Stewart Platform

Generated by Doxygen 1.8.17

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-utilis`

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:

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

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

Chapter 7

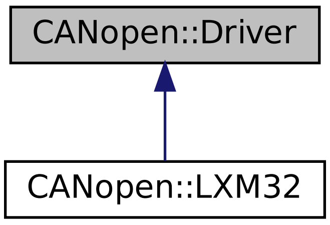
Class Documentation

7.1 CANopen::Driver Class Reference

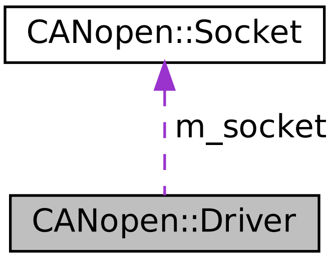
Device Profile Drives and Motion Control.

```
#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, NotReadyToSwitchtON = 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, **Valid↩Ref_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 opereration 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 axles 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: • 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: • 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$_h$01** - Speed during search for switch.
HMv_out	**6099$_h$02** - 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: • 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.

Enumerator

SwitchONDisabled	Switch ON Disabled: • 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	Ready to Switch ON: • High Voltage may be applied to the drive. • The drive parameters may be changed. • The drive function is disabled
SwitchedON	Switched ON: • 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	Operation Enabled: • 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	Fault: • The drive parameters may be changed. • A fault has occured in the drive. • The drive function is disabled.
FaultReactionActive	Fault Reaction Active: • The drive parameters may be changed. • A non-fatal fault has occured in the drive. • The Quick Stop function is being executed. • The drive function is enabled and power is applied to the motor.

Enumerator

QuickStopActive	Quick Stop Active: • 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

ifname	: Name of the CAN interface.
can_id	: Node CAN ID of the driver.
verbose←_lvl	: 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

ifname	: N
can←_id	: 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

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

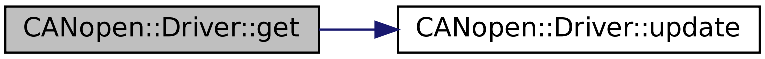
Returns

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

force_sdo	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_PD↩
O_socket() thread, the value of the pdo will be stored in the mapped parameter.

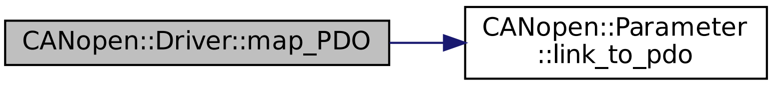
Parameters

fn	: Function code of the PDO.
param	: Parameter to map.
slot	: 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 Writting SDO message to the driver

Parameters

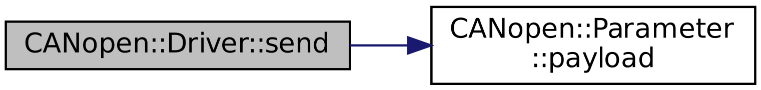
param	Parameter to send.

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.

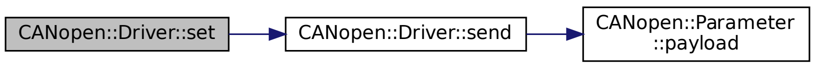
Parameters

reg	: The register to set. (In the format ind__sub)
val	: The value to store in the register.
force_sdo	: 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.
wait	: 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

ctrl	: 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 opereration mode.

Parameters

mode	Mode to set.
wait	Repeatidly test the actual operation mode register until it is eqaual 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

target	Position to reach (in internal unit)
absolute	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

offset_pos	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

target	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

target	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

param	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))

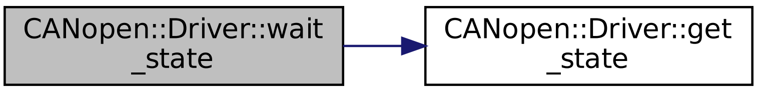
Parameters

state	State to wait for.
_mask	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:

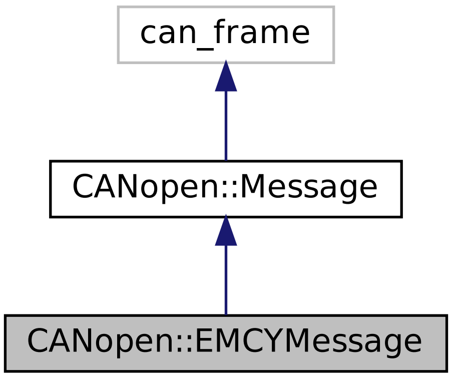
- CANopen_driver.h
- CANopen_driver.cpp

7.2 CANopen::EMCYMessage Class Reference

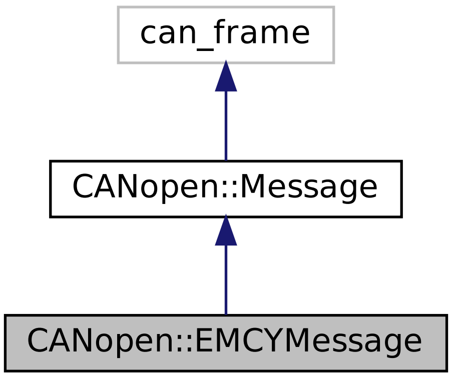
EMCY Message (Emergency Object)

```
#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)

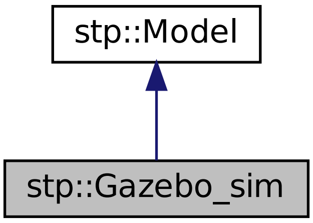
Definition at line 17 of file emcy.h.

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

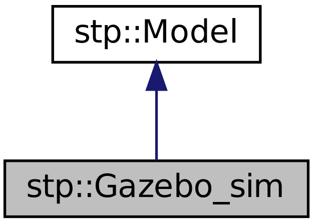
- emcy.h
- emcy.cpp

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

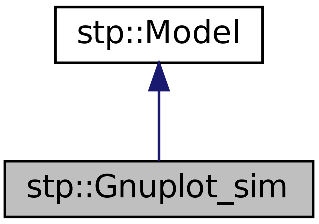
Definition at line 15 of file gazebo_sim.hpp.

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

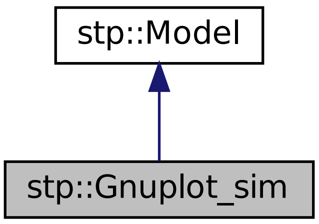
- gazebo_sim.hpp
- gazebo_sim.cpp

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:

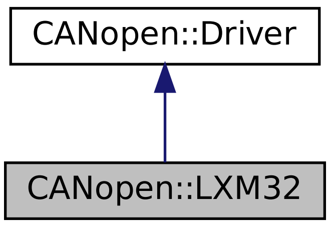
- gnuplot_sim.hpp
- gnuplot_sim.cpp

7.5 CANopen::LXM32 Class Reference

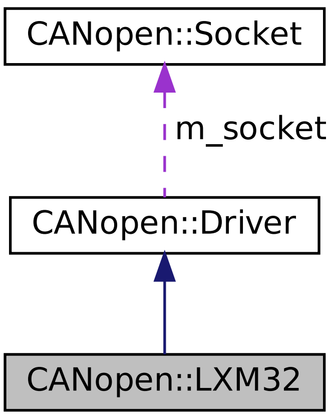
Implementation of the Driver Class for a LXM32 driver.

```
#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

ifname	: Name of the CAN interface.
can_id	: Node CAN ID of the driver.
verbose	

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

radian	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

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

Returns

True if the position was sent.

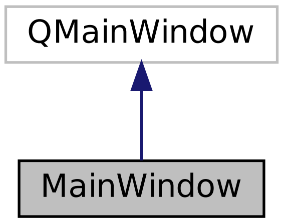
Definition at line 9 of file CANopen_lxm32.cpp.

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

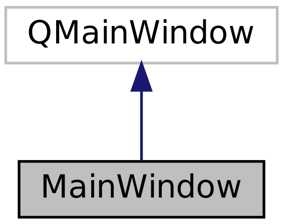
- CANopen_lxm32.h
- CANopen_lxm32.cpp

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:

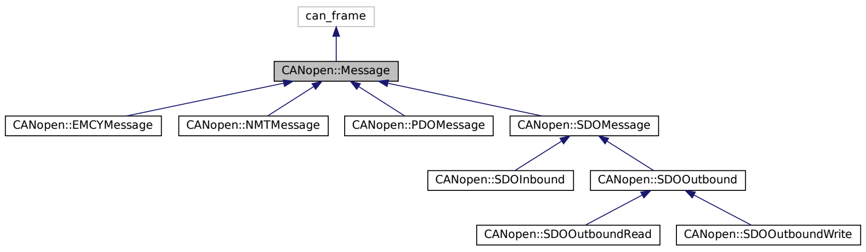
- mainwindow.h
- mainwindow.cpp

7.7 CANopen::Message Class Reference

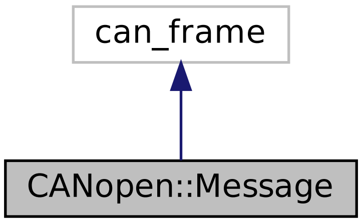
can_frame object sent and received throught CANopen socket.

