ON Disabled:</p> <ul style="list-style-type: none"> • Drive Initialisation is complete. • The drive parameters have been set up. • Drive parameters may be changed. • High Voltage may not be applied to the drive, (e.g. for safety reasons). • The drive function is disabled.
ReadyToSwitchON	<p>Ready to Switch ON:</p> <ul style="list-style-type: none"> • High Voltage may be applied to the drive. • The drive parameters may be changed. • The drive function is disabled
SwitchedON	<p>Switched ON:</p> <ul style="list-style-type: none"> • High Voltage has been applied to the drive. • The Power Amplifier is ready. • The drive parameters may be changed. • The drive function is disabled.
OperationEnabled	<p>Operation Enabled:</p> <ul style="list-style-type: none"> • No faults have been detected. • The drive function is enabled and power is applied to the motor. • The drive parameters may be changed. (This corresponds to normal operation of the drive.)
Fault	<p>Fault:</p> <ul style="list-style-type: none"> • The drive parameters may be changed. • A fault has occurred in the drive. • The drive function is disabled.
FaultReactionActive	<p>Fault Reaction Active:</p> <ul style="list-style-type: none"> • The drive parameters may be changed. • A non-fatal fault has occurred in the drive. • The Quick Stop function is being executed. • The drive function is enabled and power is applied to the motor.

Enumerator

QuickStopActive	<p>Quick Stop Active:</p> <ul style="list-style-type: none"> • The drive parameters may be changed. The Quick Stop function is being executed. • The drive function is enabled and power is applied to the motor. If the 'Quick-Stop-Option-Code' is switched to 5 (Stay in Quick-Stop), you can't leave the Quick-Stop-State, but you can transmit to 'Operation Enable' with the command 'Enable Operation'
-----------------	---

Definition at line 91 of file [CANopen_driver.h](#).

7.1.3 Constructor & Destructor Documentation

7.1.3.1 Driver()

```
CANopen::Driver::Driver (
    const char * ifname,
    uint16_t can_id,
    int verbose_lvl = 0 )
```

Constructor.

Parameters

<i>ifname</i>	: Name of the CAN interface.
<i>can_id</i>	: Node CAN ID of the driver.
<i>verbose</i> ↔ <i>_lvl</i>	: Level of verbosity.

Definition at line 4 of file [CANopen_driver.cpp](#).

References [_DCOMstatus](#).

7.1.4 Member Function Documentation

7.1.4.1 activate_PDO()

```
void CANopen::Driver::activate_PDO (
    PDOFunctionCode fn,
    bool set = true ) [protected]
```

Sends a SDO message to activate the specified PDO.

Parameters

<i>ifname</i>	: N
<i>can↔ _id</i>	: Node CAN ID of the driver.

Definition at line 59 of file [CANopen_driver.cpp](#).

7.1.4.2 ctrl_to_str()

```
std::string CANopen::Driver::ctrl_to_str (
    Control control )
```

< goto ReadySwitchON

< goto SwitchedON

< goto SwitchONDisabled

< goto QuickStopActiv

< goto OperationEnabled

Definition at line 346 of file [CANopen_driver.cpp](#).

7.1.4.3 get()

```
template<typename T >
T CANopen::Driver::get (
    Register reg,
    bool force_sdo = false ) [inline]
```

get Gets the value of a specified registers.

Parameters

<i>reg</i>	: The register to get. (In the format ind__sub)
<i>force_sdo</i>	: If true, the parameter will be updated via a reading SDO.

Returns

The value of the register in the templated format T selected.

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

References [update\(\)](#).

Here is the call graph for this function:



7.1.4.4 get_mode()

```
OperationMode CANopen::Driver::get_mode (
    bool force_sdo = true ) [inline]
```

get_mode Returns the actual operation mode of the driver.

Parameters

<i>force_sdo</i>	If set a sdo read message will be send to get the mode. Set to false to save some communication time.
------------------	---

Returns

The actual operational mode

Definition at line 267 of file [CANopen_driver.h](#).

References [_DCOMopmd_act](#).

7.1.4.5 get_position()

```
int32_t CANopen::Driver::get_position ( ) [inline]
```

get_position Returns the actual postion of the motor.

Returns

The actual postion of the motor.

Definition at line 297 of file [CANopen_driver.h](#).

References [_p_act](#).

7.1.4.6 get_state()

```
State CANopen::Driver::get_state ( ) [inline]
```

Returns the current state of the driver by reading the status word.

Returns

The current state.

Definition at line 240 of file [CANopen_driver.h](#).

References [_DCOMstatus](#).

Referenced by [wait_state\(\)](#).

7.1.4.7 get_torque()

```
int32_t CANopen::Driver::get_torque ( ) [inline]
```

get_torque Returns the actual torque of the motor.

Returns

The actual torque of the motor.

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

References [_tq_act](#).

7.1.4.8 get_velocity()

```
int32_t CANopen::Driver::get_velocity ( ) [inline]
```

get_velocity Returns the actual velocity of the motor.

Returns

The actual velocity of the motor.

Definition at line 303 of file [CANopen_driver.h](#).

References [_v_act](#).

7.1.4.9 map_PDO()

```
void CANopen::Driver::map_PDO (
    PDOFunctionCode fn,
    Parameter * param,
    int slot ) [protected]
```

Enables to map the different parameters of the driver to the Transmit PDO. When a PDO is received in the T_PDO↔O_socket() thread, the value of the pdo will be stored in the mapped parameter.

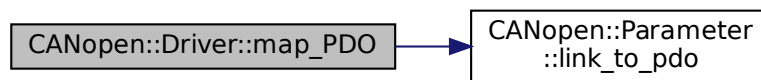
Parameters

<i>fn</i>	: Function code of the PDO.
<i>param</i>	: Parameter to map.
<i>slot</i>	: Slot of the parameter in the PDO message.

Definition at line 50 of file [CANopen_driver.cpp](#).

References [CANopen::Parameter::link_to_pdo\(\)](#).

Here is the call graph for this function:



7.1.4.10 send()

```
void CANopen::Driver::send (  
    Parameter * param ) [protected]
```

send the parameter via a Writing SDO message to the driver

Parameters

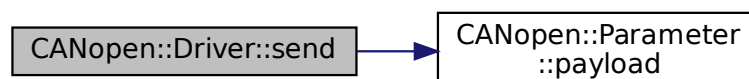
<i>param</i>	Parameter to send.
--------------	------------------------------------

Definition at line 96 of file [CANopen_driver.cpp](#).

References [CANopen::Parameter::payload\(\)](#).

Referenced by [set\(\)](#).

Here is the call graph for this function:



7.1.4.11 set()

```
template<typename T >
void CANopen::Driver::set (
    Register reg,
    T val,
    bool force_sdo = false,
    bool wait = false ) [inline]
```

Enables to store a value in a specified registers.

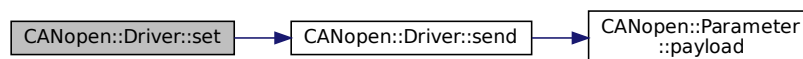
Parameters

<i>reg</i>	: The register to set. (In the format ind__sub)
<i>val</i>	: The value to store in the register.
<i>force_sdo</i>	: If true, the parameter will be send to the driver via a SDO. Else the parametr will be sent via PDO if it was mapped to an activated RPDO.
<i>wait</i>	: If set, the function is blocking and wait for the parameter to be sent via SDO.

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

References [send\(\)](#).

Here is the call graph for this function:



7.1.4.12 set_control()

```
void CANopen::Driver::set_control (
    Control ctrl )
```

`set_control` : Send transition states order.

Parameters

<i>ctrl</i>	: Control to send.
-------------	--------------------

Definition at line 439 of file [CANopen_driver.cpp](#).

7.1.4.13 set_mode()

```
void CANopen::Driver::set_mode (
    OperationMode mode,
    bool wait = false )
```

set_mode Set the desired operation mode.

Parameters

<i>mode</i>	Mode to set.
<i>wait</i>	Repeatidly test the actual operation mode register until it is equal to the selected mode.

Definition at line 429 of file [CANopen_driver.cpp](#).

7.1.4.14 set_position()

```
bool CANopen::Driver::set_position (
    int32_t target,
    bool absolute = true )
```

set_position Send the new position to reach. Has to be in ProfilPositon mode to have some effect.

Parameters

<i>target</i>	Position to reach (in internal unit)
<i>absolute</i>	If set the target will be process as an absolute value. Else it will be procecced as a relative (to the current position) value.

Returns

True if successfully sent.

Definition at line 239 of file [CANopen_driver.cpp](#).

7.1.4.15 set_position_offset()

```
void CANopen::Driver::set_position_offset (
    int32_t offset_pos ) [inline]
```

set_position_offset Set the postion offset of the motor. Rq better to have a large offset to avoid letting the motor switch off in negativ position: it would result in a wrong position when restarting

Parameters

<i>offset_pos</i>	Offest of the motor in internal Unit
-------------------	--------------------------------------

Definition at line 316 of file [CANopen_driver.h](#).

7.1.4.16 set_torque()

```
bool CANopen::Driver::set_torque (
    int16_t target )
```

set_torque Send the new torque to reach. Has to be in ProfilTorque mode to have some effect.

Parameters

<i>target</i>	Torque to reach (in internal unit)
---------------	------------------------------------

Returns

True if successfully sent.

Definition at line 275 of file [CANopen_driver.cpp](#).

7.1.4.17 set_velocity()

```
bool CANopen::Driver::set_velocity (
    int32_t target )
```

set_velocity Send the new velocity to reach. Has to be in ProfilPositon or ProfilVelocity mode to have some effect.

Parameters

<i>target</i>	Velocity to reach (in internal unit)
---------------	--------------------------------------

Returns

True if successfully sent.

Definition at line 260 of file [CANopen_driver.cpp](#).

7.1.4.18 update()

```
void CANopen::Driver::update (
    Parameter * param ) [protected]
```

Request an update of the parameter via a Reading SDO message. (the parameter has been updated when param->sdo_flag is down.

Parameters

<i>param</i>	Parameter to update.
--------------	----------------------

Definition at line 112 of file [CANopen_driver.cpp](#).

Referenced by [get\(\)](#).

7.1.4.19 wait_state()

```
void CANopen::Driver::wait_state (
    State state,
    uint16_t _mask = mask ) [inline]
```

wait_state loop until the driver state is different from the one passed while(actual_state()&mask) != (state&mask))

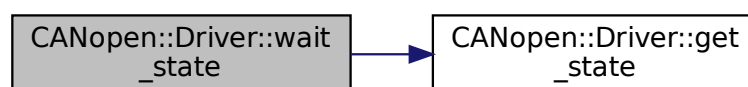
Parameters

<i>state</i>	State to wait for.
<i>_mask</i>	Mask to selected specific bits of the state.

Definition at line 248 of file [CANopen_driver.h](#).

References [get_state\(\)](#), and [mask](#).

Here is the call graph for this function:



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

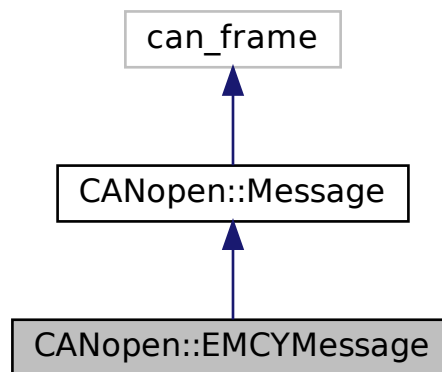
- [CANopen_driver.h](#)
- [CANopen_driver.cpp](#)

7.2 CANopen::EMCYMessage Class Reference

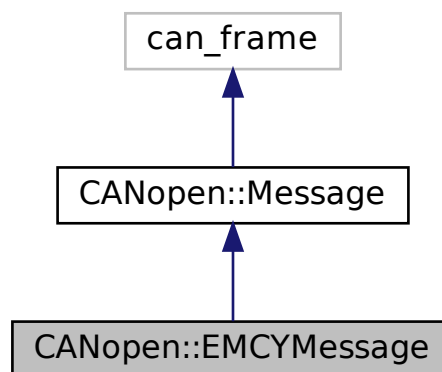
EMCY [Message](#) (Emergency Object)

```
#include <emcy.h>
```

Inheritance diagram for CANopen::EMCYMessage:



Collaboration diagram for CANopen::EMCYMessage:



Public Member Functions

- **EMCYMessage** (const can_frame &other)
- uint16_t **code** () const
- uint8_t **reg** () const

Additional Inherited Members

7.2.1 Detailed Description

EMCY [Message](#) (Emergency Object)

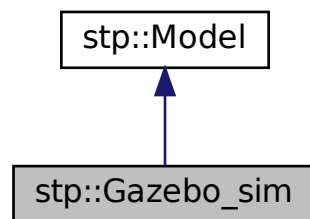
Definition at line 17 of file [emcy.h](#).

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

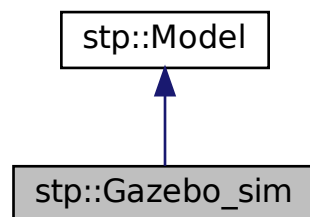
- [emcy.h](#)
- [emcy.cpp](#)

7.3 stp::Gazebo_sim Class Reference

Inheritance diagram for stp::Gazebo_sim:



Collaboration diagram for stp::Gazebo_sim:



Public Member Functions

- **Gazebo_sim** (double deltas[4], double a, double l)
- double * **new_pos** (double T[3], double theta[3])
- void **update** ()

Public Attributes

- gazebo::transport::NodePtr **m_node**
- gazebo::transport::PublisherPtr **m_pub**

Additional Inherited Members

7.3.1 Detailed Description

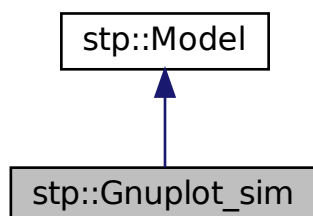
Definition at line 15 of file [gazebo_sim.hpp](#).

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

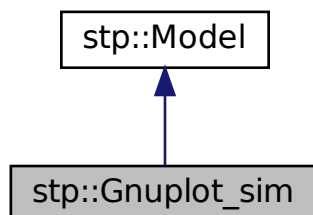
- gazebo_sim.hpp
- gazebo_sim.cpp

7.4 stp::Gnuplot_sim Class Reference

Inheritance diagram for stp::Gnuplot_sim:



Collaboration diagram for stp::Gnuplot_sim:



Public Member Functions

- **Gnuplot_sim** (double deltas[4], double a, double l)
- double * **new_pos** (double T[3], double theta[3])
- void **update_draw** ()
- void **draw** ()

Additional Inherited Members

7.4.1 Detailed Description

Definition at line 11 of file [gnuplot_sim.hpp](#).

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

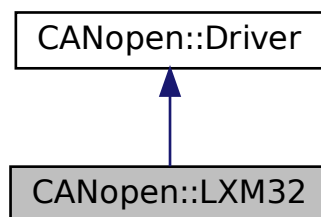
- [gnuplot_sim.hpp](#)
- [gnuplot_sim.cpp](#)

7.5 CANopen::LXM32 Class Reference

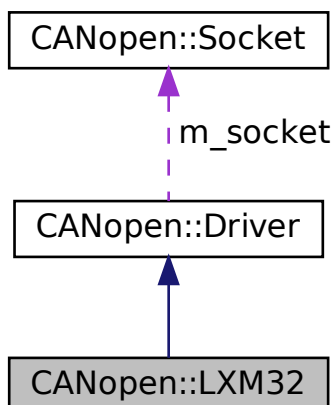
Implementation of the [Driver](#) Class for a [LXM32](#) driver.

```
#include <CANopen_lxm32.h>
```

Inheritance diagram for CANopen::LXM32:



Collaboration diagram for CANopen::LXM32:



Public Member Functions

- [LXM32](#) (const char *ifname, uint16_t can_id, bool verbose=false)
Constructor.
- bool [set_angle](#) (double ang, bool absolute=true, bool radian=true)
set_angle Sets the motor at a desired angle.
- double [get_angle](#) (bool radian=true)
get_angle Gets the motor at a desired angle.
- void [print_manufacturer_status](#) ()

Public Attributes

- int [nb_index_per_turn](#) = 737280

Additional Inherited Members

7.5.1 Detailed Description

Implementation of the [Driver](#) Class for a [LXM32](#) driver.

