THOR-NX-T/E LiftApp

Testing, Life Cycle & Cyber Security

Document Information:

Item:	Info/Comment:
Project Name:	Testing, Life Cycle & Security of the THOR LiftApp software
Document Owner:	Dipl.-Ing.(FH) Roy Schneider
File Name:	LiftApp_LifeCycle_Security.odt

Document History:

Name:	Version:	Reason/Comment:	Date:
Roy Schneider	1.1.6	Updated Password Security chapter.	28.11.23
Roy Schneider	1.1.7	Updated FR 7 CANopen interfaces.	07.12.23
Roy Schneider	1.1.8	Updated USB, FR3/4 and Development.	19.01.24
Roy Schneider	1.1.9	Updated introduction & Cloud chapter.	22.01.24
Roy Schneider	1.2.0	Added Fuzz-Testing and Port Scanning.	06.02.24
Roy Schneider	1.2.1	Note Update & Functional Safety	22.03.24
Roy Schneider	1.2.2	Added MQTT interface chapter.	23.04.24

Release Date: 23.04.2024

THOR Engineering GmbH

Table of Contents

1 Normative references

/CiA 417-1..4 version 2	CANopen application profile for lift control systems, Part 4: Detailed application object specification
/DIN EN 81-20:2020	Safety rules for the construction and installation of lifts
/DIN EN 60664/	Insulation coordination for equipment within low-voltage systems
/DIN EN 60950/	Information technology equipment and safety
/IEC 62443-4-1/2/	Security for industrial automation and control systems
/ISO 8102-20/	Electrical requirements for lifts, escalators and moving walks — Part 20: Cybersecurity

2 Company

Thor Engineering GmbH
Koblenzer Straße 96
53177 Bonn
Germany
E-Mail: hq@thor.engineering

https://www.thor.engineering/

Headquarters: Koblenzer Straße 96, 53177 Bonn
Amtsgericht Bonn, HRB 21892
USt-IdNr.: DE304473775
Member of the NeXt group

https://next-group.org/

3 Copyright

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this document and THOR Engineering GmbH was aware of a trademark claim, the designations have been emphasized printed.

WARNING: The information described in this document may contain errors or bugs and may not function as described. All information is subject to enhancement or upgrade for any reason including to fix bugs, add features or change performance. As with all upgrades full compatibility, although a goal, cannot be guaranteed and is in fact unlikely.

DISCLAIMER: This information is provided to you "as is" with out warranty of any kind, either express or implied. The entire risk as to the use of the information is assumed by you. THOR Engineering GmbH specifically does not make any representations or endorsements, regarding the use of, the results of, or performance of the information including but not limited to its appropriateness, accuracy, reliability, currentness, or otherwise. In no event will THOR Engineering GmbH be liable for direct, indirect, incidental, or consequential damages resulting from any defect in this information even if it has been advised of the possibility of such damages. Some laws do not allow the exclusion or limitation of implied warranties or liabilities for incidental or consequential damages, so the above limitation or exclusion may not apply.

4 Error Reports

In a complex technical manual, errors are often found after publication. When errors in this manual are found, they will be corrected in a subsequent version. Updates will be published via the company's homepage.

Bug reports can be sent to us by e-mail. Submitted reports must be clear, complete and concise. Reports must include an e-mail address and enough information, so that the bug can quickly be verified from the report. So please describe the bug and the steps that produce it.

5 Abstract

The THOR lift controllers are exciting high-performance microcomputers with superb user interface and multitasking capabilities. Their technologically advanced hardware is designed around a modern Embedded Linux® system and sophisticated hardware design. Thor's unique system software provides technicians with unparalleled power, flexibility and convenience in designing state-of-the-art lift applications.

6 Signs & Symbols

The used icons have been licensed from Axialis IconWorkshop™.

- In this document the term 'lift' is used rather than 'elevator.
- The term 'LiftApp' is used to refer to the lift controller application software.
- The term 'OS' is used to refer to the Embedded Linux® operating system.
- The term 'THOR NX-T/E' or simply 'THOR' refers to the unit made from the reference hardware and reference software package.

7 Purpose and Intended Use

The THOR NX-T/E lift controller is specially made for lift/elevator applications only. To ensure safe operation, the device shall only be operated in accordance with the instructions given.

8 Safety Information

Before commissioning, assembling and/or maintaining this unit, read the safety instructions carefully and pay extra attention to any warning label attached to the cabinet or units itself.

- Make sure that the warning labels are not hidden or damaged.
- Replace any missing or damaged warning label.

This device may only be installed and operated in conjunction with this documentation. Commissioning, installing and operation of the unit shall only be done by qualified employees having an electrical engineering qualification.

Qualified employees in the sense of the safety instructions in this documentation are further persons, who have the authorization to put devices, systems and electrical circuits, according to the standards of safety engineering, into operation.

9 General

Software does make at least half of the function of lift controller products. It has been grown as important as the hardware, itself, were it is running on. Therefore testing, documentation and maintaining of the software has grown as important as never before in mankind's technological history.

ISO 8102-20 describes the implementation of IEC 62443 - *Industrial Communication Network Security* for elevators.

9.1 Introduction

Because of the software being so capable and complex, testing and maintaining has become a major part of the lift controller's life cycle. Providing updates in regard to normatives, bug fixes and security related improvements are part of the customer service, provided after selling the unit. This also applies to the documentation, that needs permanently to be kept up-to-date.

In the same manner as hardware changes are documented in the form of schematics, software changes need to be documented as well as the software is very much defining the product's behaviour and function.

9.2 Threat model

The threat model and the risk analysis shall start with defining the attacker types and their intention. An attacker could be a local teenager who simply sees hacking a lift as an exciting task. But criminals who want to gain access to buildings/rooms by attacking a local lift are also conceivable. However, both of these types of attackers require local access, the damage is limited to the facility itself and this is more likely to be considered vandalism (cybervandalism). These attackers put themselves at high personal risk and it is more the responsibility of local security authorities to track them down. A real threat, which should be considered are “*crawlers*”, which are automatic scripts, that are scanning around for open network ports to step in and create damage. When we talk about Cybersecurity, it is more widespread attacks on the transport infrastructure, that should be considered. In other words, attacks that are carried out remotely by script kiddies or hackers for personal pride, blackmailing or to weaken a facility by putting the lifts out of operation.

Attacks via the remote maintenance connections, i.e. the connections between the lifts and a cloud, are the most suitable for this, as they allow the attacker to be at a safe distance from the event and enable him/her to operate covertly. The relationship between effort and achievable damage makes enough sense for the attacker to carry

out an attack in this way.

If we talk about the ISO 8102-20 for lifts, we are in need to look at the domain '*Essential*', that requires **SL(T)2**, which defines:

“**Protection against intentional misuse by simple means with few resources, general skills and low motivation.**”

9.3 Data minimization

In general, we at Thor Engineering only store the minimum of required data necessary for planning, development, testing, construction and e-mail connectivity. Data that have no direct relation to the processes at Thor Engineering is not stored non-volatile. Personal Data stored may include Name, Address, Titles, E-Mail and phone numbers as well as the kind of business relation and position inside the company. This also includes bug reports, related to software or hardware issues, that may also include the contact person as well as the technical data recorded to reproduce the issue.

9.4 Code Analysis and Automatic Documentation

The developers are supported by code analysis tools, like the ones provided by Eclipse's CDT-*Code-Analysis* and *CPPCheck*, to prevent mistakes in the process of writing the code. An example would be using wrong format specifiers in string formatting code or using unattended implicit casts or leaving local variables, non-initialized.

All functions or methods have a comment head, that confirms to DoxyGen's syntax standard. That makes it possible for other developers to get an overview about the classes or structures, using that tool.

The automatic checking and documentation is just a helper and **not** a substitute for manual checking and documentation. But we have learned that issues remarked by the documentation tool in the first place, like naming differences in declarations, can help tracking down issues early.

For more information see the chapter 'Code Analysis Tools' on page 61.

10 Product requirements

The IEC-62443 defines four main regions.

Region/Range	Description	Examples
Essential	Essential functions that are vital for the availability of the system	Call entry, position, direction detection, door movements
Safety	SIL related safety functions	Door safety bridging
Alarm	Emergency Functions	Lift Phone calls, Emergency Light
Other	Other functions	Infotainment, Music

Each functional region has a minimum security level (SL-T), that shall be achieved. IEC-62443 defines basically five levels.

Requirement	Alarm	Essential	Safety
FR 1 - Identification and authentication	2	2	3
FR 2 - Use control	1	2	2
FR 3 - System integrity	1	2	2
FR 4 - Data confidentiality	1	2	2
FR 5 - Restricted data flow	1	1	1
FR 6 - Timely response to events	1	1	1
FR 7 - Resource availability	1	2	2

The lift controller itself is in the *essential* functional region. The built-in safety circuit is an electromechanical circuit that is a component of its own and integrated as a separate assembly. Its task of checking the synchronization of the zone channels is carried out without any software. The control unit monitors the function after every trip, but this is captures by the *essential* range, as is the contactor monitoring. Only functional safety and no SIL level is defined for both.

For the lift controller in question, the SL-T would equal to { 2-2-2-2-1-1-2 }.

10.1 FR1

Identification and authentication

If browsing through the menus you find a yellow or red overlay icon on menu items. These indicate that you will have to enter a 'Service' (yellow) or 'Setup' (red) password in order to alter their value/setting.

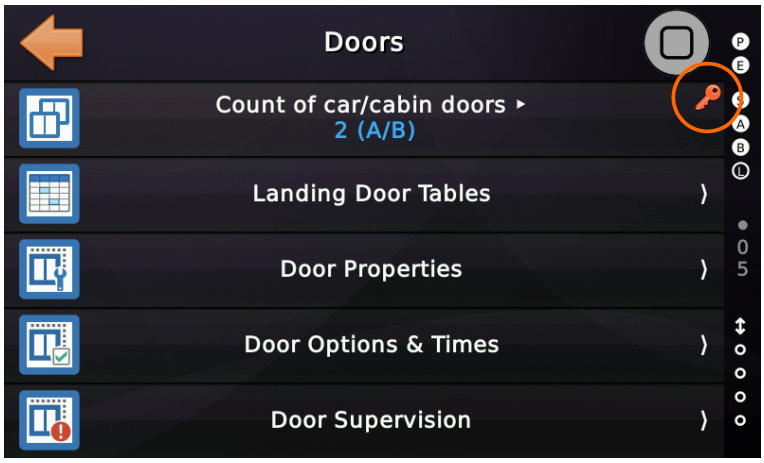


Figure 1: Menu item requiring setup password privilege

Menu item requiring 'Service' password privilege.

Menu item requiring 'Setup' password privilege.

The manual clearly states, that the installing company in agreement with the owner, have to protect the important settings in the Lift Controller by setting a proper 'Setup'-Password. You can do that via the user interface, following '*System Menu → Security*'. The password should be at least 6 characters long.

Please refer to the chapter 'Password Security' on page 35 in this document for more details.

The system does not store passwords in its non-volatile memory. Instead only the salted SHA is stored in order to make password validation possible.

10.2 FR2

Use control

The usage of Passwords is logged in the '*Logbook*' (Event Logger). Any parameter changes are recorded in the '*Parameter Change Log* (Parameter Logger). The Parameter Change Log is a logging file system, storing all changes that had been made to the lift's parameters over time. It stores the last 256 parameter changes locally and non-volatile on the controller board.

The graphical visualization can be found following 'System Menu' → 'Security' → 'Lift Parameter Change Log'.

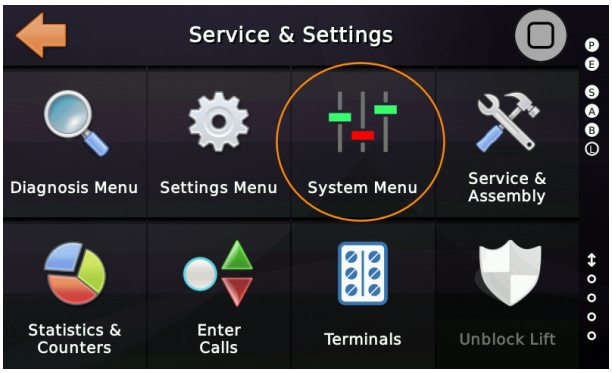


Figure 2: Lift Parameter Change Log found under System Menu → Security

The log file stores:

- What parameter had been changed (name/help text).
- At which date/time the parameter had been changed.
- How the parameter had been changed.
  - locally via the user interface
  - via the bus system
  - remotely (if possible) via the cloud solution
- What kind of privilege had been required to change the parameter (setup/service/none) privilege.
- The old and the new value(s) of the parameter, to put the parameter change in a context.

Please refer to the chapter 'Lift Parameter Change Log' on page 37 in this document for more details.

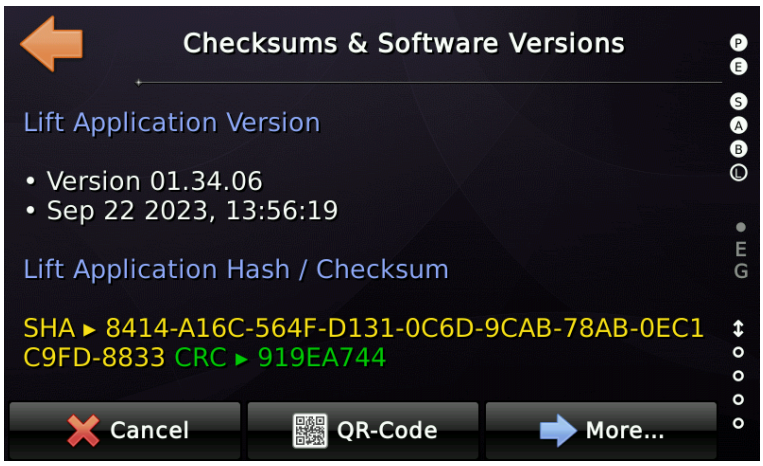
10.3 FR 3

System integrity

Application

At every application start the checksum (CRC32) of the binary is checked to make sure, it has not changed unattended. The file system used also uses checksums to detect defective sectors and not to pass on data that is not valid. When installing an update of the application, a **SHA1** of the update binary is send independently from the binary itself to make it possible for the technician to check, if the file has been tampered or manipulated. To make that process easy, the SHA is calculated by the running application and presented to the technician, that now is in charge to check it against the one he got via mail. Additionally to that the notified body is regularly checking the checksum to make sure it has not been altered, since the last check.

Note: The checksum of the update binary and the checksum later shown at the screen for the notified body may vary as the binary is '*stamped*' when being installed, making it impossible to run it on another hardware.



Please refer to the chapter 'Checksums & Software Version' on page 51 in this document for more details.