```
#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 throught 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 Transmiting Message (Process Data Object)
PDO1Receive	**200_h** PDO1 Receiving Message (Process Data Object)
PDO2Transmit	**280_h** PDO2 Transmiting Message (Process Data Object)
PDO2Receive	**300_h** PDO2 Receiving Message (Process Data Object)
PDO3Transmit	**380_h** PDO3 Transmiting Message (Process Data Object)
PDO3Receive	**400_h** PDO3 Receiving Message (Process Data Object)
PDO4Transmit	**480_h** PDO4 Transmiting Message (Process Data Object)
PDO4Receive	**500_h** PDO4 Receiving Message (Process Data Object)
SDOTransmit	**580_h** SDO Transmiting 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.

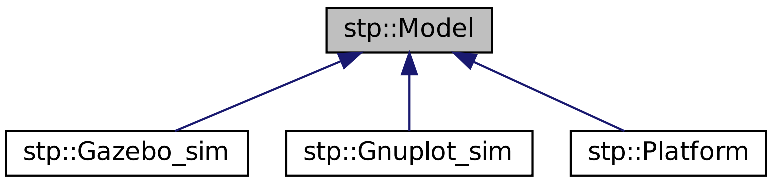
Definition at line 29 of file message.cpp.

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 **_alphal** [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 postion 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:

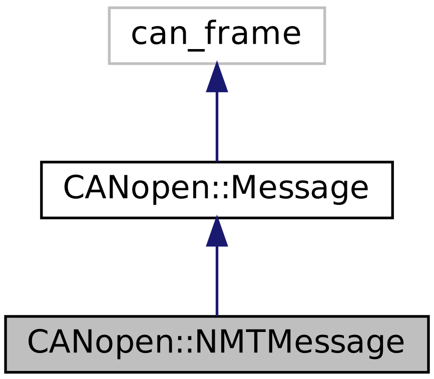
- model.hpp
- model.cpp

7.9 CANopen::NMTMessage Class Reference

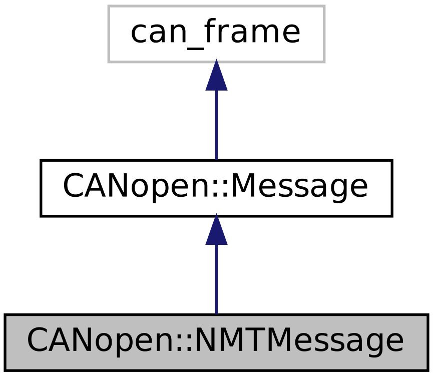
NMT Message (Network management)

```
#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

*name_*	Name of the parameter.
val	Value to store in the parameter (the type will be used to fixed the data size)
*index_*	Index of the object in the object dictionary of the device.
subindex←↩ _	Subindex of the object in the object dictionary of the device.
cb	[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

*name_*	Name of the parameter.
val	Value to store in the parameter (the type will be used to fixed the data size)
index__sub	Index and subindex of the object in the object dictionary of the device. (in the format index__sub)
cb	[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

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

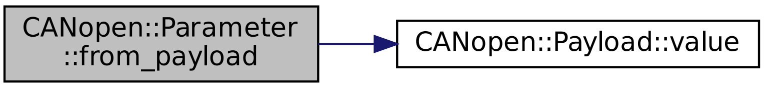
Returns

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

fn	The function code of the PDO
slot	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

val	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

val	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

val	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

val	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

val	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

val	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

should_be_sent	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

val	The value to set. (the type has to be coherent with the parameter data size)
force_update	If set the value of the Parameter is set even idf the previous value has not been sent yet to the remote device.
received_data	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:

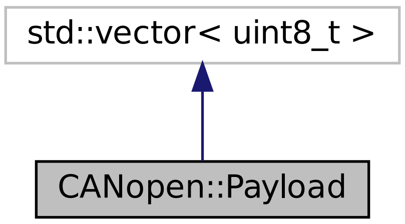
- parameter.h
- parameter.cpp

7.11 CANopen::Payload Class Reference

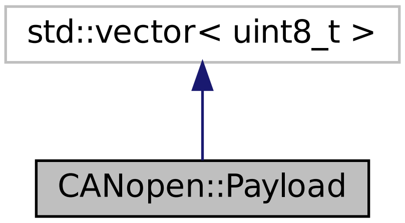
Payload of CANopen message: array of 1 to 8 bytes of data.

```
#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 transfrom 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 transfrom the n bytes of the data into a array of n bytes.

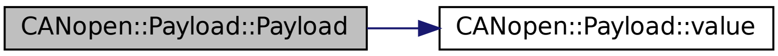
Parameters

value	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

p	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

value	The variable to store

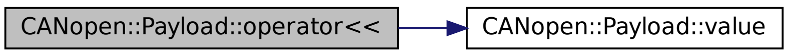
Returns

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

p	The variable to store.
slot	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

begin	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:

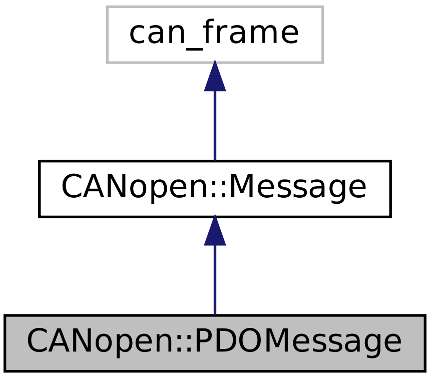
- payload.h
- payload.cpp

7.12 CANopen::PDOMessage Class Reference

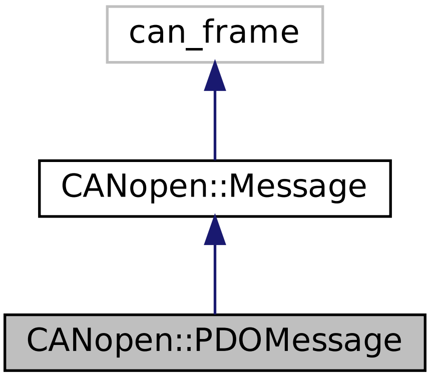
PDO Message (Process Data Object)

```
#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)

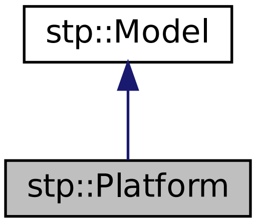
Definition at line 18 of file pdo.h.

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

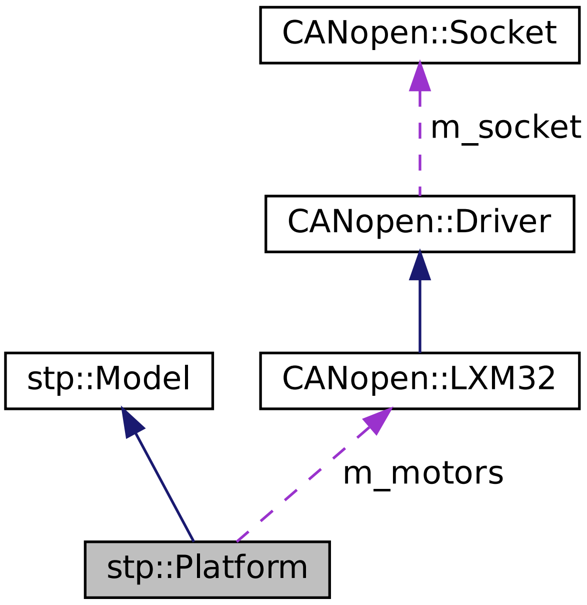
- pdo.h
- pdo.cpp

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:

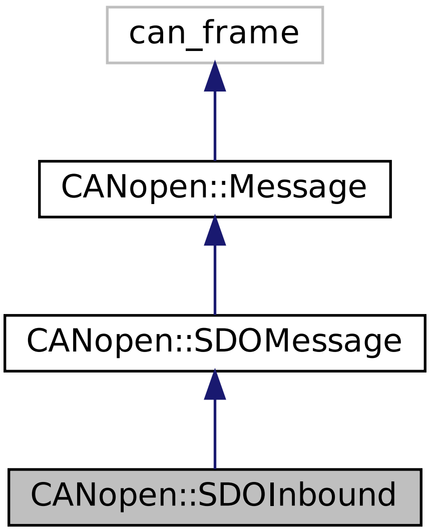
- platform.hpp
- platform.cpp

7.14 CANopen::SDOInbound Class Reference

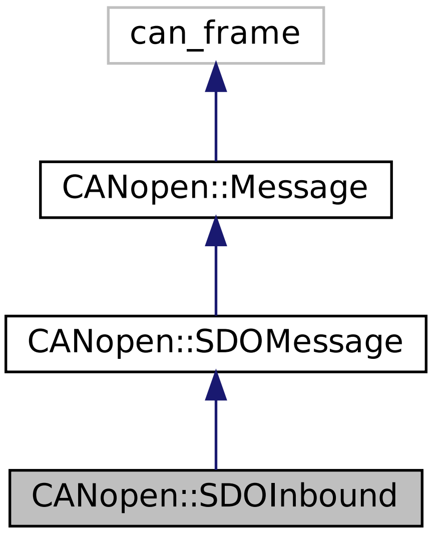
SDO received Message.

`#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:

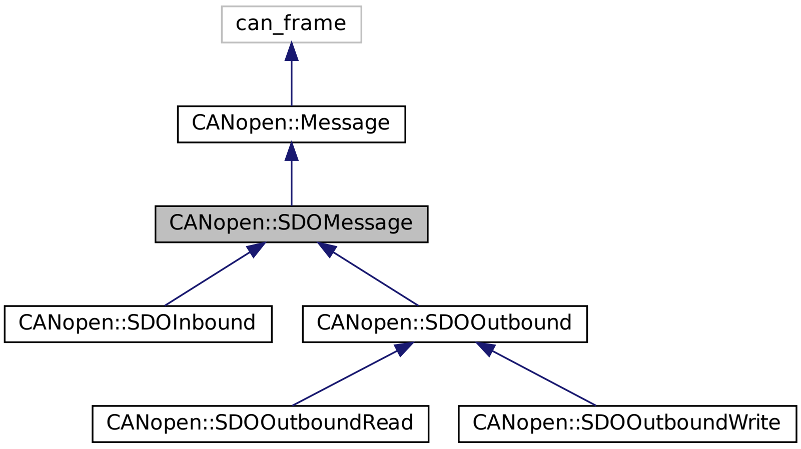
- sdo.h
- sdo.cpp

7.15 CANopen::SDOMessage Class Reference

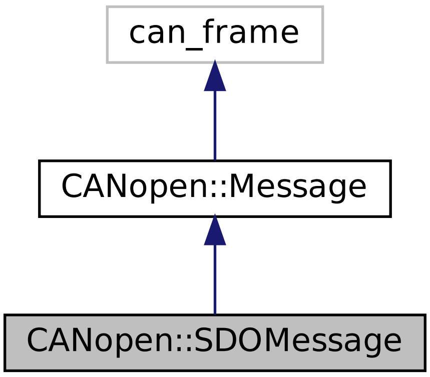
SDO Message (Service Data Object)

`#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

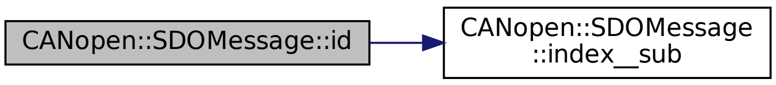
Returns the index__sub of the register.

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 throught 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:

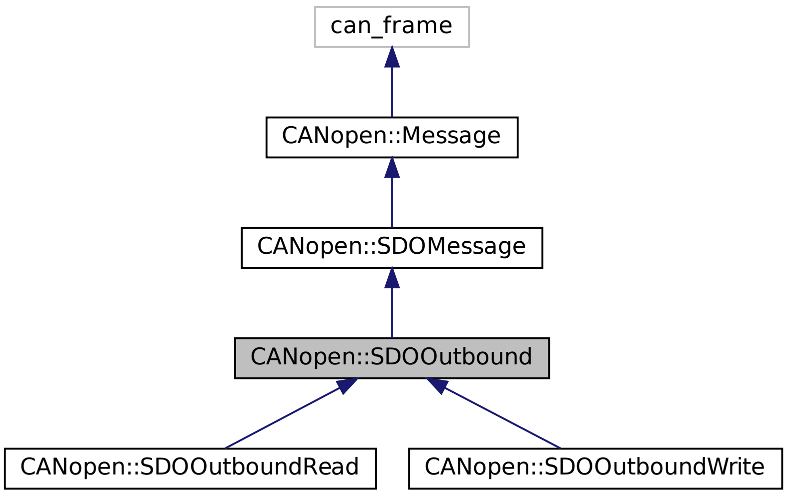
- sdo.h
- sdo.cpp

7.16 CANopen::SDOOutbound Class Reference

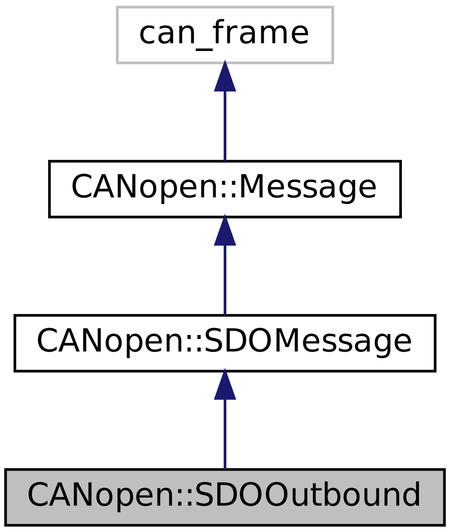
SDO Message to be sent.