Definition at line 22 of file [CANopen_lxm32.h](#).

7.5.2 Constructor & Destructor Documentation

7.5.2.1 LXM32()

```
CANopen::LXM32::LXM32 (
    const char * ifname,
    uint16_t can_id,
    bool verbose = false )
```

Constructor.

Parameters

<i>ifname</i>	: Name of the CAN interface.
<i>can_id</i>	: Node CAN ID of the driver.
<i>verbose</i>	

Definition at line 3 of file [CANopen_lxm32.cpp](#).

7.5.3 Member Function Documentation

7.5.3.1 `get_angle()`

```
double CANopen::LXM32::get_angle (
    bool radian = true )
```

`get_angle` Gets the motor at a desired angle.

Parameters

<i>radian</i>	If set the angle return in radian else degree.
---------------	--

Returns

The actual motor angle.

Definition at line 16 of file [CANopen_lxm32.cpp](#).

7.5.3.2 `set_angle()`

```
bool CANopen::LXM32::set_angle (
    double ang,
    bool absolute = true,
    bool radian = true )
```

`set_angle` Sets the motor at a desired angle.

Parameters

<i>ang</i>	Angle to reached.
<i>absolute</i>	If set the angle will be interpreted as a absolute angle. Else it will be interpreted as a relative angle.
<i>radian</i>	If set the angle will be interpreted in radian else degree.

Returns

True if the position was sent.

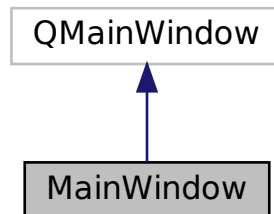
Definition at line 9 of file [CANopen_lxm32.cpp](#).

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

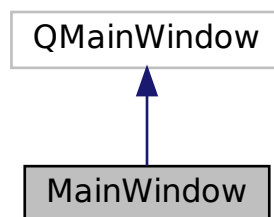
- [CANopen_lxm32.h](#)
- [CANopen_lxm32.cpp](#)

7.6 MainWindow Class Reference

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Public Member Functions

- **MainWindow** (QWidget *parent=NULLPTR)
 - void [update_pos](#) ()
update_pos Update the GUI to display the current position of the platform.
- void [loop](#) ()
loop Called every 1ms to send order depending of the mode.

7.6.1 Detailed Description

Definition at line 14 of file [mainwindow.h](#).

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

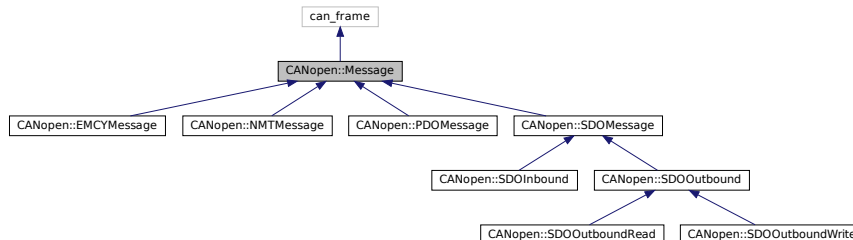
- [mainwindow.h](#)
- [mainwindow.cpp](#)

7.7 CANopen::Message Class Reference

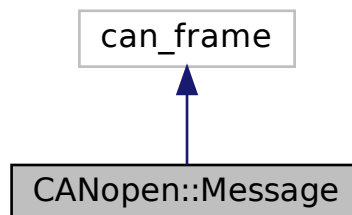
can_frame object sent and received through CANopen socket.

```
#include <message.h>
```

Inheritance diagram for CANopen::Message:



Collaboration diagram for CANopen::Message:



Public Types

- enum **FunctionCode** : uint32_t {
NMT = 0, **Emergency** = 0x80, **Sync** = 0x80, **TimeStamp** = 0x100,
PDO1Transmit = 0x180, **PDO1Receive** = 0x200, **PDO2Transmit** = 0x280, **PDO2Receive** = 0x300,
PDO3Transmit = 0x380, **PDO3Receive** = 0x400, **PDO4Transmit** = 0x480, **PDO4Receive** = 0x500,
SDOTransmit = 0x580, **SDOReceive** = 0x600, **Heartbeat** = 0x700 }

Function codes.

Public Member Functions

- **Message** (const can_frame &other)
- **Message** (uint32_t cob_id, **Payload** payload)
- **operator can_frame** * () const
- **FunctionCode** function_code () const

function_code

- uint8_t [node_id](#) () const
node_id
- virtual [Payload](#) [payload](#) () const
payload
- virtual uint32_t [id](#) () const
id
- std::string [to_string](#) () const

7.7.1 Detailed Description

can_frame object sent and received through CANopen socket.

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

7.7.2 Member Enumeration Documentation

7.7.2.1 FunctionCode

```
enum CANopen::Message::FunctionCode : uint32_t
```

Function codes.

Enumerator

NMT	0 _h NMT Message (Network management)
Emergency	80 _h EMCY Message (Emergency Object)
Sync	80 _h
TimeStamp	100 _h
PDO1Transmit	180 _h PDO1 Transmitting Message (Process Data Object)
PDO1Receive	200 _h PDO1 Receiving Message (Process Data Object)
PDO2Transmit	280 _h PDO2 Transmitting Message (Process Data Object)
PDO2Receive	300 _h PDO2 Receiving Message (Process Data Object)
PDO3Transmit	380 _h PDO3 Transmitting Message (Process Data Object)
PDO3Receive	400 _h PDO3 Receiving Message (Process Data Object)
PDO4Transmit	480 _h PDO4 Transmitting Message (Process Data Object)
PDO4Receive	500 _h PDO4 Receiving Message (Process Data Object)
SDOTransmit	580 _h SDO Transmitting Message (Service Data Object)
SDOReceive	600 _h SDO Receiving Message (Service Data Object)
Heartbeat	700 _h

Definition at line 25 of file [message.h](#).

7.7.3 Member Function Documentation

7.7.3.1 `function_code()`

```
Message::FunctionCode CANopen::Message::function_code ( ) const
```

`function_code`

Returns

Returns the function code of the message (without the node id)

Definition at line 19 of file [message.cpp](#).

7.7.3.2 `id()`

```
virtual uint32_t CANopen::Message::id ( ) const [inline], [virtual]
```

`id`

Returns

Depends of the child class : return `index__sub` if [SDOMessage](#) or pdo number if [PDOMessage](#)

Reimplemented in [CANopen::SDOMessage](#).

Definition at line 74 of file [message.h](#).

7.7.3.3 `node_id()`

```
uint8_t CANopen::Message::node_id ( ) const
```

`node_id`

Returns

Returns the ID of the node sending or getting the message (extracted from the COB ID)

Definition at line 24 of file [message.cpp](#).

7.7.3.4 payload()

```
Payload CANopen::Message::payload ( ) const [virtual]
```

payload

Returns

Returns the payload of the message.

Reimplemented in [CANopen::SDOMessage](#).

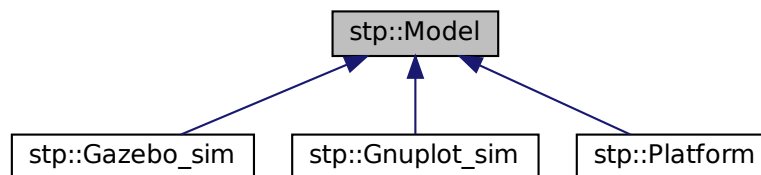
Definition at line 29 of file [message.cpp](#).

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

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

7.8 stp::Model Class Reference

Inheritance diagram for stp::Model:



Public Member Functions

- **Model** (double deltas[4], double a, double l, int verbose_level=0)
- double * **new_pos** (double T[3], double theta[3])
- double **get_T** (int i)
- double **get_theta** (int i)
- void **init_pos** ()
- void **compute_R** (double theta[3])
- void **print_pos** ()

Protected Member Functions

- void **set_T** (double t0, double t1, double t2)
- void **set_T_spd** (double t0, double t1, double t2)
- void **set_theta** (double t0, double t1, double t2)
- void **set_theta_spd** (double t0, double t1, double t2)

Protected Attributes

- double **m_radius** [2]
- double **m_a**
- double **m_a2**
- double **m_l**
- double **m_l2**
- double **_d2** [NB_LEGS]
- double **m_P** [NB_LEGS][3]
- double **m_P1** [NB_LEGS][3]
- double **m_B** [NB_LEGS][3]
- double **m_A** [NB_LEGS][3]
- double **m_parity** [NB_LEGS]
- double **m_beta** [NB_LEGS]
- double **m_alpha** [NB_LEGS]
- double **m_gamma** [2][NB_LEGS]
- double **m_T** [3]
- double **m_theta** [3]
- double **m_R** [3][3]
- double **m_trig** [6]
- double **m_T_spd** [3]
- double **m_theta_spd** [3]
- double **m_alpha_spd** [6]
- double **_alpha** [6]
- int **m_verbose_level** = false

7.8.1 Detailed Description

Definition at line 11 of file [model.hpp](#).

7.8.2 Member Data Documentation

7.8.2.1 _d2

```
double stp::Model::_d2[NB_LEGS] [protected]
```

Distance between the axe of the motors and the coresponding legs-platform articulation.

Definition at line 53 of file [model.hpp](#).

7.8.2.2 m_A

```
double stp::Model::m_A[NB_LEGS][3] [protected]
```

Position of the arms-legs articulation in the base B0.

Definition at line 65 of file [model.hpp](#).

7.8.2.3 m_a2

```
double stp::Model::m_a2 [protected]
```

Size of the motors' arms.

Definition at line 50 of file [model.hpp](#).

7.8.2.4 m_alpha

```
double stp::Model::m_alpha[NB_LEGS] [protected]
```

Angular position of the rotor.

Definition at line 69 of file [model.hpp](#).

7.8.2.5 m_alpha_spd

```
double stp::Model::m_alpha_spd[6] [protected]
```

Rotation speed of the motors.

Definition at line 80 of file [model.hpp](#).

7.8.2.6 m_B

```
double stp::Model::m_B[NB_LEGS][3] [protected]
```

Position of the motors-arms articulation in the base B0.

Definition at line 63 of file [model.hpp](#).

7.8.2.7 m_beta

```
double stp::Model::m_beta[NB_LEGS] [protected]
```

Orientation of the motor on the base.

Definition at line 68 of file [model.hpp](#).

7.8.2.8 m_gamma

```
double stp::Model::m_gamma[2][NB_LEGS] [protected]
```

Angular position of the motor on the base

Definition at line 71 of file [model.hpp](#).

7.8.2.9 m_l2

```
double stp::Model::m_l2 [protected]
```

Size of the legs.

Definition at line 51 of file [model.hpp](#).

7.8.2.10 m_P

```
double stp::Model::m_P[NB_LEGS][3] [protected]
```

Position of the legs-platform articulation in the base B0.

Definition at line 57 of file [model.hpp](#).

7.8.2.11 m_P1

```
double stp::Model::m_P1[NB_LEGS][3] [protected]
```

Position of the legs-platform articulation in the base B1.

Definition at line 60 of file [model.hpp](#).

7.8.2.12 m_parity

```
double stp::Model::m_parity[NB_LEGS] [protected]
```

Use to represent the side of the motor arm.

Definition at line 67 of file [model.hpp](#).

7.8.2.13 m_R

```
double stp::Model::m_R[3][3] [protected]
```

Matrix of rotation.

Definition at line 75 of file [model.hpp](#).

7.8.2.14 m_T

```
double stp::Model::m_T[3] [protected]
```

Postion of the center of the platform.

Definition at line 73 of file [model.hpp](#).

7.8.2.15 m_T_spd

```
double stp::Model::m_T_spd[3] [protected]
```

Velocity of the platform.

Definition at line 78 of file [model.hpp](#).

7.8.2.16 m_theta

```
double stp::Model::m_theta[3] [protected]
```

Orientation of the platform.

Definition at line 74 of file [model.hpp](#).

7.8.2.17 m_theta_spd

```
double stp::Model::m_theta_spd[3] [protected]
```

Rotation speed of the platform.

Definition at line 79 of file [model.hpp](#).

7.8.2.18 m_trig

```
double stp::Model::m_trig[6] [protected]
```

```
(cos(theta_i),sin(theta_i))
```

Definition at line 76 of file [model.hpp](#).

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

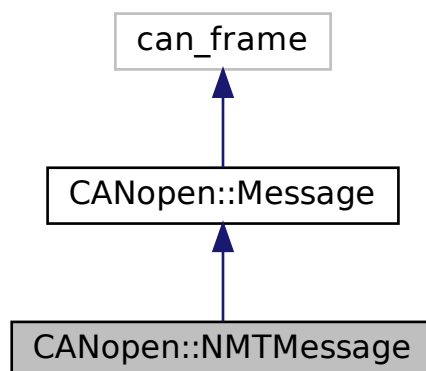
- [model.hpp](#)
- [model.cpp](#)

7.9 CANopen::NMTMessage Class Reference

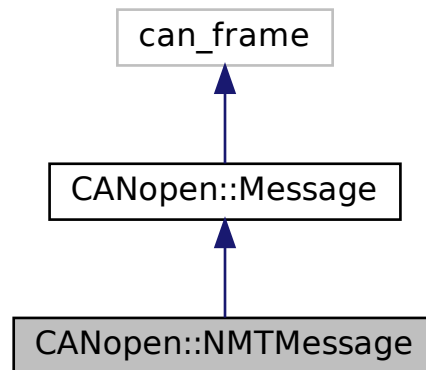
NMT [Message](#) (Network management)

```
#include <nmt.h>
```

Inheritance diagram for CANopen::NMTMessage:



Collaboration diagram for CANopen::NMTMessage:



Public Types

- enum **Code** : uint8_t {
Initialising = 0, **GoToOperational** = 0x01, **GoToStopped** = 0x02, **Stopped** = 0x04,
Operational = 0x05, **PreOperational** = 0x7f, **GoToPreOperational** = 0x80, **GoToResetNode** = 0x81,
GoToResetCommunication = 0x82 }

Public Member Functions

- NMTMessage** (const can_frame &other)
- NMTMessage** (Code code, uint8_t [node_id](#))

7.9.1 Detailed Description

NMT [Message](#) (Network management)

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

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

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

7.10 CANopen::Parameter Struct Reference

Object from the object dictionary of a remote CANopen device.

```
#include <parameter.h>
```

Public Types

- enum [PDOFunctionCode](#) : uint32_t {
PDO1Transmit = Message::PDO1Transmit, **PDO1Receive** = Message::PDO1Receive, **PDO2Transmit** = Message::PDO2Transmit, **PDO2Receive** = Message::PDO2Receive,
PDO3Transmit = Message::PDO3Transmit, **PDO3Receive** = Message::PDO3Receive, **PDO4Transmit** = Message::PDO4Transmit, **PDO4Receive** = Message::PDO4Receive }
PDO Function code.
- typedef void(* [param_cb_t](#)) ([Parameter](#) *)

Public Member Functions

- template<typename T >
[Parameter](#) (std::string name_, T val, uint16_t index_, uint8_t subindex_, [param_cb_t](#) cb=nullptr)
[Parameter](#) Constructor.
- template<typename T >
[Parameter](#) (std::string name_, T val, uint32_t index__sub, [param_cb_t](#) cb=nullptr)
[Parameter](#) Constructor.
- void [link_to_pdo](#) ([PDOFunctionCode](#) fn, int8_t slot)
[link_to_pdo](#) Links a paramet to a PDO message
- template<typename T >
bool [set](#) (T val, bool force_update=false, bool received_data=false)
sets Set the value of a parameter.
- bool [operator=](#) (int8_t val)
Sets the value of the parameter.
- bool [operator=](#) (int16_t val)
Sets the value of the parameter.
- bool [operator=](#) (int32_t val)
Sets the value of the parameter.
- bool [operator=](#) (uint8_t val)
Sets the value of the parameter.
- bool [operator=](#) (uint16_t val)
Sets the value of the parameter.
- bool [operator=](#) (uint32_t val)
Sets the value of the parameter.
- template<typename T >
T [get](#) ()
gets Returns the value (with a type T coherent with the data size)
- [operator int8_t](#) ()
operator int8_t Returns the value as an int8_t
- [operator int16_t](#) ()
operator int16_t Returns the value as an int16_t
- [operator int32_t](#) ()
operator int32_t Returns the value as an int32_t
- [operator uint8_t](#) ()
operator uint8_t Returns the value as an uint8_t
- [operator uint16_t](#) ()
operator uint16_t Returns the value as an uint16_t
- [operator uint32_t](#) ()
operator uint32_t Returns the value as an uint32_t
- bool [from_payload](#) ([Payload](#) &p, int slot=0, bool received_data=true)

- from_payload* Sets the value of the parameters with the data of a payload.
- bool [has_been_sent](#) ()
has_been_sent Returns the sending flag.
- void [callback](#) ()
callback Execute the parameter callback (If not null)
- [Payload](#) [payload](#) (bool *should_be_sent=nullptr)
payload Returns a payload filled with the parameter data.

