AN13643

i.MX RT Industrial Drive Development Platform software overview

Rev. 1.1 — 12 September 2022 Application note

Document information

Information	Content
Keywords	i.MX, RT1170, ISA/IEC 62443, multi-motor control, software
Abstract	This document provides an architectural and functional overview of the software included in the i.MX RT Industrial Drive Development Platform product support package. The i.MX RT Industrial Drive Development Platform software demonstrates how to develop secure industrial multi-motor applications, therefore reducing the development effort and time to market of industrial components and solutions.

Revision history

Revision history

Revision number	Date	Description
1.0	2022-07-29	Initial version
1.1	2022-09-12	Updated part numbers for daughter card, digital board and power stage board

1 Abbreviations

Table 1. List of acronyms

Name	Description
AD	Anomaly Detection
ADC	Analogue to Digital Converter
API	Application Programming Interface
AWDG	Authenticated Watchdog Timer
AWS	Amazon Web Services
CA	Certification Authority
CAAM	Cryptographic Acceleration and Assurance Module
DMA	Direct Memory Access
FIFO	First In First Out
FW	Firmware
HAB	High Assurance Boot
HTTP	Hypertext Transfer Protocol
IAM	Identity and Access Management
IRQ	Interrupt Request
ISR	Interrupt Service Routine
lwIP	Light Weight IP
MAD	Monitoring, Analytics and Damage control
MCU	Micro Controller Unit
MQTT	Message Queuing Telemetry Transport
NVM	Non-Volatile Memory
PC	Personal Computer
POSIX	Portable Operating System Interface
PUF	Physical Unclonable Function
PWM	Pulse Width Modulation
RDC	Resource Domain Controller
RNG	Random Number Generator
ROM	Read-Only Memory
RPC	Remote Procedure Call
RTC	Real Time Clock
SBL	Secure / Secondary Boot Loader
SCP	Secure Channel Protocol
SDP	Serial Download Protocol
SIMW	Secure IoT Middleware
SNMP	Simple Network Management Protocol
SPI	Serial Peripheral Interface
SSL	Secure Sockets Layer
SSS	Secure Sub-System
TCM	Tightly Coupled Memory
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TSN	Time Sensitive Network
XiP	Execute in Place

2 i.MX RT Industrial Drive Development Platform

The i.MX RT Industrial Drive Development Platform is a flexible and cost-effective modular-board kit that speeds up the development, evaluation and validation of complex multi-motor control applications for industrial robots, mobile robotics, multi-axis machineries, digital manufacturing, and many other industrial use cases.

The i.MX RT Industrial Drive Development Platform demonstrates how an **NXP i.MX RT1170 crossover MCU** can be leveraged to simultaneously control up to four Permanent Magnet Synchronous Motors (PMSMs) while at the same time handling advanced functionalities such as data logging, fault detection, deterministic connectivity (Ethernet TSN) and complex user interfaces. By leveraging on **NXP EdgeLock SE05x secure element** for the secure storage of keys and credentials, the i.MX RT1170 MCU also supports strong cybersecurity based on the latest, most secure cryptographic algorithms and protocols, therefore enabling the path to achieve some the highest security levels of the *ISA/IEC 62443-4-2* industrial standard.

The i.MX RT Industrial Drive Development Platform comes with a fully-featured hardware and software package that allows users to quickly start developing multi-motor control applications:

- The **i.MX RT Industrial Drive Development Platform hardware package** consists of a daughter card integrating the i.MX RT1170 crossover MCU, a digital board to expand the interfaces available to the daughter card and a power stage board to transform control commands into power signals to drive a servo motor. All the boards can be configured and adapted to meet the specific requirements of the application that is being developed. More information on the hardware package can be found in *i.MX RT Industrial Drive Development Platform hardware overview* document.
- The **i.MX RT Industrial Drive Development Platform software package** consists of a reference demo application and API that demonstrate how to take advantage of i.MX RT Industrial Drive Development Platform hardware capabilities to develop a secure, robust and reliable multi-motor control system which meets the requirements, standards and best practices required by commercial industrial products. This significantly reduces the effort required to develop multi-motor control applications and the time-to-market of the product. This document provides an overview of the i.MX RT Industrial Drive Development Platform software package.

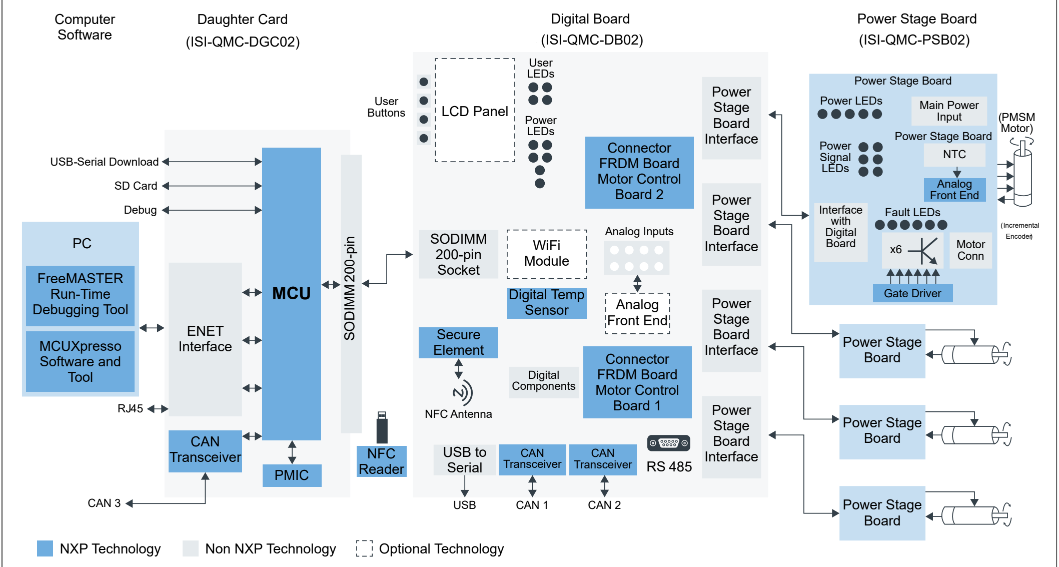


Figure 1. Hardware architecture of i.MX RT Industrial Drive Development Platform

Table 2. Hardware components of i.MX RT Industrial Drive Development Platform

Component	Part number
MCU	NXP i.MX RT1176
PMIC	NXP PF5020
Secure Element (SE)	NXP EdgeLock SE05x
Analog frontend	NXP NAFE 13388
CAN transceiver	NXPTJA115x (daughter card) NXP TJA146x (digital board)
Temperature sensor	NXP PCT2075
Gate driver	NXP GD3000
NFC reader	NXP PN7462

2.1 Introducing the i.MX RT Industrial Drive Development Platform software package

The i.MX RT Industrial Drive Development Platform software package demonstrates how to implement **secure multi-motor control applications** that leverage on the **i.MX RT1170 MCU** and the hardware, interfaces and features offered by the i.MX RT Industrial Drive Development Platform.

The i.MX RT Industrial Drive Development Platform software consists of several ***software components and libraries*** that implement functionalities for motor control, fault handling, data logging and deterministic communication, among many others. A fully-featured and reusable ***API*** exposes use-case specific functions and provides a convenient way for other software components and applications to interact using well-defined software interfaces. This also allows developers to focus on building the application logic without having to deal with complex low-level details and functions.

The different software components and APIs form the base of a ***demo application*** that demonstrates how to implement a motor control system that can be securely monitored in real-time from a cloud service and controlled from a local PC using either a serial communication or a web interface.

The i.MX RT Industrial Drive Development Platform software is built with security at its core. By leveraging on secure development practices, hardware storage of keys and credentials using EdgeLock SE05x and strong cryptographic algorithms and protocols, the i.MX RT Industrial Drive Development Platform software provides users with a reference *ISA/IEC 62443-4-2 SL3* compliant solution that can be used as a starting point to develop secure industrial products and components.

***Note:** the i.MX RT Industrial Drive Development Platform and software does not automatically grants ISA/IEC 62443-4-2 certification to a product.*

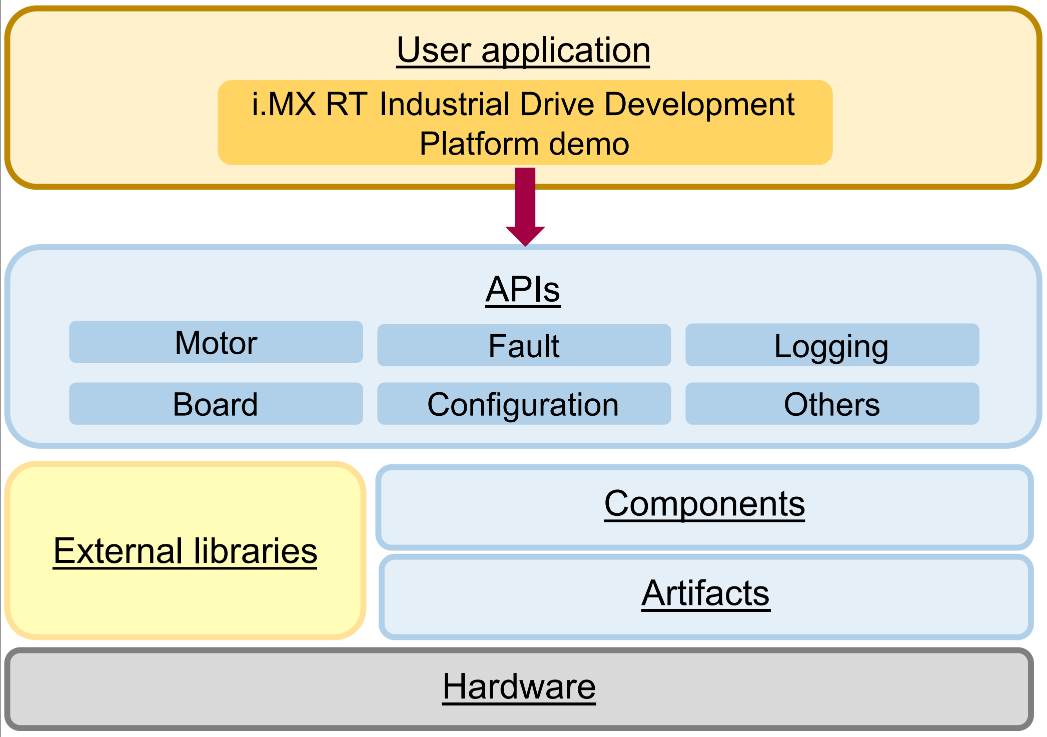


Figure 2. i.MX RT Industrial Drive Development Platform overview

2.2 Note on ISA/IEC 62443-4-2 certified software

The i.MX RT Industrial Drive Development Platform software is built with security at its core. By leveraging on secure development practices, hardware storage of keys and credentials using EdgeLock SE05x and strong cryptographic algorithms and protocols, the i.MX RT Industrial Drive Development Platform software provides users with a *ISA/IEC 62443-4-2 SL3* compliant solution that can be used as-is or as a reference to develop certified industrial products and components.

The i.MX RT Industrial Drive Development Platform software is available in two different versions:

- **Non-hardened:** with this version of the software users will have the possibility of configuring the features that they would like to enable in the platform. This means that they can disable some or all security features and enable interfaces that would typically be disabled in a production environment (e.g. debugging ports or interfaces). This version of the software is ideal for development or debugging purposes or for users who are not interested in ISA/IEC 62443-4-2 certification.

- **Hardened:** this version of the software has been designed to be compliant with ISA/IEC 62443-4-2 Security Level 3 (SL3). When running this version, a mandatory set of security features is enabled and development/debugging interfaces are disabled. The hardened software provides a reference implementation for users that want their product certified for ISA/IEC 62443-4-2 SL3.
  Note: *the i.MX RT Industrial Drive Development Platform and software does not automatically grants ISA/IEC 62443-4-2 certification to a product.*

Running the different versions of the software can be achieved through compilation options as described in Section 6.

2.3 How to use this document

This document provides an overview of the i.MX RT Industrial Drive Development Platform software and is meant to be used as a starting point to understand its architecture and main components. The document is structured as follows:

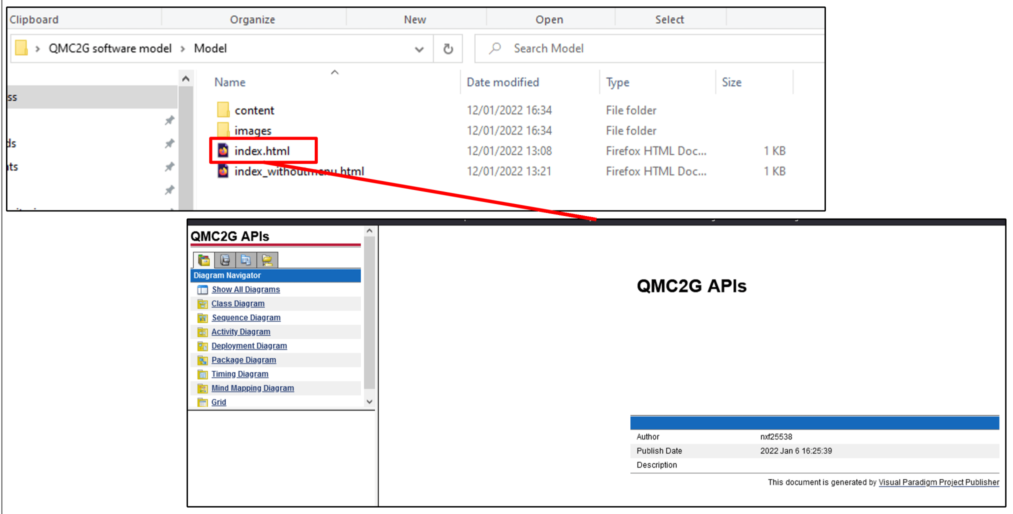
- Section 3 describes the architecture of the i.MX RT Industrial Drive Development Platform software, its main components, APIs and security features.
- Section 4 provides an overview of the software components and artifacts that implement the functionality of the i.MX RT Industrial Drive Development Platform software.
- Section 5 provides an overview of the API that is implemented by the i.MX RT Industrial Drive Development Platform software and that can be leveraged by applications to implement use-case specific functionalities.

This document is complementary to the **i.MX RT Industrial Drive Development Platform software model** that you can download from the i.MX RT Industrial Drive Development Platform website. The i.MX RT Industrial Drive Development Platform software model is an interactive HTML documentation that is easy to navigate using any web browser. It provides descriptions of i.MX RT Industrial Drive Development Platform software components, data structures and APIs, as well as diagrams showing how components are implemented and how they interact to realize a specific functionality.

Follow these instructions to learn how to download and use the i.MX RT Industrial Drive Development Platform software model:

1. Go to the i.MX RT Industrial Drive Development Platform website and download the i.MX RT Industrial Drive Development Platform software model. Unzip the content of the ZIP file in a location of your choice. You should see a *Model* folder where you unzipped the file.