```
#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:

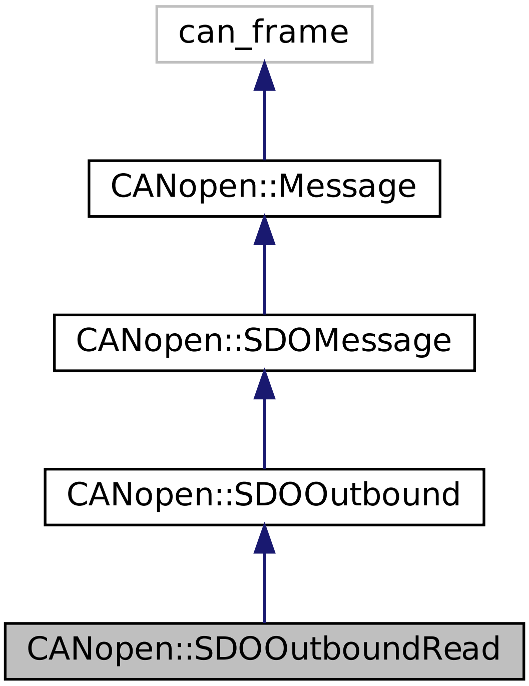
- sdo.h
- sdo.cpp

7.17 CANopen::SDOOutboundRead Class Reference

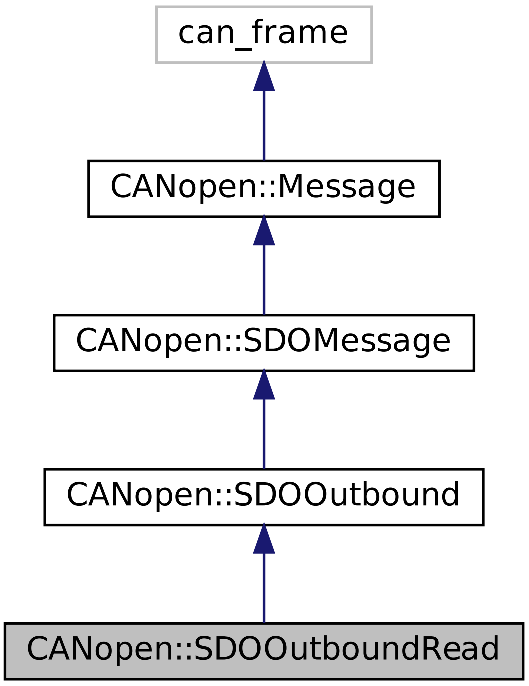
SDO Message to be sent to read the value from the object dictionary of a remote device.

```
#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:

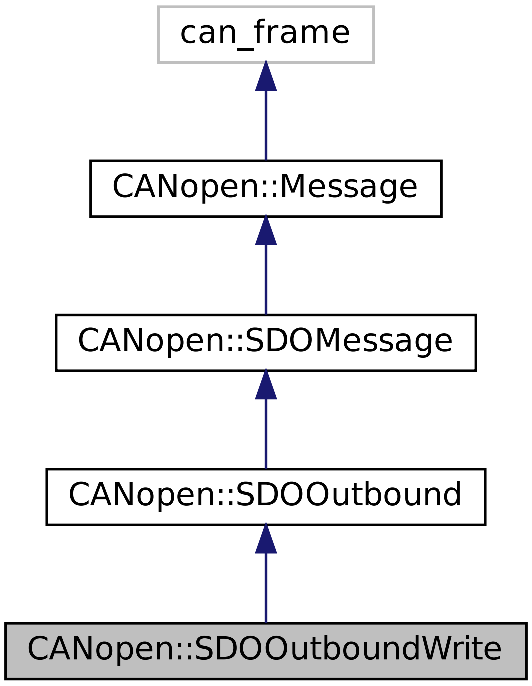
- sdo.h
- sdo.cpp

7.18 CANopen::SDOOutboundWrite Class Reference

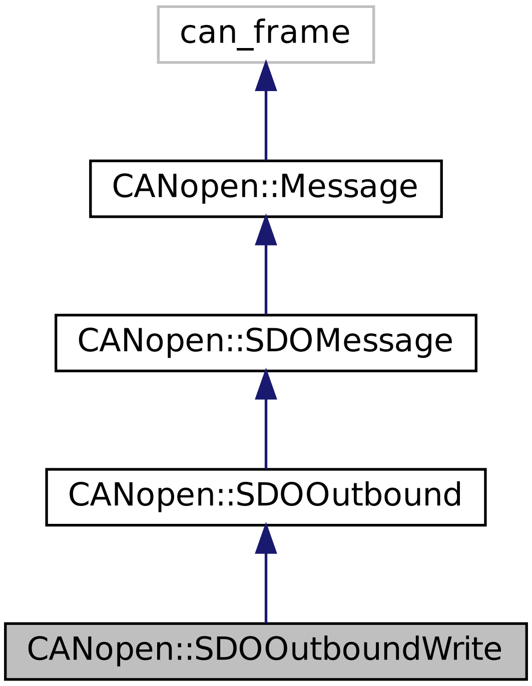
SDO Message to be sent to write the value of the object dictionary of a remote device.

```
#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.

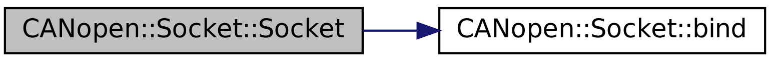
Parameters

ifname	: Name of the can interface ex:"can0"
verbose	: 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

ifname	: Name of the can interface ex:"can0"
cob_id	: Filtering only frame with this ID.
verbose	: 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

msg	: 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

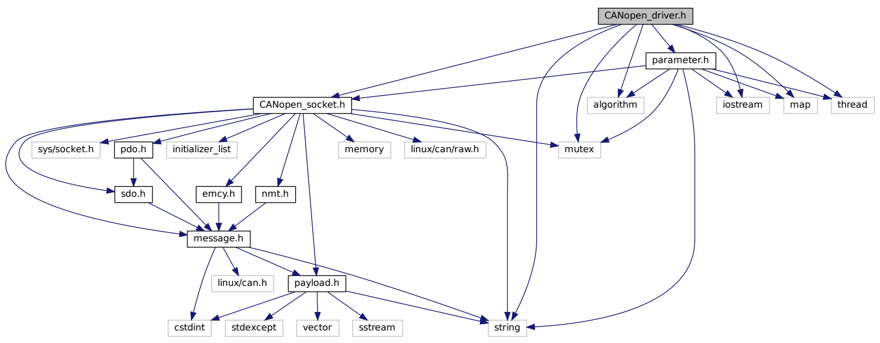
File Documentation

8.1 CANopen_driver.h File Reference

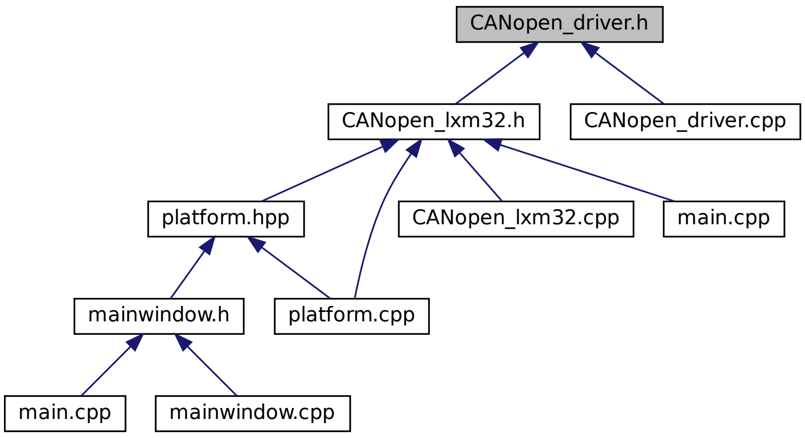
Device Profile Drives and Motion Control.

```
#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

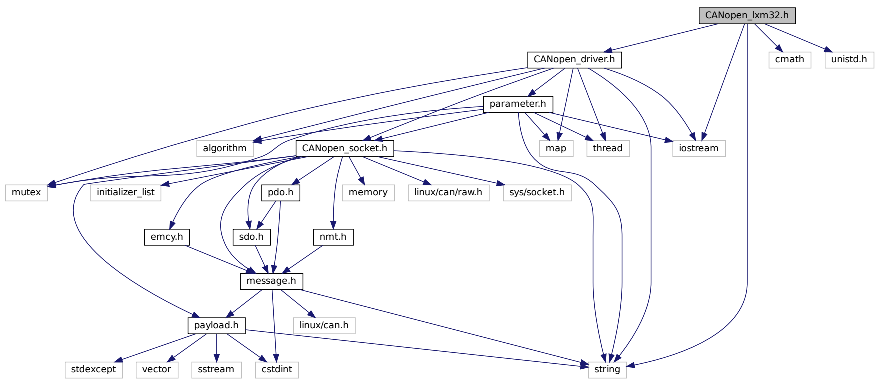
```
#ifndef _CANOPEN_DRIVER_H_
#define _CANOPEN_DRIVER_H_