Public Attributes

- size_t **size** = 0
- std::string **name**
- uint16_t **index** = 0
- uint8_t **subindex** = 0
- [PDOFunctionCode](#) **pdo_fn**
- int8_t **pdo_slot** = -1
- std::atomic_flag **sdo_flag**

7.10.1 Detailed Description

Object from the object dictionary of a remote CANopen device.

Definition at line 24 of file [parameter.h](#).

7.10.2 Member Typedef Documentation

7.10.2.1 param_cb_t

```
typedef void(* CANopen::Parameter::param_cb_t) (Parameter *)
```

[Parameter](#) Callback function type

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

7.10.3 Constructor & Destructor Documentation

7.10.3.1 Parameter() [1/2]

```
template<typename T >
CANopen::Parameter::Parameter (
    std::string name_,
    T val,
    uint16_t index_,
    uint8_t subindex_,
    param_cb_t cb = nullptr ) [inline]
```

[Parameter](#) Constructor.

Parameters

<i>name_</i>	Name of the parameter.
<i>val</i>	Value to store in the parameter (the type will be used to fixed the data size)
<i>index_</i>	Index of the object in the object dictionary of the device.
<i>subindex_</i>	Subindex of the object in the object dictionary of the device.
<i>cb</i>	[facultative] The address of a callback function that will be called each time the parameter is updated by the device.

Definition at line 56 of file [parameter.h](#).

7.10.3.2 Parameter() [2/2]

```
template<typename T >
CANopen::Parameter::Parameter (
    std::string name_,
    T val,
    uint32_t index__sub,
    param_cb_t cb = nullptr ) [inline]
```

[Parameter](#) Constructor.

Parameters

<i>name_</i>	Name of the parameter.
<i>val</i>	Value to store in the parameter (the type will be used to fixed the data size)
<i>index__sub</i>	Index and subindex of the object in the object dictionary of the device. (in the format index__sub)
<i>cb</i>	[facultative] The address of a callback function that will be called each time the parameter is updated by the device.

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

7.10.4 Member Function Documentation

7.10.4.1 from_payload()

```
bool CANopen::Parameter::from_payload (
    Payload & p,
    int slot = 0,
    bool received_data = true )
```

`from_payload` Sets the value of the parameters with the data of a payload.

Parameters

<i>p</i>	Payload to store in the parameter
<i>slot</i>	Index in the payload array where the parameter data is
<i>received_data</i>	If set the data is processed as a received data (no sending flag raised)

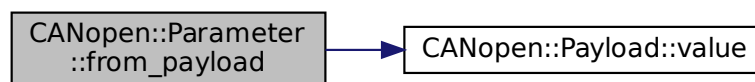
Returns

True if the parameter value has been changed.

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

References [CANopen::Payload::value\(\)](#).

Here is the call graph for this function:



7.10.4.2 get()

```
template<typename T >
T CANopen::Parameter::get ( ) [inline]
```

gets Returns the value (with a type T coherent with the data size)

Returns

The value of the parameter.

Definition at line 164 of file [parameter.h](#).

7.10.4.3 has_been_sent()

```
bool CANopen::Parameter::has_been_sent ( ) [inline]
```

has_been_sent Returns the sending flag.

Returns

The sending flag.

Definition at line 214 of file [parameter.h](#).

7.10.4.4 link_to_pdo()

```
void CANopen::Parameter::link_to_pdo (
    PDOFunctionCode fn,
    int8_t slot )
```

link_to_pdo Links a paramet to a PDO message

Parameters

<i>fn</i>	The function code of the PDO
<i>slot</i>	The index of the parameter data inside the PDO Message Payload

Definition at line 4 of file [parameter.cpp](#).

Referenced by [CANopen::Driver::map_PDO\(\)](#).

7.10.4.5 operator=() [1/6]

```
bool CANopen::Parameter::operator= (
    int16_t val ) [inline]
```

Sets the value of the parameter.

Parameters

<i>val</i>	New int16_t value to set
------------	--------------------------

Returns

True if the parameter value has been changed.

Definition at line 128 of file [parameter.h](#).

7.10.4.6 operator=() [2/6]

```
bool CANopen::Parameter::operator= (
    int32_t val ) [inline]
```

Sets the value of the parameter.

Parameters

<i>val</i>	New int32_t value to set
------------	--------------------------

Returns

True if the parameter value has been changed.

Definition at line 135 of file [parameter.h](#).

7.10.4.7 operator=() [3/6]

```
bool CANopen::Parameter::operator= (
    int8_t val ) [inline]
```

Sets the value of the parameter.

Parameters

<i>val</i>	New int8_t value to set
------------	-------------------------

Returns

True if the parameter value has been changed.

Definition at line 121 of file [parameter.h](#).

7.10.4.8 operator=() [4/6]

```
bool CANopen::Parameter::operator= (
    uint16_t val ) [inline]
```

Sets the value of the parameter.

Parameters

<i>val</i>	New uint16_t value to set
------------	---------------------------

Returns

True if the parameter value has been changed.

Definition at line 149 of file [parameter.h](#).

7.10.4.9 operator=() [5/6]

```
bool CANopen::Parameter::operator= (
    uint32_t val ) [inline]
```

Sets the value of the parameter.

Parameters

<i>val</i>	New uint32_t value to set
------------	---------------------------

Returns

True if the parameter value has been changed.

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

7.10.4.10 operator=() [6/6]

```
bool CANopen::Parameter::operator= (
    uint8_t val ) [inline]
```

Sets the value of the parameter.

Parameters

<i>val</i>	New uint8_t value to set
------------	--------------------------

Returns

True if the parameter value has been changed.

Definition at line 142 of file [parameter.h](#).

7.10.4.11 payload()

```
CANopen::Payload CANopen::Parameter::payload (
    bool * should_be_sent = nullptr )
```

payload Returns a payload filled with the parameter data.

Parameters

<i>should_be_sent</i>	If a boolean pointer is passed the sending flag is stored in the booled and then cleared.
-----------------------	---

Returns

A payload filled with the parameter data.

Definition at line 29 of file [parameter.cpp](#).

Referenced by [CANopen::Driver::send\(\)](#).

7.10.4.12 set()

```
template<typename T >
bool CANopen::Parameter::set (
    T val,
    bool force_update = false,
    bool received_data = false ) [inline]
```

sets Set the value of a parameter.

Parameters

<i>val</i>	The value to set. (the type has to be coherent with the parameter data size)
<i>force_update</i>	If set the value of the Parameter is set even idf the previous value has not been sent yet to the remote device.
<i>received_data</i>	Has to be set to True if the data being set come from the remote device (so the sending flag will not be raised)

Returns

True if the parameter value has been changed.

Definition at line 96 of file [parameter.h](#).

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

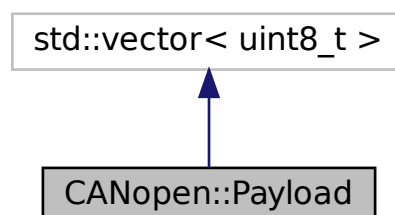
- [parameter.h](#)
- [parameter.cpp](#)

7.11 CANopen::Payload Class Reference

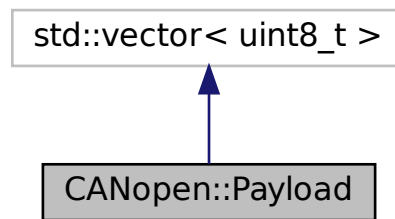
[Payload](#) of CANopen message: array of 1 to 8 bytes of data.

```
#include <payload.h>
```

Inheritance diagram for CANopen::Payload:



Collaboration diagram for CANopen::Payload:



Public Member Functions

- **Payload** (const [Payload](#) &)=default
- **Payload** (const std::vector< uint8_t > &other)
- template<typename T >
[Payload](#) (T value)
Payload Constructor from a standard data type variable. It transform the n bytes of the data into a array of n bytes.
- **Payload & operator=** (const [Payload](#) &)=default
- template<typename T >
T & [value](#) (unsigned begin=0)
value Returns the data casted as a variable of type T
- template<typename T >
[Payload](#) & **operator<<** (T &&value)
operator << Adds a variable inside the payload
- **Payload & operator<<** ([Payload](#) &&p)
- *operator << Adds a variable inside the payload*
- **Payload & store_at** ([Payload](#) &&p, int slot)
store_at Stores a variable at a specified index in the data array.
- **operator std::string** () const

7.11.1 Detailed Description

[Payload](#) of CANopen message: array of 1 to 8 bytes of data.

Definition at line 22 of file [payload.h](#).

7.11.2 Constructor & Destructor Documentation

7.11.2.1 Payload()

```

template<typename T >
CANopen::Payload::Payload (
    T value ) [inline]
  
```

[Payload](#) Constructor from a standard data type variable. It transform the n bytes of the data into a array of n bytes.

Parameters

<i>value</i>	The variable to store in the payload
--------------	--------------------------------------

Definition at line 33 of file [payload.h](#).

References [value\(\)](#).

Here is the call graph for this function:



7.11.3 Member Function Documentation

7.11.3.1 operator<<() [1/2]

```
Payload& CANopen::Payload::operator<< (
    Payload && p ) [inline]
```

operator << Adds a variable inside the payload

Parameters

<i>p</i>	The variable to store
----------	-----------------------

Returns

The [Payload](#) reference.

Definition at line 74 of file [payload.h](#).

7.11.3.2 operator<<() [2/2]

```
template<typename T >
Payload& CANopen::Payload::operator<< (
    T && value ) [inline]
```

operator << Adds a variable inside the payload

Parameters

<i>value</i>	The variable to store
--------------	-----------------------

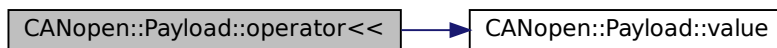
Returns

The [Payload](#) reference.

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

References [value\(\)](#).

Here is the call graph for this function:

**7.11.3.3 store_at()**

```

Payload& CANOpen::Payload::store_at (
    Payload && p,
    int slot ) [inline]
  
```

store_at Stores a variable at a specified index in the data array.

Parameters

<i>p</i>	The variable to store.
<i>slot</i>	The index in data array where the variable has to be store to.

Returns

Definition at line 87 of file [payload.h](#).

7.11.3.4 value()

```

template<typename T >
T& CANOpen::Payload::value (
    unsigned begin = 0 ) [inline]
  
```

value Returns the data casted as a variable of type T

Parameters

<i>begin</i>	The index of the data array from where the data must be returned.
--------------	---

Returns

The data as a variable of type T.

Definition at line 48 of file [payload.h](#).

Referenced by [CANopen::Parameter::from_payload\(\)](#), [operator<<\(\)](#), and [Payload\(\)](#).

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

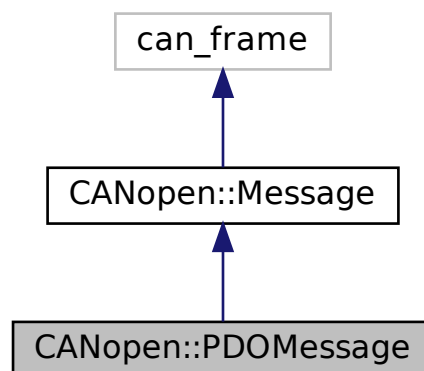
- [payload.h](#)
- [payload.cpp](#)

7.12 CANopen::PDOMessage Class Reference

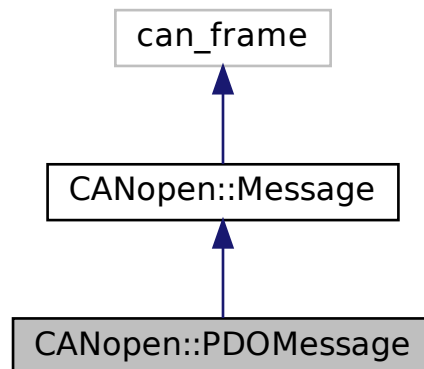
PDO [Message](#) (Process Data Object)

```
#include <pdo.h>
```

Inheritance diagram for CANopen::PDOMessage:



Collaboration diagram for CANopen::PDOMessage:



Public Types

- enum **PDOFunctionCode** : uint32_t {
 PDO1Transmit = Message::PDO1Transmit, **PDO1Receive** = Message::PDO1Receive, **PDO2Transmit** = Message::PDO2Transmit, **PDO2Receive** = Message::PDO2Receive,
 PDO3Transmit = Message::PDO3Transmit, **PDO3Receive** = Message::PDO3Receive, **PDO4Transmit** = Message::PDO4Transmit, **PDO4Receive** = Message::PDO4Receive }

Public Member Functions

- **PDOMessage** (const can_frame &other)
- **PDOMessage** (PDOFunctionCode fn, uint8_t [node_id](#), [Payload](#) [payload](#))
- uint8_t **num** ()
- uint32_t **id** ()

7.12.1 Detailed Description

PDO [Message](#) (Process Data Object)

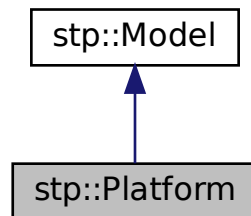
Definition at line [18](#) of file [pdo.h](#).

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

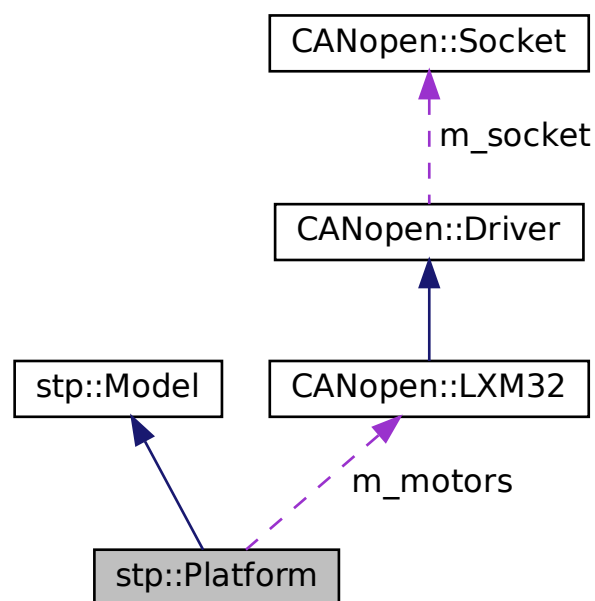
- pdo.h
- pdo.cpp

7.13 stp::Platform Class Reference

Inheritance diagram for stp::Platform:



Collaboration diagram for stp::Platform:



Public Member Functions

- **Platform** (double deltas[4], double a, double l, int verbose_level=0)
- void **init** ()
- void **start** ()
- void **pause** ()
- void **stop** ()
- double * **new_pos** (double T[3], double theta[3])
- void **update_platform** ()

Protected Attributes

- [CANopen::LXM32](#) * [m_motors](#) [NB_LEGS]

Additional Inherited Members

7.13.1 Detailed Description

Definition at line 13 of file [platform.hpp](#).

