What Piotr did to the CanModule in 2022 ?

Piotr P. Nikiel

Warning

1. I strongly discourage to apply presented achievements **without understanding**.

2. Changes were applied where deemed necessary by Piotr. Some changes might not be backwards compatible.

Statements presented are strongly **impersonal**: addressing technical matters no matter who the author is.

Why ?

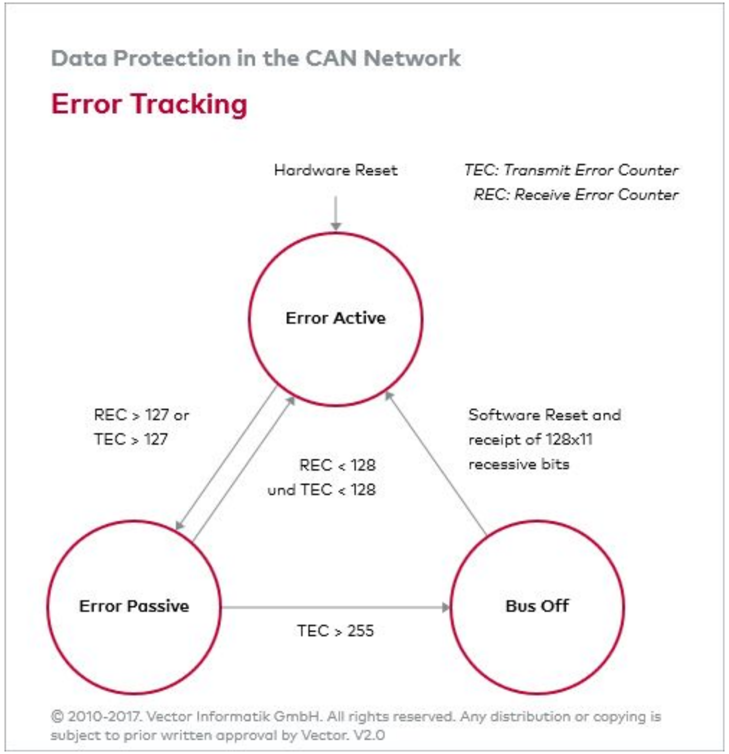
- ATLAS is paying Piotr's salary to get a **quality product** for mission-critical CANopen systems
- CANopen NG server is a **complex product** that requires **stable and clear architecture**, design and implementation discipline, good understanding of technology and product applications, and conscious decisions
  - Piotr's viewpoint: we're not yet there wrt CanModule
- it also requires technical quality (clear code, proven software techniques, …) for efficient implementation
  - Piotr's viewpoint: we're not yet there wrt CanModule
- however, the time is running out and certain activities must be happening faster
- also think about newcomers and helping them ramp up: points above apply firmly.

CanModule has docs (bravo Michael!) which can be read.

What a pro user of CanModule should find in the docs?