#include "CANopen_socket.h"
#include "parameter.h"

#include <algorithm>
#include <iostream>
#include <map>
#include <mutex>
#include <string>
#include <thread>

namespace CANopen {
void
print_status(Parameter *);
class Driver {

    static constexpr int NB_PDO = 4;
    static constexpr int MAX_PDO_SLOT = 2;

    public:
    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,
        NotReadyToSwitchtON = 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,
    };

    Driver(const char *ifname, uint16_t can_id, int verbose_lvl = 0);

    template <typename T>
    void
    set(Register reg, T val, bool force_sdo = false, bool wait = false) {

        if(m_available) {
            m_parameters[reg]->set(val);
            if(m_parameters[reg]->pdo_slot == -1 || force_sdo || wait)
                send(m_parameters[reg]);
            if(wait)
                while(m_parameters[reg]->sdo_flag.test_and_set())
                    ;
        }
    }

    template <typename T>
    T
    get(Register reg, bool force_sdo = false) {

        if(force_sdo && m_available) {
            update(m_parameters[reg]);
            while(m_parameters[reg]->sdo_flag.test_and_set())
                ;
        }
        return m_parameters[reg]->get<T>();
    };

    void
    set_control(Control ctrl);

    State
    get_state() { return m_parameters[_DCOMstatus]->get<State>(); };

    void
    wait_state(State state, uint16_t _mask = mask) {
        while((get_state() & mask) != (state & mask))
            ;
    }; //std::cout « (get_state()&mask) « " " « (state&mask)« "\n";}

    void
    set_mode(OperationMode mode, bool wait = false);

    OperationMode
    get_mode(bool force_sdo = true) { return this->get<OperationMode>(_DCOMopmd_act, force_sdo); };

    bool
    set_position(int32_t target, bool absolute = true);
    bool
    set_velocity(int32_t target);
    bool
    set_torque(int16_t target);

    int32_t
    get_position() { return m_parameters[_p_act]->get<int32_t>() - m_offset_pos; };
    int32_t
    get_velocity() { return m_parameters[_v_act]->get<int32_t>(); };
    int32_t
    get_torque() { return m_parameters[_tq_act]->get<int32_t>(); };

    void
    set_position_offset(int32_t offset_pos) { m_offset_pos = offset_pos; };

    void
    start();
    void
    pause();
    void
    stop();

    void
    profilePosition_mode();
    void
    profileVelocity_mode();
    void
    profileTorque_mode();
    void
    homing();

    Parameter *
    get_param(Register reg) { return m_parameters[reg]; };

    virtual void
    print_manufacturer_status() = 0;

    std::string
    ctrl_to_str(Control control);

    bool
    is_available() { return m_available; };

    protected:
    void
    send(Parameter *param);

    void
    update(Parameter *param);

    void
    map_PDO(PDOFunctionCode fn, Parameter *param, int slot);

    void
    activate_PDO(PDOFunctionCode fn, bool set = true);

    void
    T_socket();

    void
    RPDO_socket();

    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;
};

} // namespace CANopen
#endif
```

8.3 CANopen_lxm32.h File Reference

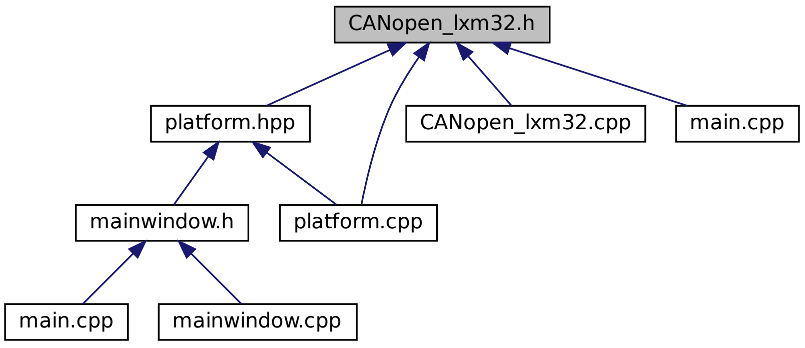
Implementation of the Driver Class for a LXM32 driver.

```
#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

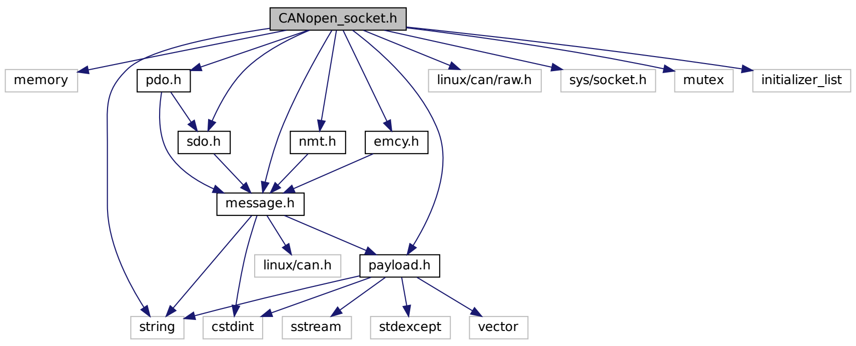
```
#ifndef _LEXIUM32A_CANOPEN_H_
#define _LEXIUM32A_CANOPEN_H_

#include "CANopen_driver.h"

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

namespace CANopen {
class LXM32 : public Driver {
    public:
    LXM32(const char *ifname, uint16_t can_id, bool verbose = false);

    bool
    set_angle(double ang, bool absolute = true, bool radian = true);

    double
    get_angle(bool radian = true);

    void
    print_manufacturer_status(){};


    int nb_index_per_turn = 737280;
};
} // namespace CANopen

#endif
```

8.5 CANopen_socket.h File Reference

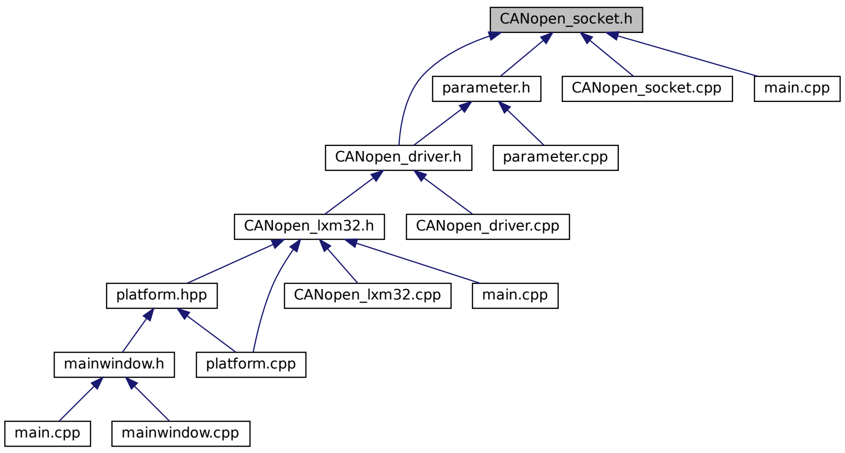
Canopen socket able to send/receive messages through a CAN interface using the UNIX socket.

```
#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 "sdo.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.

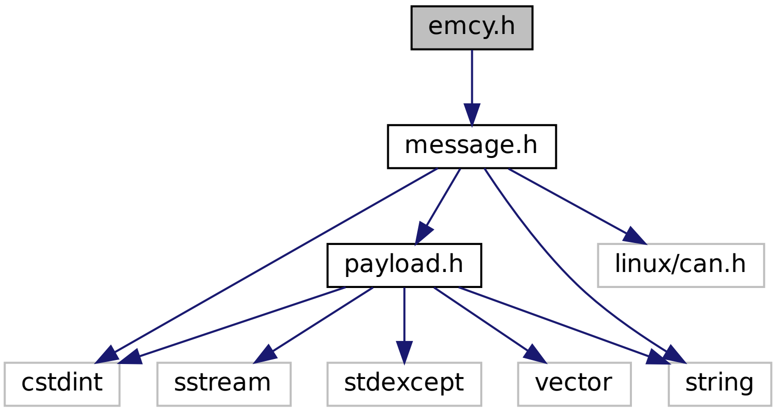
8.6 CANopen_socket.h