Safety Chain Sensing

The system integrity also touches sensing the safety chain (ISO 8102-20 A.3.9.3). The safety chain is sensed in hardware by means of a certified sensing circuitry, that fulfill the EN81-20/50 normatives. The circuitry has been validated and certified by the Lift Instituut as a notified body. The examination covered a check whether compliance with the Lift Directive 2014/33/EU is met based on the harmonized product standards

EN 81-20 and EN 81-50.

What is important during the test is that the query circuit is non-reactive and therefore cannot influence the state of the safety circuit itself (backwards). Please refer to the chapter 'Safety Chain Sensing' on page 43 in this document for more details.

10.4 FR 4

Data confidentiality

Application

The lift controller is storing events, related to the lifts operation in order to make fault tracking easier and increase the overall availability of the lift installation. Together with recorded statistical data about trips, direction changes, contactor operations, re-levelling and door movements, the wear of components can be evaluated by the maintenance company. Additionally parameter changes are recorded by the lift controller. Personal data, like 'who' has changed a parameter or 'who' has used the lift is not recorded by any means. This also applies to the optional mass storage logging, that can optionally be used for detailed fault tracking.

The connected Cloud solution is featuring encryption (TLS) and certificate based handshakes in order to make sure, that no data are leaked into the wrong hands.

But anyway it is a requirement that the users of the product or the cloud solution makes sure, that passwords or credentials are not leaked out. This also applies to employees who leave the company.

If the slightest suspect of a leakage exists, it is in their responsibility to act and change passwords and credentials.

We recommend that there is a designated person in the maintenance company who centrally manages passwords and credentials. It is also important that only the lowest level of access is passed on to employees and that not everyone has the role of an administrator!

The system does not store SETUP/SERVICE passwords in its non-volatile memory. Instead only the salted SHA is stored in order to make SETUP/SERVICE password validation possible.

Please refer to the chapter 'NeXt® Cloud Security' on page 44 as well as chapter 'Password Security' on page 35 in this document for more details.

Safety Chain Sensing

The data confidentiality also touches sensing the safety chain (ISO 8102-20 A.3.9.3). The safety chain is sensed in hardware by means of a certified sensing circuitry, that fulfills the EN81-20/50 normatives. The state of the safety chain **is not** received/read by means of any bus system. It is directly sensed by the lift controller own hardware by means of on-board components. Messages that reflect the safety chain signals are not accepted being received via the CANopen system, even that CANopen CiA-417

defines such messages. The lift controller does reflect the current electrical state of the safety chain back via bus messages for diagnostic purposes. Please refer to the chapter 'Safety Chain Sensing' on page 43 in this document for more details.

10.5 FR 5

Restricted data flow

The lift controller exchanges control and status words with the peripherals via the CANopen bus system, according to the CiA-417 Profile for Lifts. Process data flow with the cloud solutions is done accordingly to the User Agreements of the Cloud solution.

It is important that the user of the Cloud solution, usually the Maintenance Company, has agreed that kind of remote service with the owner of the lift, as the owner of the lift is not only owning the installation but also the data, that it is producing.

Please refer to the chapter 'NeXt® Cloud Security' on page 44 as well as chapter 'Password Security' on page 35 in this document for more details.

10.6 FR 6

Timely response to events

Organizational Side

In the event of an incident, it is important that we have a structured procedure in place to respond in a timely manner and, if necessary, inform our customers promptly. To do this, follow the chapter Incident/Issue Reporting on page 29. Feedback from the field is important for the qualitative further development of the product.

Technical Side

On the technical side, the lift controller itself has a build-in **hardware watchdog** that ensures that the required program parts are processed correctly. If this is not the case, the lift is performing a full-stop and all outputs are switched off and the system restarts safely. A system restart is recorded in the system's event memory.

The hardware watchdog can be tested via the Tests menu.

Additionally, there are time-based warnings/faults that are generated when:

- The door zone signal is reset belated, when the car is leaving the floor.
- The Advance Door Opening operation takes an unusually long time.
- Door opening and door closing are unusual slow or even fail.
- The time behavior of the drive is unusual, see Start Control, Run Timer Supervision, Deceleration Control, Re-levelling and Pawl Device Supervision.
- The cabin movement fails while driving for no apparent reason, for example due to a failure of the position encoder belt.
- The status words of the position sensor, the drive and the car IO-panel are monitored for their timing, when being transmitted via the bus system.

CANopen CiA-417 is featuring the producer-consumer heartbeat model, with which it is constantly checked that the modules (nodes), like the Drive Unit, the Positioning

Unit and the Car Weighing Device do not 'log off' from the bus. The nodes do monitor the heartbeat of the controller and the controller monitors the heartbeat of them. If IO-units disappear from the bus, their inputs and outputs are set to off.

10.7 FR 7

Resource availability

Energy/System Power

The system is directly monitoring the mains voltage (230V AC or 120 V AC), that the power supply is using to generate the 24V DC bus. That means that the controller knows that the 24V will drop, before it actually does. Additionally the supply voltage of the internal circuitry is monitored by a dedicated power controller (PMIC), that generate internal voltages from the 24V DC.

The voltage of the Car Light supply is also directly monitored via a 230V AC / 120V AC input. Additionally the emergency light battery charger has usually a fault output, that is connected to the lift controller as well.

Safety Chain

The safety chain is not transmitted via the bus system. It is always connected classically in hardware, connected directly onto the certified safety board. We do NOT support encoder system or other peripherals, that transmit the safety chain over the bus system. Those inputs can not be reprogrammed.

Please refer to the chapter 'Safety Chain Sensing' on page 43 for more details.

CANBus

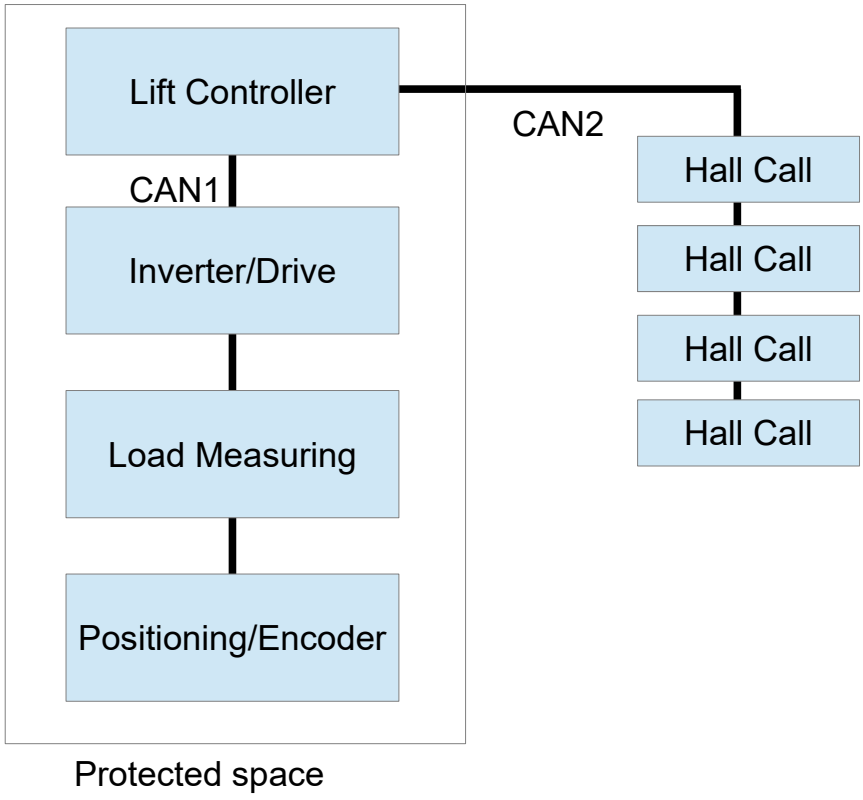
The CANopen bus status words of peripherals, like the position encoder or the drive unit are parameterized to be send on a cyclic base, making it possible to detect failures or delayed transmission properly. Additionally timeouts for control words (commands) have been useful implemented and tested. When a bus error or bus off situation would happen, the lift controller would come to a full stop and enter a secure state. The doors are kept closed outside the door zone.

The CANbus is featuring identifiers (COB-IDs) for the messages, that distinguishes and determines the priority (importance) in case of concurrence of messages in the network. The manufactures of CANopen components must take part in the regularly Plug-Fest at CiA HQ in Nuremberg in order to ensure the interoperability of the components and that they apply to the bus standard. Otherwise they are not allowed

to feature the CANopen logo for their products. **We recommend only to use components, that proudly and rightly bear the CANopen Lift logo.**

The lift controller itself is installed regarding the EN81-20/50 in a special and key-locked room, that shall prevent unattended access to the hardware and equipment, usually reffed to as the '*Machine Room*'. The CANbus interface and the corresponding wiring are in those closed and safe spaces, were even every screw, used for car panels need to be vandal-proof. Anyhow the second CAN, so CAN2, that is used for the landing calls might be more exposed as it ends up in the hall call panels at the landings. Therefore all bus components, being related to positioning, driving, load measuring and car I/O are connected to CAN1 only.

The CAN1 and CAN2 interfaces are galvanically and logically separated. The CAN1 interface goes from the controller cabinet straight through the travelling cable, passing the hoistway into the car, were the panels are hold with vandal-proof elements.



No status words from inverter/brake, position encoder, PSU or doors are accepted on CAN2. Control words cannot be sent to these modules either. There is no transparent routing of any messages between CAN1 and CAN2.

Nevertheless, the availability of a lift system can be negatively influenced by opening

external landing call panels (LOB), for example by short-circuiting the call acknowledgment lamp to ground.

Web server and Cloud Interface

While the build-in web server is only meant to be used temporarily for Repair in a local network, the Cloud solution is intended to be used remotely by means of a n internet connection. Both interfaces might be attacked. For the build-in web server, a local attack by some script kiddy might be a scenario and for the Cloud interface automatic crawlers are a real threat.

MQTT-Interface

MQTT stands for "*Message Queuing Telemetry Transport*". It is an open messaging protocol. It is usually used for M2M (machine-to-machine communication), such as the Internet of Things. The interface is switched off by factory defaults and can only be activated directly on the device (i.e. not remotely). Typically, this interface is used with our lift controller for AVG's (Automated Guided Vehicles) in factories. We recommend using the encrypted TLS Web Socket MQTT mode. Only in secured networks, such as in factories or hospital environments, where the building automation network is separate and inaccessible from outside, can a lower connection mode be selected. In principle, it is not possible to change parameters or access elements of the lift via this logical interface. However, it is possible to give calls and send door open/close button requests. This can affect the availability of the system.

Fuzz-Testing

To make sure, that our interfaces are rock-solid and hardened enough to fight back, attacks, executed by means of malformed HTTP-headers, malformed HTTP-bodies, data floating via the Web-Socket or simply fuzzing the JSON-REST-API with non-sense, in order to break the system and render the lift non-operational, we have implemented a set of Fuzzing and Penetration tests of our own, using POSTMAN as the working horse. POSTMAN is a wide spread tool, used for performing Fuzzing tests to network interfaces.

https://www.postman.com/product/what-is-postman/

To make sure, the Fuzz, used for testing the interface input does contain all thinkable data noise, we update our '*List of Naughty Strings*' from the "`minimaxir/big-list-of-naughty-strings`" repository and use a simple shell script to convert those to a .csv file and use that as input for POSTMAN.

It is especially important to use those 'Naughty Strings' for input properties of the JSON-REST-API and web interface to make sure, that the parsing code at the LiftApp

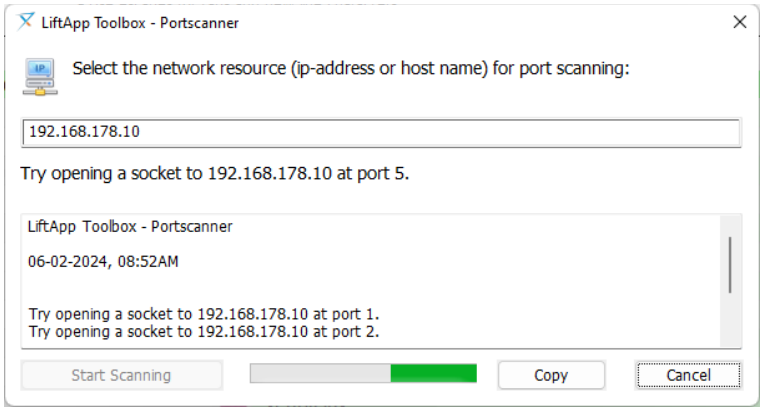
end will handle all unexpected data and does not fail or do unattended or unwanted procedures.

The POSTMAN tool is also used to check if it is possible to create issues by simply spamming the exposed network interfaces with data noise and do port scans.

The POSTMAN tool is also used to document these tests as it keeps a history of test execution.

Port Scanning

To check for open ports, we use our own port scanner utility, which we have also published as part of our LiftApp Toolbox for everyone to use for free.

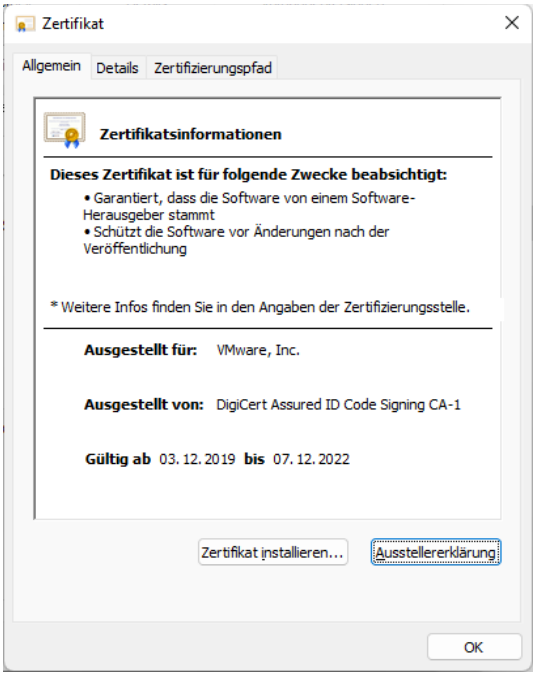


► By factory default our lifts do not have any port open!

11 Development Environment

11.1 Local Development Machines

To minimize the risk of an attack or intrusion, the development of the lift application is done within a virtual machine, that is exclusively used to write, compile and link/build the application. This virtual machine is not used for web browsing, e-mails or any other online activity, other than fetching updates, setting the clock and getting safe time stamps for digital signing. Linux® based systems are used as development environments for the lift application.



The virtual machine player, that runs the Linux® used for bit-baking is digitally signed, making sure that this is the correct released binary of the manufacturer.

The source code is stored in a repository. The password and private key used to check-in and check-out and for administration, is known to the developer only with a single back-up copy on a physical USB-stick, hold by the CEO of the company. The key files are **not** part of the source repository for obvious reasons.