7.13.2 Member Data Documentation

7.13.2.1 m_motors

```
CANopen::LXM32* stp::Platform::m_motors[NB_LEGS] [protected]
```

Driver of the platform.

Definition at line 34 of file [platform.hpp](#).

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

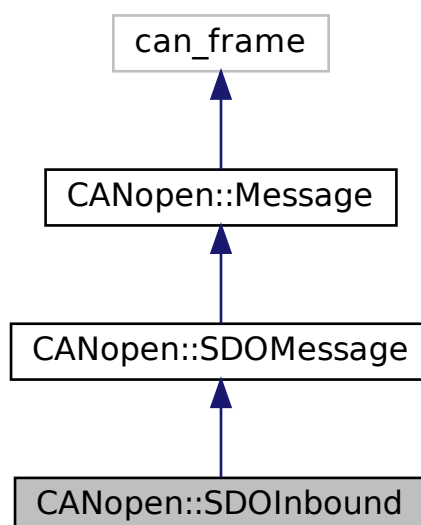
- [platform.hpp](#)
- [platform.cpp](#)

7.14 CANopen::SDOInbound Class Reference

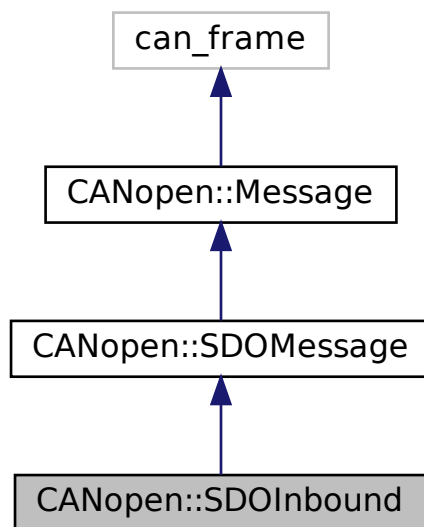
SDO received [Message](#).

```
#include <sdo.h>
```

Inheritance diagram for [CANopen::SDOInbound](#):



Collaboration diagram for CANopen::SDOInbound:



Public Member Functions

- **SDOInbound** (const can_frame &other)

Additional Inherited Members

7.14.1 Detailed Description

SDO received [Message](#).

Definition at line [118](#) of file [sdo.h](#).

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

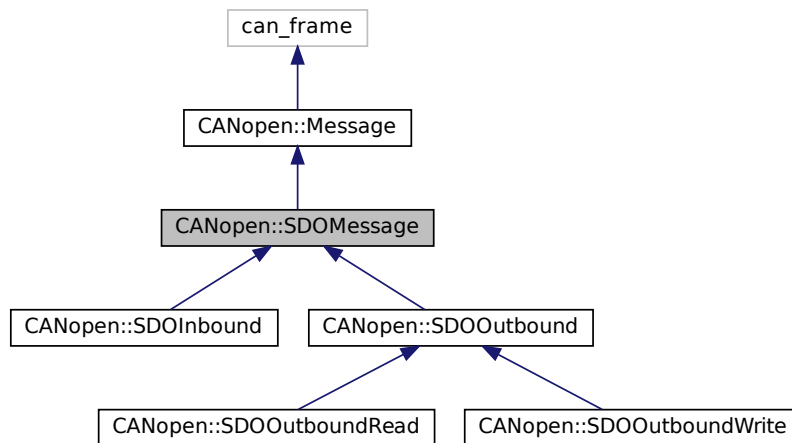
- [sdo.h](#)
- [sdo.cpp](#)

7.15 CANopen::SDOMessage Class Reference

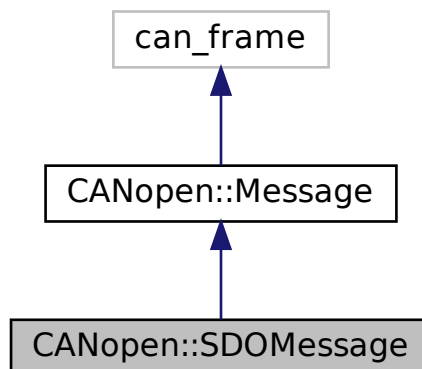
SDO [Message](#) (Service Data Object)

```
#include <sdo.h>
```

Inheritance diagram for CANopen::SDOMessage:



Collaboration diagram for CANopen::SDOMessage:



Public Types

- enum **RDWR** { **Read**, **Write** }
- enum **CCS** {
SegmentDownload = 0, **InitiateDownload** = 1, **InitiateUpload** = 2, **SegmentUpload** = 3,
AbortTransfer = 4, **BlockUpload** = 5, **BlockDownload** = 6 }

Public Member Functions

- **SDOMessage** (const can_frame &other)
- **SDOMessage** (FunctionCode fn, uint8_t node_id, CCS spec, uint8_t n, uint8_t e, uint8_t s, uint16_t index, uint8_t subindex, Payload payload)
- uint16_t index () const
index
- bool is_confirmation ()
is_confirmation
- bool is_error ()
is_error
- uint8_t subindex () const
subindex
- uint32_t index__sub () const
index__sub
- uint32_t id () const
id
- uint8_t size_data () const
size_data
- Payload payload () const
payload

7.15.1 Detailed Description

SDO Message (Service Data Object)

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

7.15.2 Member Function Documentation

7.15.2.1 id()

```
uint32_t CANopen::SDOMessage::id ( ) const [inline], [virtual]
```

id

Returns

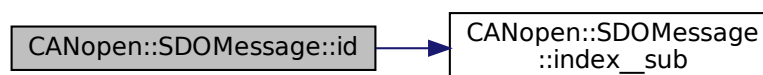
Returns the index__sub of the register.

Reimplemented from [CANopen::Message](#).

Definition at line 96 of file [sdo.h](#).

References [index__sub\(\)](#).

Here is the call graph for this function:



7.15.2.2 index()

```
uint16_t CANopen::SDOMessage::index ( ) const
```

index

Returns

Returns the index of the register.

Definition at line 29 of file [sdo.cpp](#).

7.15.2.3 index__sub()

```
uint32_t CANopen::SDOMessage::index__sub ( ) const
```

index__sub

Returns

Returns the index__sub of the register.

Definition at line 39 of file [sdo.cpp](#).

Referenced by [id\(\)](#).

7.15.2.4 is_confirmation()

```
bool CANopen::SDOMessage::is_confirmation ( ) [inline]
```

is_confirmation

Returns

Returns true if the message is a confirmation from a previous SDO write message.

Definition at line 57 of file [sdo.h](#).

7.15.2.5 is_error()

```
bool CANopen::SDOMessage::is_error ( ) [inline]
```

is_error

Returns

Returns true if the message is an error sent through SDO message.

Definition at line 69 of file [sdo.h](#).

7.15.2.6 payload()

```
Payload CANopen::SDOMessage::payload ( ) const [virtual]
```

payload

Returns

Returns the message payload

Reimplemented from [CANopen::Message](#).

Definition at line 44 of file [sdo.cpp](#).

7.15.2.7 size_data()

```
uint8_t CANopen::SDOMessage::size_data ( ) const
```

size_data

Returns

Returns the size of the data stored in the message payload

Definition at line 49 of file [sdo.cpp](#).

7.15.2.8 subindex()

```
uint8_t CANopen::SDOMessage::subindex ( ) const
```

subindex

Returns

Returns the subindex of the register.

Definition at line 34 of file [sdo.cpp](#).

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

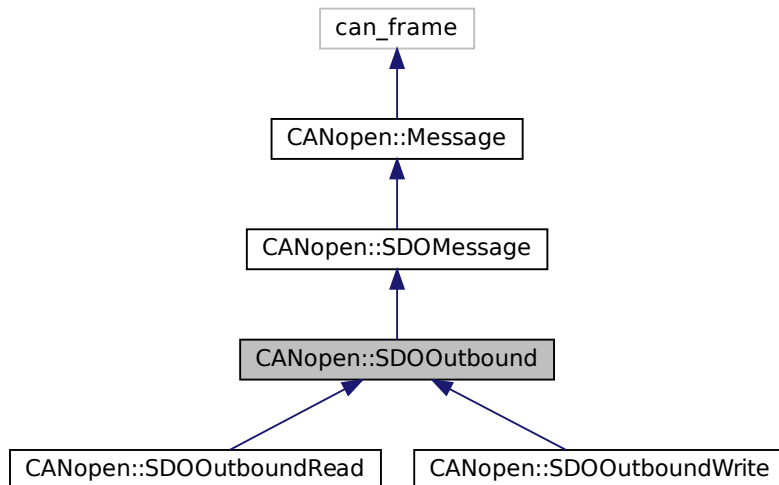
- [sdo.h](#)
- [sdo.cpp](#)

7.16 CANopen::SDOOutbound Class Reference

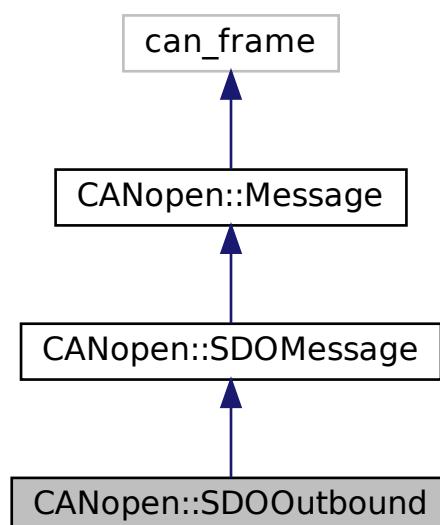
SDO [Message](#) to be sent.

```
#include <sdo.h>
```

Inheritance diagram for CANopen::SDOOutbound:



Collaboration diagram for CANopen::SDOOutbound:



Public Member Functions

- **SDOOutbound** (uint8_t [node_id](#), RDWR dir, uint16_t [index](#), uint8_t [subindex](#), [Payload payload](#))

Additional Inherited Members

7.16.1 Detailed Description

SDO [Message](#) to be sent.

Definition at line 126 of file [sdo.h](#).

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

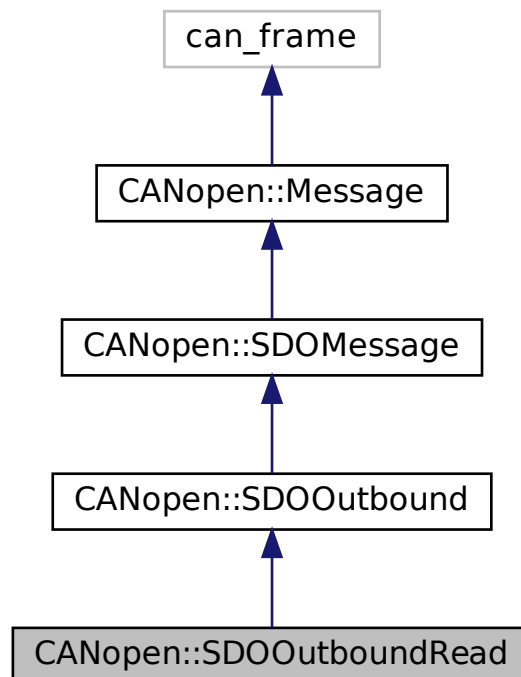
- [sdo.h](#)
- [sdo.cpp](#)

7.17 CANopen::SDOOutboundRead Class Reference

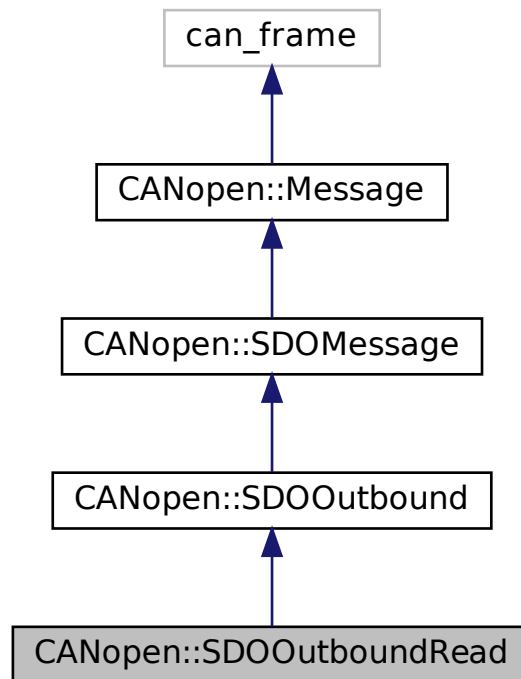
SDO [Message](#) to be sent to read the value from the object dictionary of a remote device.

```
#include <sdo.h>
```

Inheritance diagram for CANopen::SDOOutboundRead:



Collaboration diagram for CANopen::SDOOutboundRead:



Public Member Functions

- `SDOOutboundRead` (uint8_t [node_id](#), uint16_t [index](#), uint8_t [subindex](#))
- `SDOOutboundRead` (uint8_t [node_id](#), uint32_t [index__sub](#))

Additional Inherited Members

7.17.1 Detailed Description

SDO [Message](#) to be sent to read the value from the object dictionary of a remote device.

Definition at line [134](#) of file [sdo.h](#).

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

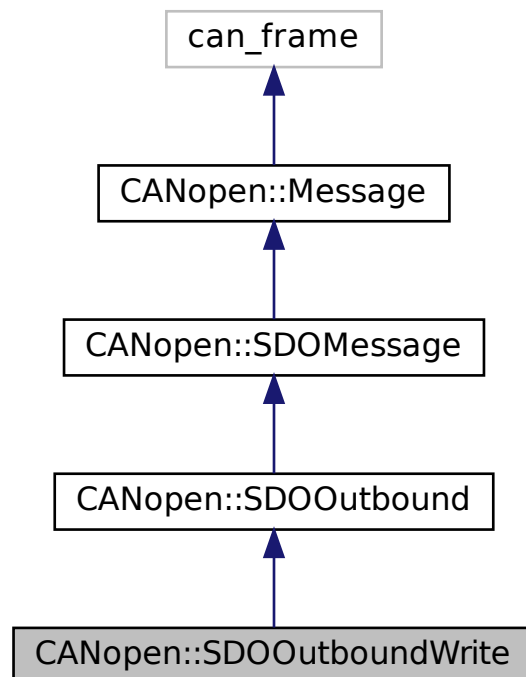
- [sdo.h](#)
- [sdo.cpp](#)

7.18 CANopen::SDOOutboundWrite Class Reference

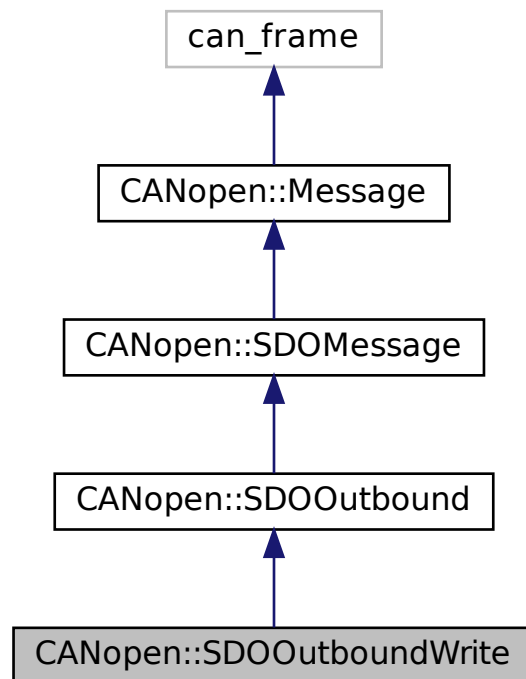
SDO [Message](#) to be sent to write the value of the object dictionary of a remote device.

```
#include <sdo.h>
```

Inheritance diagram for CANopen::SDOOutboundWrite:



Collaboration diagram for CANopen::SDOOutboundWrite:



Public Member Functions

- `SDOOutboundWrite` (uint8_t [node_id](#), uint16_t [index](#), uint8_t [subindex](#), [Payload](#) [payload](#))
- `SDOOutboundWrite` (uint8_t [node_id](#), uint32_t [index__sub](#), [Payload](#) [payload](#))

Additional Inherited Members

7.18.1 Detailed Description

SDO [Message](#) to be sent to write the value of the object dictionary of a remote device.