```
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
00034 namespace CANopen {
00035 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
00058   void add_filter(std::initializer_list<struct can_filter> rfilter);
00059
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_
```

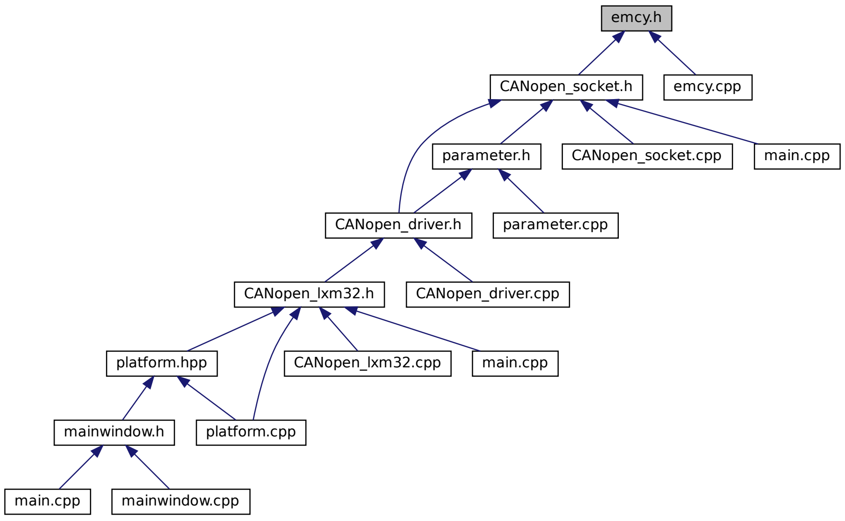
8.7 emcy.h File Reference

PDO message sent and received throught 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 throught CANopen socket.

Author

Florian Richer & Alexis Devillard

Version

1.0

Definition in file emcy.h.

8.8 emcy.h

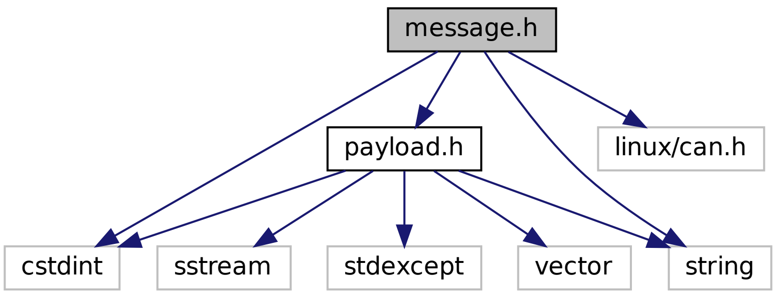
```
#ifndef _CANOPEN_EMCY_MESSAGE_H_
#define _CANOPEN_EMCY_MESSAGE_H_

#include "message.h"

namespace CANopen {
class EMCYMessage : public Message {
    public:

    EMCYMessage() = default;
    EMCYMessage(const can_frame &other);

    uint16_t code() const;
    uint8_t reg() const;
};
} // namespace CANopen

#endif // _CANOPEN_NMT_MESSAGE_H_
```

8.9 message.h File Reference

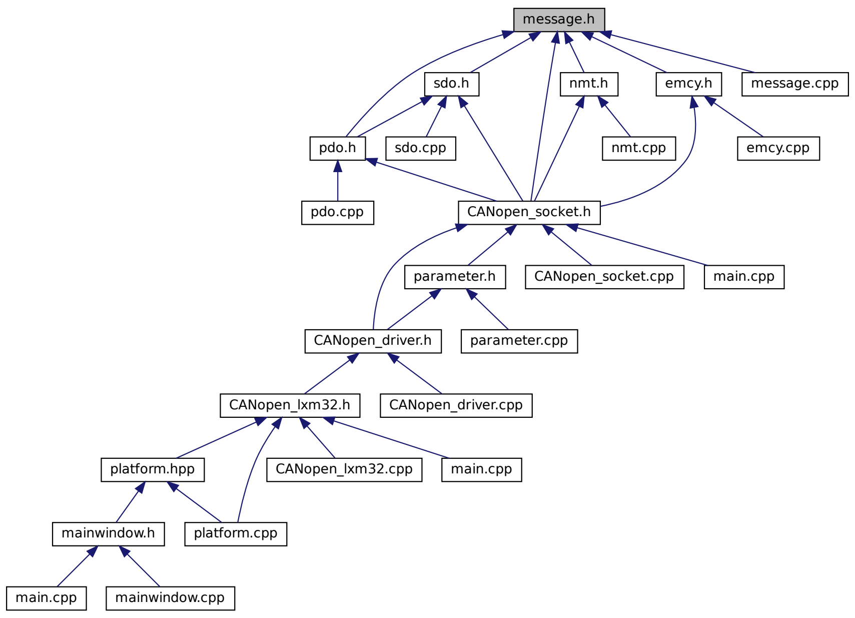
CAN_frame message sent and received throught CANopen socket.

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

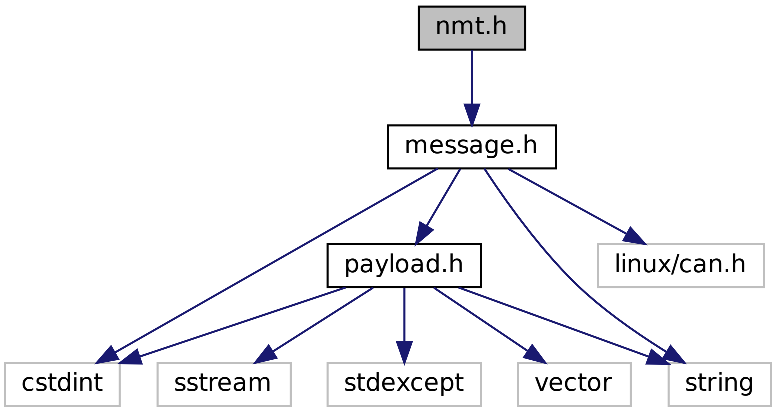
```
#ifndef MESSAGE_H
#define MESSAGE_H

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

namespace CANopen {
class Message : public can_frame {
    public:
    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
    };

    Message() = default;
    Message(const can_frame &other);
    Message(uint32_t cob_id, Payload payload);

    operator can_frame *() const { return const_cast<can_frame *>(reinterpret_cast<const can_frame
  *>(&can_id)); };

    FunctionCode
    function_code() const;

    uint8_t
    node_id() const;

    virtual Payload
    payload() const;
    virtual uint32_t
    id() const { return 0; };

    std::string
    to_string() const;
};
} // namespace CANopen

#endif // MESSAGE_H
```

8.11 nmt.h File Reference

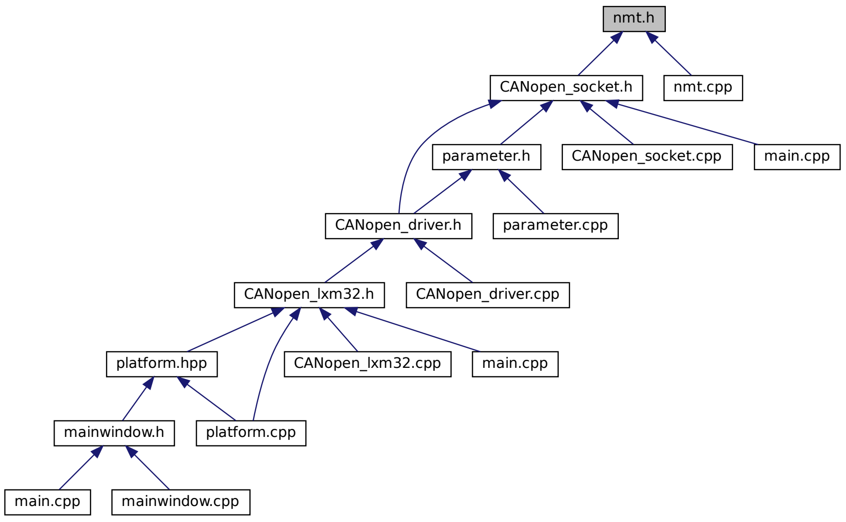
NMT message sent and received throught CANopen socket.