The outer host machine is used to communicate with the outside world. This host might be a Windows® machine. Here the developer will do browsing and e-mail or instant messaging communication. These host machine are protected by anti-virus software, monitoring the file and network communication.

All local PC's, laptop and workstations at Thor Engineering are equipped with an up-to-date virus scanner application, that locally checks the file and network I/O. It is also responsible for scanning incoming e-mails for possible threads, based on embedded scripts and hostile executable attachments.

All local PC's, laptop and workstations at Thor Engineering are equipped with a local firewall in order to minimize the risk of an intrusion from within the network.

For the Linux® and Windows® machines, available security related updates are fetched automatically on daily basis.

The development machines are turned off in the evening and over the weekends by means of being physically cut of from the mains. The feature '*Power on over Network*' is also disabled in the BIOS.

11.2 Data Protection on Routers, Switches and other Network Equipment

The firmware of the network router and manageable switches are updated as soon as new firmware is available. The external administration (WAN) console of such devices is deactivated.

- We do not use anonymous file shares or non-encrypted FTP via our networks.
- All wireless networks are secured by an 16 digit long key using WPA2.
- Guests use the '*Guest*' wireless account only, that grand no access to our local computer systems or network file storage, that may contain confidential data.

▸ This also applies to '*Guest Developers*' attending meetings at Thor Engineering!

11.3 Data Protection on involved NAS Systems

The firmware of the NAS-systems are updated as soon as new firmware is available. The external administration (WAN) console is deactivated. Accessing the NAS is secured by username and a password that complies with Thor's password policy.

11.4 Data Protection when generating Software Releases

Software that is released by Thor Engineering is in the process of compiling/linking checked for malware. Software released is either digitally signed (*Authenticode Certificate*) or (*Embedded Software*) provided with an extra SHA1 hash, to check/ensure afterwards, that the software binary is till intact and authentic. Our partner for Authenticode Certificates is the Symantec Corporation. We prefer to develop in Virtual Linux® Environments, that are only used for editing source code, compiling and linking. In those virtual machines, there is no e-mail communication and no web browser usage.

11.5 Data Protection regarding E-Mail and external File-Storage

All e-mail accounts used at Thor Engineering are accessed only featuring TLS encryption. Our trusted e-mail provider (One.com) has for its part, to ensure compliance with the GDPR.

https://help.one.com/hc/en-us/articles/360000253649-How-does-One-com-comply-with-the-GDPR-

Our provider for External File Storage is Dropbox Business which has indicated to

fulfill the GDPR on its own side being certified by ISO 27018 the internationally standard for practices in cloud privacy and data protection.

https://www.dropbox.com/en_GB/security/gdpr

11.6 Data Protection inside Thor's Lift Cloud Interface

Thor's Cloud interface is used to connect lift controllers with a cloud, mainly used for predictive maintenance. Personal data is not the main focus of this solution. Anyhow the lifts are owned by operating companies that have indirect personal data involved, like the name of the owner or maintenance technician. To make sure, there is no unattended access to the lift via the cloud solution, that could leave to the lift being attacked or data stolen, Thor's cloud solution features **TLS encryption** and **Certificate based server authentication** by default & by design without any compromises. The cloud provider, which in turn enables endpoint access, must ensure that access data does not fall into the wrong hands by complying with standards such as ISO 27017 “Information Security in Cloud Computing” and ISO 27018 “Data Protection for Cloud Services”.

11.7 Google's and DeepL's Cloud API solutions

Currently Thor Engineering is featuring the '*Translation API*' of the named cloud services, processing no personal data at all. These service are currently only used by Thor's **development host machines** and **not by the lift controller application**. When using these services only the strings to be translated, the resulting string and our API key for accounting is used.

11.8 Staying Up-To-Date about Vulnerabilities

We do a hybrid approach here. We monitor the dedicated industry news and vendor sources for our applications, that we are using on our systems. Additionally we regularly browse the BitDefender Lab News and the DigiCert News as we use their products (Virus Scanner and Code Signing) in house on our machines.

https://www.bitdefender.com/blog/labs

https://www.digicert.com/news

Additionally we regularly communicate with the web developers of MASORA AG (Switzerland) and Code Ink (Netherlands), that implement their versions of the Next® Cloud API. By doing so we make sure, that we share the news, knowledge and tech related gossip.

National Vulnerability Database (NVD)

We query the NVD, which is the National Vulnerability Database (NVD). It is the U.S. government's repository of standards-based vulnerability management data. They use a CVSS metric to measure the security issues and make it more transparent for us, how we shall react.

https://nvd.nist.gov/

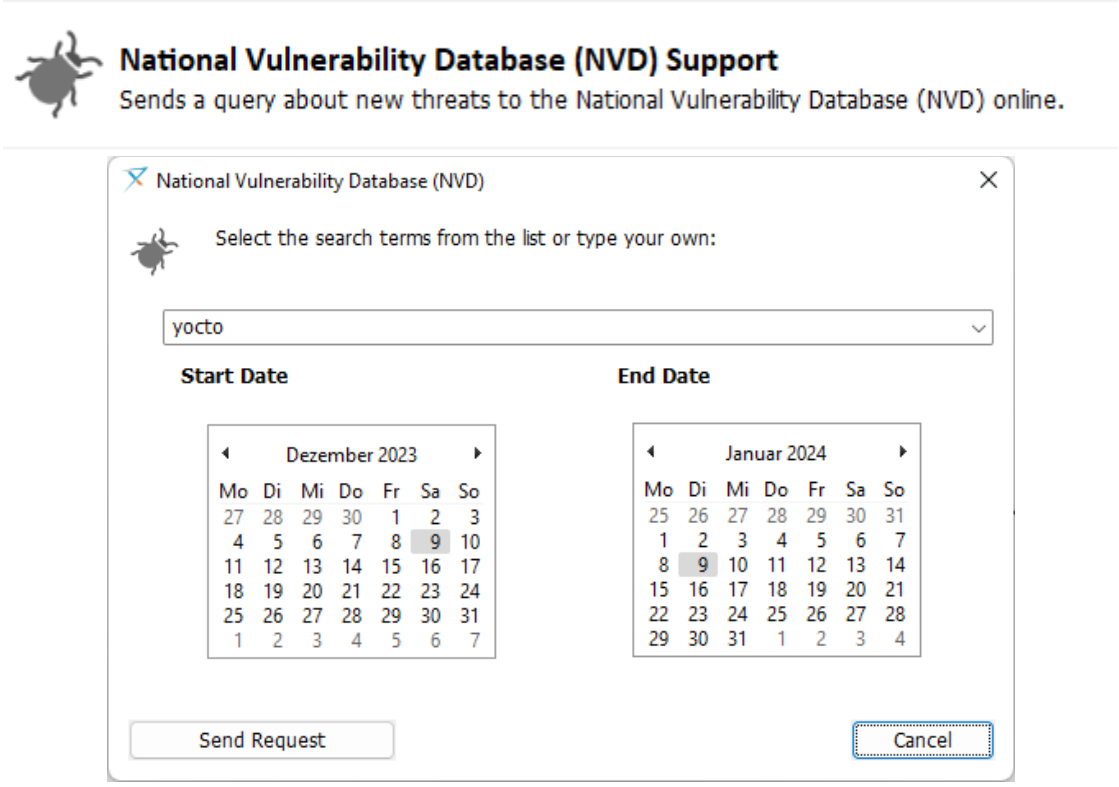
To make that more efficient, we requested an API key at the NVD, to use their REST API in order to get Linux® and Windows® related security news via REST-API requests, that are send directly via the Browser.

Example to fetch Linux based NVD records for the first December week 2023 via the browser:

https://services.nvd.nist.gov/rest/json/cves/2.0/?noRejected&pubStartDate=2023-12-01T00:00:00.000&pubEndDate=2023-12-08T00:00:00.000&keywordSearch=Linux

The result JSON-based result has then to be checked for applying to our systems.

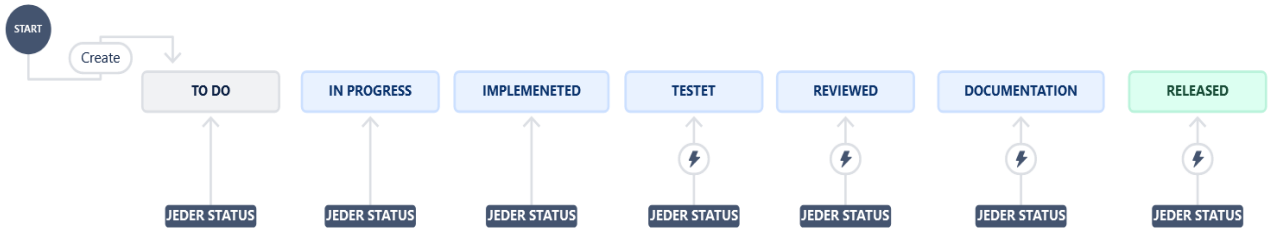
To simply the process and make it a Click-Once feature, we have created an utility, that is built into our LiftApp Toolbox:



For the Linux® and Windows® machines, available updates are fetched automatically on daily basis via the dedicated update daemon or service.

11.9 Workflow

In order to make it clear for the team to see what the current tasks are, what the status of each task is and to ensure that a task can only be moved to the final 'Release' state, when it has passed the required states before, like TESTED AND REVIEWED, the THOR Team is using a KAN Board model. The work flow can be simplified like so:



The status RELEASED can only be reached, when the task has passed the TESTED, DOCUMENTATION and the REVIEWED state before. The following RACI diagram shows, who in that work flow is responsible for what.

	Lars Gustafsson	Roy Schneider	Thomas Reul
Customer Contact (START)			R
Fetching Details (CREATE)		I	R
Checking/Validating (TO DO)		I	R
Creating Test Case (IMPLEMENTATION)		R	I
Fixing Issue (IMPLEMENTATION)		R	I
Testing Solution (TESTING / REVIEW)		I	R
Update Documentation (REVIEW)	I	R	C
Notifying Customer (RELEASE)	A		R

R – Responsible
A – Accountable
C – Consulted
I – Informed

The THOR team uses a KAN board to map the tasks to be completed and to track which current task or issue is at which stage of development or documentation.

That makes sure that software changes are not published, before they had been finalized.

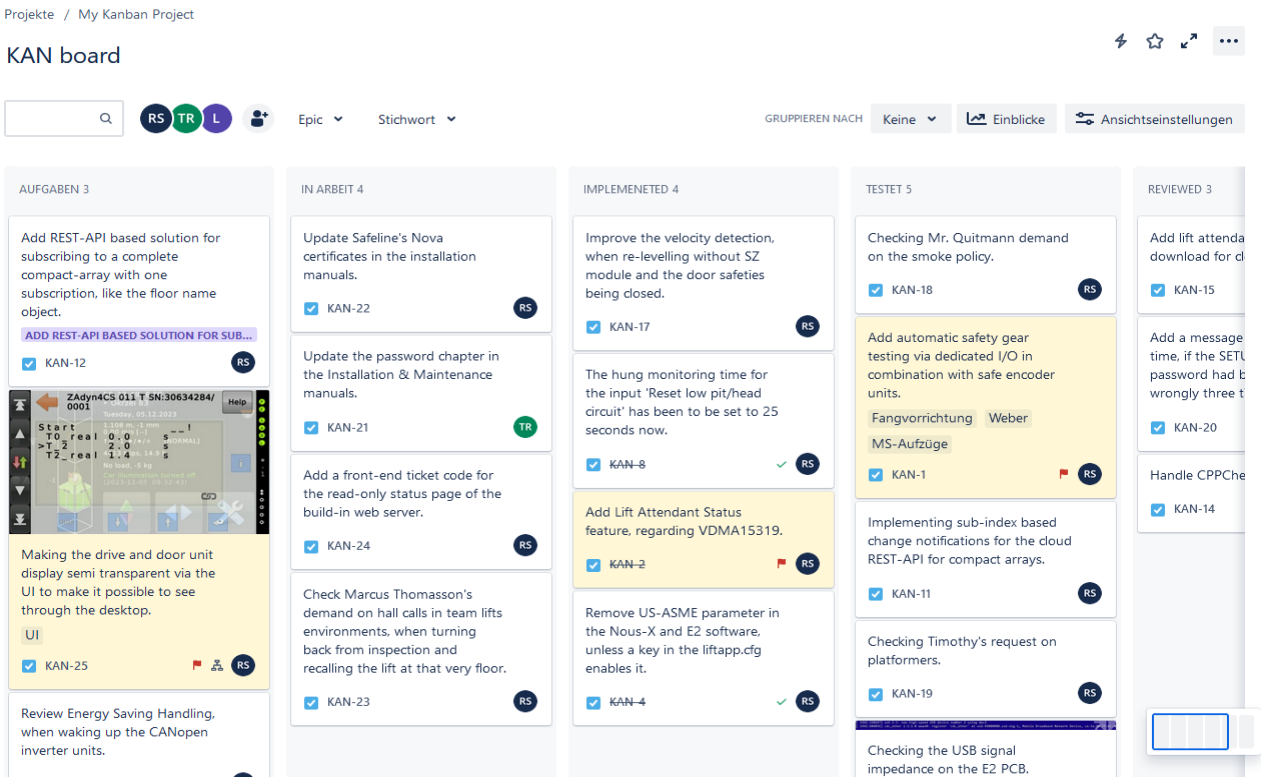


Figure 3: Jira Workflow - KAN Board

Kanban is a method in which the existing process is improved in small steps rather than in big leaps. By making small changes, the risk of errors is reduced for each release. The combination of the KAN board, in which each task is stored with its current status and associated data and documents, with a Gantt chart that shows the timeline with which the team works, provides a good overview and planning options.

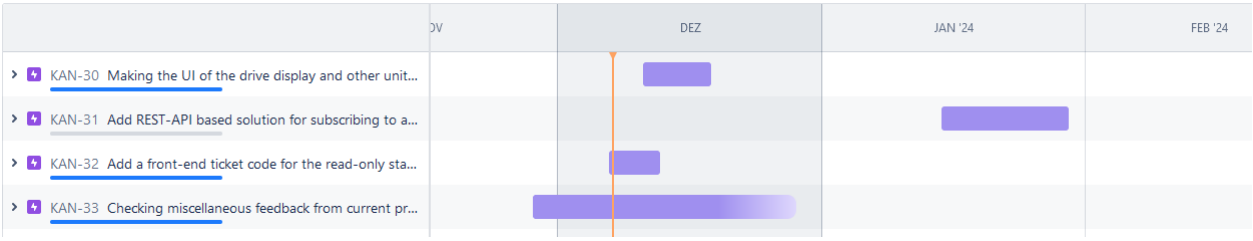


Figure 4: Jira Workflow - Gantt Diagram

This ensures that tasks do not collide and customers can be informed early, if there are delays, for example due to developers becoming ill.