2. Open the *Model* folder, then double-click on the *index.html* file to open the model in your web browser as shown in Figure 3:
   ***Note**: the QMC2G is used in the software code and documentation as an equivalent way of referring to the i.MX RT Industrial Drive Development Platform.*



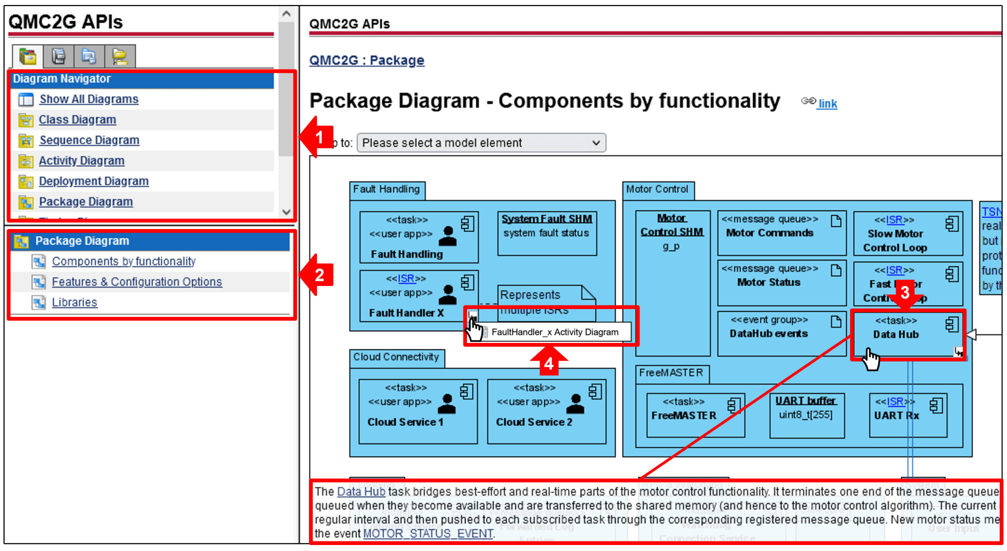
Name	Date modified	Type	Size
content	12/01/2022 16:34	File folder	
images	12/01/2022 16:34	File folder	
index.html	12/01/2022 13:08	Firefox HTML Doc...	1 KB
index_withoutmenu.html	12/01/2022 13:21	Firefox HTML Doc...	1 KB

Author	nxf25538
Publish Date	2022 Jan 6 16:25:39
Description	

This document is generated by Visual Paradigm Project Publisher

Figure 3. Open the i.MX RT Industrial Drive Development Platform software model

3. The model can be navigated by following the available links as shown in Figure 4:
   (1) Click on the links in the *diagram navigator* panel to open the different sections of the model.
   Note: *the diagram navigator is the recommended way of navigating the model. The other tabs are empty.*
   (2) After clicking on a section in the *diagram navigator*, you can select a subsection using the links that appear just below the *diagram navigator*.
   (3) Most sections include interactive diagrams. You can hover over a diagram element to see a short description. Click on the element to access the page with the complete information about the element.
   Note: *element descriptions are also provided in a list right below the diagram.*
   (4) If additional resources are available for an element, a white arrow will appear on the bottom right corner of the element. Click on it to open a popup with links to the additional resources.


QMC2G APIs
Diagram Navigator
Show All Diagrams
Class Diagram
Sequence Diagram
Activity Diagram
Deployment Diagram
Package Diagram
Package Diagram
Components by functionality
Features & Configuration Options
Libraries
QMC2G APIs
QMC2G : Package
Package Diagram - Components by functionality
link
Please select a model element
1
2
3
4
Fault Handling
<<task>> <<user app>> Fault Handling
System Fault SHM
system fault status
<<ISR>> <<user app>> Fault Handler X
Represents multiple ISRs
FaultHandler_x Activity Diagram
Motor Control
Motor Control SHM
g_p
<<message queue>> Motor Commands
<<ISR>> Slow Motor Control Loop
<<message queue>> Motor Status
<<ISR>> Fast Motor Control Loop
<<event group>> DataHub events
<<task>> Data Hub
Cloud Connectivity
<<task>> <<user app>> Cloud Service 1
<<task>> <<user app>> Cloud Service 2
FreeMASTER
<<task>> FreeMASTER
UART buffer
uint8_t[255]
<<ISR>> UART Rx

The Data Hub task bridges best-effort and real-time parts of the motor control functionality. It terminates one end of the message queue queued when they become available and are transferred to the shared memory (and hence to the motor control algorithm). The current regular interval and then pushed to each subscribed task through the corresponding registered message queue. New motor status me the event MOTOR_STATUS_EVENT.

Figure 4. Use the i.MX RT Industrial Drive Development Platform software model

3 Architecture overview of i.MX RT Industrial Drive Development Platform software

Starting from the highest possible abstraction level, the i.MX RT Industrial Drive Development Platform software consists of a **demo application** that leverages on i.MX RT Industrial Drive Development Platform hardware, and several underlying software resources, to demonstrate how to implement a secure multi-motor system that can be controlled from a local PC and monitored in real-time from a cloud service. The high-level architecture of the demo, once deployed, is shown in Figure 5.

As outlined in the diagram, the demo application relies on i.MX RT Industrial Drive Development Platform **Ethernet TSN support** to exchange both best effort and time-sensitive data with local PCs and servers and with the cloud through secure TLS connections. Upon successful connection to a cloud service, such as AWS or Azure, the application pushes motor status updates to the cloud using the MQTT protocol.

Local communication is implemented using the **NXP FreeMASTER driver** to exchange status data and motor control commands over a serial interface. Data and configurations can be set and visualized on a PC through the NXP FreeMASTER PC application. The demo application also runs a web service that allows up to five TLS connections. It serves static web pages that can then be accessed through a standard web browser and used to send motor commands and control the status of the system upon successful authentication. Different functions are available depending on the permissions granted to the logged in user.

A **secure watchdog service**, running on a separate server, can be used to monitor the i.MX RT Industrial Drive Development Platform from outside and reboot it in restricted mode in case the software hangs or the system fails. A secure connection is established between the i.MX RT Industrial Drive Development Platform and the watchdog server. Tickets, cryptographically signed by the secure watchdog, are periodically exchanged between the two entities to keep the system running. If for any reason the device stops requesting tickets or the watchdog server decides to disable ticket sending (e.g. because it does not trust anymore the device), then the system reboots in restricted mode.

All the functionalities of the demo outlined above are implemented by several software components which, through a public API, can access the different hardware and software resources made available by the i.MX RT Industrial Drive Development Platform: motor control, fault handling, logging, anomaly detection, etc.

This chapter provides a high-level overview of the different i.MX RT Industrial Drive Development Platform software components and describes how these components interact using the i.MX RT Industrial Drive Development Platform APIs.

Note: *some demo configurations, such as TSN support or FreeMASTER support, can be changed, enabled or disabled. Depending on the configuration, some components of the demo deployment might not be required. Refer to for more information on available configurations.*

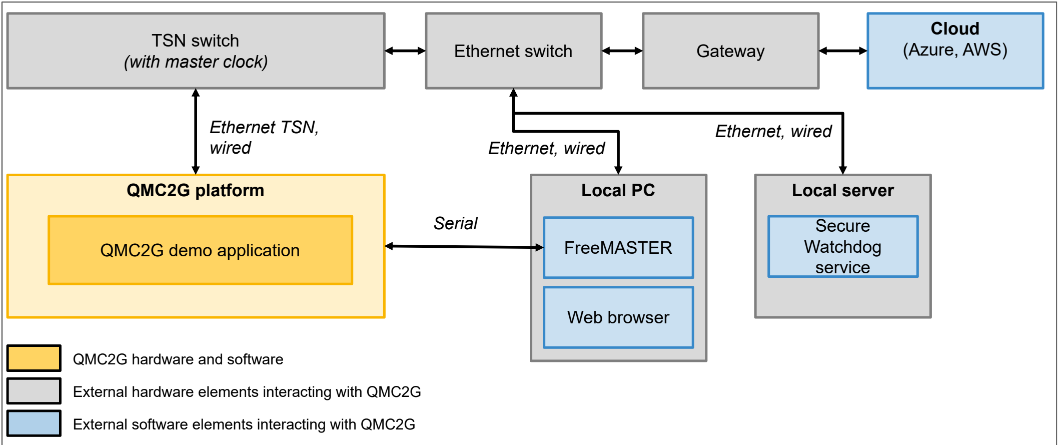


Figure 5. i.MX RT Industrial Drive Development Platform demo application deployment

3.1 Functional overview of i.MX RT Industrial Drive Development Platform software components

The i.MX RT Industrial Drive Development Platform software is a **C software** that leverages on the ***FreeRTOS operating system*** for creating and managing the different software components that together implement the functionality of the demo. Some functionalities, such as TCP/IP connectivity and cryptographic operations, are implemented using **external libraries** linked to the i.MX RT Industrial Drive Development Platform software components (see Section 7).

In the context of this document, a software component can be one of the following entities:

- **Task:** an independent function, managed by FreeRTOS, that runs on an infinite loop and implements a particular functionality. For example, a task may be responsible for implementing the logging functionality of the application.
- **Interrupt Service Routine (ISR):** a function run by the processor in response to a given interrupt event raised by hardware or software. For example, an ISR can be executed in response to a hardware fault.

Tasks and ISRs can interact with other entities, called *artifacts*, which may include:

- **Message queues:** data structures used for data communication between tasks. A message queue behaves similarly to a FIFO queue.
- **Event groups:** a set of event bits describing the state of the application (or of an aspect of the application) in a particular moment in time. These event bits can be set or retrieved by any task or ISR.
- **Shared memory (SHM)**: a type of memory that can be shared by multiple tasks to share or pass data between them.

The software components and artifacts that together form the i.MX RT Industrial Drive Development Platform software, can be logically grouped based on the functionality they implement as summarized in Figure 6:

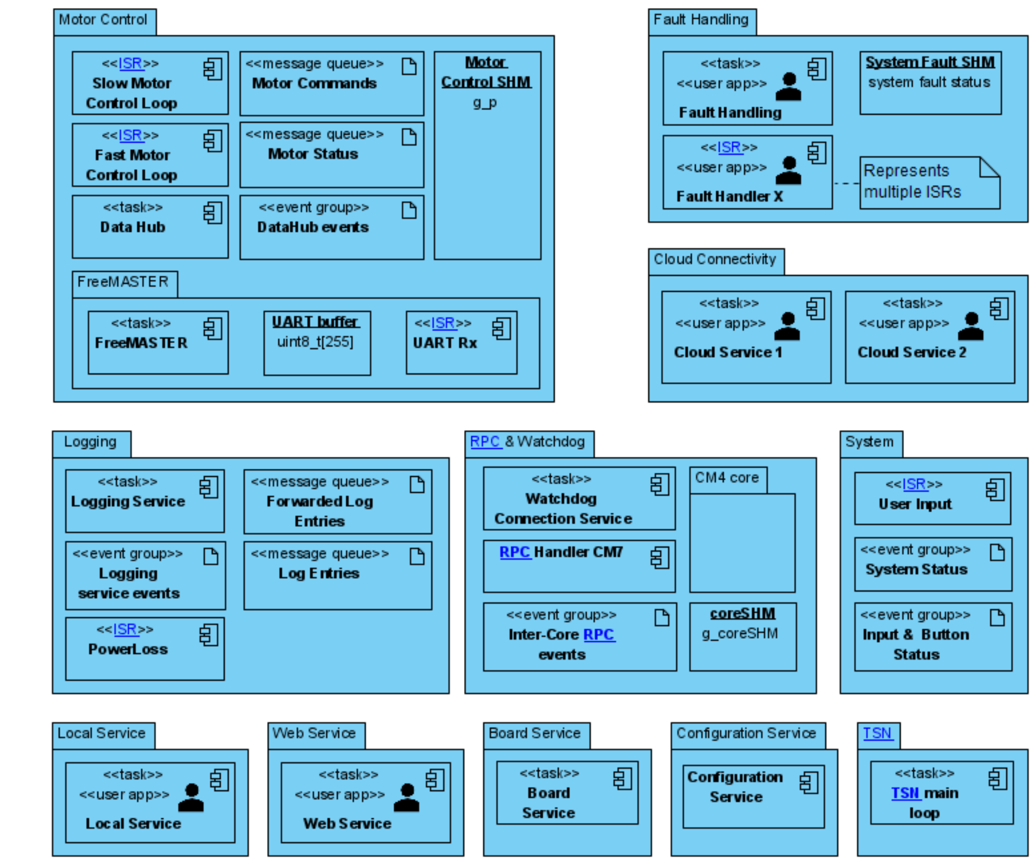

Motor Control
<<ISR>> Slow Motor Control Loop
<<message queue>> Motor Commands
Motor Control SHM g_p
<<ISR>> Fast Motor Control Loop
<<message queue>> Motor Status
<<task>> Data Hub
<<event group>> DataHub events
FreeMASTER
<<task>> FreeMASTER
UART buffer uint8_t[255]
<<ISR>> UART Rx
Fault Handling
<<task>> <<user app>> Fault Handling
System Fault SHM system fault status
<<ISR>> <<user app>> Fault Handler X
Represents multiple ISRs
Cloud Connectivity
<<task>> <<user app>> Cloud Service 1
<<task>> <<user app>> Cloud Service 2
Logging
<<task>> Logging Service
<<message queue>> Forwarded Log Entries
<<event group>> Logging service events
<<message queue>> Log Entries
<<ISR>> PowerLoss
RPC & Watchdog
<<task>> Watchdog Connection Service
CM4 core
RPC Handler CM7
<<event group>> Inter-Core RPC events
coreSHM g_coreSHM
System
<<ISR>> User Input
<<event group>> System Status
<<event group>> Input & Button Status
Local Service
<<task>> <<user app>> Local Service
Web Service
<<task>> <<user app>> Web Service
Board Service
<<task>> Board Service
Configuration Service
Configuration Service
TSN
<<task>> TSN main loop

Figure 6. Software components by functionality

- **Motor control**: these components are responsible for handling motor control commands and dealing with different aspects of motor control such as PWM control, phase current measurements and DC bus voltage measurements. A central *data hub* acts as interface for other components to send motor control commands which are then saved in shared memory and executed by the motor control algorithms. It also pushes motor status updates to other components when available. The FreeMASTER task handles serial communication through the FreeMASTER driver.
- **Fault handling**: these components react to fault events occurring in the system and determine the best action to take in response. Faults may be software faults, e.g. raised by parameter checks performed by the motor control components, or hardware faults e.g. a PWM fault.
- **Cloud connectivity:** these components implement the cloud connectivity functionality of the demo application. They allow the i.MX RT Industrial Drive Development Platform to connect to a cloud service (Azure or AWS) using a secure TLS connection and to push motor status updates using the MQTT protocol. Up to two concurrent connections to the same cloud service are supported.
- **Logging:** these components handle logs that are generated by other software components and write them in the appropriate location (e.g. internal flash memory or SD card).
- **RPC & watchdog:** these components are responsible for providing access to features implemented on the i.MX RT1170 Cortex M4 core, like SPI slave selection, RTC and secure watchdog using Remote Procedure Calls (RPCs).

- **System management:** these components are responsible for keeping the status of the system and updating the user input status (e.g. buttons pressed or released) based on user interaction with the i.MX RT Industrial Drive Development Platform input interfaces.
- **TSN:** this functionality is handled by a single component responsible for sending and receiving TSN packets. If motor control commands are received they are sent to the appropriate motor control components. The component also takes care of sending the motor and system status at a regular interval to the TSN network.
- **Anomaly detection:** these components run the anomaly detection algorithm. The algorithm uses machine learning techniques and samples from the microphone to detect and react to anomalies in the motor.
  ***Note**: the anomaly detection component will be released as part of future software releases.*
- **Others:** components are provided to deal with other operations such as reacting to user inputs and update the display (*local service*), implementing the web service functionality of the demo (*web service*), set and retrieve software configurations (*configuration service*) and monitor the board status and temperature (*board service*).

3.2 Overview of software components interaction through APIs

The i.MX RT Industrial Drive Development Platform software components interact using Application Program Interfaces (APIs). APIs are meant to simplify the development of applications by providing well-defined interfaces that abstract the low-level complexity of the i.MX RT Industrial Drive Development Platform. **API functions are implemented by tasks and can be called by other components, such as other tasks or ISRs, to access the functionality that is exposed.** The i.MX RT Industrial Drive Development Platform demo application described in Section 3 is completely built using the public API functions exposed by i.MX RT Industrial Drive Development Platform software components.

The APIs provided as part of the i.MX RT Industrial Drive Development Platform software package are listed in Table 3.