```
#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 throught CANopen socket.

Author

Florian Richer & Alexis Devillard

Version

1.0

Definition in file nmt.h.

8.12 nmt.h

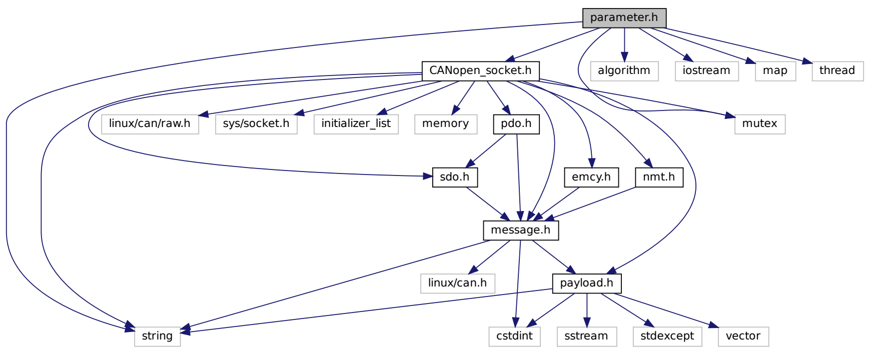
```
#ifndef _CANOPEN_NMT_MESSAGE_H_
#define _CANOPEN_NMT_MESSAGE_H_

#include "message.h"

namespace CANopen {
class NMTMessage : public Message {
    public:
    enum Code : uint8_t {
        Initialising = 0,
        GoToOperational = 0x01,
        GoToStopped = 0x02,
        Stopped = 0x04,
        Operational = 0x05,
        PreOperational = 0x7f,
        GoToPreOperational = 0x80,
        GoToResetNode = 0x81,
        GoToResetCommunication = 0x82
    };

    NMTMessage() = default;
    NMTMessage(const can_frame &other);
    NMTMessage(Code code, uint8_t node_id);
};
} // namespace CANopen

#endif // _CANOPEN_NMT_MESSAGE_H_
```

8.13 parameter.h File Reference

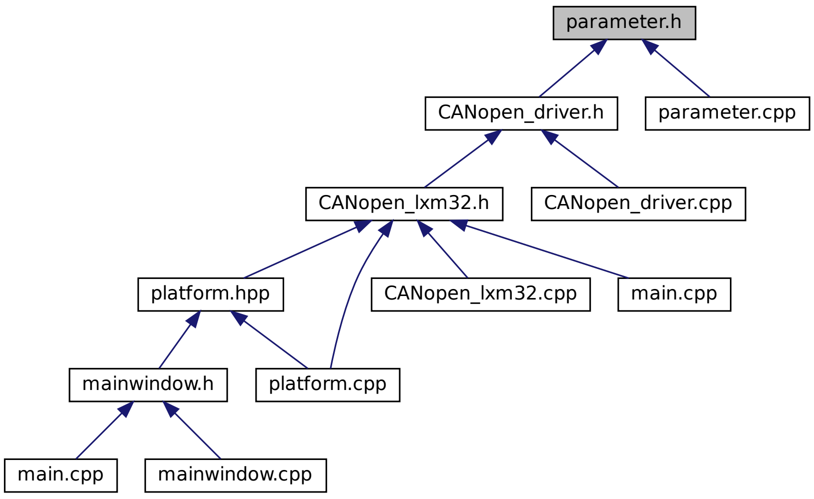
Device Object from the object dictionary of a remote CANopen device.

```
#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

```
#ifndef _PARAMETER_H_
#define _PARAMETER_H_

#include "CANopen_socket.h"

#include <algorithm>
#include <iostream>
#include <map>
#include <mutex>
#include <string>
#include <thread>
```

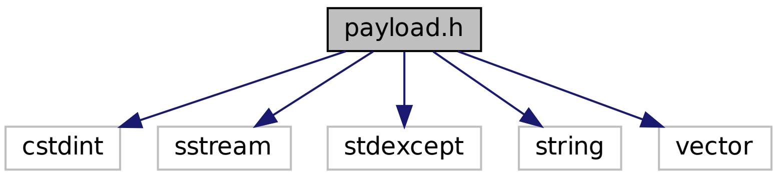
```

namespace CANopen {
struct Parameter {

    typedef void (*param_cb_t)(Parameter *); //type of the parameter callback function

    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,
    };

    Parameter() { var = new int32_t; };

    template <typename T>
    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) {
        mutex.lock();
        var = new T;
        *(T *)var = (T)val;
        m_should_be_sent = false;
        mutex.unlock();
        size = sizeof(T);
    };

    template <typename T>
    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){};

    ~Parameter() {
        delete(int32_t *)var;
    }

    void
    link_to_pdo(PDOFunctionCode fn, int8_t slot);

    template <typename T>
    bool
    set(T val, bool force_update = false, bool received_data = false) {
        bool was_updated = false;
        if(sizeof(T) == size) {
            mutex.lock();
            if(!m_should_be_sent || force_update) //if force update (even if the previous value was not sent yet) or value already update
            {
                if(*(T *)var != (T)val) {
                    was_updated = true;
                    *(T *)var = (T)val;
                }
                if(!received_data)
                    m_should_be_sent = true;
            }
            mutex.unlock();
            return was_updated;
        }
        return false;
    }

    bool
    operator=(int8_t val) { return this->set<int8_t>(val, false); } //return fakse if the assignement didn't succeed (wrong type size, not updated yet)
    bool
    operator=(int16_t val) { return this->set<int16_t>(val, false); } //return fakse if the assignement didn't succeed (wrong type size, not updated yet)
    bool
    operator=(int32_t val) { return this->set<int32_t>(val, false); } //return fakse if the assignement didn't succeed (wrong type size, not updated yet)
    bool
    operator=(uint8_t val) { return this->set<uint8_t>(val, false); } //return fakse if the assignement didn't succeed (wrong type size, not updated yet)
    bool
    operator=(uint16_t val) { return this->set<uint16_t>(val, false); } //return fakse if the assignement didn't succeed (wrong type size, not updated yet)
    bool
    operator=(uint32_t val) { return this->set<uint32_t>(val, false); } //return fakse if the assignement didn't succeed (wrong type size, not updated yet)

    template <typename T>
    T
    get() {
      T val={};
        if(sizeof(T) == size) {
            mutex.lock();
            val = *(T *)var;
            mutex.unlock();
        }
        return val;
    }

    operator int8_t() { return this->get<int8_t>(); };
    operator int16_t() { return this->get<int16_t>(); };
    operator int32_t() { return this->get<int32_t>(); };
    operator uint8_t() { return this->get<uint8_t>(); };
    operator uint16_t() { return this->get<uint16_t>(); };
    operator uint32_t() { return this->get<uint32_t>(); };

    bool
    from_payload(Payload &p, int slot = 0, bool received_data = true);

    bool
    has_been_sent() {
        const std::lock_guard<std::mutex> lock(mutex);
        return !m_should_be_sent;
    }

    void
    callback();

    Payload
    payload(bool *should_be_sent = nullptr);