12 Incident/Issue Reporting

In the case of an incident, it is vital that we have a structured procedure of reacting. Bug/error/security/issue/incident reports can be sent to us by e-mail, directly to:

hq@thor.engineering

Submitted reports must be clear, complete and concise. Reports must include an name, Lift-/Controller Identification Number, e-mail address and enough information, so that the bug can quickly be verified from the report. So, please describe the issue and the steps that produce it as complete as possible.

R – Responsible
A – Accountable
C – Consulted
I – Informed

	Lars Gustafsson	Roy Schneider	Thomas Reul
Customer Contact (START)	I*		R
Fetching Details (CREATE)		I	R
Checking/Validating (TO DO)		I	R
Creating Test Case (IMPLEMENTATION)		R	I
Fixing Issue (IMPLEMENTATION)		R	I
Testing Solution (TESTING / REVIEW)		I	R
Update Documentation (REVIEW)	I	R	C
Notifying Customer (RELEASE)	A		R

*) If the issue is security related.

13 Updating & Maintaining the Manuals

There are three most vital manuals for the Lift Controller Software:

- The Software Reference Manual with its round about 600 pages, that covers every function included in the lift controller. **This manual is updated with every software version released in English and German language.** The other translations, like French or Netherlands are translated and therefore updated on demand but at least twice a year.

- The Maintenance & Assembly Manual is updated, if the new function or feature is directly affecting the installation or maintenance of the lift, but at least twice a year.

- The small Hardware Brochure is only updated if the hardware is updated or if errors or issue had been found.

Conclusion:

The most vital document is the Software Reference Manual as this is the reference from which all other manuals and documentation are derived from. It has to be updated with every new software release and actually it reflects always for which software release the manual is valid, as shown in this screenshot:

Document History:

Name	Version	Reason/Comment	Date	LiftApp
rsc	2.0.2	Fixed MODbus register address.	26.08.21	1.25.14
rsc	2.0.3	Added docking service operation	07.09.21	1.25.18
rsc	2.0.4	Reworked date & time related MODbus registers.	01.10.21	1.26.02
rsc	2.0.4	Added note about finger protector timeout.	11.10.21	1.26.04
rsc	2.0.5	Added new MODbus registers.	20.10.21	1.26.06
rsc	2.0.6	Added hydraulic jack re-synchronization.	26.10.21	1.26.08
rsc	2.0.7	Added 'Load Time Operation' chapter.	08.11.21	1.27.02
rsc	2.0.8	Updated chapters about Em.Power and Stopover.	17.11.21	1.27.04

New software means an updated manual!

14 Versioning

In general all versions released to customers do have a manually created entry in the document *'LiftApp_Version_History.pdf'*. The customer can by using the document always check which functions, options and features had been added, altered or fixed.

An example entry can be found here. The entry shall be created in a form and wording, that can be understood by the technician of the end customer. It shall include the version number, the date and which functions had been added, improved and/or bug fixed. The Secure Hash of the release is not part of that document. The hash is send with the release notification to the customer directly. It is recommend to use the same icons as in the user interface, when referring to the functions in question, making it visually easier for the customer to get the context of the entry.

The entry shall highlight possible safety issues or risks.

14.1 Example

V1.20.14 (03-2020)

New features/functions

- Added basic support for the upcoming smart power supply units. This will be for sure improved in the future as we want to have statistical data and such. Currently the units are recognized, turned operational and a simple status page is available. More will follow...

Improvements

- When using car call code input via the buttons in the panel in the cabin, the disabled call, that requires a code input, is now flashing, as long as the code input is active and waiting for the numbers to be entered.

Bug fixes

- Interlocked door operation fixed, if used together with car call code input and locked door open button option.

14.2 Numbering

If new functions had been added, the minor version number will be incremented, making a V1.22.02 out of a V1.21.16 for example. On any kind of improvement or bug fix, the release number (last two digits) will be incremented. Those digits will always be incremented, once a version had been released. Odd release numbers do always indicate a version '*In Making*'. An even release number indicates a version, that is meant to be released to the customer. Once the version had been tested here and by our test partners, the extension '_stable' will be added. Such a version is then ready to be shipped to the customers.

The file name will then look like this:

liftapp_01_26_04_stable

14.3 Tagging

Once a version has been build and released for testing a tag shall be created via the version control system (Git) in order to be able to restore a release later on again. As also the Software Reference Manual is part of the same repository, the manual recreated would then match the version being tagged.

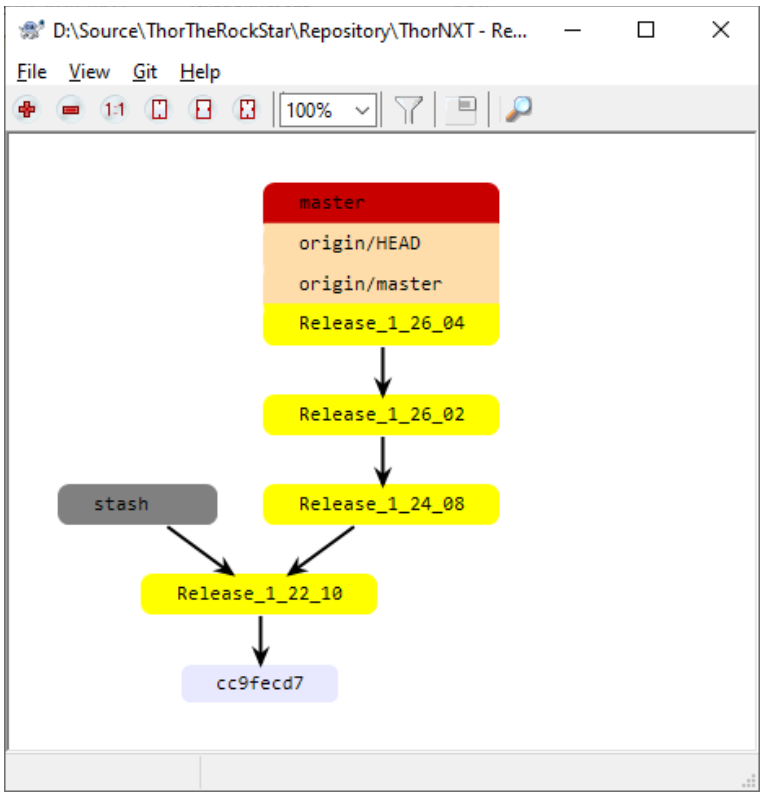


Figure 5: Revision control system

15 Release Notification & LiftApp Update

The customer is always notified about every release of the LiftApp via e-mail.

This includes mainly our OEM-partners, that build controller cabinets, sold to their end-customers. This might also include Engineering or Planning offices or end-customers, that applied for being notified of any new software release. This includes also our development partners in the Lift Industry, like the drive, door and encoder manufacturers for example.

These release notes contain the Versioning Entry plus the Secure hash, that the downloadable application shall have.

Always note, that the end-customer **shall check** the **SHA** when doing the update, ensuring that the software being installed had not been tampered since it had been released. This SHA gives you additionally security over the **automated CRC32** validation process.

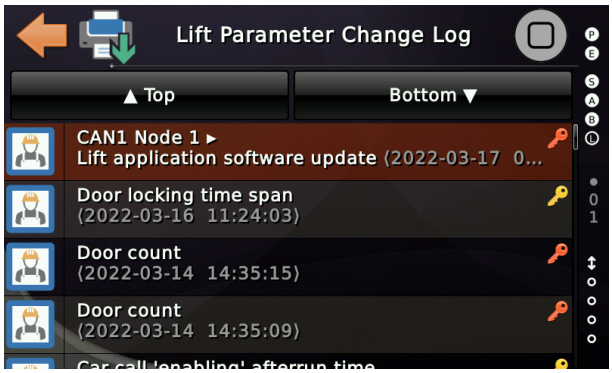
An update is generally only possible, if the lift has been turned to inspection, emergency electrical or emergency stop operation mode. The Setup-Password (🔑) has to be entered correctly and the automatic validation process has to be successfully passed. Nevertheless manually checking the SHA is recommended. We would like to refer to the float chart on page 41 as well.

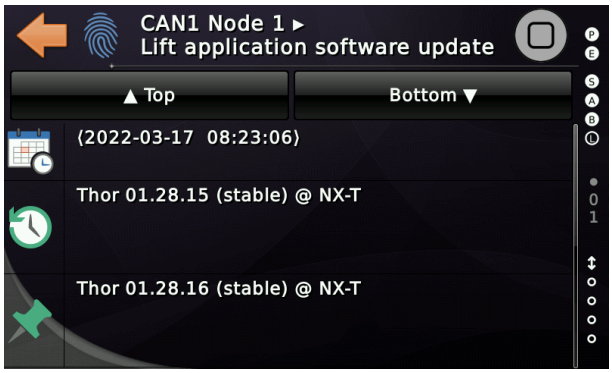
If the update is not downloaded from a local USB/SD mass storage but directly from the cloud, we force the technician to manually enter the last 4 digits of the hash, before the hash is displayed on the screen, to ensure, that he/she has read the e-mail beforehand.

The update can be downloaded manually from an USB mass storage device, a micro SD card or a cloud connection. A PUSH of an update to the controller is not planned and is classified by us as a safety risk, as we are of the opinion that after an update the technician always is in need to check the functional safety of the system, as well as the correct functioning of elements with which the passengers interact directly, such as doors and light curtains.

15.1 Update Documentation

In the lift controller the software update is documented, including the time and date, the old version and the new version, that had been installed. This change log is 256 entries large and can not be erased. But if it has reached 256 entries, a new entry will then dismiss the oldest entry in the list.





The Lift Parameter Change Log can be found by pressing the 'Favorites' button ( ) and then following 'System Menu' → 'Security' → 'Lift Parameter Change Log'.

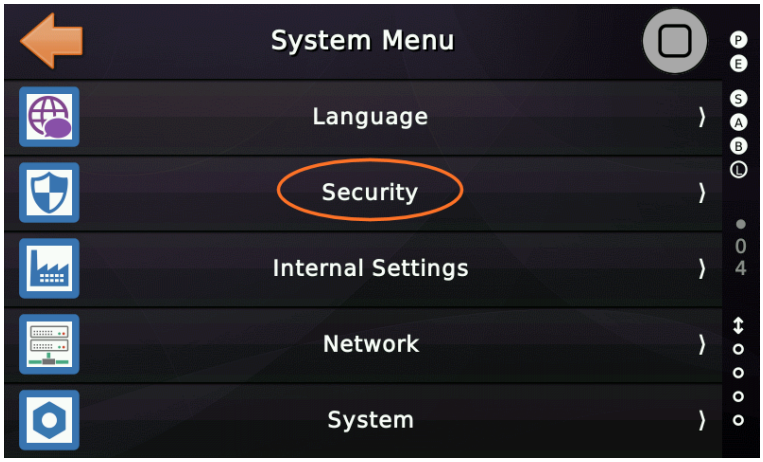
15.2 Update & Functional Security

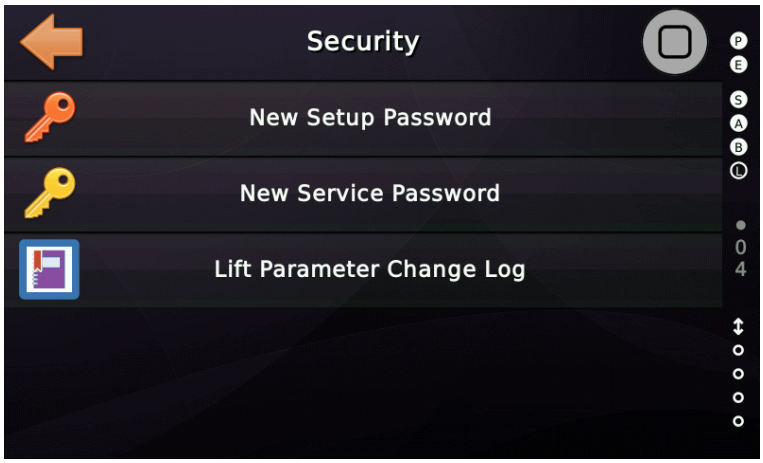
If an update is carried out on an lift controller, it does not affect any SIL-3 relevant functions, as they are implemented either in hardware or by external components, like a position supervisor unit.

The software update does not change any operating parameters of the controller.

16 Password Security

We highly recommend, that the customers do setup a **Setup Code** and **Service Code** to protect unattended usage of the lift controllers user interface.





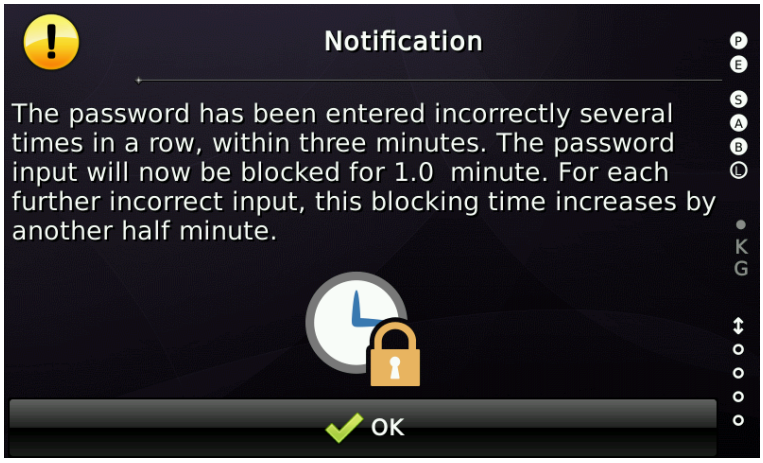
The Setup and Service password shall be 8 digits, containing characters and numbers.

The setup password (red key 🔑) secures parameters such as contactor monitoring times or the orientation of the position encoder used. The service password (yellow key 🔑) secures settings such as parking times or settings for the onward journey display.

Passwords are generally **not** stored in the lift controller's storage. Instead a salted SHA-1 (hash) of the password is stored. That means the lift controller can safely check the password input for being legit but it is not possible to calculate back from the hash to the readable (visible) password string.