Table 3. i.MX RT Industrial Drive Development Platform APIs

Name	Description
Motor API	Provides functions to access the motor control algorithm. It allows applications to queue motor control commands which are then prepared for execution. The motor API can only be called by other tasks and not by ISRs.
Motor API ISR	This API is used internally by the i.MX RT Industrial Drive Development Platform software to read or write the motor control status in shared memory. These functions can be called by both tasks and ISRs.
Logging API	Provides functions to queue new log entries and retrieve log records.
Fault API	Provides access to the fault handling system. These functions can only be called from tasks.
Fault API ISR	Provides access to the fault handling system. These functions can only be called from ISRs.
User management API	Provides functions to change user configuration such as adding/ removing a user from the system or checking user credentials.
Configuration API	Provides functions to access the application configurations as key-value pairs. It allows to get or set the value of i.MX RT Industrial Drive Development Platform software configurations.

Table 3. i.MX RT Industrial Drive Development Platform APIs*...continued*

Name	Description
Board API	Provides access to board-specific features implemented on the i.MX RT1170 Cortex M7 core, like temperature sensors, display, etc. It also provides functions for initialization e.g. pinmux setting, clock setting, etc.
RPC API	Provides access to features implemented on the i.MX RT1170 Cortex M4 core, like SPI slave selection, RTC and secure watchdog.
Anomaly detection API	Provides access to the anomaly detection model executed by the Anomaly Detection Service. ***Note****: the anomaly detection API will be released as part of future software releases.*

An overview of all software components, and how these components interact through the available APIs is shown in the diagram depicted in Figure 7. Table 4 describes how to read the diagram. An interactive version of the diagram is available in the i.MX RT Industrial Drive Development Platform software model in the *deployment diagram → subsystems* section.

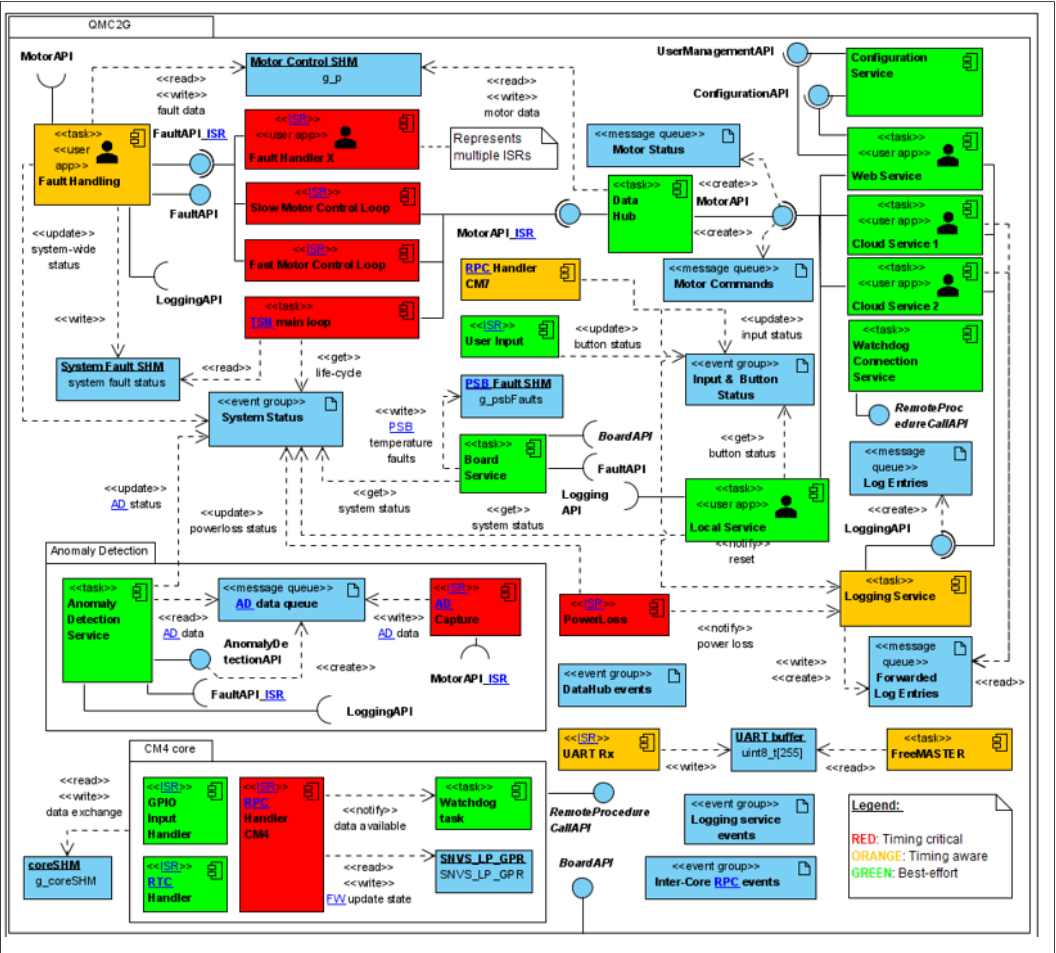
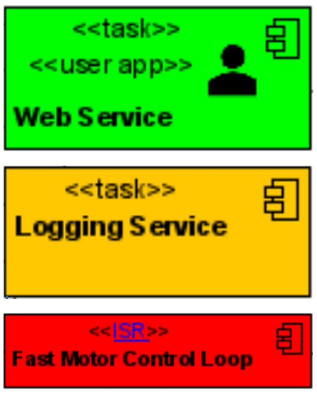
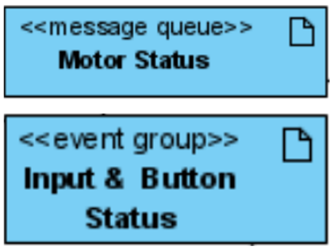
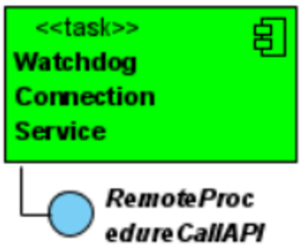
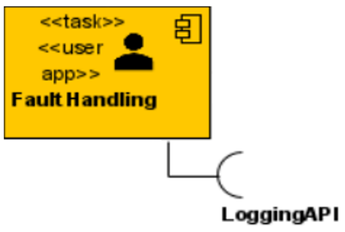


Figure 7. Software components and interaction through APIs

Table 4. Legend

Symbol	Description
 <<task>> <<user app>> Web Service <<task>> Logging Service <<ISR>> Fast Motor Control Loop	Green, red and yellow rectangles represent software components (tasks or ISRs). The name of the component is written in bold, while the type of the component (task or ISR) is specified between <<>>. The color of the rectangle defines the priority assigned to the component: red for timing critical components which must be executed with high priority, yellow for time-aware components and green for low-priority components that will be executed using a best-effort policy. Some tasks are identified by a user icon and are additionally marked with the **<<`user app`>> label**. These tasks implement the functionality of the i.MX RT Industrial Drive Development Platform demo by leveraging on the public API that is exposed by the other components.
 <<message queue>> Motor Status <<event group>> Input & Button Status	Blue rectangles represents artifacts (event groups and message queues) or other data structures (e.g. shared memory).
 <<task>> Watchdog Connection Service RemoteProcedureCallAPI	Blue circles represent APIs that are implemented by the task to which the circle is connected to. The name of the API is written in bold next to the circle.
	Empty semicircles, connected to a software component, indicate that the software component is using an API, implemented by another task, to access some functionality. The name of the API that is used is written in bold next to the semicircle.

Note: *the diagram shown in Figure 7 is not meant to provide a detailed and complete description of all components, artifacts and APIs in the i.MX RT Industrial Drive Development Platform software. Some elements might not be included and relations between elements might have been simplified or removed to improve legibility.*

3.3 Overview of main software security features

The i.MX RT Industrial Drive Development Platform has been built with security at its core. In fact, the i.MX RT Industrial Drive Development Platform hardware and software has been designed to comply with ISA/IEC 62443-4-2 standard aiming for one of the highest security levels in the standard (*SL3*). To achieve this, several security features have been implemented in the i.MX RT Industrial Drive Development Platform software. The main blocks are listed below:

Note: *security features can be enabled/disabled using software configurations. The ISA/IEC 62443 compliant version of the software (see Section 2.2) automatically activates all security features required to comply with ISA/IEC 62443-4-2 SL3 requirements.*

- **Secure boot**: the i.MX RT Industrial Drive Development Platform implements a Secure Boot Loader (SBL) whose integrity, authenticity and confidentiality is guaranteed by the High Assurance Boot (HAB) and Encrypted XiP managed by the i.MX RT1170 boot ROM. The authenticity and integrity of the main firmware is verified, before loading, using EdgeLock SE05x. For additional security, the EdgeLock SE05x SE is bonded to the SBL using Secure Channel Protocol 03 (SCP03).
- **Secure (remote) update:** the i.MX RT Industrial Drive Development Platform software supports OTA updates of the firmware. The updated firmware is sent to i.MX RT Industrial Drive Development Platform using a secure channel. The firmware update image is pre-authenticated by the main application firmware to make sure the firmware update image is valid. The verification of the firmware update image is handled by EdgeLock SE05x. The new firmware is encrypted and verified by EdgeLock SE05x as well. In case of any issues, the firmware update image is erased from the storage and the event logged and notified. In case of successful pre-authentication, the control is passed to SBL to finalize the firmware update process. Local update using SD card is also supported and is directly handled by the SBL.
- **Logging:** log records are signed and encrypted using keys stored in EdgeLock SE05x before saving them to the SD card or sending them to an external service (e.g. a cloud service).
- **TCP/IP connections**: whenever a TCP/IP connection is required, e.g. to communicate with an external cloud service, a secure TLS connection is always established to ensure data confidentiality and authentication.

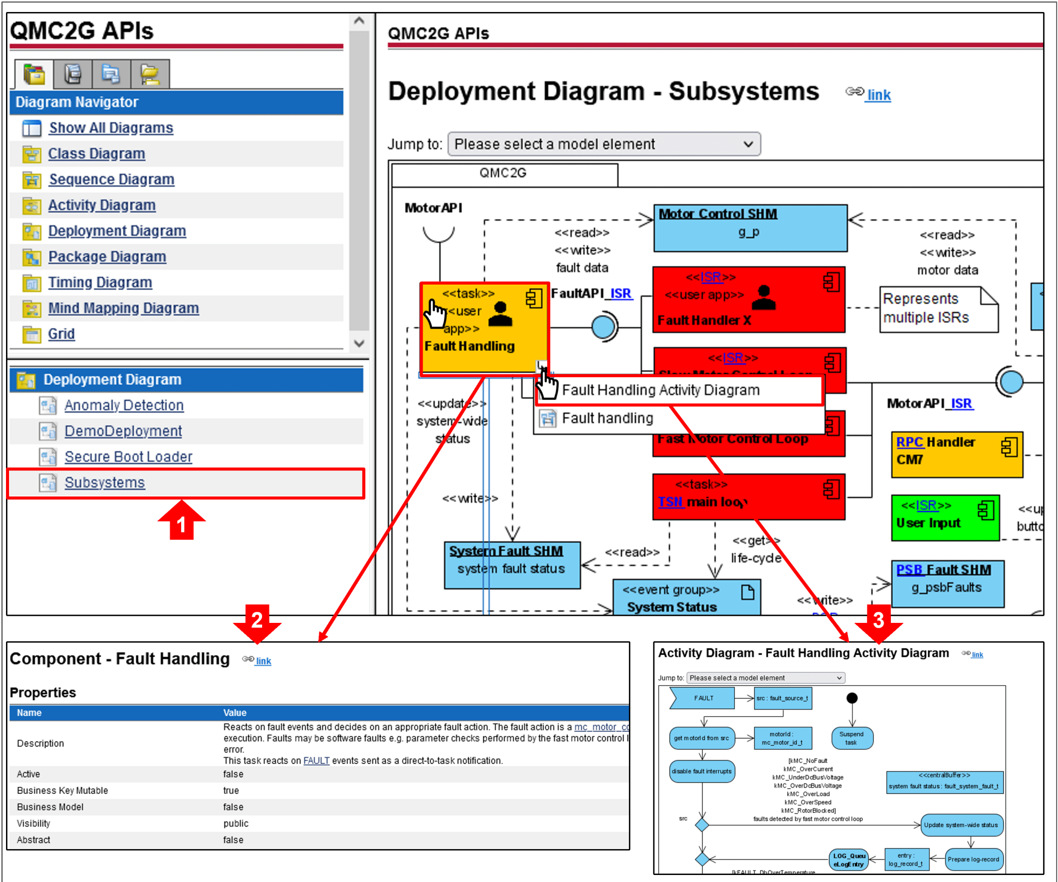
4 Software components and artifacts

This section lists the components and artifacts of the i.MX RT Industrial Drive Development Platform software grouped by functionality. For each component or artifact a brief description is provided.

- Motor control and TSN
- Fault handling
- Cloud connectivity
- Logging
- RPC and watchdog
- System
- Anomaly detection
- Others

More detailed information about a particular component or artifact, including activity diagrams, sequence diagrams and other resources, can be access from the i.MX RT Industrial Drive Development Platform software model as shown in Figure 8:

1. Open the i.MX RT Industrial Drive Development Platform software model and navigate to *deployment diagram* → *subsystems*.
2. Click on the component/artifact that you are interested in to access the page with the description of the component.
3. Click on the white arrow in the bottom right corner of the element to access additional resources. For example, the *fault handling* task additionally provides an activity diagram depicting the state machine of the task.



Component - Fault Handling link

Properties

Name	Value
Description	Reacts on fault events and decides on an appropriate fault action. The fault action is a mc_motor_c execution. Faults may be software faults e.g. parameter checks performed by the fast motor control l error. This task reacts on FAULT events sent as a direct-to-task notification.
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Abstract	false

Figure 8. Access components/artifacts resources in i.MX RT Industrial Drive Development Platform software model

4.1 Motor control and TSN

These components and artifacts deal with the motor control and TSN functionality of the i.MX RT Industrial Drive Development Platform. Components are highlighted in Figure 9.

The *data hub* task implements the *Motor API* and *Motor API ISR* and it acts as interface for all components to queue motor commands in the *motor commands* message queue. Commands are then sent to shared memory (*Motor_Control_SHM*) where they are executed by the motor control algorithm. The *data hub* also pushes motor status at regular interval to the *motor status* message queue(s) which have been previously registered with the data hub through the *Motor API*. A queue must be registered for each task that wants to subscribe to motor status updates.

The *slow motor control loop* and *fast motor control loop* components are called repeatedly for each motor. They put *motor status* updates into shared memory where they can be fetched and queued by the *datahub* task. Queue events can be detected by other tasks by reading the bits set in the *data hub events* event group.

The *FreeMASTER* task interfaces through serial communication with the corresponding PC application. Commands are received by *UART Rx* ISR and pushed to the *UART buffer* where they are finally read by *FreeMASTER* task and queued for execution. The FreeMASTER task has access to all global variables, including the inner variables of *slow motor control loop* and *fast motor control loop*.