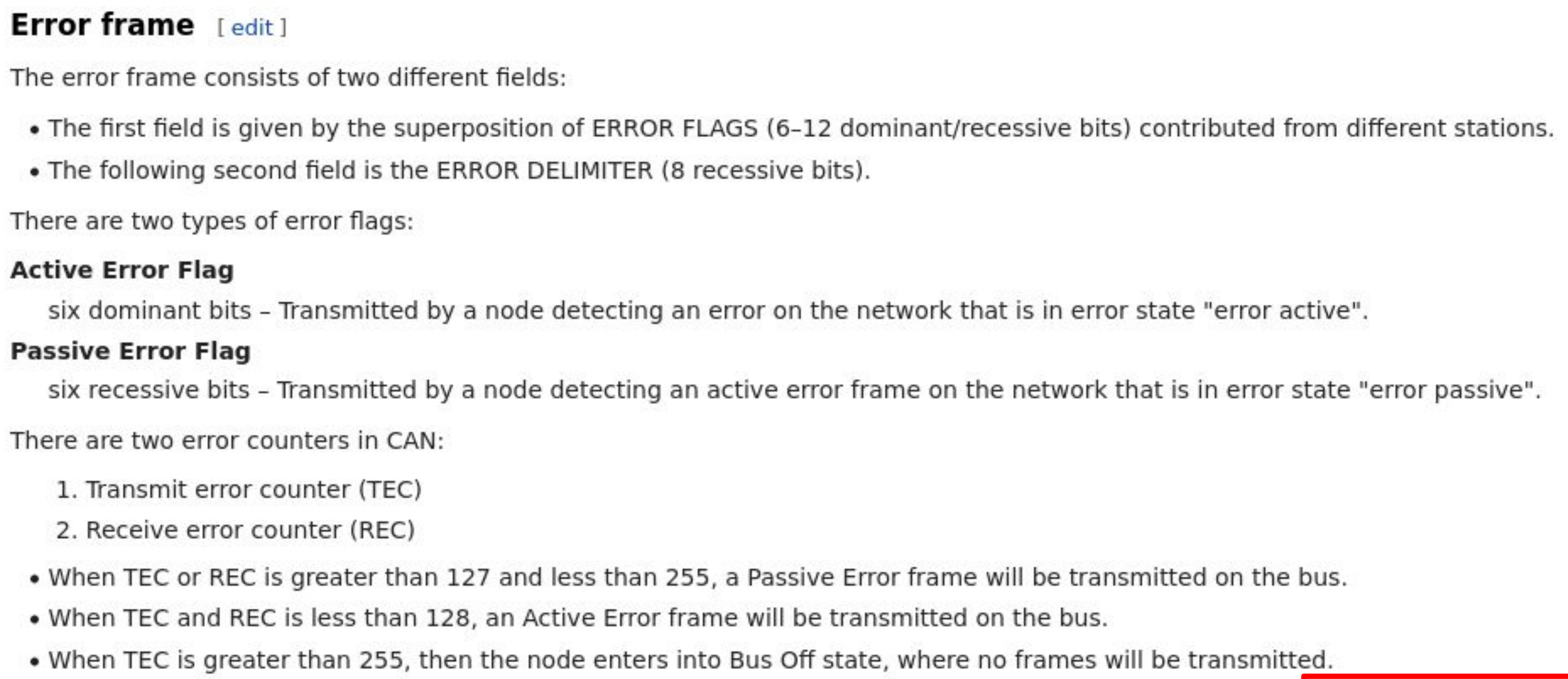
- What can block, what can't block, what may block, what by definition won't block?
- Performance considerations?
- Conditions on the callbacks?
- Are functions reentrant?
- What is the error handling model? Are exceptions used and/or supposed to be caught?
- …
- the above is not in the docs, and it seems sometimes wrong (e.g. mutually incoherent) in the present code base.

CAN state FSM


Data Protection in the CAN Network
Error Tracking
Hardware Reset
TEC: Transmit Error Counter
REC: Receive Error Counter
Error Active
REC > 127 or
TEC > 127
REC < 128
und TEC < 128
Software Reset and
receipt of 128x11
recessive bits
Error Passive
TEC > 255
Bus Off
© 2010-2017. Vector Informatik GmbH. All rights reserved. Any distribution or copying is subject to prior written approval by Vector. V2.0

The picture is from Vector Informatik GmbH, a German company, so it must be correct.

What is status, what is error and what is statistics ? (1)



Error frame [edit]

The error frame consists of two different fields:

- The first field is given by the superposition of ERROR FLAGS (6–12 dominant/recessive bits) contributed from different stations.
- The following second field is the ERROR DELIMITER (8 recessive bits).

There are two types of error flags:

Active Error Flag

six dominant bits – Transmitted by a node detecting an error on the network that is in error state "error active".

Passive Error Flag

six recessive bits – Transmitted by a node detecting an active error frame on the network that is in error state "error passive".