Definition at line [143](#) of file [sdo.h](#).

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

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

7.19 CANopen::Socket Class Reference

CANopen object able to send [Message](#) through a CAN interface using UNIX sockets.

```
#include <CANopen_socket.h>
```

Public Member Functions

- [Socket](#) (std::string ifname, int verbose_level=0)
Constructor.
- [Socket](#) (std::string ifname, uint32_t cob_id, int verbose_level=0)
Constructor.
- void **add_filter** (std::initializer_list< struct can_filter > rfilter)
- int [bind](#) ()
return true if the can interface is successfully bound
- void [send](#) (const [Message](#) &&msg)
Function to send a CAN message.
- std::shared_ptr< [Message](#) > **receive** ()

7.19.1 Detailed Description

CANopen object able to send [Message](#) through a CAN interface using UNIX sockets.

Definition at line 39 of file [CANopen_socket.h](#).

7.19.2 Constructor & Destructor Documentation

7.19.2.1 Socket() [1/2]

```
CANopen::Socket::Socket (
    std::string ifname,
    int verbose_level = 0 )
```

Constructor.

Parameters

<i>ifname</i>	: Name of the can interface ex:"can0"
<i>verbose</i>	: Display the different message sent.

Definition at line 19 of file [CANopen_socket.cpp](#).

References [bind\(\)](#).

Here is the call graph for this function:



7.19.2.2 Socket() [2/2]

```
CANopen::Socket::Socket (
    std::string ifname,
    uint32_t cob_id,
    int verbose_level = 0 )
```

Constructor.

Parameters

<i>ifname</i>	: Name of the can interface ex:"can0"
<i>cob_id</i>	: Filtering only frame with this ID.
<i>verbose</i>	: Display the different message sent.

Definition at line 30 of file [CANopen_socket.cpp](#).

7.19.3 Member Function Documentation

7.19.3.1 send()

```
void CANopen::Socket::send (
    const Message && msg )
```

Function to send a CAN message.

Parameters

<i>msg</i>	: CAN frame to send
------------	---------------------

Definition at line 73 of file [CANopen_socket.cpp](#).

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

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

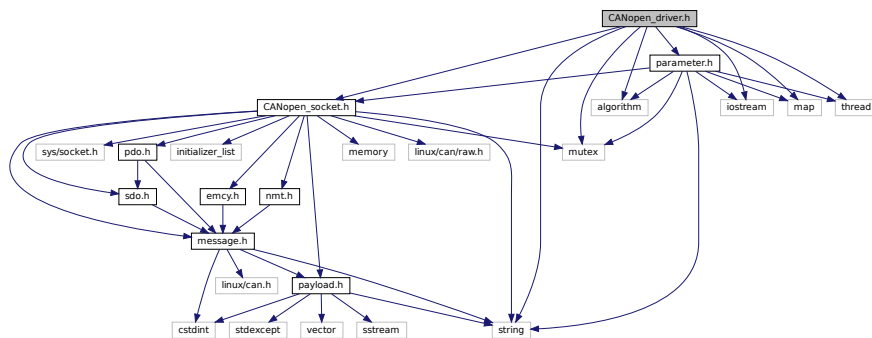
Chapter 8

File Documentation

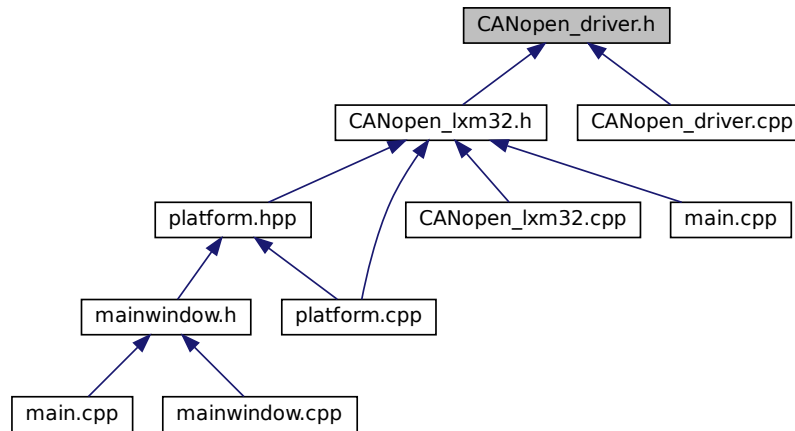
8.1 CANopen_driver.h File Reference

Device Profile Drives and Motion Control.

```
#include "CANopen_socket.h"  
#include "parameter.h"  
#include <algorithm>  
#include <iostream>  
#include <map>  
#include <mutex>  
#include <string>  
#include <thread>  
Include dependency graph for CANopen_driver.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [CANopen::Driver](#)
Device Profile Drives and Motion Control.

Functions

- void **CANopen::print_status** (Parameter *)

8.1.1 Detailed Description

Device Profile Drives and Motion Control.

Author

Alexis Devillard

Version

1.0

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

8.2 CANopen_driver.h

```

00001 #ifndef _CANOPEN_DRIVER_H_
00002 #define _CANOPEN_DRIVER_H_
00003
00011 #include "CANopen_socket.h"
00012 #include "parameter.h"
00013
00014 #include <algorithm>
00015 #include <iostream>
00016 #include <map>
00017 #include <mutex>
00018 #include <string>
00019 #include <thread>
00020
00021 namespace CANopen {
00022 void
00023 print_status(Parameter *);
00027 class Driver {
00028
00029     static constexpr int NB_PDO = 4;
00030     static constexpr int MAX_PDO_SLOT = 2;
00031
00032     public:
00034     enum Register : uint32_t {
00035         _DCOMstatus = 0x60410000,
00039         DCOMcontrol = 0x60400000,
00043         DCOMopmode = 0x60600000,
00044         _DCOMopmd_act = 0x60610000,
00046         PPp_target = 0x607A0000,
00051         PPv_target = 0x60810000,
00054         PVv_target = 0x60FF0000,
00055         PTtq_target = 0x60710000,
00057         RAMP_v_acc = 0x60830000,
00059         RAMP_v_dec = 0x60840000,
00061         _p_act = 0x60640000,
00062         _v_act = 0x606C0000,
00064         _tq_act = 0x60770000,
00066         HMmethod = 0x60980000,
00067         HMv = 0x60990001,
00068         HMv_out = 0x60990002
00069     };
00070
00072     enum OperationMode : int8_t {
00073         ProfilePosition = 1,
00075         Velocity = 2,
00077         ProfileVelocity = 3,
00079         ProfileTorque = 4,
00083         Homing = 6,
00084         InterpolatedPosition = 7,
00088     };
00089
00091     enum State : uint16_t {
00092         mask = 0x006f,
00093         NotReadyToSwitchtON = 0x0000,
00099         SwitchONDisabled = 0x0040,
00106         ReadyToSwitchON = 0x0021,
00111         SwitchedON = 0x0023,
00117         OperationEnabled = 0x0037,
00122         Fault = 0x000f,
00127         FaultReactionActive = 0x000f,
00133         QuickStopActive = 0x0007
00139     };
00140     enum StatusBits : uint16_t {
00141         ReadyToSwitchOn_bit = 0x0001,
00142         SwitchedOn_bit = 0x0002,
00143         OperationEnabled_bit = 0x0004,
00144         Fault_bit = 0x0008,
00145         VoltageEnabled_bit = 0x0010,
00146         QuickStop_bit = 0x0020,
00147         SwitchONDisabled_bit = 0x0040,
00148         Error0_bit = 0x0080,
00149         HaltRequest_bit = 0x0100,
00150         Remote_bit = 0x0200,
00151         TargetReached_bit = 0x0400,
00152         InternalLimitReached_bit = 0x0800,
00153         OperationMode_bit = 0x1000,
00154         BlockingError_bit = 0x2000,
00155         OperationModeStart_bit = 0x4000,
00156         ValidRef_bit = 0x8000
00157     };
00158
00160     enum Control : uint16_t {
00161         Shutdown = 0x0006,
00162         SwitchON = 0x0007,
00163         DisableVoltage = 0x0000,
00164         QuickStop = 0x0002,

```

```

00165         DisableOperation = 0x0007,
00166         EnableOperation = 0x000f,
00167         FaultResest = 0x0080
00168     };
00169
00170     enum PDOFunctionCode : uint32_t {
00171         PDO1Transmit = Message::PDO1Transmit,
00172         PDO1Receive = Message::PDO1Receive,
00173         PDO2Transmit = Message::PDO2Transmit,
00174         PDO2Receive = Message::PDO2Receive,
00175         PDO3Transmit = Message::PDO3Transmit,
00176         PDO3Receive = Message::PDO3Receive,
00177         PDO4Transmit = Message::PDO4Transmit,
00178         PDO4Receive = Message::PDO4Receive,
00179     };
00180
00181     Driver(const char *ifname, uint16_t can_id, int verbose_lvl = 0);
00182
00183     template <typename T>
00184     void
00185     set(Register reg, T val, bool force_sdo = false, bool wait = false) {
00186         if(m_available) {
00187             m_parameters[reg]->set(val);
00188             if(m_parameters[reg]->pdo_slot == -1 || force_sdo || wait)
00189                 send(m_parameters[reg]);
00190             if(wait)
00191                 while(m_parameters[reg]->sdo_flag.test_and_set())
00192                     ;
00193         }
00194     }
00195
00196     template <typename T>
00197     T
00198     get(Register reg, bool force_sdo = false) {
00199         if(force_sdo && m_available) {
00200             update(m_parameters[reg]);
00201             while(m_parameters[reg]->sdo_flag.test_and_set())
00202                 ;
00203         }
00204         return m_parameters[reg]->get<T>();
00205     };
00206
00207     void
00208     set_control(Control ctrl);
00209
00210     State
00211     get_state() { return m_parameters[_DCOMstatus]->get<State>(); };
00212
00213     void
00214     wait_state(State state, uint16_t _mask = mask) {
00215         while((get_state() & mask) != (state & mask))
00216             ;
00217     }; //std::cout << (get_state()&mask) << " " << (state&mask)<< "\n";}
00218
00219     void
00220     set_mode(OperationMode mode, bool wait = false);
00221
00222     OperationMode
00223     get_mode(bool force_sdo = true) { return this->get<OperationMode>(_DCOMopmd_act, force_sdo); };
00224
00225     bool
00226     set_position(int32_t target, bool absolute = true);
00227     bool
00228     set_velocity(int32_t target);
00229     bool
00230     set_torque(int16_t target);
00231
00232     int32_t
00233     get_position() { return m_parameters[_p_act]->get<int32_t>() - m_offset_pos; };
00234     int32_t
00235     get_velocity() { return m_parameters[_v_act]->get<int32_t>(); };
00236     int32_t
00237     get_torque() { return m_parameters[_tq_act]->get<int32_t>(); };
00238
00239     void
00240     set_position_offset(int32_t offset_pos) { m_offset_pos = offset_pos; };
00241
00242     void
00243     start();
00244     void
00245     pause();
00246     void
00247     stop();
00248
00249     void

```

```

00338     profilePosition_mode();
00342     void
00343     profileVelocity_mode();
00347     void
00348     profileTorque_mode();
00352     void
00353     homing();
00354
00355     Parameter *
00356     get_param(Register reg) { return m_parameters[reg]; };
00357
00358     virtual void
00359     print_manufacturer_status() = 0;
00360
00361     std::string
00362     ctrl_to_str(Control control);
00363
00367     bool
00368     is_available() { return m_available; };
00369
00370     protected:
00375     void
00376     send(Parameter *param);
00377
00382     void
00383     update(Parameter *param);
00384
00391     void
00392     map_PDO(PDOFunctionCode fn, Parameter *param, int slot);
00393
00399     void
00400     activate_PDO(PDOFunctionCode fn, bool set = true);
00401
00402     void
00403     T_socket();
00404
00405     void
00406     RPDO_socket();
00407
00408     std::thread *m_rpdo_socket_thread;
00409     std::atomic_flag rpdo_socket_flag;
00410     std::mutex rpdo_mutex;
00411     std::thread *m_t_socket_thread;
00412     std::atomic_flag t_socket_flag;
00413
00414     const char *m_ifname;
00415     int m_verbose_level;
00416     bool m_available;
00417
00418     CANopen::Socket m_socket;
00419
00420     std::map<PDOFunctionCode, std::vector<Parameter *>> m_PDO_map;
00421     std::map<Register, Parameter *> m_parameters;
00422
00423     uint8_t m_node_id;
00424     uint16_t m_can_baud;
00425     int32_t m_offset_pos = 0;
00426 };
00427
00428 } // namespace CANopen
00429 #endif

```

8.3 CANopen_Ixm32.h File Reference

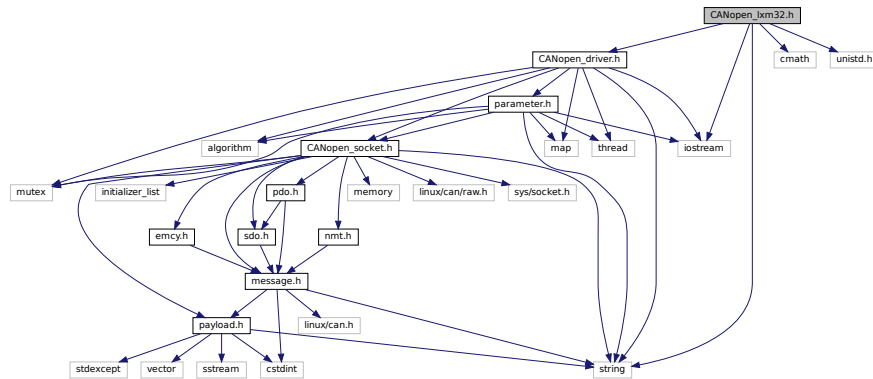
Implementation of the Driver Class for a LXM32 driver.

```

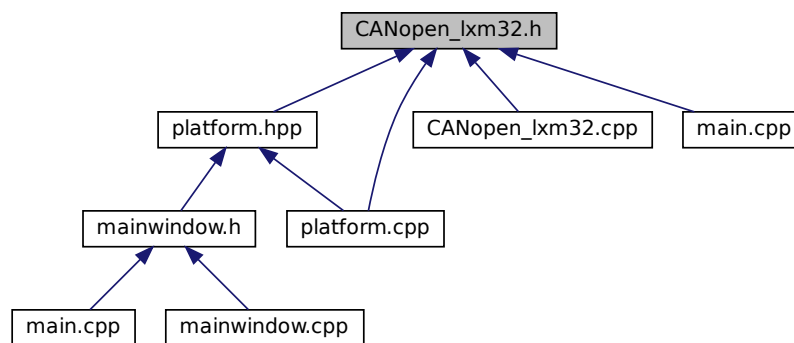
#include "CANopen_driver.h"
#include <cmath>
#include <iostream>
#include <string>
#include <unistd.h>

```

Include dependency graph for CANopen_lxm32.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [CANopen::LXM32](#)
Implementation of the [Driver](#) Class for a [LXM32](#) driver.

8.3.1 Detailed Description

Implementation of the Driver Class for a LXM32 driver.

Author

Alexis Devillard

Version

1.0

Definition in file [CANopen_lxm32.h](#).

8.4 CANopen_lxm32.h

```

00001 #ifndef _LEXIUM32A_CANOPEN_H_
00002 #define _LEXIUM32A_CANOPEN_H_
00003
00011 #include "CANopen_driver.h"
00012
00013 #include <cmath>
00014 #include <iostream>
00015 #include <string>
00016 #include <unistd.h>
00017
00018 namespace CANopen {
00022 class LXM32 : public Driver {
00023     public:
00030     LXM32(const char *ifname, uint16_t can_id, bool verbose = false);
00031
00039     bool
00040     set_angle(double ang, bool absolute = true, bool radian = true);
00041
00047     double
00048     get_angle(bool radian = true);
00049
00050     void
00051     print_manufacturer_status(){};
00052
00053     int nb_index_per_turn = 737280;
00054 };
00055 // namespace CANopen
00056 }
00057 #endif

```

8.5 CANopen_socket.h File Reference

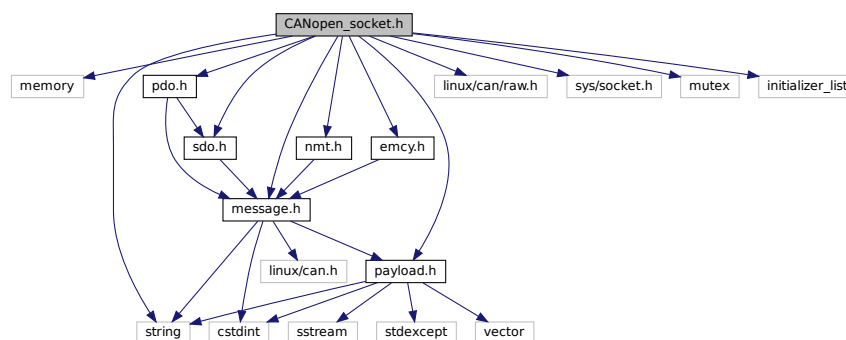
Canopen socket able to send/receive messages through a CAN interface using the UNIX socket.