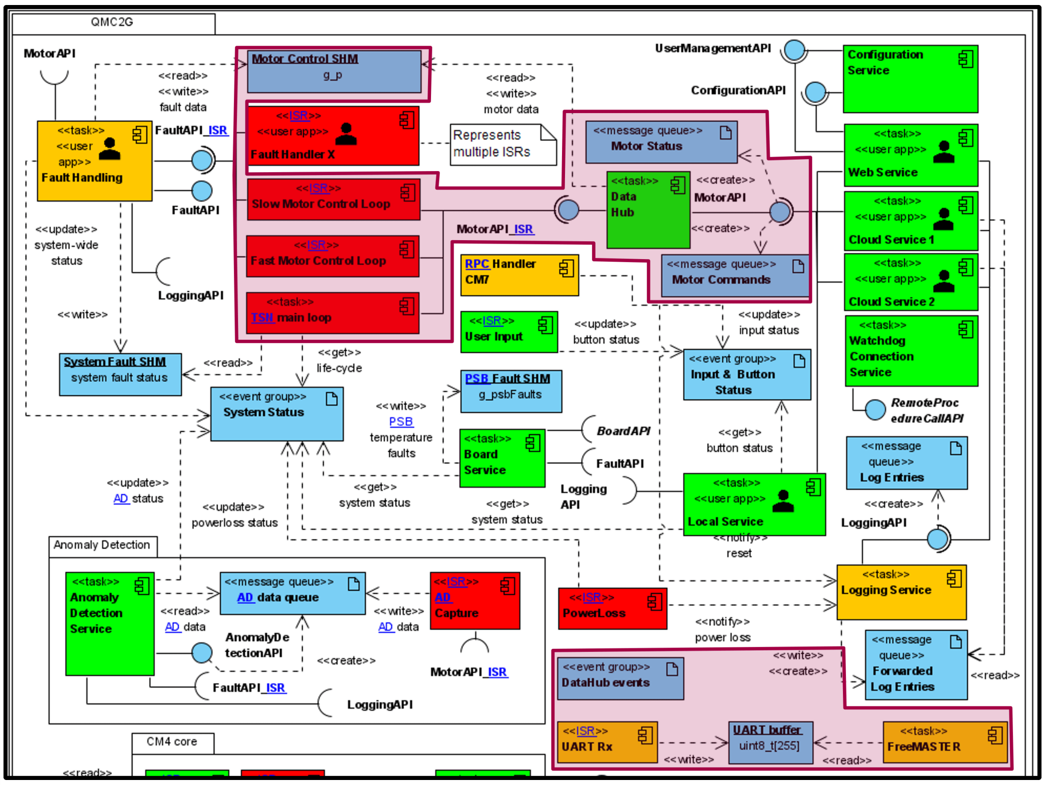


Figure 9. Motor control and TSN components and artifacts overview

Table 5. Motor control and TSN components

Name	Type	Description
Data hub	Task	The Data Hub task bridges best-effort and real-time parts of the motor control functionality. It terminates one end of the message queues of the *Motor API*. Motor commands are de-queued when they become available and are transferred to the shared memory (and hence to the motor control algorithm). The current motor status for each motor is polled in a regular interval and then pushed to each subscribed task through the corresponding registered message queue. New motor status messages in the queue are announced by emitting the event *MOTOR_STATUS_EVENT*.
Slow motor control loop	ISR	Deals with the slow aspects of motor control like speed control and position control. It is reading motor commands from shared memory and executes them as well as updating part of the motor status values by using *MotorAPI_ISR*.
Fast motor control loop	ISR	Deals with the fast aspects of motor control like current control. It is updating part of the motor status values by using *MotorAPI_ISR*. Also, it can update the fault status of a motor and emit FAULT events by using *FaultAPI_ISR*.
TSN main loop	ISR	Handles sending and receiving TSN packets. Whenever a packet via TSN is received, that contains new motor commands, it sets the motor commands for execution using *MotorAPI_ISR*. Also, it sends the motor and system status at a regular interval to the TSN network.

Table 5. Motor control and TSN components*...continued*

Name	Type	Description
FreeMASTER	Task	This task runs the back-end part of the FreeMASTER application. It has access to a dedicated serial port (UART) to connect to a corresponding front-end application running on a PC. A compile-time configuration option allows users to disable this task if required. FreeMASTER has direct access to all global variables, including the inner variables of the *Fast Motor Control Loop* and *Slow Motor Control Loop*.
UART Rx	ISR	UART receive handler. Writes incoming data to UART buffer for further processing by FreeMASTER. This handler is only instantiated and registered if FreeMASTER is enabled in the build-configuration.

Table 6. Motor control and TSN artifacts

Name	Type	Description
Motor commands	Message queue	Message queue for motor commands. Elements get queued by using the *MC_QueueMotorCommand* call of the Motor API. The Data Hub then de-queues the commands and transfers them to the shared memory for execution.
Motor status	Message queue	Message queues for motor status values. Elements get queued by the Data Hub. A task can de-queue them by using the *MC_DequeueMotorStatus* call of the *Motor API*. Each task that wants to receive motor status values must register their own queue with the Data Hub by using the *Motor API*. Maximal number of queues available can be defined at compile-time; each queue has its own event bit.
Data hub events	Event group	This event group holds the event bits for the message queues and timers used by the Data Hub. A set event bit indicates that the queue contains new messages for reading or the timer has elapsed, respectively. Consult the i.MX RT Industrial Drive Development Platform software model for a description of all event bits of this event group.
Motor control SHM	Other	Shared memory that holds motor commands and motor status values for each motor. It can be read from and written to by using *Motor API, MotorAPI_ISR and FaultAPI_ISR*. For a detailed description of the shared memory, refer to its data type definition (*mc_control_SHM*) in i.MX RT Industrial Drive Development Platform software model.
UART buffer	Other	Data buffer for incoming UART communication. This buffer is only instantiated if FreeMASTER is enabled in the build-configuration.

4.2 Fault handling

These components and artifacts deal with the fault handling functionality of i.MX RT Industrial Drive Development Platform. Components are highlighted in Figure 10.

Fault events are detected by the fast motor control loop, the board service task and in the case of buffer/queue overflow by the *Fault API*. These components then notify the fault to the *fault handling* task which takes care of sending the appropriate motor command for execution and updates the system fault status (*system fault SHM*). Additionally, the *fault handling* task also logs information about the fault.

Note: *user-defined fault handler ISRs will be available in future versions of the i.MX RT Industrial Drive Development Platform software.*

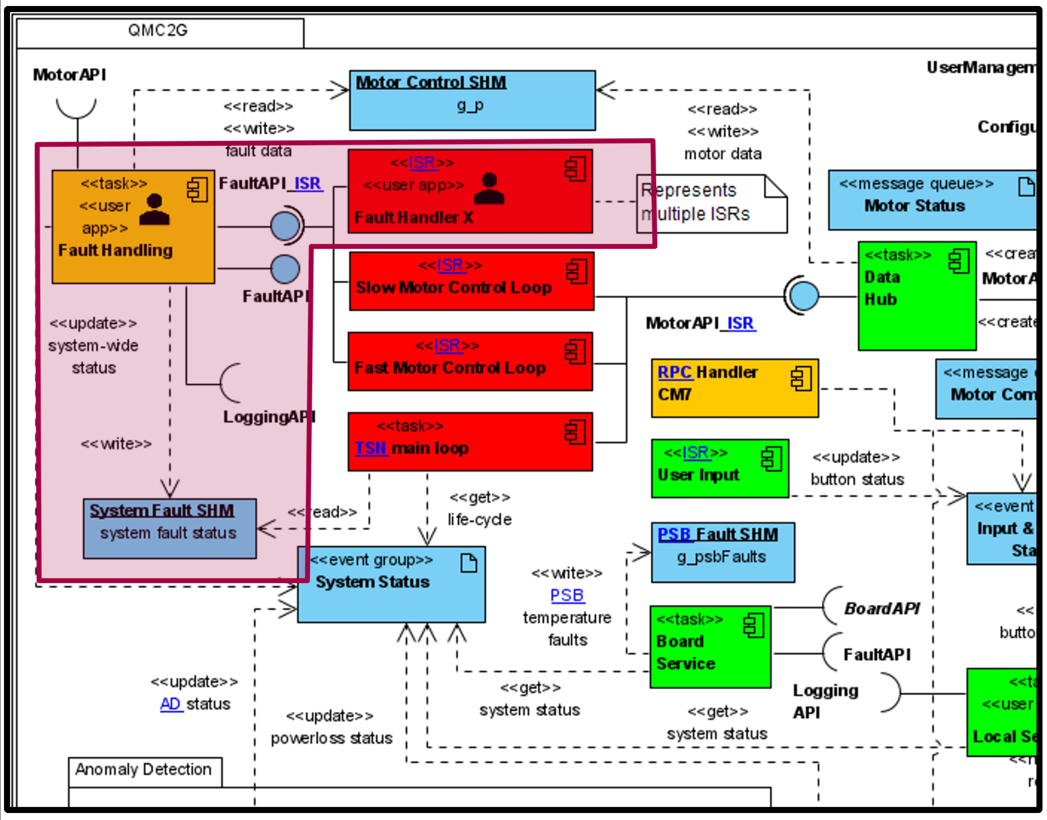


Figure 10. Fault handling components and artifacts overview

Table 7. Fault handling components

Name	Type	Description
Fault handling	Task (*user app*)	Reacts on fault events and decides on an appropriate fault action. The fault action is a motor command (*mc_motor_command_t*) sent via the *Motor API* for execution. Faults may be software faults e.g. parameter checks performed by the fast motor control loop, or hardware faults e.g. a PWM error. This task reacts on FAULT events sent as a direct-to-task notification.
Fault handler X	ISR (*user app*)	Represents a user-definable handler for a hardware fault interrupt request. There may be multiple of these handlers; each with their own behavior. It shall send a FAULT event indicating the *fault_source_t* to the fault handling task using *FAULT_RaiseFaultEvent_fromISR* from *FaultAPI_ISR*. The demo application is using this feature for the MCU over-temperature and PWM fault events only.

Table 8. Fault handling artifacts

Name	Type	Description
System fault SHM	Other	Holds the system-wide fault status. The corresponding event bit in *System Status* indicates only that the system-wide fault status is different from *kFAULT_NoFault* (no faults detected).

4.3 Cloud connectivity

These components and artifacts implement the cloud connectivity functionality of i.MX RT Industrial Drive Development Platform demo application. Components are highlighted in Figure 11.

The i.MX RT Industrial Drive Development Platform leverages on *cloud service 1* and *cloud service 2* tasks to open up to two concurrent cloud connections. Motor status updates, when made available by the *data hub* task, are pushed to the cloud using the MQTT protocol over a secure TLS connection. You can find the list of MQTT topics that the *cloud service* tasks publish to in the i.MX RT Industrial Drive Development Platform software model (*mind mapping diagram → MQTT topics*).

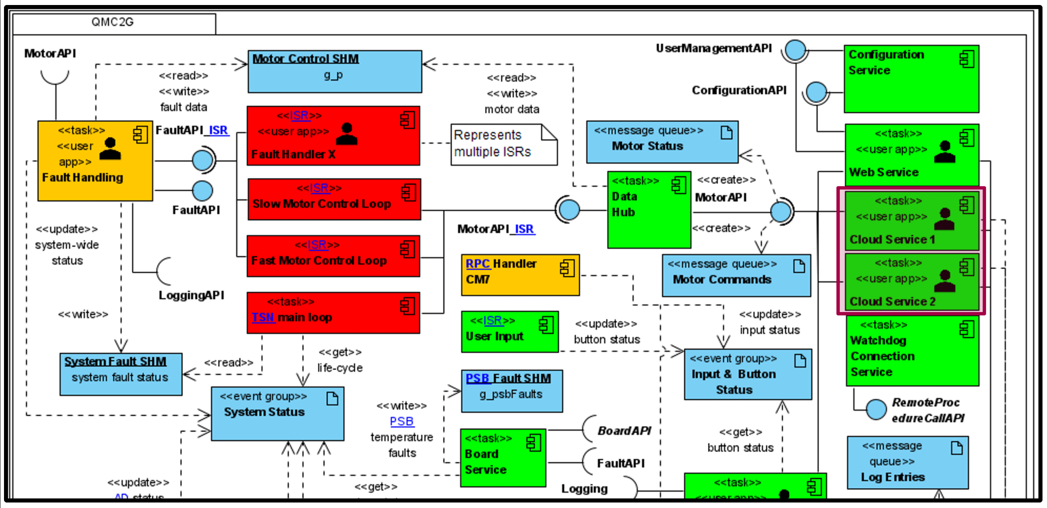

QMC2G
MotorAPI
<<read>> <<write>> fault data
Motor Control SHM g_p
<<read>> <<write>> motor data
<<task>> <<user app>> Fault Handling
FaultAPI_ISR
<<ISR>> <<user app>> Fault Handler X
Represents multiple ISRs
FaultAPI
<<ISR>> Slow Motor Control Loop
<<ISR>> Fast Motor Control Loop
<<task>> TSN main loop
MotorAPI_ISR
<<update>> system-wide status
LoggingAPI
<<write>>
System Fault SHM system fault status
<<read>>
<<get>> life-cycle
<<event group>> System Status
<<write>> PSB temperature faults
<<update>> AD status
<<get>>
RPC Handler CM7
<<ISR>> User Input
PSB Fault SHM g_psbFaults
<<task>> Board Service
BoardAPI
FaultAPI
Logging
UserManagementAPI
ConfigurationAPI
Configuration Service
<<message queue>> Motor Status
<<task>> Data Hub
<<create>>
MotorAPI
<<create>>
<<message queue>> Motor Commands
<<task>> <<user app>> Web Service
<<task>> <<user app>> Cloud Service 1
<<task>> <<user app>> Cloud Service 2
<<task>> Watchdog Connection Service
<<update>> button status
<<update>> input status
<<event group>> Input & Button Status
<<get>> button status
RemoteProcedureCallAPI
<<message queue>> Log Entries
<<task>>

Figure 11. Cloud connectivity components and artifacts overview

Table 9. Cloud connectivity components

Name	Type	Description
Cloud service 1	Task *(user app)*	If configuration parameters for a cloud service are set, the *cloud service 1* task establishes a TLS secured connection to the configured cloud service, otherwise it goes to "suspend" state. Upon a successful connection to the cloud, it serves the various MQTT topics for motor status and keeps them updated whenever it receives a motor status update from the *data hub* task. It is the MQTT brokers/clouds responsibility to provide history access to the data where it is required (e.g. logs).
Cloud service 2	Task *(user app)*	Same as *cloud service 1* task, but for a second connection parameter set (for the same type of service).

4.4 Logging

These components and artifacts deal with the logging functionality of i.MX RT Industrial Drive Development Platform. Components are highlighted in Figure 12.

The *logging service* task centralizes the log management of the whole application through the *logging API*. Logs are read from the *log entries* message queue and, if required, forwarded in encrypted form to subscribed tasks (*forwarded log entries* message queue(s)).

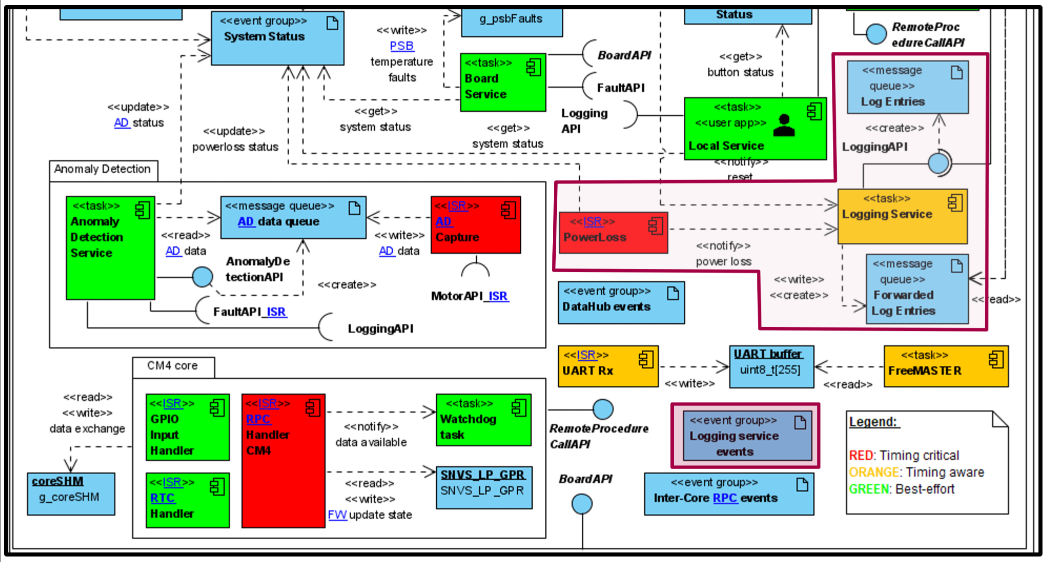


Figure 12. Logging components and artifacts overview

Table 10. Logging components

Name	Type	Description
Logging service	Task	Implements the *Logging API*. It manages the audit log stored on the flash and SD card. New log entries get de-queued from the queue and added to the log. Every log message needs to be written to the dedicated internal flash memory. When a POWER_LOSS event is received, all unnecessary components are disabled and log entries are written to the flash immediately. In case of a WD_RESET event, log entries must also be written to the flash immediately, but it is not required to disable any other components. If applicable, certain log entries can be forwarded to each registered *Forwarded Log Entries* queue. Several log entries can be combined into one when forwarding.
Power loss	ISR	Signals the power loss event to the *logging service* and updates the *system status* event group.