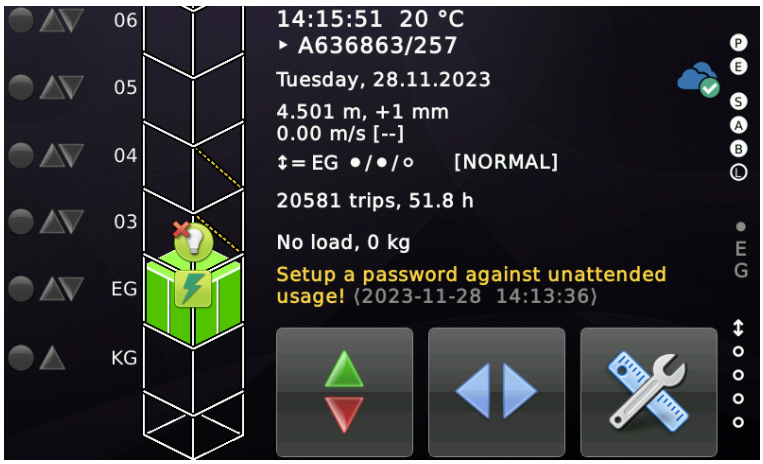
Since version V1.24.18 (12-2023)

If the password has been entered incorrectly three times in a row, within three minutes, then the password input will be blocked for one minute. For each further incorrect input, this blocking time increases by another half minute. If fifteen minutes have been passed, since the last wrong attempt, the internal counters are reset and three wrong password attempts are granted again.



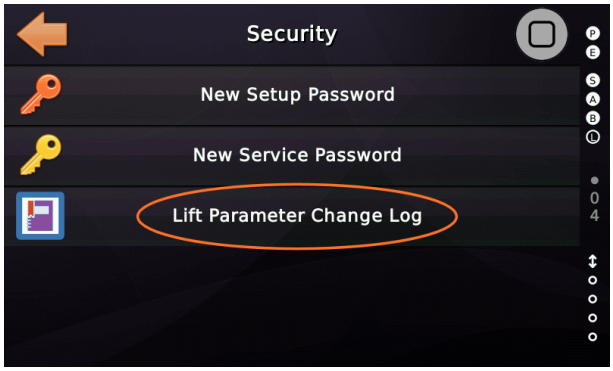
Since version V1.34.02 (09-2023)

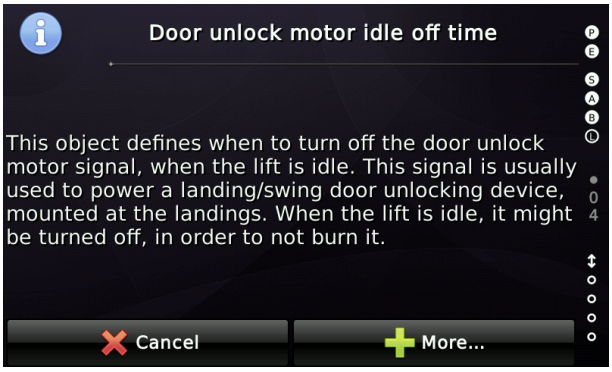
After completing the setting trip, the technician is asked to create at least one SETUP password.

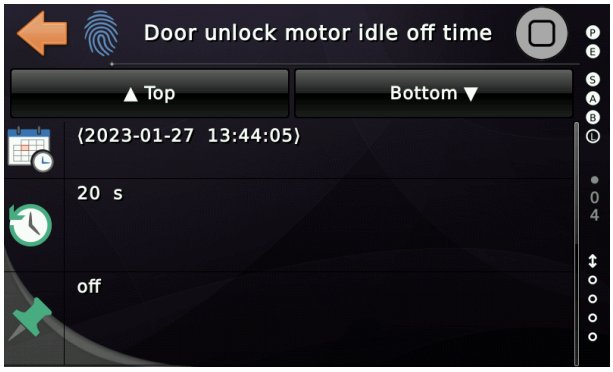


17 Lift Parameter Change Log

In order to record any parameter changes done to the lift controller, changes are recorded non-volatile in the unit. This log is not supposed to be erased by the end-customer. It can only be erased by the manufacturer. The changed parameter, the old parameter value, the new parameter value and the time of the change are saved.







The symbol of an entry signals whether the parameter was changed locally, by the bus system, the cloud or an assistant (e.g. teach-in the braking distances).

18 Network connection

18.1 General

Our lift controller does not have cellular technology or any other wireless communication on board.

The device can be connected to a router via the integrated wired network connector (RJ-45), which in turn provides a network connection. If the connection is wireless, it is mandatory to encrypt this temporary network with WPA2-PSK and to protect access with a secure (8-digit) password. For security reasons, the controller features a randomized MAC address by default. However, this can be changed if necessary when integrating the unit into permanent networks (see below).

When connecting to a building management network (SCADA), as is the case in hospitals for example, we recommend using a separate VLAN for connecting technology such as lifts, air conditioning, lighting, etc. A so-called VLAN is a virtual local area network, i.e. a logical sub-network of a physical local area network (LAN). The Virtual Local Area Network forms its own network segment and broadcast domain.

We advise against connecting the lift controller to the same logical network as printers, office PCs and similar equipment, as their physical access points (network connections) are often easy to reach.

We also recommend using managed switches, where the MAC of the participant on a network connection can be specified. For this purpose, the randomized MAC address in the controller can be replaced by a fixed MAC address (which can be specified by the end customer).

We do not recommend connecting the lift controller to a fixed WiFi® in a building, even if it has been properly protected.

18.2 Fuzzing the Interfaces

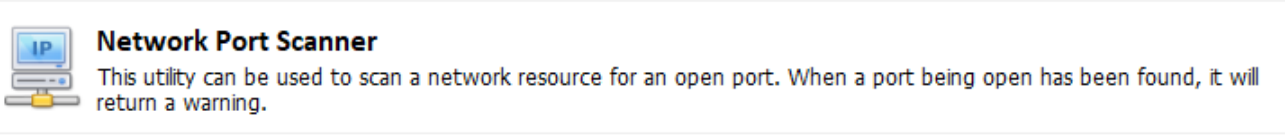
The network interfaces are fuzzed using 'Naughty Strings' using the POSTMAN tool before releasing a new firmware. These fuzzing tests include malformed HTTP headers, HTTP bodies, and incorrect input and input featuring unexpected syntax, like URLs, SQL commands and such. It is also being examined whether mass flooding of the interfaces can have a negative impact on the performance of the lift application.

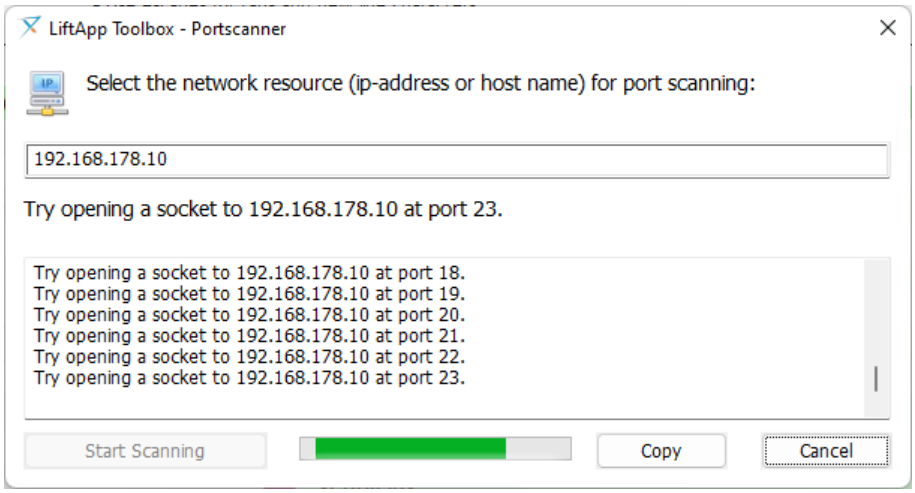
► See also Web server and Cloud Interface on page 21.

18.3 Open Network Ports

By factory default **no network port** is open at all. But the customer might open a port by means of activating a service, that is required for the project or end-customer demands, like the web server. In order for us to check, if by default no network port is open, we have created our own Port Scanner Tool, that has been build into our LiftApp Toolbox.

Before a new software is released, that scanner is used to check, if the new firmware release would have any port unintentionally open, for example when the software developer would simply forgot to have turned off a debug port.





The tool delivers a protocol, that we archive, when releasing a firmware.

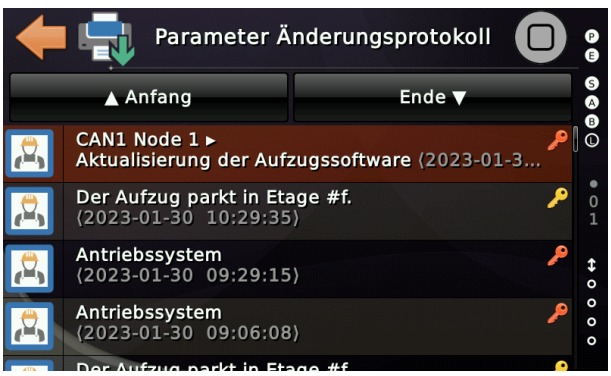
```
LiftApp Toolbox - Portscanner

06-02-2024, 08:57AM

Try opening a socket to 192.168.178.10 at port 1.
Try opening a socket to 192.168.178.10 at port 2.
Try opening a socket to 192.168.178.10 at port 3.
…
Try opening a socket to 192.168.178.10 at port 65535.
```

19 USB/Micro-SD Security

The unit is provided with USB host connectors and a Micro-SD card, that supports mass storage for File I/O. Newer devices also support the temporary connection of USB network routers that use the USB-CDC class. The USB and the SD-Card slot can be used to store text printouts, like the fault history or the parameter change log. The USB and SD-Card mass storage can also be used for **updating the firmware**.

Updating the firmware can only be done, if...

- The lift has to be turned to inspection, emergency electrical operation or emergency stop operation.
- To update the firmware, the **Setup Code** has to be entered locally on site.
- The firmware file is validated by the lift controller. For that purpose the ELF-token, the build in **CRC32** of the file, the vendor-id and the product code are checked.
- Additionally, the onsite engineer must verify the **SHA** of the file previously specified in the release notes. The release note has been previously sent to the technician, typically by e-mail, not in the same way as the file, which is usually transferred via a file sharing service. **The existing lift software calculates the SHA** of the requested file from the USB stick/micro SD card and displays it on the screen in an easy-to-read manner.
- Only if all requirements are fulfilled, the software can be updated.
- Any update of the software is recorded in the Parameter Change Log as well, that cannot be erased by the technician.



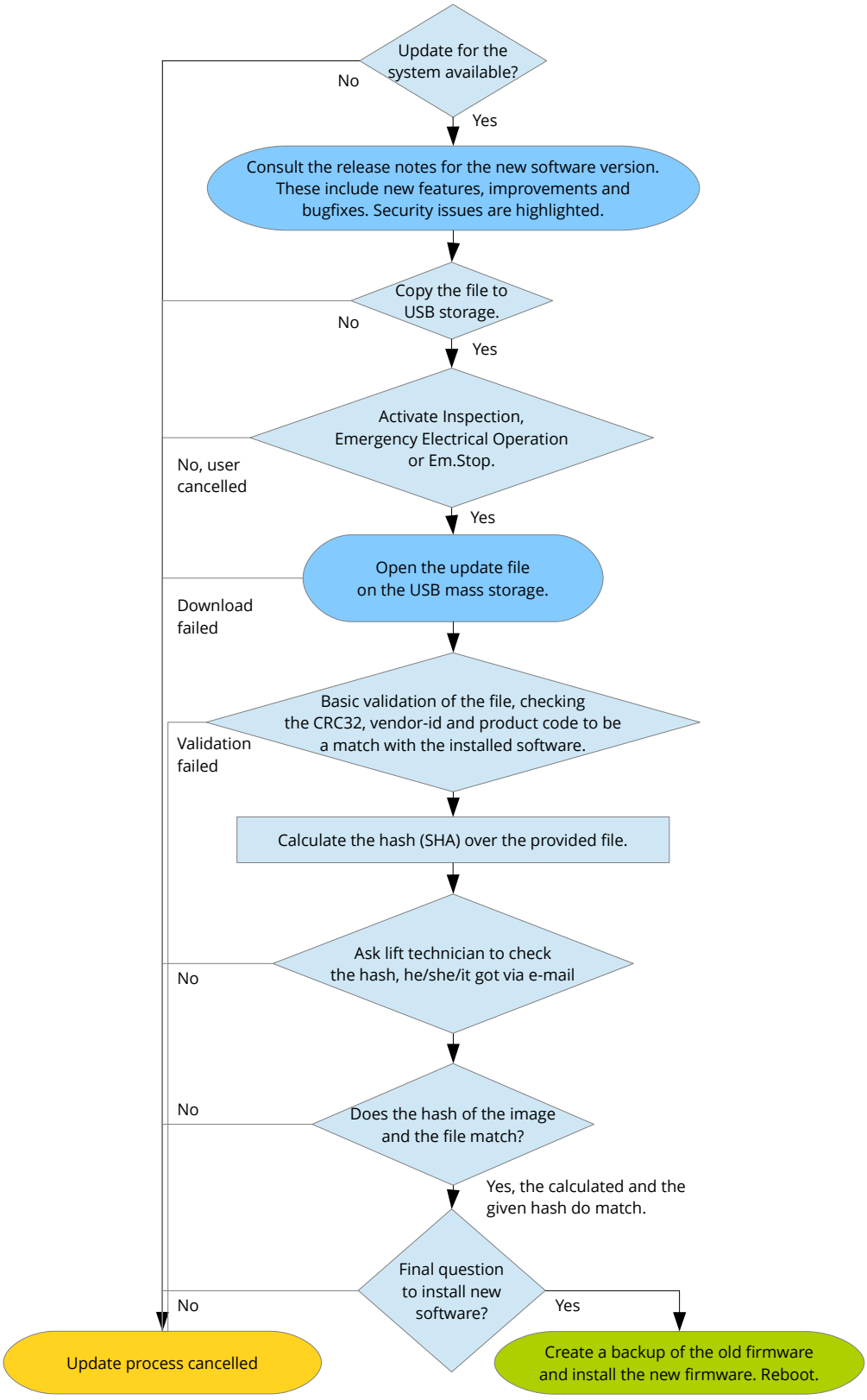


Figure 6: Float Chart Application Update

20 DEBUG Interface

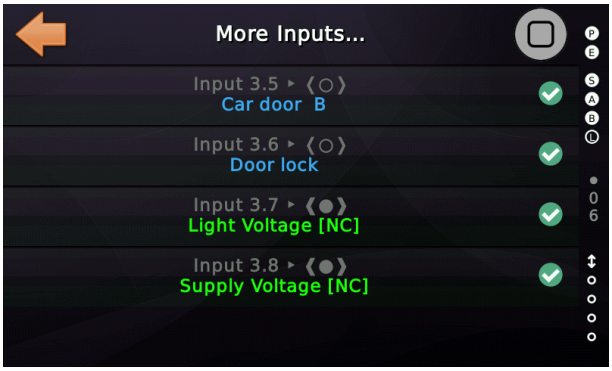
The units have a TTL-3.3V-UART DEBUG interface, that is **not populated** on the PCB by means of a connector or header. If someone would solder a connector on, reverse engineer the pin-out and would use a special hardware adapter, the intruder would get the boot log on that UART, that does not contain any relevant or secure data, no serial numbers and such. If the LiftApp takes over, the interface is rendered completely non-operational. To make this interface usable, a special development version of the LiftApp has to be installed first by an engineer of the manufacturer.