```

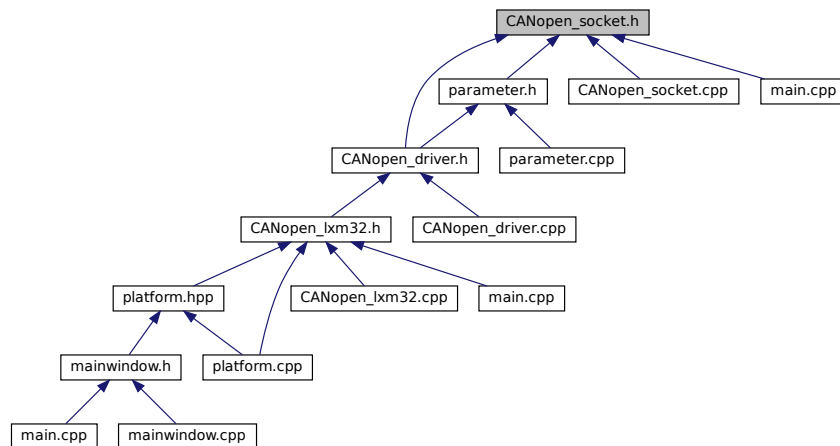
#include <memory>
#include <string>
#include <linux/can/raw.h>
#include <sys/socket.h>
#include <mutex>
#include <initializer_list>
#include "message.h"
#include "payload.h"
#include "pdo.h"
#include "nmt.h"
#include "emcy.h"

```

Include dependency graph for CANopen_socket.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [CANopen::Socket](#)
CANopen object able to send [Message](#) through a CAN interface using UNIX sockets.

Macros

- `#define IF_VERBOSE(lvl, cmd, m_lvl)`

Variables

- `std::mutex CANopen::g_verbose_mutex`

8.5.1 Detailed Description

Canopen socket able to send/receive messages through a CAN interface using the UNIX socket.

Author

Florian Richer & Alexis Devillard

Version

1.0

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

8.5.2 Macro Definition Documentation

8.5.2.1 IF_VERBOSE

```
#define IF_VERBOSE(
    lvl,
    cmd,
    m_lvl )
```

Value:

```
if (m_lvl >= lvl) { \
    CANopen::g_verbose_mutex.lock(); \
    cmd; \
    CANopen::g_verbose_mutex.unlock(); \
}
```

Definition at line 26 of file [CANopen_socket.h](#).

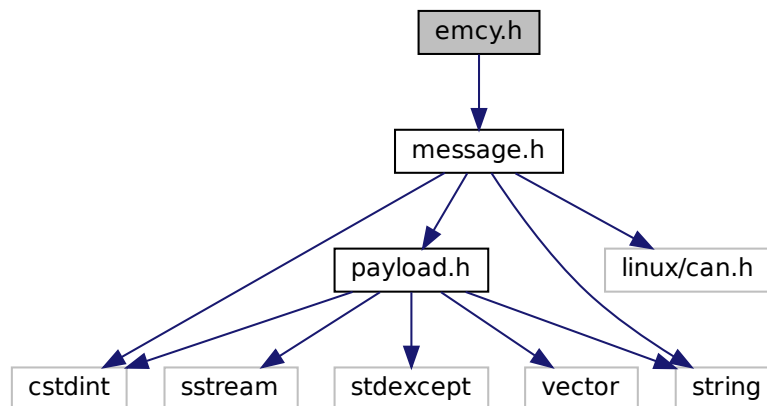
8.6 CANopen_socket.h

```
00001 #ifndef _CANOPEN_SOCKET_H_
00002 #define _CANOPEN_SOCKET_H_
00003
00011 #include <memory>
00012 #include <string>
00013 #include <linux/can/raw.h>
00014 #include <sys/socket.h>
00015 #include <mutex>
00016 #include <initializer_list>
00017
00018 #include "message.h"
00019 #include "payload.h"
00020 #include "sdo.h"
00021 #include "pdo.h"
00022 #include "nmt.h"
00023 #include "emcy.h"
00024
00025
00026 #define IF_VERBOSE(lvl, cmd, m_lvl) \
00027     if (m_lvl >= lvl) { \
00028         CANopen::g_verbose_mutex.lock(); \
00029         cmd; \
00030         CANopen::g_verbose_mutex.unlock(); \
00031     }
00032
00033 namespace CANopen {
00034 extern std::mutex g_verbose_mutex;
00039 class Socket {
00040 public:
00046     Socket(std::string ifname, int verbose_level = 0);
00047
00054     Socket(std::string ifname, uint32_t cob_id, int verbose_level = 0);
00055
00056
00057     void add_filter(std::initializer_list<struct can_filter> rfilter);
00058
00063     int bind();
00064
00069     void send(const Message&& msg);
00070
00071     std::shared_ptr<Message> receive();
00072
00073 private:
00074     int m_socket;
00075     std::string m_ifname;
00077     int m_verbose_level;
00078 };
00079 }
00080
00081 #endif // _CANOPEN_SOCKET_H_
```

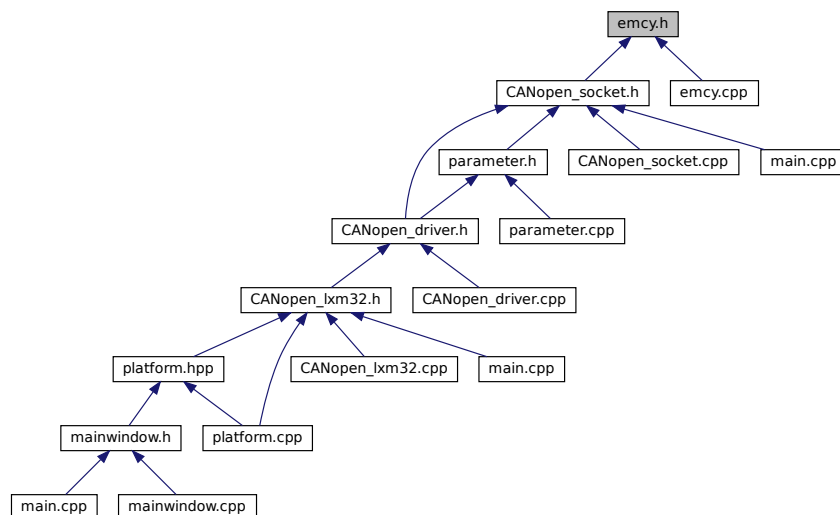
8.7 emcy.h File Reference

PDO message sent and received through CANopen socket.

```
#include "message.h"
Include dependency graph for emcy.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [CANopen::EMCYMessage](#)
EMCY Message (Emergency Object)

8.7.1 Detailed Description

PDO message sent and received through CANopen socket.

Author

Florian Richer & Alexis Devillard

Version

1.0

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

8.8 emcy.h

```

00001 #ifndef _CANOPEN_EMCY_MESSAGE_H_
00002 #define _CANOPEN_EMCY_MESSAGE_H_
00003
00011 #include "message.h"
00012
00013 namespace CANopen {
00017 class EMCYMessage : public Message {
00018     public:
00019
00020     EMCYMessage() = default;
00021     EMCYMessage(const can_frame &other);
00022
00023     uint16_t code() const;
00024     uint8_t reg() const;
00025 };
00026 } // namespace CANopen
00027
00028 #endif // _CANOPEN_NMT_MESSAGE_H_

```

8.9 message.h File Reference

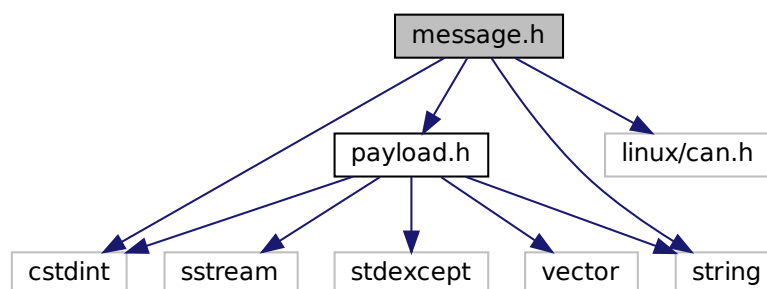
CAN_frame message sent and received through CANopen socket.

```

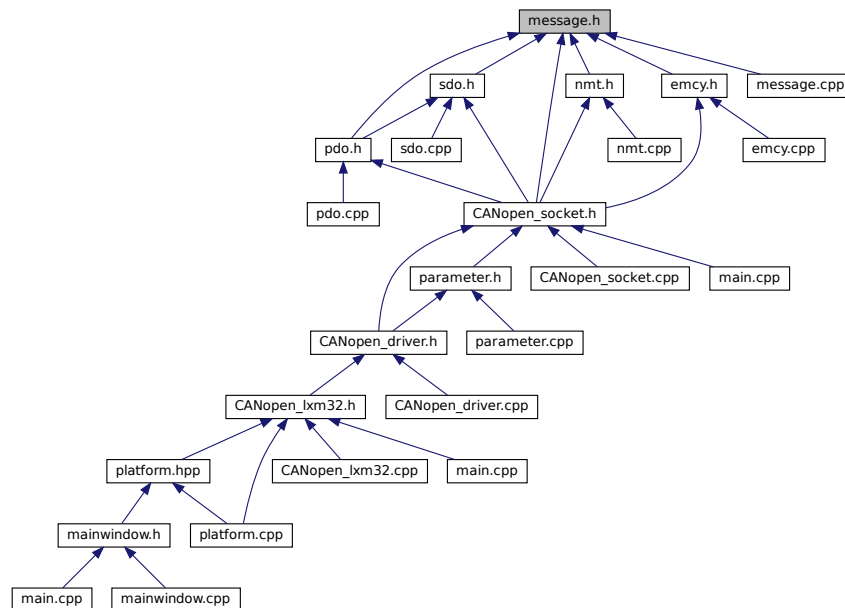
#include "payload.h"
#include <stdint>
#include <linux/can.h>
#include <string>

```

Include dependency graph for message.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [CANopen::Message](#)

can_frame object sent and received throught CANopen socket.

8.9.1 Detailed Description

CAN_frame message sent and received throught CANopen socket.

Author

Florian Richer & Alexis Devillard

Version

1.0

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

8.10 message.h

```

00001 #ifndef MESSAGE_H
00002 #define MESSAGE_H
00003
00011 #include "payload.h"
00012 #include <cstdint>
00013 #include <linux/can.h>
00014 #include <string>
00015
00016 namespace CANopen {
00020 class Message : public can_frame {
00021     public:
00025     enum FunctionCode : uint32_t {
00026         NMT = 0,
00027         Emergency = 0x80,
00028         Sync = 0x80,
00029         TimeStamp = 0x100,
00030         PDO1Transmit = 0x180,
00031         PDO1Receive = 0x200,
00032         PDO2Transmit = 0x280,
00033         PDO2Receive = 0x300,
00034         PDO3Transmit = 0x380,
00035         PDO3Receive = 0x400,
00036         PDO4Transmit = 0x480,
00037         PDO4Receive = 0x500,
00038         SDOTransmit = 0x580,
00039         SDOReceive = 0x600,
00040         Heartbeat = 0x700
00041     };
00042
00043     Message() = default;
00044     Message(const can_frame &other);
00045     Message(uint32_t cob_id, Payload payload);
00046
00047     operator can_frame *() const { return const_cast<can_frame *>(reinterpret_cast<const can_frame
*>(&can_id)); };
00048
00053     FunctionCode
00054     function_code() const;
00055
00060     uint8_t
00061     node_id() const;
00062
00067     virtual Payload
00068     payload() const;
00073     virtual uint32_t
00074     id() const { return 0; };
00075
00076     std::string
00077     to_string() const;
00078 };
00079 } // namespace CANopen
00080
00081 #endif // MESSAGE_H

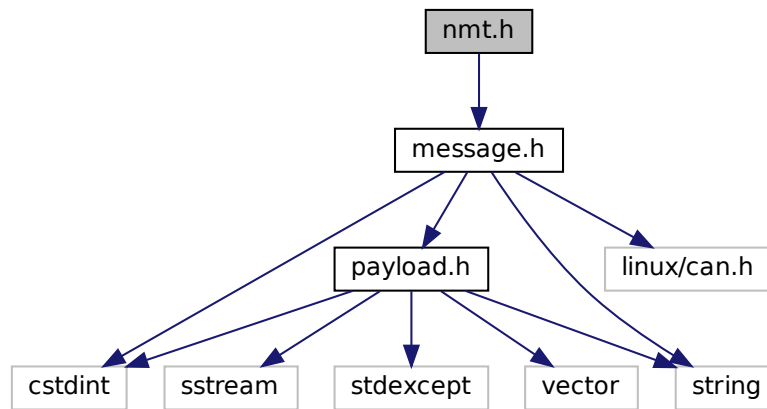
```

8.11 nmt.h File Reference

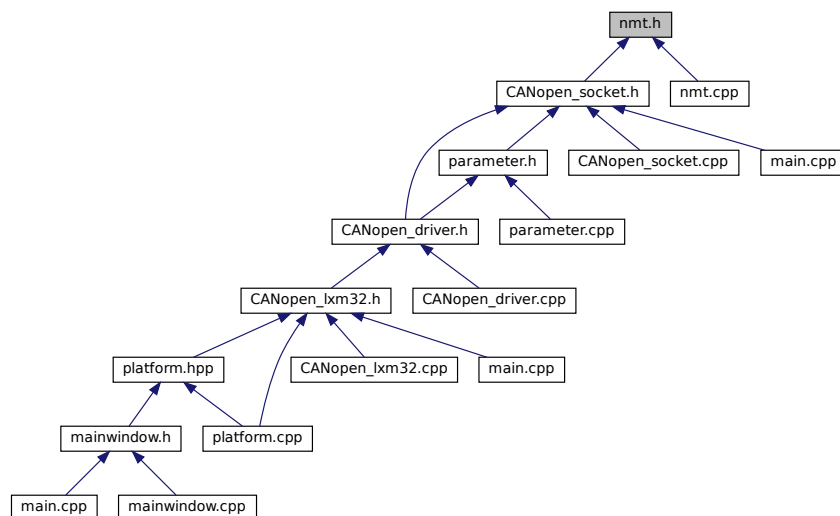
NMT message sent and received through CANopen socket.

```
#include "message.h"
```

Include dependency graph for nmt.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [CANopen::NMTMessage](#)
NMT Message (Network management)

8.11.1 Detailed Description

NMT message sent and received through CANopen socket.