    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;

    private:
    void *var = nullptr;
    param_cb_t _cb = nullptr;
    bool m_should_be_sent;
    std::mutex mutex;
};
} // namespace CANopen
#endif
```

8.15 payload.h File Reference

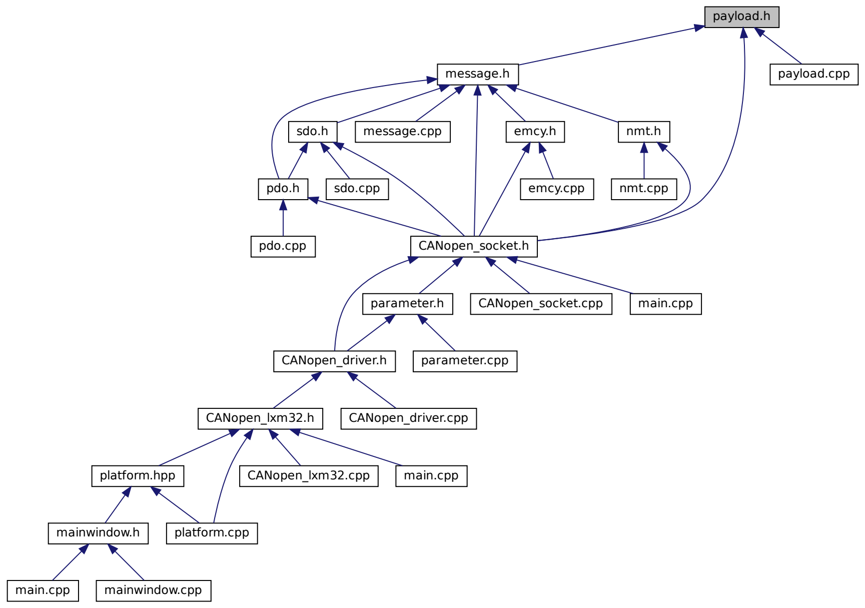
Payload of CANopen message: array of 1 to 8 bytes of data.

```
#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

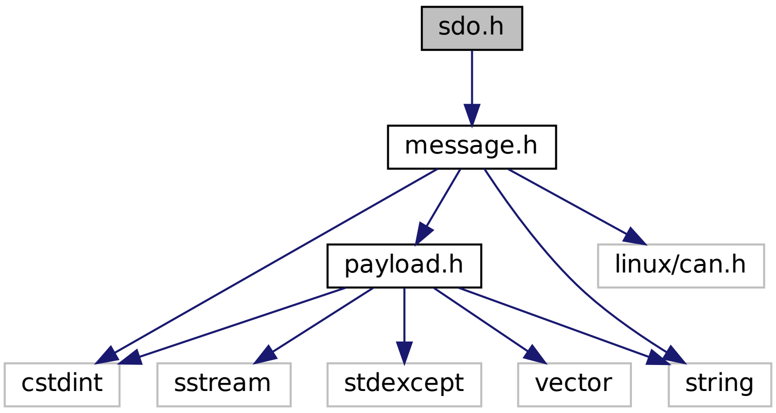
```
#ifndef _CANOPEN_PAYLOAD_H_
#define _CANOPEN_PAYLOAD_H_

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


namespace CANopen {
class Payload : public std::vector<uint8_t> {
    public:
    Payload() = default;
    Payload(const Payload &) = default;
    Payload(const std::vector<uint8_t> &other);

    template <typename T>
    Payload(T value) {
        for(int i = 0; i < sizeof(T); i++)
            push_back(*((uint8_t *)(&value) + i));
    }

    Payload &
    operator=(const Payload &) = default;

    template <typename T>
    T &
    value(unsigned begin = 0) {
        if(empty())
            throw std::runtime_error(std::string("Empty payload."));
        return *(T *)(data() + begin);
    };


    template <typename T>
    Payload &
    operator«(T &&value) {
        for(int i = 0; i < sizeof(T); i++)
            push_back(*((uint8_t *)(&value) + i));
        return *this;
    };

    Payload &
    operator«(Payload &&p) {
        for(int i = 0; i < p.size(); i++)
            push_back(*((uint8_t *)(&p[i])));
        return *this;
    };

    Payload &
    store_at(Payload &&p, int slot) {
        for(int i = this->size(); i < slot ; i++)
            this->push_back(0);
        for(int i = slot; i < slot + p.size(); i++)
        {
            if(i < this->size())
                (*this)[i]=*((uint8_t *)(&p[i]));
            else
                    this->push_back(*((uint8_t *)(&p[i-slot])));
         }
        return *this;
    };

    operator std::string() const;
};
} // namespace CANopen

std::ostream &
operator«(std::ostream &out, const CANopen::Payload &p);

#endif // _CANOPEN_PAYLOAD_H_
```

8.17 sdo.h File Reference

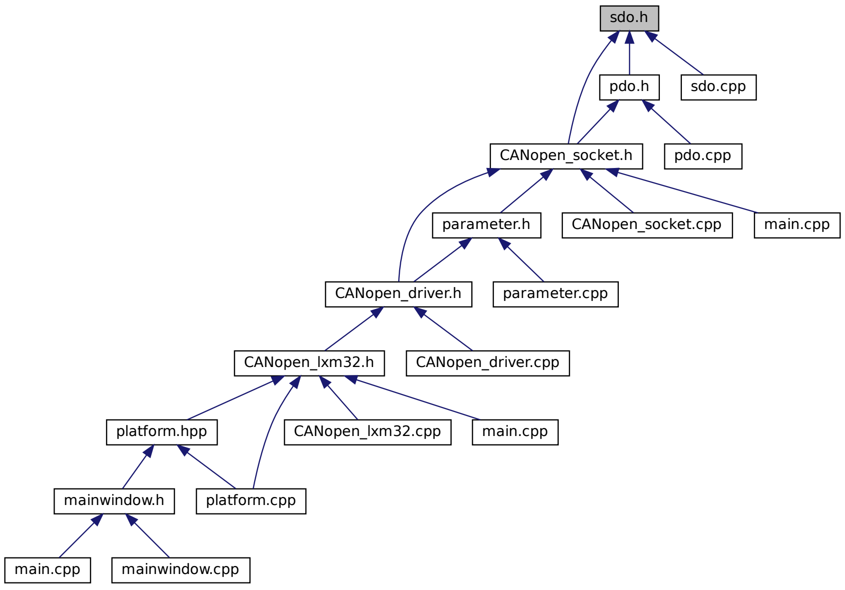
PDO message sent and received throught CANopen socket.

`#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 throught CANopen socket.

SDO message sent and received throught CANopen socket.

Author

Florian Richer & Alexis Devillard

Version

1.0

Definition in file sdo.h.

8.18 sdo.h

```
#ifndef _CANOPEN_SDO_MESSAGE_H_
#define _CANOPEN_SDO_MESSAGE_H_

#include "message.h"

namespace CANopen {
class SDOMessage : public Message {
    public:
    enum RDWR {
        Read,
        Write
    };

    enum CCS {
        SegmentDownload = 0,
        InitiateDownload = 1,
        InitiateUpload = 2,
        SegmentUpload = 3,
        AbortTransfer = 4,
        BlockUpload = 5,
        BlockDownload = 6
    };

    SDOMessage() = default;
    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;

    bool is_confirmation()
    {
        if(data[0]==0x60)
            return true;
        else
```

```
            return false;
    };

    bool is_error()
    {
        if(data[0]==0x80)
            return true;
        else
            return false;
    };

    uint8_t
    subindex() const;

    uint32_t
    index__sub() const;

    uint32_t
    id() const{
        return index__sub();
    }

    uint8_t
    size_data() const;

    Payload
    payload() const;
};

class SDOInbound : public SDOMessage {
    public:
    SDOInbound(const can_frame &other);
};

class SDOOutbound : public SDOMessage {
    public:
    SDOOutbound(uint8_t node_id, RDWR dir, uint16_t index, uint8_t subindex, Payload payload);
};

class SDOOutboundRead : public SDOOutbound {
    public:
    SDOOutboundRead(uint8_t node_id, uint16_t index, uint8_t subindex);
    SDOOutboundRead(uint8_t node_id, uint32_t index__sub);
};

class SDOOutboundWrite : public SDOOutbound {
    public:
    SDOOutboundWrite(uint8_t node_id, uint16_t index, uint8_t subindex, Payload payload);
    SDOOutboundWrite(uint8_t node_id, uint32_t index__sub, Payload payload);
};
} // namespace CANopen

#endif // _CANOPEN_SDO_MESSAGE_H_
```

Index