21 Micro USB Connector

An USB micro connector is installed on some devices. This is not intended for customer use, only available to the manufacturer for repair purposes. This connection is protected with an 8-digit alphanumeric and **random** password. Only the manufacturer has a table with the assignment between serial number and password. Via this interface, it is possible for the manufacturer to reset the parameter change record, if a device is sent in for repair and is not returned to the customer. There is no such thing as a '*master key*' and there never will be!

22 Safety Chain Sensing

The safety chain signals are monitored via our type-tested hardware-based sensing circuit. We **do not support** software-based transmission of the states of the car doors, shaft doors or door locks via the bus system. These signals have to be wired and are processed by us directly on the controller board. The functions of these terminals cannot be reprogrammed either locally or remotely – not even via the bus system.



The attempt to access these terminals via the bus system is not executed and is answered with an abort code:

```
Node    1, ▸ RSDO initiate download 0x6100:0x13, number of bytes 6
Node    1, ▸ TSDO abort 0x6100:0x13 code 0x08000020 'Data cannot be transferred or stored to the application.'
```

It is not possible to manipulate the signal states of the blocking device chain either locally, via the bus system, or remotely.

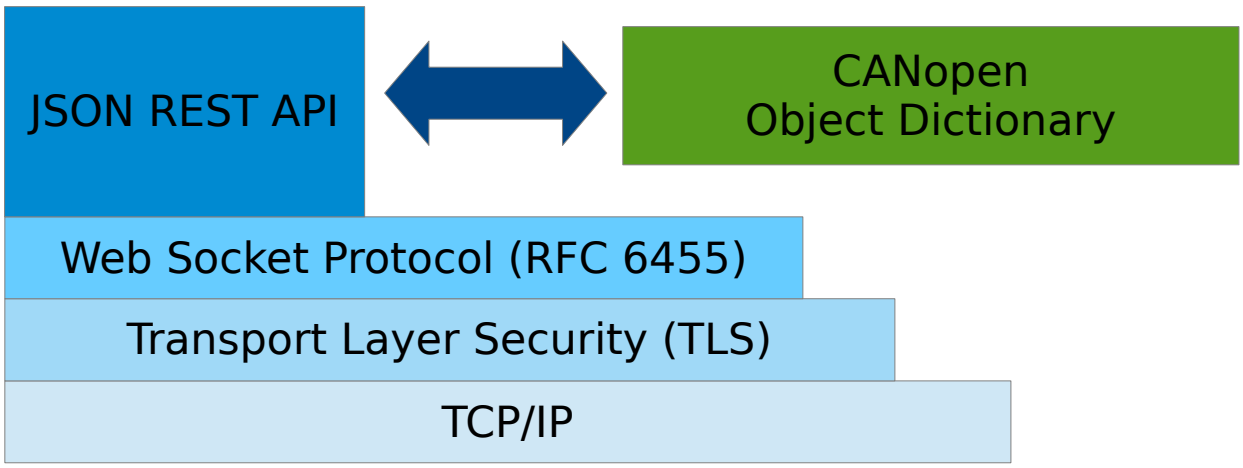
We do not support the input functions of the safety chain signals defined in CANopen because we have never supported the transmission of the safety chain signals via the bus.

23 NeXt® Cloud Security

To make sure, there is no unattended access to the lift via the cloud solution, that could cause the lift being attacked or data stolen, Thor's cloud solution features TLS encryption and a certificate based server authentication by default & by design without any compromises.

The Transport Layer Security (TLS) is the successor of the older and meanwhile deprecated Secure Sockets Layer (SSL). It is a cryptographic protocol, that have been designed to provide communication security via a computer network. This protocol has already been widely used in applications such as e-mail, online banking, instant messaging.

This protocol layer includes encryption and a certificate based handshake, in order to check if the cloud server is really the one, that the lift wants to connect to and not a '*fake*' created via a DNS attack. The server certificate used contains the server name and the trusted certificate authority (CA) that has issued the certificate of authenticity. It also contains the server's public encryption key, that is used for encrypting the payload data.



Accessing the screen remotely comes with limitations. Features and parameter that clearly are not intended to be used remotely are labeled as 'Engineer on Site' and can not be operated remotely. The protection of parameters (SETUP & SERVICE) passwords are the same.

We highly recommend to protect the lift controller by means of a SETUP & SERVICE password, when being connected to the Cloud solution. Under no circumstances should these passwords be stored in the “notes” for the lift in the cloud itself.

24 MQTT Interface Security

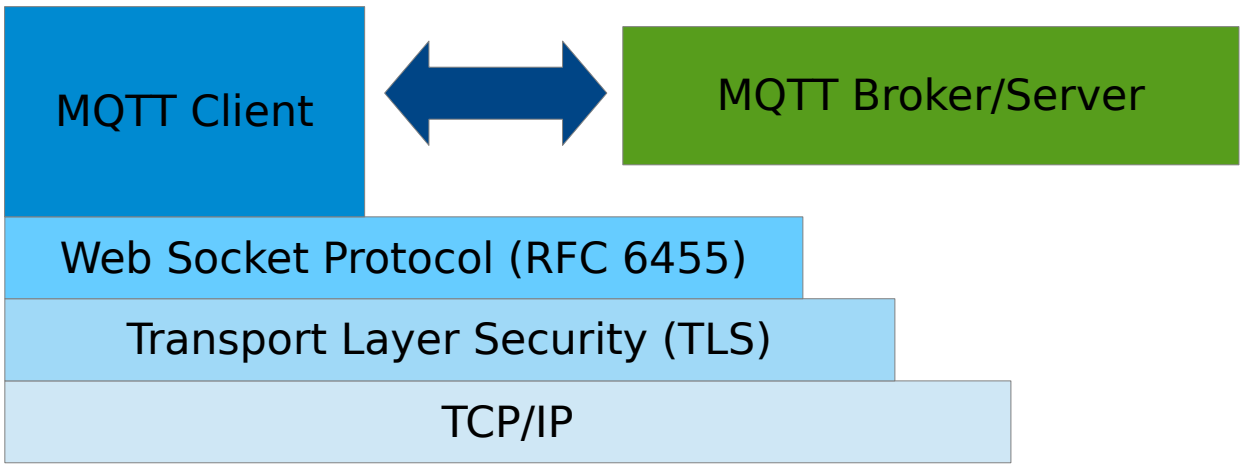
MQTT means "*Message Queuing Telemetry Transport*". It is an open messaging protocol. It is usually used for M2M solutions (*machine-to-machine communication*), such as in the "*Internet of Things*".

The lift controller software LiftApp provides a MQTT client, that had been developed in-house with the focus on robustness and security. This interface must be explicitly activated locally on the device. This is not possible remotely.

But beside writing robust code and making sure, that invalid MQTT message will not corrupt or crash the system, it is vital that the MQTT Broker system on the customer side is robust as well and properly protected against unattended access.

The customer must ensure that new security updates are installed in time and that their MQTT system remains protected from unauthorized external access.

To make sure, that the connection and transportation of the MQTT message from and to the broker/service is secure as well, we suggest to use the build-in TLS support. That means that the MQTT message will be transported via a TLS-encrypted WebSocket connection.



The secure socket is the preferred connection mode, when connecting via the internet. If you run the system in a factory or hospital environment, were a secure and encapsulated network is used for technical facilities, like lifts, you might go for the simpler connection modes.

We regularly do fuzz test the MQTT interface by means of a testing method, that creates thousands of malformed MQTT messages, that contain invalid message types, invalid remaining length indicators, invalid headers and payload data and even white noise. These messages are then fired against the MQTT message parser, in order to check, that all thinkable and non-expected error cases are properly handled, when receiving and processing MQTT message from an external broker.

24.1 MQTT Settings and Connection Status

The MQTT support settings can be found by pressing the hardware button 'Favorites' and then go further to 'System Menu' → 'Network' → 'More...' → 'Even more...' → 'Much more...' → 'MQTT Support'.

The connection status can be found on the last page of the MQTT settings. In this example, an encrypted TLS WebSocket is used to connect to a Broker, featuring QoS Level 1.

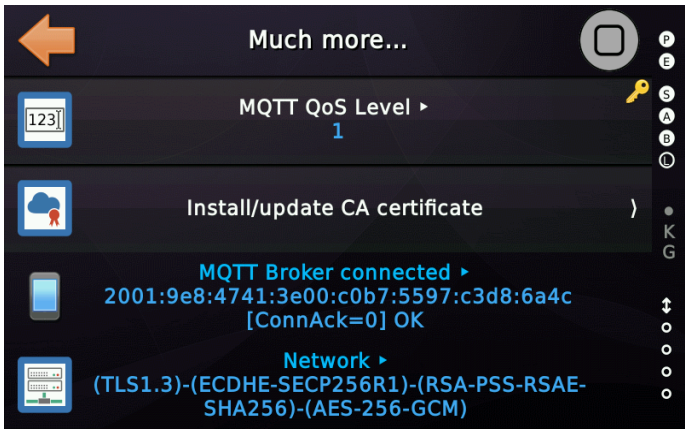


Figure 7: MQTT Network Connection Status

24.2 MQTT access to the Lift

Access to the lift controller via MQTT is restricted. It is only possible to make calls, press the door open/close button and switch special fieldbus terminals via MQTT. The interface is primarily intended for factory environments in which automated vehicles use the lift. But MQTT is also steadily replacing older fieldbus systems in the area of building automation in hospitals for example.

25 Testing a Release Candidate

Before a version is marked as stable and finally released to the OEM partners, it has to pass several testing procedures. This includes the function test done by the author of the software as well as a 'gray' test done by our service colleagues, that had **not** been included in the development process.

The standard test procedure includes:

- Checking the new release being able to be downgraded by the release before.
  It would be a fatal issue, if a software being delivered, can't be updated anymore, without the unit being send back to the factory.

- Testing the warm start and cold start feature.
  Changing certain parameters require the unit for restart. In order to make that as quick as possible, the software internally supports a quick warm start, were the POSIX process is kept alive and runs to the destructor and constructor code again.

- Checking function to backup and restore the parameter set.
  Backing up the parameter of the THOR unit and restoring them from a backup is a quite vital feature. This test includes doing this via USB-stick but also checking if the background task is still creating automatic backups on the pushed-in Micro-SD-card.

- Checking the function to reset to factory defaults and to reset the onboard terminals.
  If boards are replaced or swapped, resetting them to factory defaults or resetting the on-board terminals is required. As this function is not used so often, a failure will not be detected immediately. So testing it here is vital.

- Checking the functions used to printout the parameters, the logbook, the parameter change log and the quantity list of faults.
  This test shall be done in two languages, German and English to ensure we spot issues with umlauts.

- Checking normal driving in position profile and velocity profile operation mode. By doing this test, it makes sense to test also the Quickstart feature in both operating modes, as internally that makes quite a difference.
  Testing is done featuring ZA, Nidec and B&F drives and the iCON simulator, using CANopen 417. Additionally we test DCP3/4+ via the Nidec with a different option module. We test as well 4-valve drive operation via IO simulation. Include in the test

also the 'Drive unit control enable signal', which we use for soft-starters to enable the valves.

- Testing the inspection, emergency electrical operation and testing operating both inspection panels at the same time. Checking that inspection takes precedence over the emergency electrical operation.
  When testing that pit and top inspection can drive in the same direction if the same buttons have been pressed, ensure that this actually also applies to the 'Fast' buttons, if the lift controller would have some.

- Testing the pit inspection reset operation, via a classical input, via the display (Sweden) and via the alternative input method, using a pulse code.
- Checking re-levelling and SZ-fault detection as well as generating warnings for the zone signal being dropped belated.
- Checking Emergency Stop functions, Out Of Order Operation, Maintenance Mode and Assembly Operation Mode.
- Erasing all floors level position and operate the unit in Assembly Mode, via the Emergency Electrical Operation.
- Doing a manual and automatic learning trip, using a simple encoder (Wachendorff, ELGO 2M) and a safe encoder ELGO33CP and Safe ANTS.
- Using the UCM test assistant, the limit switch test assistant, the buffer test assistant, Runtime test assistant, Safety circuit bridge test assistant.
- Testing the contactor monitoring and brake contact monitoring supervision functions.
- Testing the power fail and car light power drop detection.
- Testing Phase Failure detection.
- Testing the non-volatile blocking for the passive safety chain.
- Testing the landing control off function and the car preference (independent mode) functionality.
- Testing hydraulic homing, start supervision, runtime supervision and deceleration supervision timer.
- Testing rotation supervision and car movement supervision.
- Testing the Cloud interface and the build web server.
- Testing the safety chain bridge detection.

- Testing No-Load, Full-Load and Overload handling.
- Testing Energy Saving and Standby functions. That includes the timers for the floor displays as well.
- Testing collective and SFR call operation modes.
- Testing Fire Alarm (simple, dynamic and fire alarm center mode).
- Testing Fire Recall/Service in the EN81 and US-ASME variants.
- Testing the Emergency Power functionality. This test includes checking the signals, used for doing Emergency Power of several lifts in a sequence.
- Testing the Emergency Evacuation (Shutter Break) feature.
- Testing the low pit and head solution, including the car fence operation.
- Testing the custom temperature threshold inputs and also the detection of the environment temperature, if exceeding the maximum allowed values as stated in the normatives.
- Testing the IO terminals of the NX-T2/3, the M18 and the Nous boards. This test includes starting them with inverted inputs and power-up.
- Testing the interlocked door operation.
- Testing the Extra Door Supervision.
- Testing the Separating Door Supervision.
- Testing of fully power driven automatic doors, swing doors with an automatic car door, swing doors only, manual car door gates and safety light curtain operation.
- Test advance door opening, including dropping the door zone, not receiving the door zone and the door zone being stuck/hung. Check that the blocking operation is non-volatile.
- Check the option to define the maximum door re-openings by a landing call.
- Check the swing door opener feature.
- Check the option to keep the retiring cam locked outside the floor level (not door zone) for the Belgium market.
- Check the motion detector functionality.
- Check the detection of a permanently interrupted light curtain.
- Check the door nudging operation and in detail if the door machine is featuring reduced force, when closing.
- Check the finger protector handling, using the Meiller MiDrive for testing.

- Checking if the 'Limit switch «closed» bridge/hung detection' does work.
- Test erasing the logbook and resetting of statistical counters.
- Testing the QR code generator.

The extended test procedure includes:

- A detailed test of the new functions on the release list.
- A detailed test of the updated functions on the release list.
- A test of the new and/or updated functions by one college of the service stuff, that takes the newly or updated chapter of the Software Reference Manual and then simply without any further instructions tries to get the function running properly. This might result in the dedicated chapter of the manual being updated again, before the software has been updated.