Author

Florian Richer & Alexis Devillard

Version

1.0

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

8.12 nmt.h

```

00001 #ifndef _CANOPEN_NMT_MESSAGE_H_
00002 #define _CANOPEN_NMT_MESSAGE_H_
00003
00011 #include "message.h"
00012
00013 namespace CANopen {
00017 class NMTMessage : public Message {
00018     public:
00019         enum Code : uint8_t {
00020             Initialising = 0,
00021             GoToOperational = 0x01,
00022             GoToStopped = 0x02,
00023             Stopped = 0x04,
00024             Operational = 0x05,
00025             PreOperational = 0x7f,
00026             GoToPreOperational = 0x80,
00027             GoToResetNode = 0x81,
00028             GoToResetCommunication = 0x82
00029         };
00030
00031         NMTMessage() = default;
00032         NMTMessage(const can_frame &other);
00033         NMTMessage(Code code, uint8_t node_id);
00034     };
00035 } // namespace CANopen
00036
00037 #endif // _CANOPEN_NMT_MESSAGE_H_

```

8.13 parameter.h File Reference

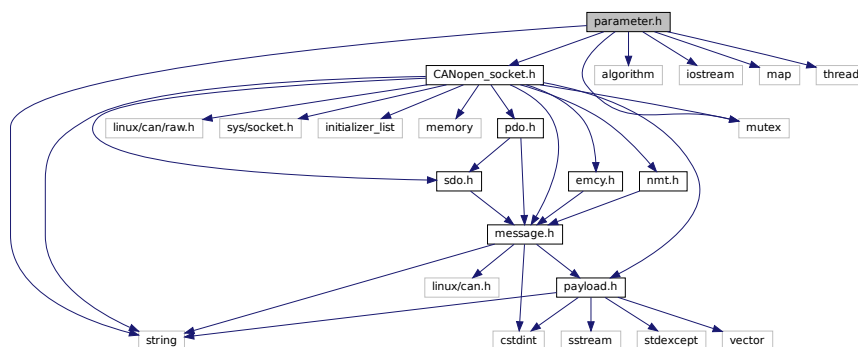
Device Object from the object dictionary of a remote CANopen device.

```

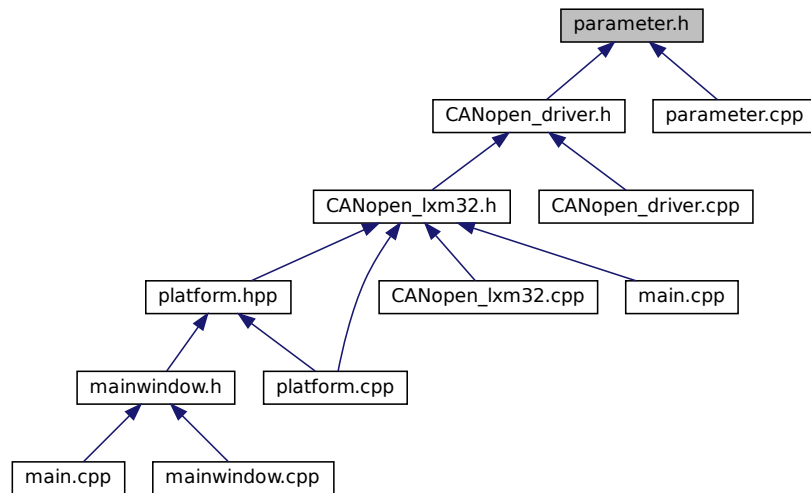
#include "CANopen_socket.h"
#include <algorithm>
#include <iostream>
#include <map>
#include <mutex>
#include <string>
#include <thread>

```

Include dependency graph for parameter.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [CANopen::Parameter](#)

Object from the object dictionary of a remote CANopen device.

8.13.1 Detailed Description

Device Object from the object dictionary of a remote CANopen device.

Author

Alexis Devillard

Version

1.0

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

8.14 parameter.h

```

00001 #ifndef _PARAMETER_H_
00002 #define _PARAMETER_H_
00003
00011 #include "CANopen_socket.h"
00012
00013 #include <algorithm>
00014 #include <iostream>
00015 #include <map>
00016 #include <mutex>
00017 #include <string>
00018 #include <thread>

```



```

00019
00020 namespace CANopen {
00024 struct Parameter {
00025
00029     typedef void (*param_cb_t)(Parameter *); //type of the parameter callback function
00030
00034     enum PDOFunctionCode : uint32_t {
00035         PDO1Transmit = Message::PDO1Transmit,
00036         PDO1Receive = Message::PDO1Receive,
00037         PDO2Transmit = Message::PDO2Transmit,
00038         PDO2Receive = Message::PDO2Receive,
00039         PDO3Transmit = Message::PDO3Transmit,
00040         PDO3Receive = Message::PDO3Receive,
00041         PDO4Transmit = Message::PDO4Transmit,
00042         PDO4Receive = Message::PDO4Receive,
00043     };
00044
00045     Parameter() { var = new int32_t; };
00046
00047     template <typename T>
00056     Parameter(std::string name_, T val, uint16_t index_, uint8_t subindex_, param_cb_t cb = nullptr) :
name(name_), index(index_), subindex(subindex_), _cb(cb) {
00057         mutex.lock();
00058         var = new T;
00059         *(T *)var = (T)val;
00060         m_should_be_sent = false;
00061         mutex.unlock();
00062         size = sizeof(T);
00063     };
00064
00065     template <typename T>
00073     Parameter(std::string name_, T val, uint32_t index__sub, param_cb_t cb = nullptr) :
Parameter(name_, val, (uint16_t)(index__sub >> 16), (uint8_t)index__sub, cb){};
00074
00075     ~Parameter() {
00076         delete(int32_t *)var;
00077     }
00078
00084     void
00085     link_to_pdo(PDOFunctionCode fn, int8_t slot);
00086
00087     template <typename T>
00095     bool
00096     set(T val, bool force_update = false, bool received_data = false) {
00097         bool was_updated = false;
00098         if(sizeof(T) == size) {
00099             mutex.lock();
00100             if(!m_should_be_sent || force_update) //if force update (even if the previous value was
not sent yet) or value already update
00101             {
00102                 if(*(T *)var != (T)val) {
00103                     was_updated = true;
00104                     *(T *)var = (T)val;
00105                 }
00106                 if(!received_data)
00107                     m_should_be_sent = true;
00108             }
00109             mutex.unlock();
00110             return was_updated;
00111         }
00112         return false;
00113     }
00114
00120     bool
00121     operator=(int8_t val) { return this->set<int8_t>(val, false); } //return false if the assignment
didn't succeed (wrong type size, not updated yet)
00127     bool
00128     operator=(int16_t val) { return this->set<int16_t>(val, false); } //return false if the
assignment didn't succeed (wrong type size, not updated yet)
00134     bool
00135     operator=(int32_t val) { return this->set<int32_t>(val, false); } //return false if the
assignment didn't succeed (wrong type size, not updated yet)
00141     bool
00142     operator=(uint8_t val) { return this->set<uint8_t>(val, false); } //return false if the
assignment didn't succeed (wrong type size, not updated yet)
00148     bool
00149     operator=(uint16_t val) { return this->set<uint16_t>(val, false); } //return false if the
assignment didn't succeed (wrong type size, not updated yet)
00155     bool
00156     operator=(uint32_t val) { return this->set<uint32_t>(val, false); } //return false if the
assignment didn't succeed (wrong type size, not updated yet)
00157
00158     template <typename T>
00163     T
00164     get() {
00165         T val={};
00166         if(sizeof(T) == size) {

```

```

00167         mutex.lock();
00168         val = *(T *)var;
00169         mutex.unlock();
00170     }
00171     return val;
00172 }
00173
00177 operator int8_t() { return this->get<int8_t>(); };
00181 operator int16_t() { return this->get<int16_t>(); };
00185 operator int32_t() { return this->get<int32_t>(); };
00189 operator uint8_t() { return this->get<uint8_t>(); };
00193 operator uint16_t() { return this->get<uint16_t>(); };
00197 operator uint32_t() { return this->get<uint32_t>(); };
00198
00206 bool
00207 from_payload(Payload &p, int slot = 0, bool received_data = true);
00208
00213 bool
00214 has_been_sent() {
00215     const std::lock_guard<std::mutex> lock(mutex);
00216     return !m_should_be_sent;
00217 }
00218
00222 void
00223 callback();
00224
00230 Payload
00231 payload(bool *should_be_sent = nullptr);
00232
00233 size_t size = 0;
00234 std::string name;
00235 uint16_t index = 0;
00236 uint8_t subindex = 0;
00237
00238 PDOFunctionCode pdo_fn;
00239 int8_t pdo_slot = -1;
00240
00241 std::atomic_flag sdo_flag;
00242
00243 private:
00244     void *var = nullptr;
00245     param_cb_t _cb = nullptr;
00246     bool m_should_be_sent;
00247     std::mutex mutex;
00248 };
00249 } // namespace CANopen
00250 #endif

```

8.15 payload.h File Reference

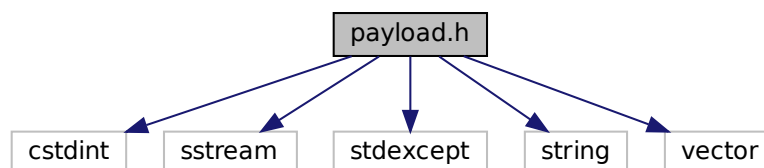
Payload of CANopen message: array of 1 to 8 bytes of data.

```

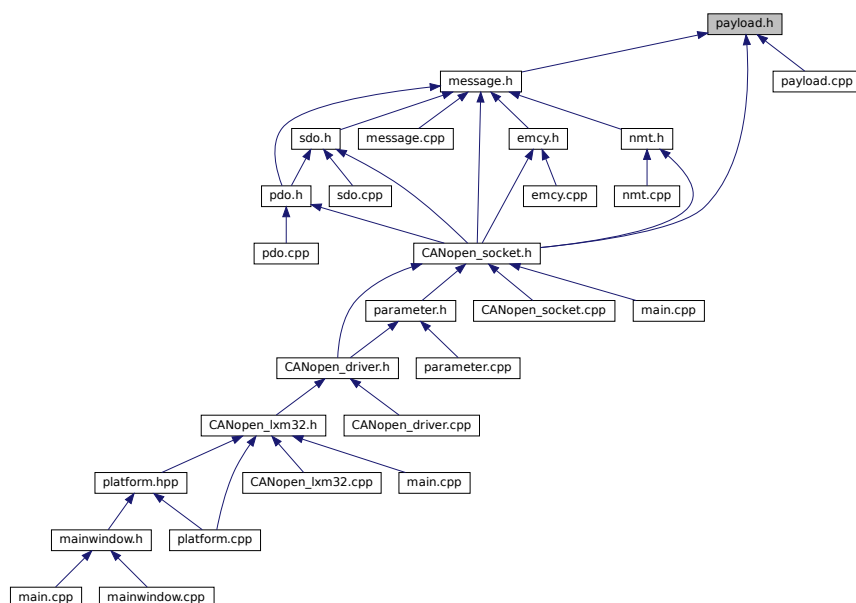
#include <cstdint>
#include <sstream>
#include <stdexcept>
#include <string>
#include <vector>

```

Include dependency graph for payload.h:



This graph shows which files directly or indirectly include this file:



Classes

- class `CANopen::Payload`
Payload of *CANopen* message: array of 1 to 8 bytes of data.

Functions

- `std::ostream & operator<< (std::ostream &out, const CANopen::Payload &p)`

8.15.1 Detailed Description

Payload of CANopen message: array of 1 to 8 bytes of data.

Author

Florian Richer & Alexis Devillard

Version

1.0

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

8.16 payload.h

```

00001 #ifndef _CANOPEN_PAYLOAD_H_
00002 #define _CANOPEN_PAYLOAD_H_
00003
00011 #include <stdint>
00012 #include <sstream>
00013 #include <stdexcept>
00014 #include <string>
00015 #include <vector>
00016
00017
00018 namespace CANopen {
00022 class Payload : public std::vector<uint8_t> {
00023     public:
00024         Payload() = default;
00025         Payload(const Payload &) = default;
00026         Payload(const std::vector<uint8_t> &other);
00027
00028         template <typename T>
00033         Payload(T value) {
00034             for(int i = 0; i < sizeof(T); i++)
00035                 push_back(*(uint8_t *)(&value) + i);
00036         }
00037
00038         Payload &
00039         operator=(const Payload &) = default;
00040
00041         template <typename T>
00047         T &
00048         value(unsigned begin = 0) {
00049             if(empty())
00050                 throw std::runtime_error(std::string("Empty payload."));
00051             return *(T *) (data() + begin);
00052         };
00053
00054
00055         template <typename T>
00061         Payload &
00062         operator<<(T &&value) {
00063             for(int i = 0; i < sizeof(T); i++)
00064                 push_back(*(uint8_t *)(&value) + i);
00065             return *this;
00066         };
00067
00073         Payload &
00074         operator<<(Payload &&p) {
00075             for(int i = 0; i < p.size(); i++)
00076                 push_back(*(uint8_t *)(&p[i]));
00077             return *this;
00078         };
00079
00086         Payload &
00087         store_at(Payload &&p, int slot) {
00088             for(int i = this->size(); i < slot ; i++)
00089                 this->push_back(0);
00090             for(int i = slot; i < slot + p.size(); i++)
00091             {
00092                 if(i < this->size())
00093                     (*this)[i]=*(uint8_t *)(&p[i]);
00094                 else
00095                     this->push_back(*(uint8_t *)(&p[i-slot]));
00096             }
00097             return *this;
00098         };
00099
00100         operator std::string() const;
00101     };
00102 } // namespace CANopen
00103
00104 std::ostream &
00105 operator<<(std::ostream &out, const CANopen::Payload &p);
00106
00107 #endif // _CANOPEN_PAYLOAD_H_

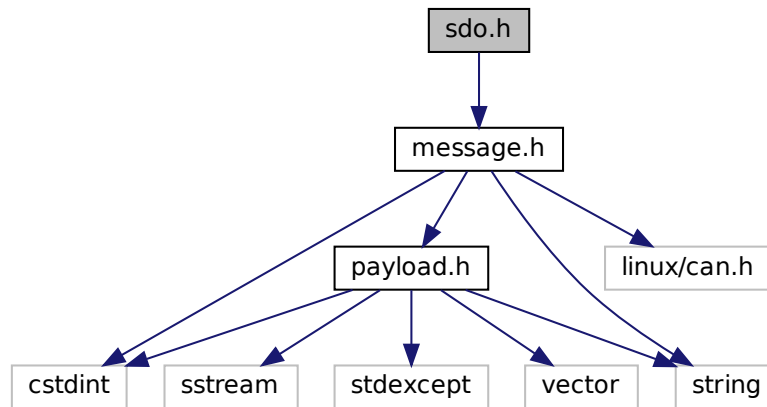
```

8.17 sdo.h File Reference

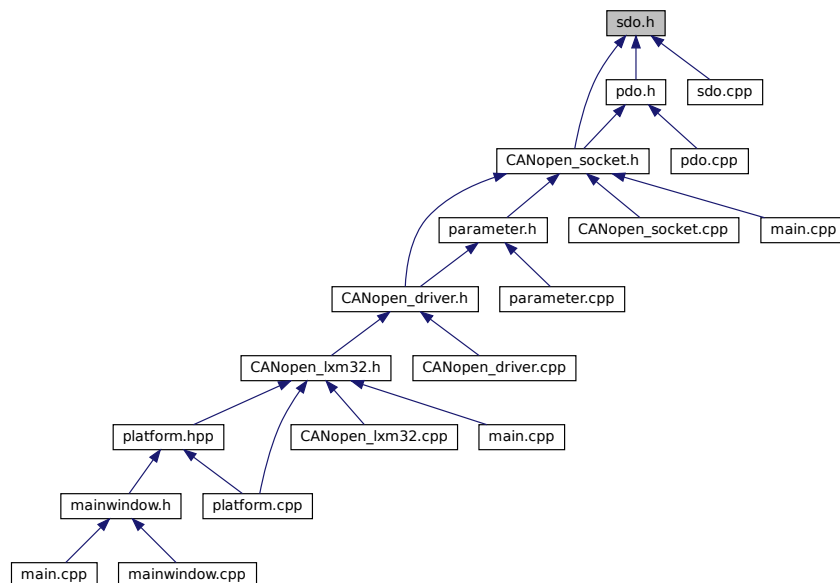
PDO message sent and received throught CANopen socket.

```
#include "message.h"
```

Include dependency graph for sdo.h:



This graph shows which files directly or indirectly include this file:



Classes

- class `CANopen::SDOMessage`
SDO Message (Service Data Object)
- class `CANopen::SDOInbound`
SDO received Message.
- class `CANopen::SDOOutbound`

SDO Message to be sent.