Table 11. Logging artifacts

Name	Type	Description
Log entries	Message queue	Message queue for log entries. Elements get queued by using a call to the *Logging API*. The *Logging Service* then de-queues and processes these elements.
Forwarded log entries	Message queue	Message queues for encrypted log messages that are forwarded to some external processing e.g. cloud storage. Elements get queued by the *Logging Service*. A task can de-queue them by using the *LOG_DequeueEncryptedLogEntry* call of the *Logging API*. Each task that wants to receive logging messages for further processing must register their own queue with the *Logging Service* by using the *Logging API*. Maximum number of queues available can be defined at compile-time; each queue has its own event bit.

Table 11. Logging artifacts*...continued*

Name	Type	Description
Logging service events	Event group	This event group holds the event bits for the message queues used in the logging service. A set event bit indicates that the queue contains new messages for reading.

4.5 RPC and watchdog

These components and artifacts deal with the RPC and watchdog functionality of the i.MX RT Industrial Drive Development Platform. Components are highlighted in Section 4.5.

The Cortex M4 core of the i.MX RT1170 MCU provides functionalities such as SPI slave selection, RTC and secure watchdog. In particular the *watchdog task* implements both functional watchdogs and the secure watchdog. The *RPC Handler CM4* ISR processes calls to the *RPC API* and trigger the appropriate action in Cortex M4 core; e.g., kicking the watchdog, setting output pins on SNVS domain or get the time from the RTC. In an equivalent way, the *RPC Handler CM7* ISR processes calls to the *RPC API* as well as events from Cortex M4 and trigger the appropriate action in Cortex M7 core.

Note: *the watchdog task is NOT, in this case, a FreeRTOS task. The CM4 core does not run FreeRTOS and functions are implemented using a bare metal implementation.*

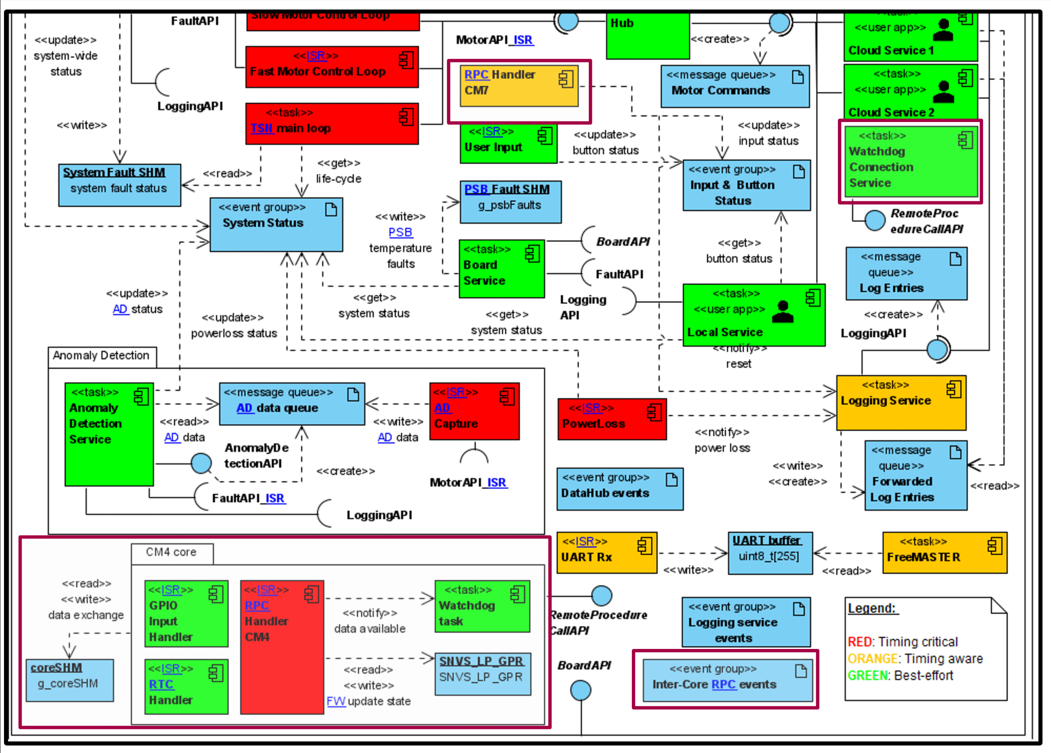


Figure 13. RPC and watchdog components and artifacts overview

Table 12. RPC and watchdog components

Name	Type	Description
Watchdog task	Task	Implements the functional watchdogs and the secure watchdog feature. The number of functional watchdogs and their intervals is configured at compile-time. ***Note:*** *the watchdog task is NOT, in this case, a FreeRTOS task. The CM4 core does not run FreeRTOS and functions are implemented using a bare metal implementation.*
RPC handler CM4	ISR	Triggered by the global interrupt (*GINT / IOMUX_GPR->GPR7*). It processes the remote procedure calls from the *RPC API* and takes the appropriate action, e.g. kicking the watchdog, setting the output pins on the SNVS domain, etc
GPIO Input handler	ISR	Triggered by rising and falling edges on either of the four slow SNVS domain user input pins. It performs input debouncing, then triggers an event via the RPC interface. This is done to prevent a slow-down of the Cortex M7 core, which would occur when accessing the SNVS domain directly.
RTC handler	ISR	Interrupt handler for the SNVS RTC interrupts. Used for ticking the watchdogs, triggering a notification about an upcoming system reset in case of watchdog expiration and resetting the system.
RPC handler CM7	ISR	Triggered by the global interrupt (*GINT / IOMUX_GPR->GPR7*). It processes the remote procedure calls from the *RPC API* as well as events generated by the Cortex M4 core.
Watchdog connection service	Task	Connects to the back-end service of the secure watchdog feature. Data received via network is forwarded to the watchdog task in the Cortex M4 core and vice versa. Data forwarding uses the functionality defined by the *RPC API*.

Table 13. RPC and watchdog artifacts

Name	Type	Description
coreSHM	Shared memory	Shared memory that holds the necessary data for the *RPC API*.
SNVS_LP_GPR	Data structure	Represents the battery backed-up SNVS_LP_GPR registers.
Inter-Core RPC events	Event group	Represents the status of the remote procedure calls (called on the Cortex M7, executed on the Cortex M4).

4.6 System

These components and artifacts deal with system-related functionalities of i.MX RT Industrial Drive Development Platform. Components are highlighted in Figure 14.

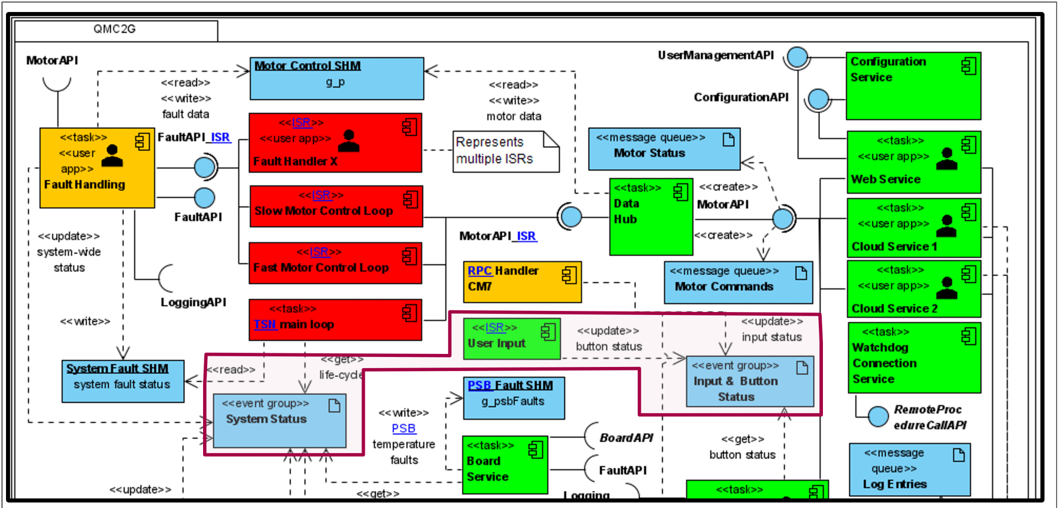

QMC2G
MotorAPI
Motor Control SHM
g_p
<<read>> <<write>> fault data
<<read>> <<write>> motor data
UserManagementAPI
ConfigurationAPI
Configuration Service
<<task>> <<user app>> Fault Handling
FaultAPI_ISR
FaultAPI
<<ISR>> <<user app>> Fault Handler X
<<ISR>> Slow Motor Control Loop
<<ISR>> Fast Motor Control Loop
<<task>> TSN main loop
Represents multiple ISRs
<<message queue>> Motor Status
<<task>> Data Hub
<<create>>
MotorAPI
<<task>> <<user app>> Web Service
<<task>> <<user app>> Cloud Service 1
<<task>> <<user app>> Cloud Service 2
<<task>> Watchdog Connection Service
MotorAPI_ISR
RPC Handler CM7
<<message queue>> Motor Commands
LoggingAPI
<<update>> system-wide status
<<write>>
System Fault SHM
system fault status
<<read>>
<<get>> life-cycle
<<event group>> System Status
<<ISR>> User Input
<<update>> button status
<<update>> input status
<<event group>> Input & Button Status
PSB Fault SHM
g_psbFaults
<<write>> PSB temperature faults
<<task>> Board Service
BoardAPI
FaultAPI
Logging
<<get>> button status
RemoteProcedureCallAPI
<<message queue>> Log Entries
<<update>>
<<get>>
<<task>>

Figure 14. System components and artifacts overview

Table 14. System components

Name	Type	Description
User input	ISR	This ISR is triggered by rising and falling edges on the fast GPIO input pins and the pins used for user buttons and protection lid. It updates the global *input & button status* event group according to the button status (pressed/released).

Table 15. System artifacts

Name	Type	Description
System status	Event group	This event group represents the current system status. Event bits signal life-cycle state of the system, motor and system faults, configuration changes, shutdown events, etc. For a complete list of event bits, please refer to the i.MX RT Industrial Drive Development Platform software model.
Input & button status	Event group	This event group represents the status (pressed/released) of user input buttons and the protective lid as well as the status (high / low) of user input pins. It features event bits for both status, pressed / low and released / high, as FreeRTOS only allows to wait for event bits getting set.

4.7 Anomaly detection

These components and artifacts deal with the anomaly detection functionality of i.MX RT Industrial Drive Development Platform. Components are highlighted in Figure 15.

The anomaly detection functionality is implemented by the *anomaly detection service*. The service runs an anomaly detection algorithm that feeds on data captured by i.MX RT Industrial Drive Development Platform microphone and tries to detect malfunctioning of the motors based on the analysis of the samples received and on the motor status. The *AD capture* ISR queues the samples in the *AD data queue* from which *anomaly detection service* reads data from.

Important note: *the anomaly detection functionality is NOT part of the i.MX RT Industrial Drive Development Platform ISA/IEC 63442-4-2 compliancy.*

***Note**: the anomaly detection components will be released as part of future software releases.*

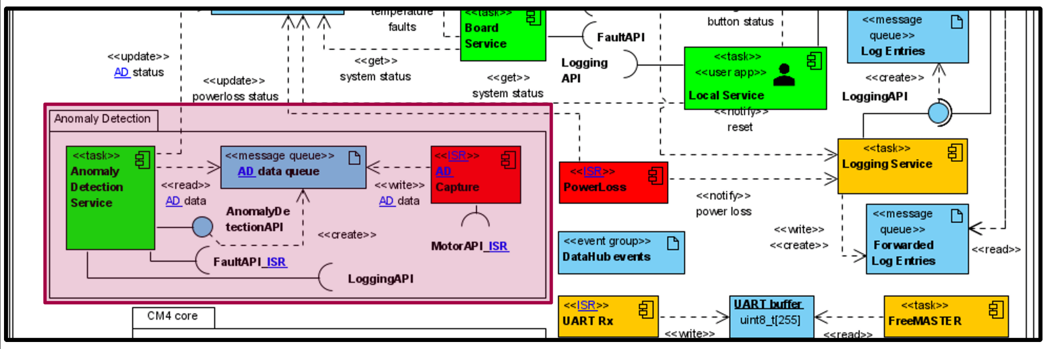


Figure 15. Anomaly detection components and artifacts overview

Table 16. Anomaly detection components

Name	Type	Description
AD Capture	ISR	*AD Capture ISR* reads blocks of samples from the digital microphone via DMA as well as blocks of motor status values via *MC_ GetFastMotorStatusSync_fromISR* of the *MotorAPI_ISR*. It aligns the blocks and queues them into *AD data queue* for further processing by *Anomaly Detection Service*.
Anomaly Detection Service	Task	This tasks runs the actual anomaly detection algorithm. The captured data is read from the *AD data queue*.

Table 17. Anomaly detection artifacts

Name	Type	Description
AD data queue	Message queue	Message queue that transfers the capture data from the *AD Capture ISR* to the *Anomaly Detection Service*. Queued entries are described by *ADE_SampleBlock_t*: a data structure that contains one block of microphone samples and one block of motor status values that are aligned to each other.

4.8 Others

These components and artifacts deal with other functionalities of i.MX RT Industrial Drive Development Platform not covered in the previous sections. Components are highlighted in Figure 16.

