


CANopenNode / CANopenNode

CANopen protocol stack

 Apache-2.0 License

☆ 607 stars  373 forks

☆ Star

 Watch ▾

Code

Issues 8

Pull requests

Actions

Projects

Wiki

Security

Insights

 master ▾

...



CANopenNode ...

✓ 7 days ago



[View code](#)

README.md

CANopenNode

CANopenNode is free and open source CANopen protocol stack.

CANopen is the internationally standardized (EN 50325-4) ([CiA301](#)) higher-layer protocol for embedded control system built on top of CAN. For more information on CANopen see <http://www.can-cia.org/>

CANopenNode is written in ANSI C in object-oriented way. It runs on different microcontrollers, as standalone application or with RTOS. Linux implementation with CANopen master functionalities is included.

Variables (communication, device, custom) are ordered in CANopen Object Dictionary and are accessible from both: C code and from CANopen network.

CANopenNode homepage is <https://github.com/CANopenNode/CANopenNode>

Characteristics

CANopen

- [Object Dictionary](#) offers clear and flexible organisation of any variables.

- [NMT](#) slave to start, stop, reset device. Simple NMT master.
- [Heartbeat](#) producer/consumer error control.
- [PDO](#) linking and dynamic mapping for fast exchange of process variables from Object Dictionary.
- [SDO](#) expedited, segmented and block transfer for service access to all Object Dictionary variables.
- [SDO](#) client.
- [Emergency](#) producer/consumer.
- [Sync](#) producer/consumer.
- [Time-stamp](#) protocol producer/consumer.
- [LSS](#) master and slave, LSS fastscan.
- [CANopen gateway](#), CiA309-3 Ascii command interface for NMT master, LSS master and SDO client.
- CANopen Safety, EN 50325-5 "PDO like" communication in safety-relevant networks

Other

- [Suitable for 16-bit microcontrollers and above](#)
- [Multithreaded, real-time](#)
- [Object Dictionary editor](#)
- Non-volatile storage for Object Dictionary variables.
- [Power saving possible](#)
- [Bootloader possible](#) (for firmware update)

Documentation, support and contributions

Documentation with [Getting started](#), [LSS usage](#) and [Trace usage](#) is in `doc` directory. Code is documented in header files. Running [doxygen](#) in project base directory will produce complete html documentation. Just open `CANopenNode/doc/html/index.html` in the browser. Alternatively browse documentation [online](#).

Report issues on <https://github.com/CANopenNode/CANopenNode/issues>

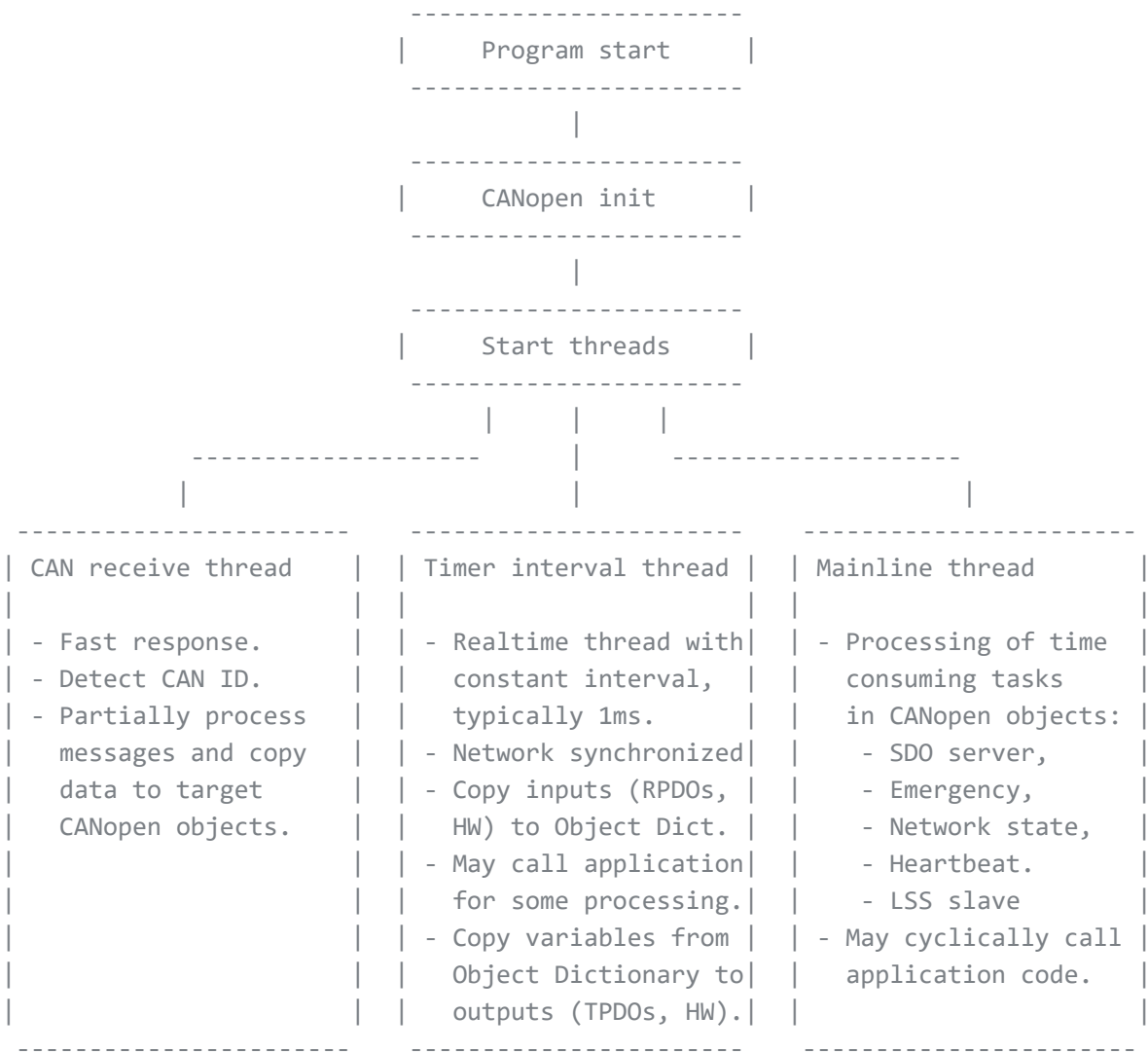
For discussion on [Slack](#) see: <https://github.com/robincornelius/libedsssharp>