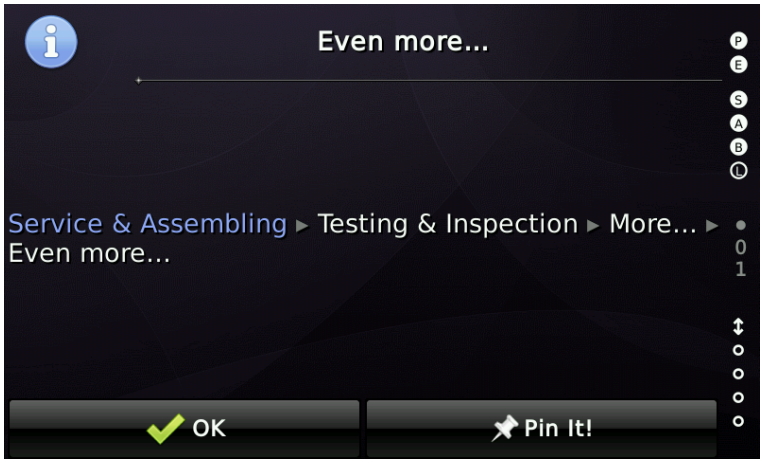
- Fuzz Testing the network interface (Build-in webserver and JSON REST-API) my means of using POSTMAN and our Test Collections, using an up-to-date 'List of Naughty Strings' fired at the input fields and used to create malformed HTTP-Headers, bodies and valid JSON-request but containing invalid or unexpected data.

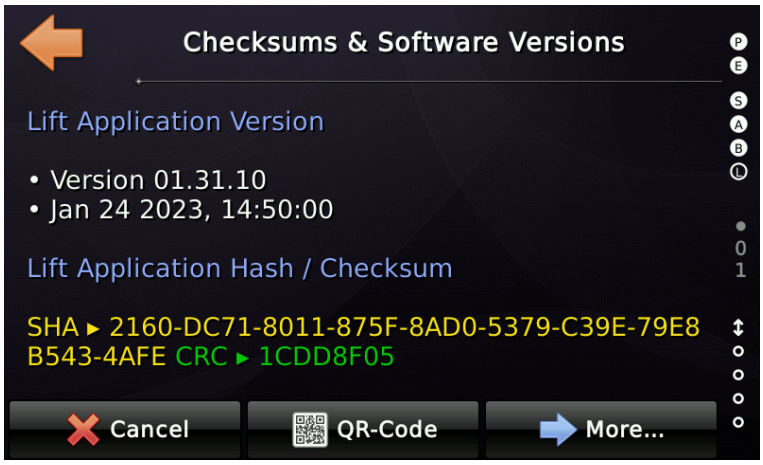
- Fuzz testing of the MQTT interface using a specially written test method, that produces randomly incorrect MQTT messages (including incorrect length identifiers).

26 Checksums & Software Version

The lift controller provides an easy way to check, which version is running in the lift controller and the checksum of the current running application.

You find that page here:

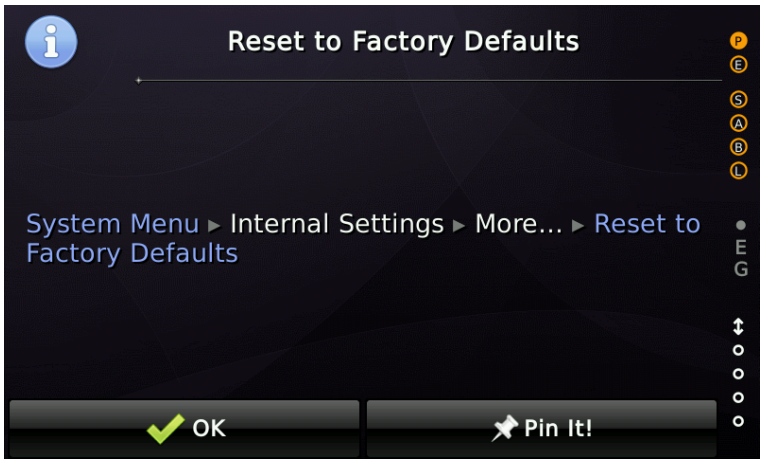




When the lift application is started, the integrity and checksum of the application are recalculated and only started if they match. This prevents changes to the application due to hardware failure. The file system used also uses checksums to detect defective sectors and not to pass on data that is not valid.

27 Decommissioning

If the lift controller has been removed, for example, during a renovation, it should be reset to factory settings before disposal. This will ensure that the lift number and controller number are removed from the unit. Personal data is generally not stored on the unit and therefore does not need to be erased. The unit should then be disposed of properly and in accordance with local regulations.



If the device was equipped with a micro SD memory card during operation that is used for data backup or voice announcement, this must be safely deleted (completely formatted) after decommissioning and then disposed of in accordance with the Electronic Waste Ordinance. The DIN standard 66399 regulates the secure destruction of “office and data technology data carriers” by law. This should then be applied.

28 Coding rules

Using common coding rules within the development team will ensure, that code can be double checked, reviewed by colleagues, making it more likely that issues, bugs and non caught error conditions can be found in the reviewing, debugging and testing life cycle part of the software. These coding rules are always a matter of discussion and are improved step by steps as the team is growing and getting more experienced.

28.1 Abstract

The following programming and coding rules are meant to be a resource for writing good, reliable and readable code, making it easier for the team and programmers of the project to review, extend, understand and learn from the existing code.

These rule shall also ensure, that the code is robust and less likely to be incorrect. Good readable and easy to understand code is a good choice, if we want to make it less likely to end up with bugs and issue, that are later encountered in the field by customers. These rules are not meant to keep someone in leading-strings. They will help the team to avoid bugs and help each other to read the code of the colleges.

Easily readable and easy-to-understand code is a good prerequisite for avoiding Cybersecurity vulnerabilities from the beginning.

28.2 Basic and General Directives

- Never access data structures shared over threads directly without the proper mutual exclusion (*locked by a Semaphore, Mutex or Critical Sections*). Remember that other threads may be accessing the same structures in the very same moment.
- Prefer event notification over polling methods.
- Provide a useful error response (to the log) if a resource is not available, when the application needs it.
- Never tie up system or application resources, unless it is absolutely necessary.
- Always make use of the simple structure conventions!
    - All reserved or currently unused fields/elements should be initialized to

zero for future compatibility.

- Fields/elements that are not defined to have a particular initial value must be initialized to zero. This shall include pointer fields.
- Always keep in mind the alignment of the fields/elements. Modern processors expect that types like WORD or DWORD will be stored in memory at addresses that are multiples of their own length. Some processors may allow unaligned access but payed with a performance penalty.

- Do not use signed variables or signed math for addresses.

- Avoid deep nesting of code if possible. It will make it more readable for others and that means that the code can easier be understood and verified for errors.

- Do not repeat code all over again. Create a function or method instead. If performance is a crucial factor, make this function or method '**inline**' to avoid unnecessary overhead by jumping into and returning from that sub-function or method.

- Avoid packing code into macros. It might be sometimes required but use this method with care as it makes code harder to review.

- If local variables do contain constant values, declare them as 'const' making it impossible to alter the values by mistake or use them as 'temporary' variables for computing that had been added on later in the software's life cycle.

- Never use CPU delay loops. Instead use the timer functions 'addclock/diffclock' or the 'Board_Sleep' function. Under Linux® the use of 'nanosleep()' shall be preferred over the older 'msleep() and usleep()' functions as those functions may not have been tied to the MONOTONIC clock, creating issues when setting the system time.

28.3 Rules and Definitions

Functions/Methods and Attributes

The function/method comment block will feature the “DoxyGen” like reserved key words.

Method having no parameters (Definition)

```
/**
 * Returns if the lift is currently in priority call operation mode.
 *
 * @return    TRUE/FALSE
 */

int CliftPilot::Is_Prio_Call_Operation(void) const
{
    return(m_priority_call_state ? TRUE : FALSE);
}
```

Method having no parameters (Declaration)

```
public:

    /* Returns if the lift is currently in prio call operation mode. */

    int Is_Prio_Call_Operation(void) const;
```

- A function having no parameters will defined explicit as “void” in the declaration/definition.
- The way the brackets are set will not be strictly defined, but for better readability a project should use just one bracket schematics.
- Enumeration/Type definitions will have the word “Enum” or “Type” at the end of their names.
- Attributes have a “m_” in the beginning of their names.
- Global variables are rarely used and will have a “g_” in the beginning of their names.

- All class names start with a capital “C” or “Q” for Qt® classes.
- Function providing a “Getter” functionality should start with “Get”, “Is” or “Are”.
- Function that provide a “Setter” functionality should start with “Set”.
- Handler (cyclic/event triggered) should start with “Handle” or “Do”.

Deriving classes

- If deriving classes, polymorphism should be avoided.
- The derived class name may contain the name of its super class.
- If the super class is the base class itself, the word “Base” should be removed from the derived class name.

Deriving classes (Declaration)

```
/**
 * Door class for a typical automatic car/landing door combination.
 */

class CLiftDoorCntrlAuto : public CLiftDoorCntrlBase
{

};
```

Deriving classes (Declaration)

```
/**
 * Door class for a automatic car door and manual (swing) landing door combination.
 */

class CLiftDoorCntrlAutoSwing : public CLiftDoorCntrlAuto
{

};
```

Jumps

- Absolute jumps like "`goto xyz`" should be banned. :-)

- Using "`continue;`" should be used with care. A common mistake is to increment a pointer in the end of a loop and bypassing this incremental instruction by mistake, because some branch is using a Continue in a switch/case construct.

Type definitions including enumerations and bit fields

Declaration of types and enumerations

```
/**
 * Lift attendant (lift boy)
 */

typedef enum LiftBoyOperationStateType
{
    LIFT_BOY_OPERATION_STATE_OFF = 0,       // Operating mode is off.
    LIFT_BOY_OPERATION_STATE_ON = 1,        // Operating mode is on.
    LIFT_BOY_OPERATION_STATE_START = 2,     // On, waiting for the start button.
    LIFT_BOY_OPERATION_STATE_RUN = 3,       // On, driving to next floor.
    LIFT_BOY_OPERATION_STATE_ERROR = 4,     // On, error mode.

} LiftBoyOperationStateType;
```

Structure declaration

```
/**
 * Structure for storing pending call acknowledge (lamps) cancellation.
 */

typedef struct CallAckCancellationType
{
    uint8_t floor;              // Floor of call

    struct
    {
        uint8_t call_type : 4;    // Call type
        uint8_t door_mask : 4;    // Door bit mask

    } info;

    uint8_t count;                // Timer triggered counter

} CallAckCancellationType;
```

Keep in mind that the internal organization of bit fields are big/little endian depended.

Switch/Case/Default constructions

A switch/case should always have a default path even if it is empty or just contains a debug message, like this:

Switch/Case/Default

```
    /* Filter door signals. */

    switch (sigid)
    {
    case APP_LIFT_DOOR_INPUT_CLOSED:

        /* Route floor/door selective door limits input signals. */

        if ((floor == APP_LIFT_FLOOR_ALL) || (!floor))
        {
              <snip>
        }
        break;

  <snip>

    default:
        GURU0(("Door %d: Unknown signal %d passed. ", m_door_id, sigid));
        break;
    }
```

So called 'Fall thru' constructs within a switch-case are undesirable as they are prune to lead to errors and mistakes.

Long if/else constructs

To make long if/else constructs better readable, sometimes it is a good idea to add even the empty else paths. This also applies to indicate that the 'else' path had not been forgotten.

If/else

```
    /* Door lock rule on Safety light curtains. */

    if (data->idoor_count)
    {

        /* Some (very) long code... */

    }
    else
    {
        GURU0(("CUnitBase: Should never be executed."));
    }

<snip>
```

Source and Header files

To make the understanding for the purpose of classes and files obvious and easy to follow, each source and header file shall contain an initial comment block, giving basic information about function, purpose, language, toolchain, original file name, project name and the author(s). The additional date is redundant as the repository system (GIT) is keeping track of it. Nevertheless, in practice it has turned out to makes sense to update the date manually.

Source/Header initial comment block

```
/**
 *  Copyright © 2016 Thor Engineering GmbH
 *
 *  liftpilot.cpp
 *
 *  Implementation of the basic states, the lift passes through while
 *  processing calls - or better destinations, which are defined by
 *  one or more calls. Each destination is a 3-tuple of a floor, a
 *  door-mask (containing one or more doors attached) and a call type.
 *  By reaching a destination one or more of these 3-tuples will be
 *  canceled. The main goal of the "LiftPilot" class is to finish
 *  all destinations in the shortest time possible.
 *
 *  Project:            LiftApp for the NeXt project
 *
 *  Programmer:         Roy Schneider
 *  Last Change:        18.05.2016
 *
 *  Language:           C/C++
 *  Toolchain:          GCC/GNU-Make
 */

#include "../main.h"

#include "liftdata.h"
#include "liftparam.h"
#include "liftapp.h"
#include "liftpilot.h"

#include "../../logfile/logfile.h"

<snip>
```

Classic C-String operations

- When possible and useful prefer a string class over classic string operations.
- Usage of deprecated string functions, like strcpy, strcat, strlen or sprintf shall be avoided. Instead the variants, like snprintf or strncpy, strncat, strnlen shall be used, that provide a parameter for limiting the destination buffer length.
- Double check, that the given 'max' value is matching the destination string remaining buffer size.
- Always use the _countof() instead of the sizeof() macro when determining string buffer sizes, to make sure, that the buffer size is correctly calculated, even when dealing with wchar_t (multi bytes characters) instead of simple char.
- Make sure that the string buffers are always zero terminated. When using strncpy keep in mind, that it will not terminate the destination buffer with a zero, when having reached the maximum character count. This is a different behaviour from snprintf for example.
- When appending strings to strings, check the remaining buffer space.

String operations

```
char sztemp[32];

/* Always make sure that the destination string will be zero terminated
 * and that the destination buffer does not overflow.
 * Keep in mind, that strncpy will not terminate the string with a zero
 * when reaching the maximum character count given! */

if (pstr)
{
    strncpy(sztemp, pstr, _countof(sztemp) - 1);
    sztemp[_countof(sztemp) - 1] = 0;
}

/* Always make sure that the destination string will be zero terminated
 * and that the destination buffer does not overflow. */

static const char s_fmtin[] = "Debug: %d";

snprintf(sztemp, _countof(sztemp), s_fmtin, ival);

/* When appending strings to strings, check the remaining buffer space. */

size_t slen = strnlen(sztemp, _countof(sztemp) - 1);

strncat(sztemp, "TEST", _countof(sztemp) - 1 - slen);
sztemp[_countof(sztemp) - 1] = 0;
```

29 Code Analysis Tools

To ease our work, of finding potential and real bugs and issues of several types, we make use of static analysis tools, that are available in the market which helps to analyze the code during the development and detect fatal defects early in the development phase.