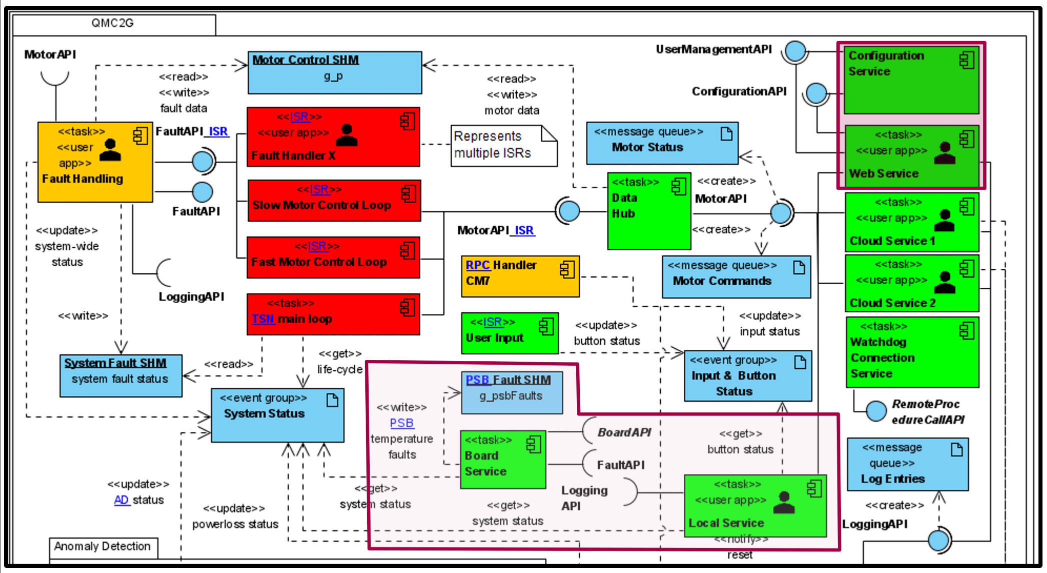

QMC2G
MotorAPI
Motor Control SHM
g_p
<<read>> <<write>> fault data
<<read>> <<write>> motor data
<<task>> <<user app>> Fault Handling
FaultAPI_ISR
FaultAPI
<<ISR>> <<user app>> Fault Handler X
Represents multiple ISRs
<<ISR>> Slow Motor Control Loop
<<ISR>> Fast Motor Control Loop
<<task>> TSN main loop
LoggingAPI
<<update>> system-wide status
<<write>>
System Fault SHM
system fault status
<<read>>
<<get>> life-cycle
<<event group>> System Status
<<update>> AD status
<<update>> powerloss status
Anomaly Detection
UserManagementAPI
ConfigurationAPI
Configuration Service
<<task>> <<user app>> Web Service
<<message queue>> Motor Status
<<task>> Data Hub
<<create>>
MotorAPI
MotorAPI_ISR
RPC Handler CM7
<<message queue>> Motor Commands
<<ISR>> User Input
<<update>> button status
<<update>> input status
<<event group>> Input & Button Status
PSB Fault SHM
g_psbFaults
<<write>> PSB temperature faults
<<task>> Board Service
BoardAPI
FaultAPI
Logging API
<<get>> system status
<<get>> system status
<<get>> button status
<<task>> <<user app>> Local Service
<<notify>> reset
<<task>> <<user app>> Cloud Service 1
<<task>> <<user app>> Cloud Service 2
<<task>> Watchdog Connection Service
RemoteProcedureCallAPI
<<message queue>> Log Entries
<<create>>
LoggingAPI

Figure 16. Other components and artifacts overview

Table 18. Other components

Name	Type	Description
Configuration service	Task	Implements the *Configuration API*. It serves decrypted key-value pairs from the encrypted configuration in flash upon request. It updates the encrypted configuration if changes to the configuration were made and an update was requested. The set of valid keys (identifiers) is set during compile time and hence is fixed for a specific firmware image.
Web service	Task (*user app*)	Implements a TCP server for up to five concurrent connections. Each connection is secured by TLS. After the TLS connection is established, the web interface is transferred to the client. Now, the user must log in to authenticate her/himself to the system. The authentication result will be sent to the *logging service* using the *Logging API*. Upon failed authentication, the connection is terminated. Upon successful authentication, the web interface is populated with the information that is appropriate for the users role and the connection is maintained to wait for user requests e.g. motor commands or motor status update requests. One of the five connections is reserved for high-priority users. If a low-priority user connects to it, the connection shall be closed directly after authentication.
Local service	Task (*user app*)	*Local Service* can be resumed by changes in the *input & button status* or *system status* event groups. It reacts on user input (button presses) and implements a state machine for the local part of the demo application. Also, it updates the display.
Board service	Task	Runs a very slow polling cycle and can be resumed early by changes in the system status event group. It mainly performs temperature measurements and emits a FAULT event if the temperature exceeds a certain limit. Upon changes of the system status, e.g. due to a fault, a log entry is prepared in the *fault handling task* and sent to the logging service.

Table 19. Other artifacts

Name	Type	Description
PSB_Fault SHM	Other	Holds the power stage board temperature faults detected by the *board service*. It will be read by *MC_SetFastMotorStatus_fromISR* function to include the information into the motor status when *FEATURE_MC_PSB_TEMPERATURE_FAULTS* configuration is enabled.

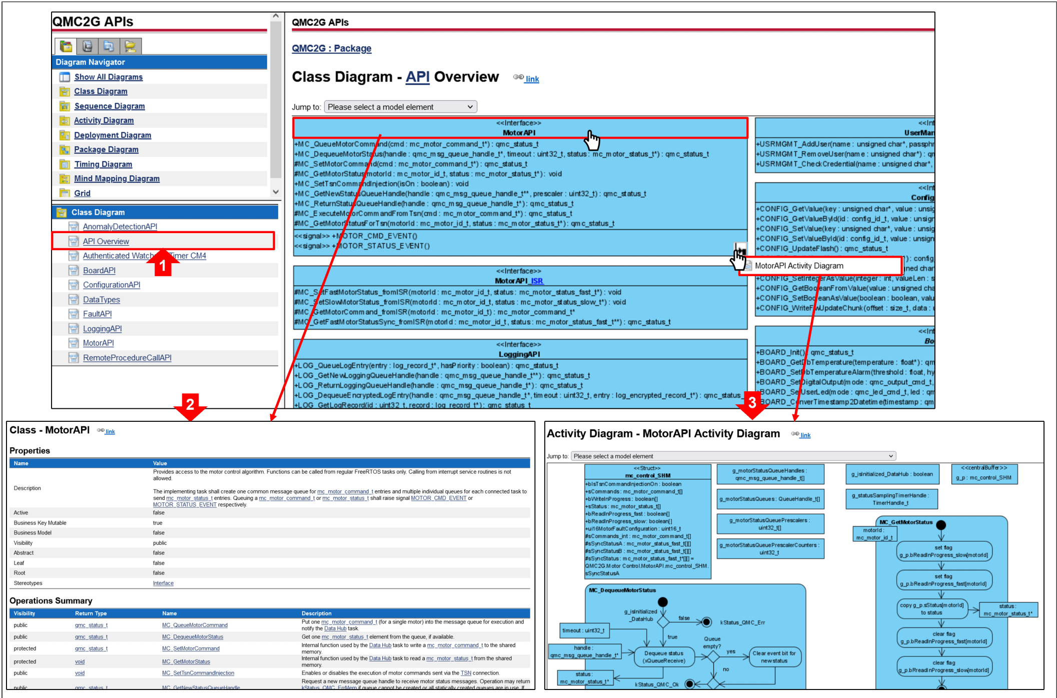
5 Software APIs

This section lists the APIs provided in i.MX RT Industrial Drive Development Platform software. For each API function a brief description is provided.

- Motor API and Motor API ISR
- Fault API and Fault API ISR
- Logging API
- User management API
- Configuration API
- Board API
- Remote procedure call API
- Anomaly detection API

Detailed information about a particular API or function, such as input and output parameters or activity diagrams, can be obtained from the i.MX RT Industrial Drive Development Platform software model as shown in Figure 17:

1. Open the i.MX RT Industrial Drive Development Platform software model and navigate to *class diagram → API overview*.
2. Click on the API that you are interested in to access the page with the detailed description of the API and all its functions.
3. Click on the white arrow in the bottom right corner of the API element to access additional resources. For example, the *motor API* additionally provides an activity diagram depicting the flow of the API functions.



```
<<Interface>>
MotorAPI
+MC_QueueMotorCommand(cmd : mc_motor_command_t*) : qmc_status_t
+MC_DequeueMotorStatus(handle : qmc_msg_queue_handle_t*, timeout : uint32_t, status : mc_motor_status_t*) : qmc_status_t
#MC_SetMotorCommand(cmd : mc_motor_command_t*) : qmc_status_t
#MC_GetMotorStatus(motorId : mc_motor_id_t, status : mc_motor_status_t*) : void
+MC_SetTsnCommandInjection(isOn : boolean) : void
+MC_GetNewStatusQueueHandle(handle : qmc_msg_queue_handle_t**, prescaler : uint32_t) : qmc_status_t
+MC_ReturnStatusQueueHandle(handle : qmc_msg_queue_handle_t*) : qmc_status_t
#MC_ExecuteMotorCommandFromTsn(cmd : mc_motor_command_t*) : qmc_status_t
#MC_GetMotorStatusForTsn(motorId : mc_motor_id_t, status : mc_motor_status_t*) : qmc_status_t
<<signal>> +MOTOR_CMD_EVENT()
<<signal>> +MOTOR_STATUS_EVENT()

<<Interface>>
MotorAPI_ISR
#MC_SetFastMotorStatus_fromISR(motorId : mc_motor_id_t, status : mc_motor_status_fast_t*) : void
#MC_SetSlowMotorStatus_fromISR(motorId : mc_motor_id_t, status : mc_motor_status_slow_t*) : void
#MC_GetMotorCommand_fromISR(motorId : mc_motor_id_t) : mc_motor_command_t*
#MC_GetFastMotorStatusSync_fromISR(motorId : mc_motor_id_t, status : mc_motor_status_fast_t**) : qmc_status_t

<<Interface>>
LoggingAPI
+LOG_QueueLogEntry(entry : log_record_t*, hasPriority : boolean) : qmc_status_t
+LOG_GetNewLoggingQueueHandle(handle : qmc_msg_queue_handle_t**) : qmc_status_t
+LOG_ReturnLoggingQueueHandle(handle : qmc_msg_queue_handle_t*) : qmc_status_t
+LOG_DequeueEncryptedLogEntry(handle : qmc_msg_queue_handle_t*, timeout : uint32_t, entry : log_encrypted_record_t*) : qmc_status
+LOG_GetLogRecord(id : uint32_t, record : log_record_t*) : qmc_status_t

<<Int
UserMan
+USRMGMT_AddUser(name : unsigned char*, passphr
+USRMGMT_RemoveUser(name : unsigned char*) : q
+USRMGMT_CheckCredential(name : unsigned char*,

<<Int
Config
+CONFIG_GetValue(key : unsigned char*, value : unsig
+CONFIG_GetValueById(id : config_id_t, value : unsign
+CONFIG_SetValue(key : unsigned char*, value : unsig
+CONFIG_SetValueById(id : config_id_t, value : unsign
+CONFIG_UpdateFlash() : qmc_status_t
): config
ned char
+CONFIG_SetIntegerAsValue(integer : int, valueLen : s
+CONFIG_GetBooleanFromValue(value : unsigned cha
+CONFIG_SetBooleanAsValue(boolean : boolean, valu
+CONFIG_WriteFwUpdateChunk(offset : size_t, data :

<<Int
Bo
+BOARD_Init() : qmc_status_t
+BOARD_GetDbTemperature(temperature : float*) : qm
+BOARD_SetDbTemperatureAlarm(threshold : float, hy
+BOARD_SetDigitalOutput(mode : qmc_output_cmd_t,
+BOARD_SetUserLed(mode : qmc_led_cmd_t, led : qm
+BOARD_ConverTimestamp2Datetime(timestamp : qm
```

Class - MotorAPI link

Properties

Name	Value
Description	Provides access to the motor control algorithm. Functions can be called from regular FreeRTOS tasks only. Calling from interrupt service routines is not allowed. The implementing task shall create one common message queue for mc_motor_command_t entries and multiple individual queues for each connected task to send mc_motor_status_t entries. Queuing a mc_motor_command_t or mc_motor_status_t shall raise signal MOTOR_CMD_EVENT or MOTOR_STATUS_EVENT respectively.
Active	false
Business Key Mutable	true
Business Model	false
Visibility	public
Abstract	false
Leaf	false
Root	false
Stereotypes	Interface

Operations Summary

Visibility	Return Type	Name	Description
public	qmc_status_t	MC_QueueMotorCommand	Put one mc_motor_command_t (for a single motor) into the message queue for execution and notify the Data Hub task.
public	qmc_status_t	MC_DequeueMotorStatus	Get one mc_motor_status_t element from the queue, if available.
protected	qmc_status_t	MC_SetMotorCommand	Internal function used by the Data Hub task to write a mc_motor_command_t to the shared memory.
protected	void	MC_GetMotorStatus	Internal function used by the Data Hub task to read a mc_motor_status_t from the shared memory.
public	void	MC_SetTsnCommandInjection	Enables or disables the execution of motor commands sent via the TSN connection.
public	qmc_status_t	MC_GetNewStatusQueueHandle	Request a new message queue handle to receive motor status messages. Operation may return kStatus_QMC_ErrMem if queue cannot be created or all statically created queues are in use. If

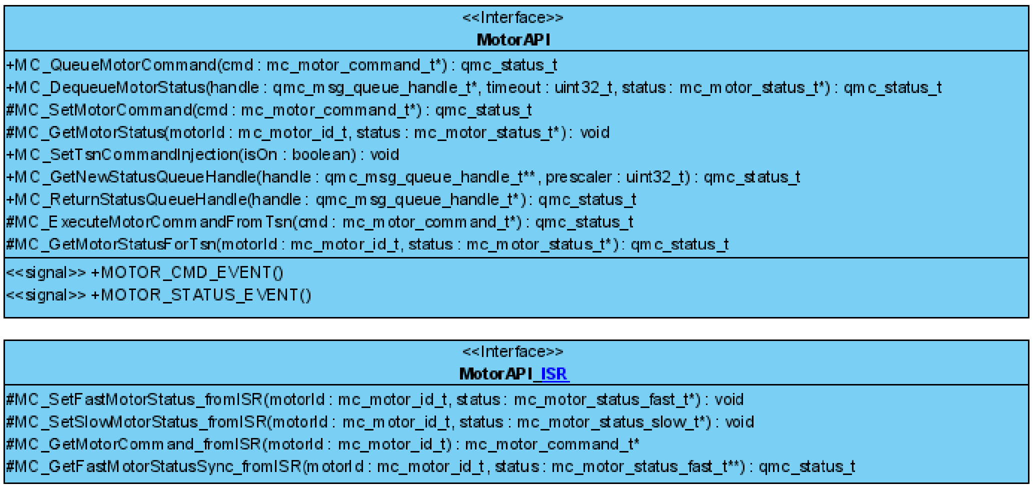
```
<<Struct>>
mc_control_SHM
+bIsTsnCommandInjectionOn : boolean
+sCommands : mc_motor_command_t[]
+bWriteInProgress : boolean[]
+sStatus : mc_motor_status_t[]
+bReadInProgress_fast : boolean[]
+bReadInProgress_slow : boolean[]

g_motorStatusQueueHandles : qmc_msg_queue_handle_t[]
g_motorStatusQueues : QueueHandle_t[]
g_motorStatusQueuePrescalers : uint32_t[]
g_motorStatusQueuePrescalerCounters : uint32_t
g_isInitialized_DataHub : boolean
g_statusSamplingTimerHandle : TimerHandle_t
<<centralBuffer>> g_p : mc_control_SHM
```

Figure 17. Access API resources in i.MX RT Industrial Drive Development Platform software model

5.1 Motor API and Motor API ISR

The *motor API* and *motor API ISR* provide access to motor control functionality through the *data hub* task. An overview of the available functions is shown in Figure 18. Table 20 and Table 21 provide a brief description of each function. More details can be obtained from the i.MX RT Industrial Drive Development Platform software model as described in Section 5.