Older discussion group is on Sourceforge:

<http://sourceforge.net/p/canopennode/discussion/387151/>

Contributions are welcome. Best way to contribute your code is to fork a project, modify it and then send a pull request. Some basic formatting rules should be followed: Linux style with indentation of 4 spaces. There is also a `codingStyle` file with example and a configuration file for `clang-format` tool.

Flowchart of a typical CANopenNode implementation



```
-----
| SDO client (optional) |
|                         |
| - Can be called by    |
|   external application|
| - Can read or write   |
|   any variable from   |
|   Object Dictionary   |
|   from any node in the|
|   CANopen network.    |
|                         |
|-----|
```

```
-----
| LSS Master (optional) |
|                         |
| - Can be called by    |
|   external application|
| - Can do LSS requests |
| - Can request node    |
|   enumeration         |
|                         |
|-----|
```

File structure

- **301/** - CANopen application layer and communication profile.
 - **CO_config.h** - Configuration macros for CANopenNode.
 - **CO_driver.h** - Interface between CAN hardware and CANopenNode.
 - **CO_ODinterface.h/.c** - CANopen Object Dictionary interface.
 - **CO_Emergency.h/.c** - CANopen Emergency protocol.
 - **CO_HBconsumer.h/.c** - CANopen Heartbeat consumer protocol.
 - **CO_NMT_Heartbeat.h/.c** - CANopen Network management and Heartbeat producer protocol.
 - **CO_PDO.h/.c** - CANopen Process Data Object protocol.
 - **CO_SDOclient.h/.c** - CANopen Service Data Object - client protocol (master functionality).
 - **CO_SDOserver.h/.c** - CANopen Service Data Object - server protocol.
 - **CO_SYNC.h/.c** - CANopen Synchronisation protocol (producer and consumer).
 - **CO_TIME.h/.c** - CANopen Time-stamp protocol.
 - **CO_fifo.h/.c** - Fifo buffer for SDO and gateway data transfer.
 - **crc16-ccitt.h/.c** - Calculation of CRC 16 CCITT polynomial.
- **303/** - CANopen Recommendation
 - **CO_LEDs.h/.c** - CANopen LED Indicators
- **304/** - CANopen Safety.
 - **CO_SRDO.h/.c** - CANopen Safety-relevant Data Object protocol.
 - **CO_GFC.h/.c** - CANopen Global Failsafe Command (producer and consumer).
- **305/** - CANopen layer setting services (LSS) and protocols.
 - **CO_LSS.h** - CANopen Layer Setting Services protocol (common).
 - **CO_LSSmaster.h/.c** - CANopen Layer Setting Service - master protocol.
 - **CO_LSSslave.h/.c** - CANopen Layer Setting Service - slave protocol.
- **309/** - CANopen access from other networks.
 - **CO_gateway_ascii.h/.c** - Ascii mapping: NMT master, LSS master, SDO client.
- **extra/**
 - **CO_trace.h/.c** - CANopen trace object for recording variables over time.
- **example/** - Directory with basic example, should compile on any system.
 - **CO_driver_target.h** - Example hardware definitions for CANopenNode.
 - **CO_driver_blank.c** - Example blank interface for CANopenNode.
 - **CO_OD.h/.c** - CANopen Object dictionary. Automatically generated files.
 - **main_blank.c** - Mainline and other threads - example template.
 - **Makefile** - Makefile for example.