There are two error counters in CAN:

1. Transmit error counter (TEC)
2. Receive error counter (REC)

- When TEC or REC is greater than 127 and less than 255, a Passive Error frame will be transmitted on the bus.
- When TEC and REC is less than 128, an Active Error frame will be transmitted on the bus.
- When TEC is greater than 255, then the node enters into Bus Off state, where no frames will be transmitted.

source: wikipedia.

What is status, what is error and what is statistics ? (2)

- **error frame** is a frame (flows through the wires from possibly many sending nodes to many receiving nodes)
  - error frame is by nature a transient thing.
- **state** is one of ERROR_ACTIVE, ERROR_WARNING, ERROR_PASSIVE, BUS_OFF
  - state is by nature a thing that persists for certain time (i.e. it's a state).
  - ERROR_ACTIVE is the CAN way of saying "port is OK" (!!!)
- **an incoming error frame** is usually correlated with state however it's an approximation.
- **statistics** is Piotr's concept from 2015 which counts frames, derives rates etc.
  - **by intention** statistics is not to be confused with state and neither status! statistics object is intentionally atomic and it's read-back should not be a random thing (because it starts a new measurement period).

What model is chosen for error handling?

- this is not clear from the present code base; strong suggestion that it is "return value" model

Exceptions

- the original code was rather using integer return values to notify errors, *or nothing at all*
- uniformly converted to use standard exceptions:
    - runtime_error for plenty of possible runtime issues (port can't be opened, serious port issue, …)
    - logic_error for logic errors
    - **note** transitions into ERROR_ACTIVE or any other non-OK CAN state are considered part of life and as such not considered exceptional

Build system

- some problems, already reported in the autumn 2021, were fixed
- some serious MrProper was applied to CMakeLists

Linking

- CANopen NG links to the CanModule statically. This is a striking difference because we're passing a strict C++ boundary.

Aliasing rules of select()

- Found by using CentOs 8's gcc that the SocketCAN use of select() violates the strict aliasing (__restrict__ flags) of select() -> fixed.
- Could result in broken FDSET.

Statistics

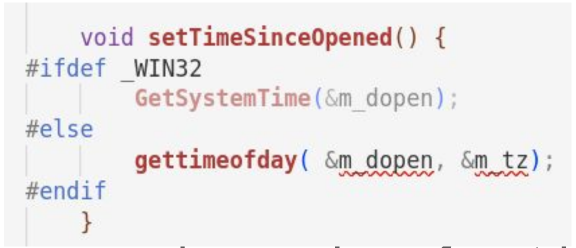
- what… why…
- getStatistics does not come for free - it invoked beginNewRun() (so, getPortStatus() can't use getStatistics)
- and also it's not the place for it.
- also, MrProper applied to #ifdefs… (OMG)

Coding quality

- … … …

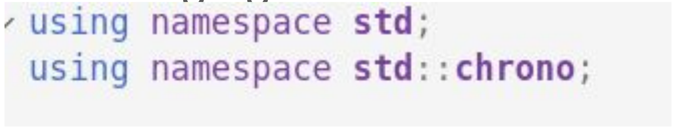
Mess and redundancy

- removed this in multiple occurrences (why … why … why ?!?!?!)



```
    void setTimeSinceOpened() {
#ifdef _WIN32
        GetSystemTime(&m_dopen);
#else
        gettimeofday( &m_dopen, &m_tz);
#endif
    }
```

- removed nanosleep for std::chrono
- could this **not** be done in the public **headers**? thank you. (BTW whole std:: was hanging on this from the Statistics…)



```
using namespace std;
using namespace std::chrono;
```

- removed non-trivial implementation from headers (there was plenty)
- lot of redundant code → applied write(), select() wrappers etc.

Undisclosed category

- Textual representation of can_state (needed in many applications, incl. CANopen)
    - now it is done – in Piotr's opinion it was a missing but really important thing
- Messages longer than 8 bytes are just sent as 8 bytes
    - OMG why
- IDs longer than 11 bits are truncated to 11 bits
    - OMG why, this does not make sense!

General summary from Piotr

- we have a developer and maintainer but we also need an architect.
- further clean-up is probably necessary.