Note: *functions whose name is preceded by a + symbol are public functions that can be used by applications. Functions whose name is preceded by the # symbol are internal (protected) i.MX RT Industrial Drive Development Platform functions that by default are not accessible to applications.*



```
<<Interface>>
MotorAPI
+MC_QueueMotorCommand(cmd : mc_motor_command_t*) : qmc_status_t
+MC_DequeueMotorStatus(handle : qmc_msg_queue_handle_t*, timeout : uint32_t, status : mc_motor_status_t*) : qmc_status_t
#MC_SetMotorCommand(cmd : mc_motor_command_t*) : qmc_status_t
#MC_GetMotorStatus(motorId : mc_motor_id_t, status : mc_motor_status_t*) : void
+MC_SetTsnCommandInjection(isOn : boolean) : void
+MC_GetNewStatusQueueHandle(handle : qmc_msg_queue_handle_t**, prescaler : uint32_t) : qmc_status_t
+MC_ReturnStatusQueueHandle(handle : qmc_msg_queue_handle_t*) : qmc_status_t
#MC_ExecuteMotorCommandFromTsn(cmd : mc_motor_command_t*) : qmc_status_t
#MC_GetMotorStatusForTsn(motorId : mc_motor_id_t, status : mc_motor_status_t*) : qmc_status_t
<<signal>> +MOTOR_CMD_EVENT()
<<signal>> +MOTOR_STATUS_EVENT()

<<Interface>>
MotorAPI_ISR
#MC_SetFastMotorStatus_fromISR(motorId : mc_motor_id_t, status : mc_motor_status_fast_t*) : void
#MC_SetSlowMotorStatus_fromISR(motorId : mc_motor_id_t, status : mc_motor_status_slow_t*) : void
#MC_GetMotorCommand_fromISR(motorId : mc_motor_id_t) : mc_motor_command_t*
#MC_GetFastMotorStatusSync_fromISR(motorId : mc_motor_id_t, status : mc_motor_status_fast_t**) : qmc_status_t
```

Figure 18. Motor control API and Motor control API ISR functions

Table 20. Motor API functions

Name	Visibility	Description
MC_QueueMotorCommand	Public	Put one *mc_motor_command_t* (for a single motor) into the message queue for execution and notify the *Data Hub* task.
MC_DequeueMotorStatus	Public	Get one *mc_motor_status_t* element from the queue, if available.
MC_SetMotorCommand	Protected	Internal function used by the Data Hub task to write a *mc_motor_command_t* to the shared memory.
MC_GetMotorStatus	Protected	Internal function used by the Data Hub task to read a *mc_motor_status_t* from the shared memory.
MC_SetTsnCommandInjection	Public	Enables or disables the execution of motor commands sent via the TSN connection.
MC_GetNewStatusQueueHandle	Public	Request a new message queue handle to receive motor status messages.
MC_ReturnStatusQueueHandle	Public	Hand back a message queue handle obtained by *MC_GetNewStatusQueueHandle*.
MC_ExecuteMotorCommandFromTsn	Protected	Function used by the *TSN main loop* task to set a *mc_motor_command_t* for execution.

Table 20. Motor API functions*...continued*

Name	Visibility	Description
MC_GetMotorStatusForTsn	Protected	Function to be used by the TSN main loop to retrieve a *mc_motor_status_t*.

Table 21. Motor API ISR functions

Name	Visibility	Description
MC_SetFastMotorStatus_fromISR	Protected	Internal function used by the *Fast Motor Control Loop* to write part of a *mc_motor_status_t* to the shared memory.
MC_SetSlowMotorStatus_fromISR	Protected	Internal function used by the *Slow Motor Control Loop* to write part of a *mc_motor_status_t* to the shared memory.
MC_GetMotorCommand_fromISR	Protected	Internal function used by the *Slow Motor Control Loop* to get the current *mc_motor_command_t* for execution.
MC_GetFastMotorStatusSync_fromISR	Protected	Internal function used by the *AD Capture handler* to get the part of the *mc_motor_status_t* written by the *Fast Motor Control Loop* (phase current values) synchronously.

5.2 Fault API and Fault API ISR

The *Fault API* and *Fault API ISR* provides access to the fault handling system respectively to tasks and ISRs. An overview of the available functions is shown in Figure 19. Table 23 provides a brief description of each function. More details can be obtained from the i.MX RT Industrial Drive Development Platform software model as described in Section 5.

Note: *functions whose name is preceded by a + symbol are public functions that can be used by user applications. Functions whose name is preceded by the # symbol are internal (protected) i.MX RT Industrial Drive Development Platform functions that by default are not accessible to user applications.*

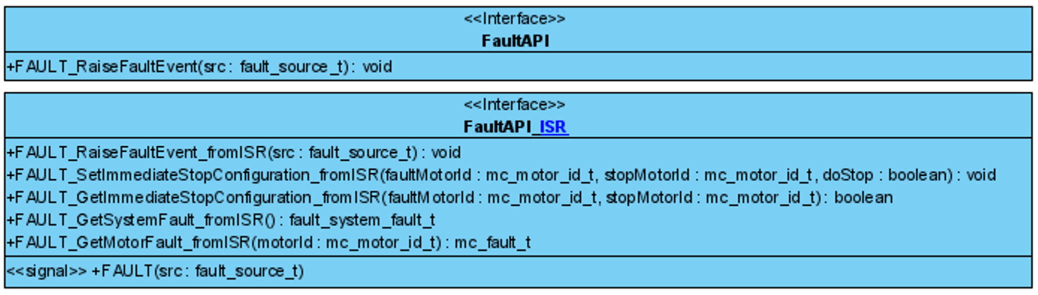

<<Interface>>
FaultAPI
+FAULT_RaiseFaultEvent(src : fault_source_t) : void
<<Interface>>
FaultAPI_ISR
+FAULT_RaiseFaultEvent_fromISR(src : fault_source_t) : void
+FAULT_SetImmediateStopConfiguration_fromISR(faultMotorId : mc_motor_id_t, stopMotorId : mc_motor_id_t, doStop : boolean) : void
+FAULT_GetImmediateStopConfiguration_fromISR(faultMotorId : mc_motor_id_t, stopMotorId : mc_motor_id_t) : boolean
+FAULT_GetSystemFault_fromISR() : fault_system_fault_t
+FAULT_GetMotorFault_fromISR(motorId : mc_motor_id_t) : mc_fault_t
<<signal>> +FAULT(src : fault_source_t)

Figure 19. Motor control API and Motor control API ISR functions

Table 22. Fault API functions

Name	Visibility	Description
FAULT_RaiseFaultEvent	Public	Sends a FAULT signal to the Fault Handling service indicating the given fault source (*fault_source_t*). For valid fault sources see *mc_fault_t* documentation in i.MX RT Industrial Drive Development Platform software model.

Table 23. Fault API ISR functions

Name	Visibility	Description
FAULT_RaiseFaultEvent_fromISR	Public	Sends a FAULT signal to the *Fault Handling* task indicating the given fault source (*fault_source_t*). For valid fault sources see *mc_fault_t* documentation in i.MX RT Industrial Drive Development Platform software model.
FAULT_SetImmediateStopConfiguration_fromISR	Public	Set the immediate motor stop behavior for the motor defined by *stopMotorId* that should be applied in case the motor defined by *faultMotorId* experiences a fault.
FAULT_GetImmediateStopConfiguration_fromISR	Public	Get the immediate motor stop behavior for the motor defined by *stopMotorId* that should be applied in case the motor defined by *faultMotorId* experiences a fault.
FAULT_GetSystemFault_fromISR	Public	Get the current system fault status.
FAULT_GetMotorFault_fromISR	Public	Get the current motor fault status for the given motor ID.

5.3 Logging API

The *logging API* provides access to the logging functionality of the i.MX RT Industrial Drive Development Platform software. An overview of the available functions is shown in Figure 20. Table 24 provides a brief description of each function. More details can be obtained from the i.MX RT Industrial Drive Development Platform software model as described in Section 5.

***Note:** functions whose name is preceded by a + symbol are public functions that can be used by user applications. Functions whose name is preceded by the # symbol are internal (protected) i.MX RT Industrial Drive Development Platform functions that by default are not accessible to user applications.*

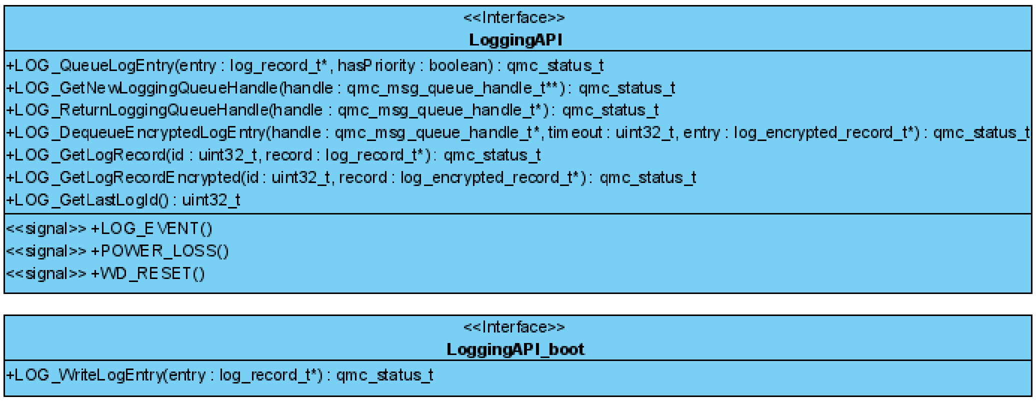

<<Interface>>
LoggingAPI
+LOG_QueueLogEntry(entry : log_record_t*, hasPriority : boolean) : qmc_status_t
+LOG_GetNewLoggingQueueHandle(handle : qmc_msg_queue_handle_t**) : qmc_status_t
+LOG_ReturnLoggingQueueHandle(handle : qmc_msg_queue_handle_t*) : qmc_status_t
+LOG_DequeueEncryptedLogEntry(handle : qmc_msg_queue_handle_t*, timeout : uint32_t, entry : log_encrypted_record_t*) : qmc_status_t
+LOG_GetLogRecord(id : uint32_t, record : log_record_t*) : qmc_status_t
+LOG_GetLogRecordEncrypted(id : uint32_t, record : log_encrypted_record_t*) : qmc_status_t
+LOG_GetLastLogId() : uint32_t
<<signal>> +LOG_EVENT()
<<signal>> +POWER_LOSS()
<<signal>> +WD_RESET()
<<Interface>>
LoggingAPI_boot
+LOG_WriteLogEntry(entry : log_record_t*) : qmc_status_t

Figure 20. Logging API functions

Table 24. Logging API functions

Name	Visibility	Description
LOG_QueueLogEntry	Public	Put one log record (*log_record_t*) into the message queue and notify the *logging service* task. If the *hasPriority* flag is set, the log record is put at the beginning of the queue instead of its end.
LOG_GetNewLoggingQueueHandle	Public	Request a new message queue handle to receive logging messages.
LOG_ReturnLoggingQueueHandle	Public	Hands back a message queue handle obtained by *LOG_GetNewLoggingQueueHandle*.
LOG_DequeueEncryptedLogEntry	Public	Get one encrypted log record (*log_encrypted_record_t*) from the previously registered queue, if available.
LOG_GetLogRecord	Public	Get the *log_record_t* with the given ID.
LOG_GetLogRecordEncrypted	Public	Get the log record with the given ID and encrypt it for the external log reader.
LOG_GetLastLogId	Public	Returns the ID of the latest log entry.
LOG_WriteLogEntry	Public	Write a log entry to the flash and if an SD card is available, also to the SD card. This functionality is required by the bootloader. It can be used by the bootloader only.

5.4 User management API

The *user management API* provides functionalities to change user access configurations of the i.MX RT Industrial Drive Development Platform software. An overview of the available functions is shown in Figure 21. Table 25 provides a brief description of each function. More details can be obtained from the i.MX RT Industrial Drive Development Platform software model as described in Section 5.

Note: *functions whose name is preceded by a + symbol are public functions that can be used by user applications. Functions whose name is preceded by the # symbol are internal (protected) i.MX RT Industrial Drive Development Platform functions that by default are not accessible to user applications.*

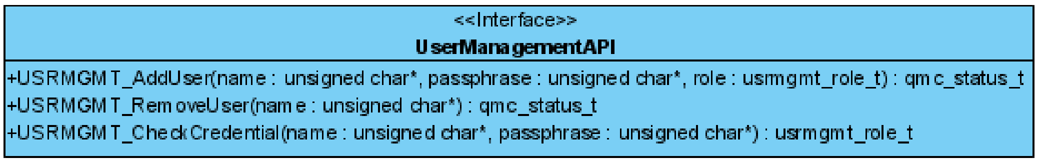

<<Interface>>
UserManagementAPI
+USRMGMT_AddUser(name : unsigned char*, passphrase : unsigned char*, role : usrmgmt_role_t) : qmc_status_t
+USRMGMT_RemoveUser(name : unsigned char*) : qmc_status_t
+USRMGMT_CheckCredential(name : unsigned char*, passphrase : unsigned char*) : usrmgmt_role_t

Figure 21. User management API functions

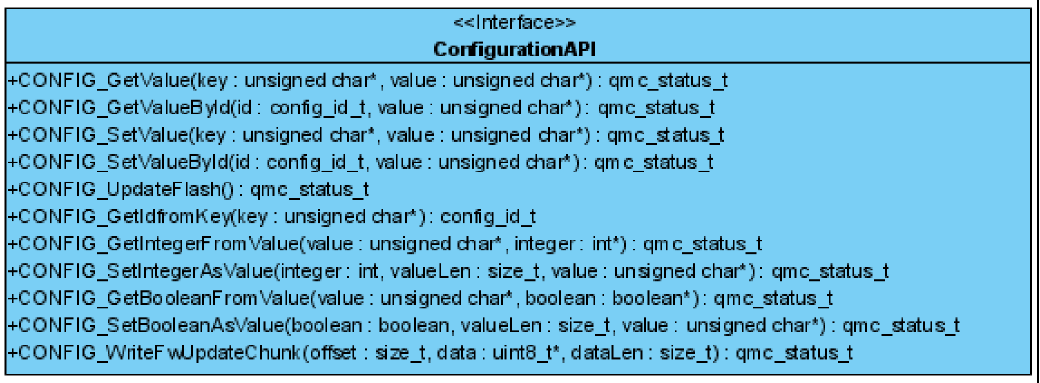
Table 25. User management API functions

Name	Visibility	Description
USRMGMT_AddUser	Public	Adds a new user to the system (name, passphrase and role). If adding the user was successful, configuration in flash is updated. The accepted user roles are defined in *usermgmt_role_t*: • *kUSRMGMT_RoleNone*: user doesn't have a role e.g. user isn't registered or authentication failed; • *kUSRMGMT_RoleEmpty*: pseudo role used by the user management to indicate free user configuration slots; • *kUSRMGMT_RoleMaintenance*: maintenance head role; • *kUSRMGMT_RoleSupervisor*: production supervisor role; • *kUSRMGMT_RoleOperator*: operator role; • *kUSRMGMT_RoleLocalSd*: user that is not authenticated cryptographically, but via a mechanical key. This role is used for logging SD card related activities e.g. insert/remove SD card or open the SD card lid; • *kUSRMGMT_RoleLocalButton*: user that is not authenticated cryptographically, but via a mechanical key. This role is used for logging activities related to the user buttons e.g. starting/stopping a motor; • *kUSRMGMT_RoleLocalEmergency*: used for logging emergency events (emergency stop button). User has not undergone authentication.
USRMGMT_RemoveUser	Public	Removes a user from the system. If removing the user was successful, configuration in flash is updated to prevent further access to the system.
USRMGMT_CheckCredential	Public	Checks the passed user credentials. It returns the role of the user if authentication was successful. Returns *kUSRMGMT_RoleNone*, if authentication failed or user doesn't exist.

5.5 Configuration API

The *configuration API* provides access to the application configuration as key-value pairs. An overview of the available functions is shown in Figure 22. Table 26 provides a brief description of each function. More details can be obtained from the i.MX RT Industrial Drive Development Platform software model as described in Section 5.