- class [CANopen::SDOOutboundRead](#)

SDO Message to be sent to read the value from the object dictionary of a remote device.

- class [CANopen::SDOOutboundWrite](#)

SDO Message to be sent to write the value of the object dictionary of a remote device.

8.17.1 Detailed Description

PDO message sent and received through CANopen socket.

SDO message sent and received through CANopen socket.

Author

Florian Richer & Alexis Devillard

Version

1.0

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

8.18 sdo.h

```

00001 #ifndef _CANOPEN_SDO_MESSAGE_H_
00002 #define _CANOPEN_SDO_MESSAGE_H_
00003
00011 #include "message.h"
00012
00013 namespace CANopen {
00017 class SDOMessage : public Message {
00018     public:
00019         enum RDWR {
00020             Read,
00021             Write
00022         };
00023
00024         enum CCS {
00025             SegmentDownload = 0,
00026             InitiateDownload = 1,
00027             InitiateUpload = 2,
00028             SegmentUpload = 3,
00029             AbortTransfer = 4,
00030             BlockUpload = 5,
00031             BlockDownload = 6
00032         };
00033
00034         SDOMessage() = default;
00035         SDOMessage(const can_frame &other);
00036         SDOMessage(FunctionCode fn,
00037             uint8_t node_id,
00038             CCS spec,
00039             uint8_t n,
00040             uint8_t e,
00041             uint8_t s,
00042             uint16_t index,
00043             uint8_t subindex,
00044             Payload payload);
00045
00050         uint16_t
00051         index() const;
00052
00057         bool is_confirmation()
00058         {
00059             if(data[0]==0x60)
00060                 return true;
00061             else

```

```
00062         return false;
00063     };
00064
00069     bool is_error()
00070     {
00071         if(data[0]==0x80)
00072             return true;
00073         else
00074             return false;
00075     };
00076
00081     uint8_t
00082     subindex() const;
00083
00088     uint32_t
00089     index__sub() const;
00090
00095     uint32_t
00096     id() const{
00097         return index__sub();
00098     }
00099
00104     uint8_t
00105     size_data() const;
00106
00111     Payload
00112     payload() const;
00113 };
00114
00118 class SDOInbound : public SDOMessage {
00119     public:
00120     SDOInbound(const can_frame &other);
00121 };
00122
00126 class SDOOutbound : public SDOMessage {
00127     public:
00128     SDOOutbound(uint8_t node_id, RDWR dir, uint16_t index, uint8_t subindex, Payload payload);
00129 };
00130
00134 class SDOOutboundRead : public SDOOutbound {
00135     public:
00136     SDOOutboundRead(uint8_t node_id, uint16_t index, uint8_t subindex);
00137     SDOOutboundRead(uint8_t node_id, uint32_t index__sub);
00138 };
00139
00143 class SDOOutboundWrite : public SDOOutbound {
00144     public:
00145     SDOOutboundWrite(uint8_t node_id, uint16_t index, uint8_t subindex, Payload payload);
00146     SDOOutboundWrite(uint8_t node_id, uint32_t index__sub, Payload payload);
00147 };
00148 } // namespace CANopen
00149
00150 #endif // _CANOPEN_SDO_MESSAGE_H_
```


Index

- [_DCOMopmd_act](#)
 - [CANopen::Driver, 18](#)
 - [_DCOMstatus](#)
 - [CANopen::Driver, 17](#)
 - [_d2](#)
 - [stp::Model, 40](#)
 - [_p_act](#)
 - [CANopen::Driver, 18](#)
 - [_tq_act](#)
 - [CANopen::Driver, 18](#)
 - [_v_act](#)
 - [CANopen::Driver, 18](#)
- [activate_PDO](#)
 - [CANopen::Driver, 20](#)
- [CANopen::Driver, 13](#)
 - [_DCOMopmd_act, 18](#)
 - [_DCOMstatus, 17](#)
 - [_p_act, 18](#)
 - [_tq_act, 18](#)
 - [_v_act, 18](#)
 - [activate_PDO, 20](#)
 - [Control, 16](#)
 - [ctrl_to_str, 21](#)
 - [DCOMcontrol, 17](#)
 - [DCOMopmode, 17](#)
 - [DisableOperation, 16](#)
 - [DisableVoltage, 16](#)
 - [Driver, 20](#)
 - [EnableOperation, 16](#)
 - [Fault, 19](#)
 - [FaultReactionActive, 19](#)
 - [FaultResest, 16](#)
 - [get, 21](#)
 - [get_mode, 22](#)
 - [get_position, 22](#)
 - [get_state, 22](#)
 - [get_torque, 23](#)
 - [get_velocity, 23](#)
 - [HMmethod, 18](#)
 - [HMv, 18](#)
 - [HMv_out, 18](#)
 - [Homing, 17](#)
 - [InterpolatedPosition, 17](#)
 - [map_PDO, 23](#)
 - [mask, 18](#)
 - [NotReadyToSwitchtON, 18](#)
 - [OperationEnabled, 19](#)
 - [OperationMode, 16](#)
 - [PPp_target, 18](#)
 - [PPv_target, 18](#)
 - [ProfilePosition, 17](#)
 - [ProfileTorque, 17](#)
 - [ProfileVelocity, 17](#)
 - [PTtq_target, 18](#)
 - [PVv_target, 18](#)
 - [QuickStop, 16](#)
 - [QuickStopActive, 20](#)
 - [RAMP_v_acc, 18](#)
 - [RAMP_v_dec, 18](#)
 - [ReadyToSwitchON, 19](#)
 - [Register, 17](#)
 - [send, 24](#)
 - [set, 24](#)
 - [set_control, 25](#)
 - [set_mode, 25](#)
 - [set_position, 26](#)
 - [set_position_offset, 26](#)
 - [set_torque, 27](#)
 - [set_velocity, 27](#)
 - [Shutdown, 16](#)
 - [State, 18](#)
 - [SwitchedON, 19](#)
 - [SwitchON, 16](#)
 - [SwitchONDisabled, 19](#)
 - [update, 27](#)
 - [Velocity, 17](#)
 - [wait_state, 28](#)
- [CANopen::EMCYMessage, 28](#)
- [CANopen::LXM32, 32](#)
 - [get_angle, 33](#)
 - [LXM32, 33](#)
 - [set_angle, 34](#)
- [CANopen::Message, 36](#)
 - [Emergency, 37](#)
 - [function_code, 38](#)
 - [FunctionCode, 37](#)
 - [Heartbeat, 37](#)
 - [id, 38](#)
 - [NMT, 37](#)
 - [node_id, 38](#)
 - [payload, 38](#)
 - [PDO1Receive, 37](#)
 - [PDO1Transmit, 37](#)
 - [PDO2Receive, 37](#)
 - [PDO2Transmit, 37](#)
 - [PDO3Receive, 37](#)
 - [PDO3Transmit, 37](#)

- PDO4Receive, 37
 - PDO4Transmit, 37
 - SDOReceive, 37
 - SDOTransmit, 37
 - Sync, 37
 - TimeStamp, 37
- CANopen::NMTMessage, 44
- CANopen::Parameter, 45
 - from_payload, 48
 - get, 49
 - has_been_sent, 49
 - link_to_pdo, 49
 - operator=, 50–52
 - param_cb_t, 47
 - Parameter, 47, 48
 - payload, 52
 - set, 52
- CANopen::Payload, 53
 - operator<, 55
 - Payload, 54
 - store_at, 56
 - value, 56
- CANopen::PDOMessage, 57
- CANopen::SDOInbound, 60
- CANopen::SDOMessage, 62
 - id, 63
 - index, 63
 - index_sub, 64
 - is_confirmation, 64
 - is_error, 64
 - payload, 64
 - size_data, 65
 - subindex, 65
- CANopen::SDOOutbound, 66
- CANopen::SDOOutboundRead, 67
- CANopen::SDOOutboundWrite, 69
- CANopen::Socket, 70
 - send, 72
 - Socket, 71, 72
- CANopen_driver.h, 73, 75
- CANopen_lxm32.h, 77, 79
- CANopen_socket.h, 79, 81
 - IF_VERBOSE, 81
- Control
 - CANopen::Driver, 16
- ctrl_to_str
 - CANopen::Driver, 21
- DCOMcontrol
 - CANopen::Driver, 17
- DCOMopmode
 - CANopen::Driver, 17
- DisableOperation
 - CANopen::Driver, 16
- DisableVoltage
 - CANopen::Driver, 16
- Driver
 - CANopen::Driver, 20
- emcy.h, 82, 83
- Emergency
 - CANopen::Message, 37
- EnableOperation
 - CANopen::Driver, 16
- Fault
 - CANopen::Driver, 19
- FaultReactionActive
 - CANopen::Driver, 19
- FaultResest
 - CANopen::Driver, 16
- from_payload
 - CANopen::Parameter, 48
- function_code
 - CANopen::Message, 38
- FunctionCode
 - CANopen::Message, 37
- get
 - CANopen::Driver, 21
 - CANopen::Parameter, 49
- get_angle
 - CANopen::LXM32, 33
- get_mode
 - CANopen::Driver, 22
- get_position
 - CANopen::Driver, 22
- get_state
 - CANopen::Driver, 22
- get_torque
 - CANopen::Driver, 23
- get_velocity
 - CANopen::Driver, 23
- has_been_sent
 - CANopen::Parameter, 49
- Heartbeat
 - CANopen::Message, 37
- HMmethod
 - CANopen::Driver, 18
- HMv
 - CANopen::Driver, 18
- HMv_out
 - CANopen::Driver, 18
- Homing
 - CANopen::Driver, 17
- id
 - CANopen::Message, 38
 - CANopen::SDOMessage, 63
- IF_VERBOSE
 - CANopen_socket.h, 81
- index
 - CANopen::SDOMessage, 63
- index_sub
 - CANopen::SDOMessage, 64
- InterpolatedPosition
 - CANopen::Driver, 17

- is_confirmation
 - CANopen::SDOMessage, 64
- is_error
 - CANopen::SDOMessage, 64
- link_to_pdo
 - CANopen::Parameter, 49
- LXM32
 - CANopen::LXM32, 33
- m_A
 - stp::Model, 40
- m_a2
 - stp::Model, 40
- m_alpha
 - stp::Model, 41
- m_alpha_spd
 - stp::Model, 41
- m_B
 - stp::Model, 41
- m_beta
 - stp::Model, 41
- m_gamma
 - stp::Model, 41
- m_I2
 - stp::Model, 42
- m_motors
 - stp::Platform, 60
- m_P
 - stp::Model, 42
- m_P1
 - stp::Model, 42
- m_parity
 - stp::Model, 42
- m_R
 - stp::Model, 42
- m_T
 - stp::Model, 43
- m_T_spd
 - stp::Model, 43
- m_theta
 - stp::Model, 43
- m_theta_spd
 - stp::Model, 43
- m_trig
 - stp::Model, 43
- MainWindow, 35
- map_PDO
 - CANopen::Driver, 23
- mask
 - CANopen::Driver, 18
- message.h, 83, 85
- NMT
 - CANopen::Message, 37
- nmt.h, 85, 87
- node_id
 - CANopen::Message, 38
- NotReadyToSwitchtON
 - CANopen::Driver, 18
- OperationEnabled
 - CANopen::Driver, 19
- OperationMode
 - CANopen::Driver, 16
- operator<<
 - CANopen::Payload, 55
- operator=
 - CANopen::Parameter, 50–52
- param_cb_t
 - CANopen::Parameter, 47
- Parameter
 - CANopen::Parameter, 47, 48
- parameter.h, 87, 88
- Payload
 - CANopen::Payload, 54
- payload
 - CANopen::Message, 38
 - CANopen::Parameter, 52
 - CANopen::SDOMessage, 64
- payload.h, 90, 92
- PDO1Receive
 - CANopen::Message, 37
- PDO1Transmit
 - CANopen::Message, 37
- PDO2Receive
 - CANopen::Message, 37
- PDO2Transmit
 - CANopen::Message, 37
- PDO3Receive
 - CANopen::Message, 37
- PDO3Transmit
 - CANopen::Message, 37
- PDO4Receive
 - CANopen::Message, 37
- PDO4Transmit
 - CANopen::Message, 37
- PPp_target
 - CANopen::Driver, 18
- PPv_target
 - CANopen::Driver, 18
- ProfilePosition
 - CANopen::Driver, 17
- ProfileTorque
 - CANopen::Driver, 17
- ProfileVelocity
 - CANopen::Driver, 17
- PTtq_target
 - CANopen::Driver, 18
- PVv_target
 - CANopen::Driver, 18
- QuickStop
 - CANopen::Driver, 16
- QuickStopActive
 - CANopen::Driver, 20

- RAMP_v_acc
 - CANopen::Driver, [18](#)
- RAMP_v_dec
 - CANopen::Driver, [18](#)
- ReadyToSwitchON
 - CANopen::Driver, [19](#)
- Register
 - CANopen::Driver, [17](#)
- sdo.h, [92](#), [94](#)
- SDOReceive
 - CANopen::Message, [37](#)
- SDOTransmit
 - CANopen::Message, [37](#)
- send
 - CANopen::Driver, [24](#)
 - CANopen::Socket, [72](#)
- set
 - CANopen::Driver, [24](#)
 - CANopen::Parameter, [52](#)
- set_angle
 - CANopen::LXM32, [34](#)
- set_control
 - CANopen::Driver, [25](#)
- set_mode
 - CANopen::Driver, [25](#)
- set_position
 - CANopen::Driver, [26](#)
- set_position_offset
 - CANopen::Driver, [26](#)
- set_torque
 - CANopen::Driver, [27](#)
- set_velocity
 - CANopen::Driver, [27](#)
- Shutdown
 - CANopen::Driver, [16](#)
- size_data
 - CANopen::SDOMessage, [65](#)
- Socket
 - CANopen::Socket, [71](#), [72](#)
- State
 - CANopen::Driver, [18](#)
- store_at
 - CANopen::Payload, [56](#)
- stp::Gazebo_sim, [30](#)
- stp::Gnuplot_sim, [31](#)
- stp::Model, [39](#)
 - _d2, [40](#)
 - m_A, [40](#)
 - m_a2, [40](#)
 - m_alpha, [41](#)
 - m_alpha_spd, [41](#)
 - m_B, [41](#)
 - m_beta, [41](#)
 - m_gamma, [41](#)
 - m_I2, [42](#)
 - m_P, [42](#)
 - m_P1, [42](#)
 - m_parity, [42](#)
 - m_R, [42](#)
 - m_T, [43](#)
 - m_T_spd, [43](#)
 - m_theta, [43](#)
 - m_theta_spd, [43](#)
 - m_trig, [43](#)
- stp::Platform, [59](#)
 - m_motors, [60](#)
- subindex
 - CANopen::SDOMessage, [65](#)
- SwitchedON
 - CANopen::Driver, [19](#)
- SwitchON
 - CANopen::Driver, [16](#)
- SwitchONDisabled
 - CANopen::Driver, [19](#)
- Sync
 - CANopen::Message, [37](#)
- TimeStamp
 - CANopen::Message, [37](#)
- update
 - CANopen::Driver, [27](#)
- value
 - CANopen::Payload, [56](#)
- Velocity
 - CANopen::Driver, [17](#)
- wait_state
 - CANopen::Driver, [28](#)