Such defects can be eliminated before the code is actually pushed for functional. A defect found later is always expensive to fix.

We are using...

- **CDT Code Analysis**, that is running in the background while the developer is typing and writing. This tools detects for example non-initialized variables right while typing the code.

- **GNU Code Diagnostic** that detects a hole bunch of issues right while compiling, such as mismatches between variable data types and format specifiers in C-format strings.

- **CPPCheck** Static Source Code Analysis Tool is an open source analysis tool for C and C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. The goal is to detect only real errors in the code, and generate as few false warnings as possible.

- **DoxyGen** is not really a Code diagnostic tool but while checking the documentation tags it also discovers mismatches, like variable naming discrepancies in declarations and definitions. It also throws additionally warnings about unclear method overloading.

30 SHA implementation

The following codes reflects the implementation of the Secure Hash Algorithm as being used in the lift controller application.

```
/**
 * Copyright (c) 2017-19 Thor Engineering GmbH
 *
 * sha.cpp       The "Secure Hash Algorithm" SHA1 implementation.
 *
 * Project:      LiftApp for the NeXt project
 *
 * Programmer:   Roy Schneider
 * Last Change:  19.08.2019
 *
 * Language:     C/C++
 * Toolchain:    GCC/GNU-Make
 *
 * NOTE:
 * This implementation in C++ was inspired by the published work of
 * John Halleck (University of Utah).
 */

#include "../main.h"
#include "../base/base_types.h"
#include "bitutils.h"
#include "sha1.h"

/**
 * Constructor
 */

CSHA1Provider::CSHA1Provider()
{
    memset(&m_context, 0, sizeof(m_context));
}

/**
 * Destructor
 */

CSHA1Provider::~CSHA1Provider()
{
}

/**
 * Initialize the SHA provider instance.
 *
 * @return OK/ERROR
 */

int CSHA1Provider::Init (void)
{
    /* Init */

    memset(&m_context, 0, sizeof(m_context));
```

```
    register unsigned long *_pd = m_context.cprocess;

    *_pd++ = 0x67452301;
    *_pd++ = 0xEFCDAB89;
    *_pd++ = 0x98BADCFE;
    *_pd++ = 0x10325476;
    *_pd   = 0xC3D2E1F0;

    /* Return */

    return(OK);
}

/**
 * Execute the SHA rounds and transform the data.
 */

inline void CSHA1Provider::Transform (void)
{
    int ival;
    CryptSHA1ContextType *pc;
    unsigned long dwval, *pdw;
    unsigned long dwA, dwB, dwC, dwD, dwE;
    unsigned long dw[128];

    /* Init */

    pc = &m_context;

    /* Check */

    /* Init */

    register unsigned long *_ps = pc->cprocess;

    dwA = *_ps++;
    dwB = *_ps++;
    dwC = *_ps++;
    dwD = *_ps++;
    dwE = *_ps;

    ival = APP_SHA_1_BLOCKWORDSIZE;
    pdw  = dw;

    register unsigned long *_pd = pc->ldata;

    while(likely(ival--))
    {
        *pdw++ = *_pd;
        *_pd++ = 0;
    }

    ival = 16;

    while(likely(ival < 80))
    {
        _pd = dw + ival;

        *_pd = *(_pd - 3) ^ *(_pd - 8 ) ^ *(_pd - 14) ^ *(_pd - 16);
```

```
        *_pd = ROTINT32(1, *_pd);

        ival++;
    }

    ival = 0;
    _pd  = dw;

    while(likely(ival < 20))
    {
        dwval = (*_pd++) + ROTINT32(5, dwA) + dwE + 0x5A827999L + \
                ((dwB & dwC) | (~dwB & dwD));

        dwE = dwD;
        dwD = dwC;
        dwC = ROTINT32(30, dwB);
        dwB = dwA;
        dwA = dwval;

        ival++;
    }

    while(likely(ival < 40))
    {
        dwval = (*_pd++) + ROTINT32(5, dwA) + dwE + 0x6ED9EBA1L + \
                (dwB ^ dwC ^ dwD);

        dwE = dwD;
        dwD = dwC;
        dwC = ROTINT32(30, dwB);
        dwB = dwA;
        dwA = dwval;

        ival++;
    }

    while(likely(ival < 60))
    {
        dwval = (*_pd++) + ROTINT32(5, dwA) + dwE + \
                 0x8F1BBCDCL + ((dwB & dwC) | (dwB & dwD) | (dwC & dwD));

        dwE = dwD;
        dwD = dwC;
        dwC = ROTINT32(30, dwB);
        dwB = dwA;
        dwA = dwval;

        ival++;
    }

    while(likely(ival < 80))
    {
        dwval = (*_pd++) + ROTINT32(5, dwA) + dwE + 0xCA62C1D6L + \
                (dwB ^ dwC ^ dwD);

        dwE = dwD;
        dwD = dwC;
        dwC = ROTINT32(30, dwB);
        dwB = dwA;
```

```
        dwA = dwval;

        ival++;
    }

    _pd = pc->cprocess;
    register unsigned long lmsk = 0xFFFFFFFF;

    *_pd   += dwA;
    *_pd++ &= lmsk;
    *_pd   += dwB;
    *_pd++ &= lmsk;
    *_pd   += dwC;
    *_pd++ &= lmsk;
    *_pd   += dwD;
    *_pd++ &= lmsk;
    *_pd   += dwE;
    *_pd   &= lmsk;

    pc->iword = 0;
    pc->ibyte = 0;
}

/**
 * Update the SHA context with the given string.
 *
 * @param pbuf  Pointer to the data buffer used to update the hash.
 * @param icnt  Length (or count of) bytes in the buffer given by 'pbuf'.
 *
 * @return OK/ERROR
 */

int CSHA1Provider::Update(const unsigned char *pbuf, int icnt)
{
    int ierr;
    int iword;
    CryptSHA1ContextType *pc;
    unsigned long dwval, dwmask;

    /* Init */

    pc = &m_context;

    /* Check */

    if (likely((pc) && (pbuf) && (icnt > 0)))
    {
        /* Init */

        dwmask = 0x1FFFFFFF; // 29 bit mask

        pc->lcount_hi  += icnt >> 29;
        pc->lcount_low += icnt & dwmask;
        pc->lcount_hi  += pc->lcount_low >> 29;
        pc->lcount_low &= dwmask;

        iword = pc->iword;
        dwval = pc->ldata[iword];
```

```
        while(likely(icnt--))
        {
            dwval = (*pbuf++) | (dwval << 8);

            pc->ibyte++;

            if (unlikely(pc->ibyte >= 4 /*32 bit*/))
            {
                pc->ldata[iword++] = dwval;

                dwval = 0;

                if (unlikely(iword >= APP_SHA_1_BLOCKWORDSIZE))
                {
                    Transform();

                    iword = 0;
                }

                pc->ibyte = 0;
            }
        }

        pc->iword = iword;
        pc->ldata[iword] = dwval;

        ierr = OK;
    }
    else
    {
        ierr = ERROR;
    }

    /* Return */

    return(ierr);
}

/**
 * Pad (fill up) the SHA buffer.
 */

void CSHA1Provider::Pad(void)
{
    CryptSHA1ContextType *pc;
    int ival;

    /* Init */

    pc = &m_context;

    /* Init */

    unsigned long *_pd = pc->ldata + pc->iword;

    *_pd <<= 8;
    *_pd  |= BIT7;

    switch(pc->ibyte)
```

```
    {
    case 2:
        *_pd <<= 8;
        break;
    case 1:
        *_pd <<= 16;
        break;
    case 0:
        *_pd <<= 24;
        break;
    default:
        break;
    }

    ival = pc->iword + 1;
    _pd  = pc->ldata + ival;

    while(likely(ival < APP_SHA_1_BLOCKWORDSIZE))
    {
        *_pd++ = 0;

        ival++;
    }

    pc->iword = 0;
    pc->ibyte = 0;
}

/**
 * Return the SHA bytes in the given buffer.
 *
 * @param phash  Hash result buffer. See CryptSHA1Digest8Type for details.
 *
 * @return OK/ERROR
 */

int CSHA1Provider::GetBytes(CryptSHA1Digest8Type phash)
{
    CryptSHA1ContextType *pc;
    int ival;
    int ierr;
    unsigned long cval;

    /* Init */

    pc = &m_context;

    /* Check */

    if (likely((pc) && (phash)))
    {
        /* Init */

        ival = 0;

        unsigned char *ph = phash;

        while(likely(ival < APP_SHA_1_DIGESTWORDSIZE))
        {
```

```
            cval = pc->cprocess[ival];

            *ph++ = (unsigned char) LOINT8((cval >> 24));
            *ph++ = (unsigned char) LOINT8((cval >> 16));
            *ph++ = (unsigned char) LOINT8((cval >>  8));
            *ph++ = (unsigned char) LOINT8((cval));

            ival++;
        }

        ierr = OK;
    }
    else
    {
        ierr = ERROR;
    }

    /* Return */

    return(ierr);
}

/**
 * Finalize the SHA hash.
 *
 * @param phash  Hash result buffer. See CryptSHA1Digest8Type for details.
 *
 * @return OK/ERROR
 */

int CSHA1Provider::Final(CryptSHA1Digest8Type phash)
{
    CryptSHA1ContextType *pc;
    int ival;
    int n;
    int ierr;

    /* Init */

    pc = &m_context;

    /* Check */

    if (likely((pc) && (phash)))
    {
        /* Init */

        ival = (pc->iword << 2) + pc->ibyte + 1;

        Pad();

        if (unlikely(ival > APP_SHA_1_PADDING_REMAINDER))
        {
            Transform();

            unsigned long * _pl = pc->ldata;

            n = (APP_SHA_1_BLOCKWORDSIZE - APP_SHA_1_PADDING_WORDS);
```

```
            while(likely(n--))
            {
                *_pl++ = 0;
            }

            pc->iword = APP_SHA_1_BLOCKWORDSIZE;
            pc->ibyte = 0;
        }

        unsigned long * _pl = &pc->ldata[14];

        *_pl++ = pc->lcount_hi;
        *_pl   = pc->lcount_low << 3;

        Transform();

        ierr = GetBytes(phash);
    }
    else
    {
        ierr = ERROR;
    }

    return(ierr);
}

/**
 * Create a SHA hash from the given string.
 *
 * @param phash    Hash result buffer. See CryptSHA1Digest8Type for details.
 * @param pbuf     Pointer to the buffer containing the string to hash.
 * @param icnt     Count of characters in the buffer containing the string
 *                 to hash.
 *
 * @return OK/ERROR
 */

int CSHA1Provider::HashIt(CryptSHA1Digest8Type *phash, \
                          const unsigned char *pbuf, int icnt)
{
    int ierr;

    /* Init */

    memset(&m_context, 0, sizeof(m_context));
    memset(phash, 0, sizeof(CryptSHA1Digest8Type));

    /* Check */

    ierr = ERROR;

    if (likely((pbuf) && (phash) && (icnt > 0)))
    {
        if (likely(Init() == OK))
        {
            if (likely(Update(pbuf, icnt) == OK))
            {
                ierr = Final(*phash);
            }
```

```
        }
    }

    return(ierr);
}

/**
 * Verify (compare) two SHA1 hashs.
 *
 * @param pplain1 [in] First plain SHA hash.
 * @param pplain2 [in] Second plain SHA hash used to verify (compare) the first.
 *
 * @return OK/ERROR
 */

int CSHA1Provider::VerifyIt(CryptSHA1Digest8Type pplain1, \
                            CryptSHA1Digest8Type pplain2)
{
    int ierr;

    /* Init */

    ierr = ERROR;

    if (likely((pplain1) && (pplain2)))
    {
        if (likely(!memcmp(pplain1, pplain2, sizeof(CryptSHA1Digest8Type))))
        {
            ierr = OK;
        }
    }

    return(ierr);
}

/**
 * Calculate SHA1 of the given file stream and finally return it to the caller.
 *
 * <Beware that this code is optimized for 32/64 bit memory
 *  block alignment for little endian processors.>
 *
 * @param pfile  Pointer to the file, returned by fopen().
 * @param phash  Hash result buffer. See CryptSHA1Digest8Type for details.
 *
 * @return OK/ERROR
 */

int CSHA1Provider::CalculateSHAFile(FILE *pfile, CryptSHA1Digest8Type *phash)
{
    int ierr;
    size_t ilen;
    unsigned char *ptmp;

    /* Init */

    ierr = ERROR;

    memset(&m_context, 0, sizeof(m_context));
```

```
    if (likely(phash))
    {
        memset(phash, 0, sizeof(CryptSHA1Digest8Type));

        /* Check */

        if (likely(pfile))
        {
            /* Alloc */

            ilen = 16 * 1024;
            ptmp = new unsigned char[ilen];

            if (likely(ptmp))
            {
                if (likely(Init() == OK))
                {
                    if (likely(!fseek(pfile, 0, SEEK_SET)))
                    {
                        unsigned long ulread = 0;

                        do
                        {
                            ulread = fread(ptmp, 1 /*byte*/, ilen, pfile);

                            if (likely(ulread))
                            {
                                ierr = Update(ptmp, (int) ulread);

                                if (unlikely(ierr != OK))
                                {
                                    ierr = ERROR;
                                    break;
                                }
                            }
                        }
                        while(likely(ulread));

                        /* Finalize */

                        if (likely(ierr != ERROR))
                        {
                            ierr = Final(*phash);
                        }
                    }
                }

                /* Free memory */

                delete(ptmp);

            } // if (likely(ptmp))
        }
    }

    /* Return */

    return(ierr);
}
```

```
/**
 * Calculate the SHA of the given file name and finally return it to the caller.
 *
 * <Beware that this code is optimized for 32/64 bit memory
 *  block alignment for little endian processors.>
 *
 * @param pfilename  Pointer to the file, returned by fopen().
 * @param phash      Hash result buffer. See CryptSHA1Digest8Type for details.
 *
 * @return OK/ERROR
 */

int CSHA1Provider::CalculateSHAFile(const char *pfilename, \
                                    CryptSHA1Digest8Type *phash)
{
    int ierr;

    /* Init */

    ierr = ERROR;

    /* The file exists, open it as a binary. */

    if (likely(pfilename))
    {
        if (likely(*pfilename))
        {
            FILE *pfile = fopen (pfilename, "rb");

            if (likely(pfile))
            {
                ierr = CalculateSHAFile(pfile, phash);

                fclose(pfile);
            }
        }
    }

    return(ierr);
}

/* sha1.cpp */
```

List of figures