- **IO.eds** - Standard CANopen EDS file, which may be used from CANopen configuration tool. Automatically generated file.
- **_project.xml** - XML file contains all data for CANopen Object dictionary. It is used by *Object dictionary editor* application, which generates other files.
- **_project.html** - *Object dictionary editor* launcher.
- **socketCAN/** - Directory for Linux socketCAN interface.
 - **CO_driver_target.h** - Linux socketCAN specific definitions for CANopenNode.
 - **CO_driver.c** - Interface between Linux socketCAN and CANopenNode.
 - **CO_error.h/.c** - Linux socketCAN Error handling object.
 - **CO_error_msgs.h** - Error definition strings and logging function.
 - **CO_epoll_interface.h/.c** - Helper functions for Linux epoll interface to CANopenNode.
 - **CO_OD_storage.h/.c** - Object Dictionary storage object for Linux SocketCAN.
 - **CO_main_basic.c** - Mainline for socketCAN (basic usage).
- **doc/** - Directory with documentation
 - **CHANGELOG.md** - Change Log file.
 - **deviceSupport.md** - Information about supported devices.
 - **gettingStarted.md, LSSusage.md, traceUsage.md** - Getting started and usage.
 - **objectDictionary.md** - Description of CANopen object dictionary interface.
 - **html** - Directory with documentation - must be generated by Doxygen.
- **CANopen.h/.c** - Initialization and processing of CANopen objects.
- **codingStyle** - Example of the coding style.
- **.clang-format** - Definition file for the coding style.
- **Doxyfile** - Configuration file for the documentation generator *doxygen*.
- **Makefile** - Makefile for Linux socketCAN.
- **canopend** - Executable for Linux, build with `make`.
- **LICENSE** - License.
- **README.md** - This file.

Object dictionary editor

Object Dictionary is one of the most essential parts of CANopen.

Its implementation in this version of CANopenNode (up to v2) is outdated and new version (v4 and up) contains new approach. Anyway, current version is fully operational and works well.

To customize the Object Dictionary it is necessary to use external application: [libedssharp](#). Latest pre-compiled [binaries](#) are also available. Just extract the zip file and run the `EDSEditor.exe`. In Linux it runs with `mono`, which is available by default on Ubuntu. Just set file permissions to "executable" and then execute the program.

In program, in preferences, set exporter to "CANopenNode_Legacy". Then open `example/_project.xml` or your project file for editing.

Many project file types are supported, EDS, XDD v1.0, XDD v1.1, old custom XML format. Generated project file can then be saved in XDD v1.1 file format (`xmlns="http://www.canopen.org/xml/1.1"`). Project file can also be exported to other formats, it can be used to generate documentation and CANopenNode source files for Object Dictionary.

To clone, add or delete, select object(s) and use right click. Some knowledge of CANopen is required to correctly set-up the custom Object Dictionary. Separate objects can also be inserted from another project. `example/_project.xml` can be used as a reference for the (<= v2) projects.

Please note: since rearrangement in directory structure it may be necessary to manually update `CO_OD.c` file - it must include: `301/CO_driver.h`, `CO_OD.h` and `301/CO_SD0server.h`. Latest `libedssharp` includes `CO_VERSION_MAJOR` macro, to include the correct files. For this to work, C macro `CO_VERSION_MAJOR=2` has to be added to project options (or `-DCO_VERSION_MAJOR=2` to Makefile CFLAGS).

Device support

CANopenNode can run on many different devices. Each device (or microcontroller) must have own interface to CANopenNode. CANopenNode can run with or without operating system.

It is not practical to have all device interfaces in a single project. For that reason CANopenNode project only includes interface to Linux `socketCAN`. Interfaces to other microcontrollers are in separate projects. See [deviceSupport.md](#) for list of known device interfaces.

Some details

RTR

RTR (remote transmission request) is a feature of CAN bus. Usage of RTR is not recommended for CANopen and it is not implemented in CANopenNode.

Self start

Object **0x1F80** from Object Dictionary enables the NMT slaves to start automatically or allows it to start the whole network. It is specified in DSP302-2 standard. Standard allows two values for slaves for object 0x1F80:

- Object 0x1F80, value = **0x8** - "NMT slave shall enter the NMT state Operational after the NMT state Initialization autonomously (self starting)"
- Object 0x1F80, value = **0x2** - "NMT slave shall execute the NMT service start remote node with node-ID set to 0"

Error control

When node is started (in NMT operational state), it is allowed to send or receive Process Data Objects (PDO). If Error Register (object 0x1001) is set, then NMT operational state is not allowed.

Power saving

All CANopen objects calculates next timer info for OS. Calculation is based on various timers which expire in known time. Can be used to put microcontroller into sleep and wake at the calculated time.

Change Log

See [CHANGELOG.md](#)


License

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Releases 7

 Old layout with some drivers included Latest
on Apr 27, 2020

[+ 6 releases](#)

Packages

No packages published

Contributors 21



+ 10 contributors

Languages

● C 85.6% ● HTML 14.2% ● Makefile 0.2%