Note: *functions whose name is preceded by a + symbol are public functions that can be used by user applications. Functions whose name is preceded by the # symbol are internal (protected) i.MX RT Industrial Drive Development Platform functions that by default are not accessible to user applications.*



```
<<Interface>>
ConfigurationAPI
+CONFIG_GetValue(key : unsigned char* , value : unsigned char*) : qmc_status_t
+CONFIG_GetValueById(id : config_id_t, value : unsigned char*) : qmc_status_t
+CONFIG_SetValue(key : unsigned char*, value : unsigned char*) : qmc_status_t
+CONFIG_SetValueById(id : config_id_t, value : unsigned char*) : qmc_status_t
+CONFIG_UpdateFlash() : qmc_status_t
+CONFIG_GetIdfromKey(key : unsigned char*) : config_id_t
+CONFIG_GetIntegerFromValue(value : unsigned char* , integer : int*) : qmc_status_t
+CONFIG_SetIntegerAsValue(integer : int, valueLen : size_t, value : unsigned char*) : qmc_status_t
+CONFIG_GetBooleanFromValue(value : unsigned char* , boolean : boolean*) : qmc_status_t
+CONFIG_SetBooleanAsValue(boolean : boolean, valueLen : size_t, value : unsigned char*) : qmc_status_t
+CONFIG_WriteFwUpdateChunk(offset : size_t, data : uint8_t*, dataLen : size_t) : qmc_status_t
```

Figure 22. Configuration API functions

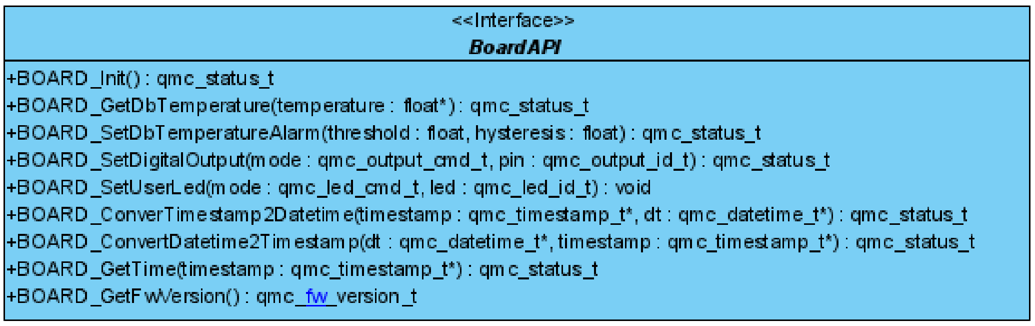
Table 26. Configuration API functions

Name	Visibility	Description
CONFIG_GetValue	Public	Get the value indicated by key input parameter.
CONFIG_GetValueById	Public	Get the value indicated by id.
CONFIG_SetValue	Public	Set the new value for the given key.
CONFIG_SetValueById	Public	Set the new value for the given id.
CONFIG_UpdateFlash	Public	Encrypt the current configuration and write it to the flash.
CONFIG_GetIdfromKey	Public	Return the config id that matches the given key.
CONFIG_GetIntegerFromValue	Public	Tries to parse an integer from the configuration value retrieved by *CONFIG_GetValue* or *CONFIG_GetValueById*
CONFIG_SetIntegerAsValue	Public	Convert an integer to a string in order to store it as a configuration value. This value is then stored in the configuration by calling *CONFIG_SetValue* or *CONFIG_SetValueById*
CONFIG_GetBooleanFromValue	Public	Tries to parse a boolean from the configuration value retrieved by *CONFIG_GetValue* or *CONFIG_GetValueById*. Values such as "true"/"false", "yes"/"no", "on"/"off", "1"/"0" shall be recognized by the function.
CONFIG_SetBooleanAsValue	Public	Convert a boolean to a string in order to store it as a configuration value. This value is then stored in the configuration by calling *CONFIG_SetValue* or *CONFIG_SetValueById*
CONFIG_WriteFwUpdateChunk	Public	Write a chunk of data to the firmware update location.

5.6 Board API

The *board API* provides access to board-specific features implemented on the Cortex M7 core, like temperature sensors and display. It also provides functions for initialization. An overview of the available functions is shown in Figure 23. Table 27 provides a brief description of each function. More details can be obtained from the i.MX RT Industrial Drive Development Platform software model as described in Section 5.

Note: *functions whose name is preceded by a + symbol are public functions that can be used by user applications. Functions whose name is preceded by the # symbol are internal (protected) i.MX RT Industrial Drive Development Platform functions that by default are not accessible to user applications.*



```
<<Interface>>
BoardAPI
+BOARD_Init() : qmc_status_t
+BOARD_GetDbTemperature(temperature : float*) : qmc_status_t
+BOARD_SetDbTemperatureAlarm(threshold : float, hysteresis : float) : qmc_status_t
+BOARD_SetDigitalOutput(mode : qmc_output_cmd_t, pin : qmc_output_id_t) : qmc_status_t
+BOARD_SetUserLed(mode : qmc_led_cmd_t, led : qmc_led_id_t) : void
+BOARD_ConverTimestamp2Datetime(timestamp : qmc_timestamp_t*, dt : qmc_datetime_t*) : qmc_status_t
+BOARD_ConvertDatetime2Timestamp(dt : qmc_datetime_t*, timestamp : qmc_timestamp_t*) : qmc_status_t
+BOARD_GetTime(timestamp : qmc_timestamp_t*) : qmc_status_t
+BOARD_GetFwVersion() : qmc_fw_version_t
```

Figure 23. Board API functions

Table 27. Board API functions

Name	Visibility	Description
BOARD_Init	Public	Initialize the *Board API*. Scheduler must be already running.
BOARD_GetDbTemperature	Public	Get the current temperature in degree Celsius from the temperature sensor on the digital board.
BOARD_SetDbTemperatureAlarm	Public	Set alarm parameters for the temperature sensor on the digital board. The temperature sensor on the digital board can set a pin if the temperature exceeds the value indicated by *threshold* input parameter. If the temperature drops below the value indicated by *hysteresis* input parameter, the pin is cleared again. This hardware signal may be used as an interrupt source. However, it is not connected to the system by default.
BOARD_SetDigitalOutput	Public	Sets the output state of the given user output pin. For slow output pins on the SNVS domain, the call is forwarded to the internal function *RPC_SetSnvsOutput*.
BOARD_SetUserLed	Public	Sets the LED state indicated by *qmc_led_cmd_t* (LED on/off/toggle) of the given user LED.
BOARD_ConverTimestamp2Datetime	Public	Utility function. Converts a *qmc_timestamp_t* to a *qmc_datetime_t* structure.

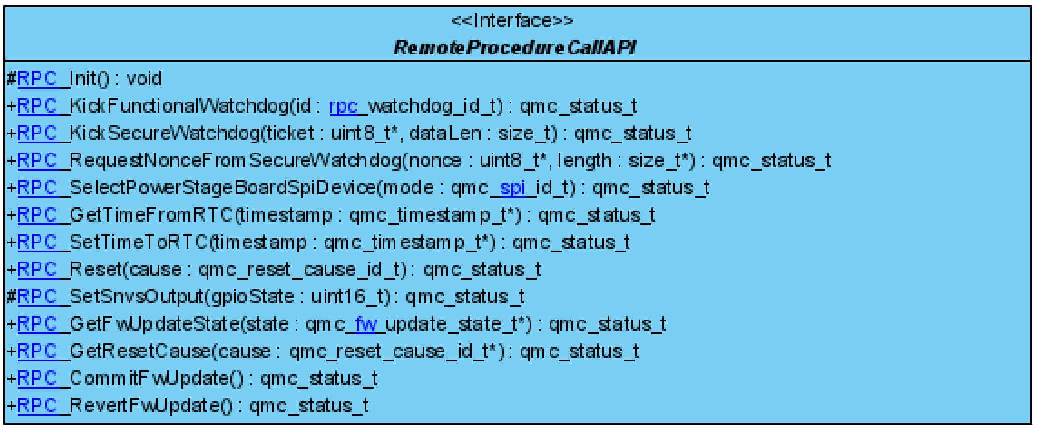
Table 27. Board API functions*...continued*

Name	Visibility	Description
BOARD_ConvertDatetime2Timestamp	Public	Utility function. Convert a *qmc_datetime_t* structure to a *qmc_timestamp_t*.
BOARD_GetTime	Public	Get a timestamp derived from the internal systick counter. The function shall add the offset to the wall-clock time retrieved by *RPC_GetTimeFromRTC* during system start-up.
BOARD_GetFwVersion	Public	Get the version of the currently running firmware. It is retrieved from the firmware manifest section in flash.

5.7 Remote Procedure Call API

The *remote procedure call API* provides functionalities to execute operations in the Cortex M4 core of the platform like SPI slave selection, RTC and secure watchdog. An overview of the available functions is shown in Figure 24. More details can be obtained from the i.MX RT Industrial Drive Development Platform software model as described in Section 5.

Note: *functions whose name is preceded by a + symbol are public functions that can be used by user applications. Functions whose name is preceded by the # symbol are internal (protected) i.MX RT Industrial Drive Development Platform functions that by default are not accessible to user applications.*



```
<<Interface>>
RemoteProcedureCallAPI
#RPC_Init() : void
+RPC_KickFunctionalWatchdog(id : rpc_watchdog_id_t) : qmc_status_t
+RPC_KickSecureWatchdog(ticket : uint8_t*, dataLen : size_t) : qmc_status_t
+RPC_RequestNonceFromSecureWatchdog(nonce : uint8_t*, length : size_t*) : qmc_status_t
+RPC_SelectPowerStageBoardSpiDevice(mode : qmc_spi_id_t) : qmc_status_t
+RPC_GetTimeFromRTC(timestamp : qmc_timestamp_t*) : qmc_status_t
+RPC_SetTimeToRTC(timestamp : qmc_timestamp_t*) : qmc_status_t
+RPC_Reset(cause : qmc_reset_cause_id_t) : qmc_status_t
#RPC_SetSnvsOutput(gpioState : uint16_t) : qmc_status_t
+RPC_GetFwUpdateState(state : qmc_fw_update_state_t*) : qmc_status_t
+RPC_GetResetCause(cause : qmc_reset_cause_id_t*) : qmc_status_t
+RPC_CommitFwUpdate() : qmc_status_t
+RPC_RevertFwUpdate() : qmc_status_t
```

Figure 24. Remote Procedure Call API functions

Table 28. Remote Procedure Call API functions

Name	Visibility	Description
RPC_Init	Protected	Registers IRQ handlers and Initializes event groups,mutexes and data structures used by the API. This function is meant to be called by the main function / startup task.
RPC_KickFunctionalWatchdog	Public	Kick the functional watchdog referenced by the provided ID.
RPC_KickSecureWatchdog	Public	Kick the secure watchdog by giving it an encrypted update message (ticket) received from the *Watchdog Connection Service*.

Table 28. Remote Procedure Call API functions*...continued*

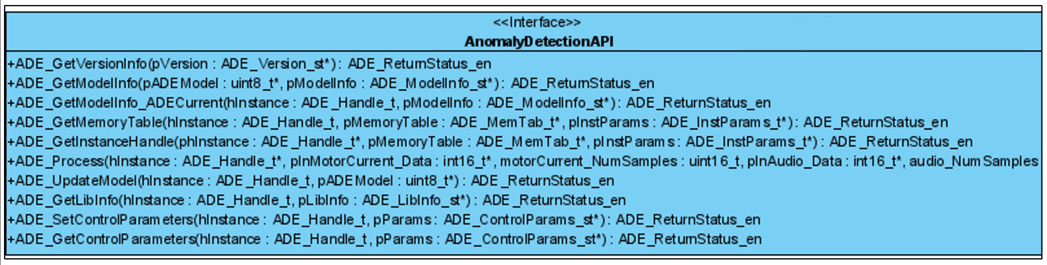
Name	Visibility	Description
RPC_RequestNonceFromSecureWatchdog	Public	Request a nonce from the secure watchdog. This function is meant to be called by the *Watchdog Connection Service*.
RPC_SelectPowerStageBoardSpiDevice	Public	Sets the output state of the SPI selection pins in the SNVS domain such that the SPI device referenced by mode is selected.
RPC_GetTimeFromRTC	Public	Request a timestamp from the real time clock implemented in the CM4 core.
RPC_SetTimeToRTC	Public	Try to set the real time clock to the given timestamp.
RPC_Reset	Public	Trigger a reset. This function creates a log entry and send it to the *Logging Service*.
RPC_SetSnvsOutput	Protected	Set the four output pins / two SPI select pins on the SNVS domain to the desired state.
RPC_GetFwUpdateState	Public	Retrieve the reset-proof firmware update status from the battery backed-up SNVS_LP_GPR register.
RPC_GetResetCause	Public	Get the reason why a reset was triggered by the watchdog implementation of the Cortex M4 core.
RPC_CommitFwUpdate	Public	Communicate a commit firmware update request to the bootloader. During next boot the new firmware will be committed and a recovery image will be created.
RPC_RevertFwUpdate	Public	Communicate a revert request to the bootloader. During next boot the old firmware will be restored.

5.8 Anomaly Detection API

The *anomaly detection API* provides access to the anomaly detection model executed by the *anomaly detection service* component. Through this API, it is possible to retrieve information about the anomaly detection model, get/set anomaly detection module parameters and provide motor status and audio samples to the module. The detection result is accessible via the *system status* event group.

An overview of the available functions is shown in Figure 25. Table 29 provides a brief description of each function. More details can be obtained from the i.MX RT Industrial Drive Development Platform software model as described in Section 5.

Note: *functions whose name is preceded by a + symbol are public functions that can be used by user applications. Functions whose name is preceded by the # symbol are internal (protected) i.MX RT Industrial Drive Development Platform functions that by default are not accessible to user applications.*



```
<<Interface>>
AnomalyDetectionAPI
+ADE_GetVersionInfo(pVersion : ADE_Version_st*): ADE_ReturnStatus_en
+ADE_GetModelInfo(pADEModel : uint8_t*, pModelInfo : ADE_ModelInfo_st*): ADE_ReturnStatus_en
+ADE_GetModelInfo_ADECurrent(hInstance : ADE_Handle_t, pModelInfo : ADE_ModelInfo_st*): ADE_ReturnStatus_en
+ADE_GetMemoryTable(hInstance : ADE_Handle_t, pMemoryTable : ADE_MemTab_t*, pInstParams : ADE_InstParams_t*): ADE_ReturnStatus_en
+ADE_GetInstanceHandle(phInstance : ADE_Handle_t*, pMemoryTable : ADE_MemTab_t*, pInstParams : ADE_InstParams_t*) : ADE_ReturnStatus_en
+ADE_Process(hInstance : ADE_Handle_t*, pInMotorCurrent_Data : int16_t*, motorCurrent_NumSamples : uint16_t, pInAudio_Data : int16_t*, audio_NumSamples
+ADE_UpdateModel(hInstance : ADE_Handle_t, pADEModel : uint8_t*) : ADE_ReturnStatus_en
+ADE_GetLibInfo(hInstance : ADE_Handle_t, pLibInfo : ADE_LibInfo_st*) : ADE_ReturnStatus_en
+ADE_SetControlParameters(hInstance : ADE_Handle_t, pParams : ADE_ControlParams_st*): ADE_ReturnStatus_en
+ADE_GetControlParameters(hInstance : ADE_Handle_t, pParams : ADE_ControlParams_st*) : ADE_ReturnStatus_en
```

Figure 25. Anomaly detection API functions

Table 29. Anomaly detection API functions

Name	Visibility	Description
ADE_GetVersionInfo	Public	This function is used to retrieve information about the library's version.
ADE_GetModelInfo	Public	This function is used to retrieve information about an ADE model.
ADE_GetModelInfo_ADECurrent	Public	This function is used to retrieve information about the model currently used by the ADE engine.
ADE_GetMemoryTable	Public	This function is used for memory allocation and free.
ADE_GetInstanceHandle	Public	This function is used to create a bundle instance. It returns the created instance handle through *phInstance*. All parameters are set to their default values.
ADE_Process	Public	Process function for the ADE module.
ADE_UpdateModel	Public	This function is used to update multi modal machine learning model.
ADE_GetLibInfo	Public	This function is used to get library and model information.
ADE_SetControlParameters	Public	Set the ADE module parameters.
ADE_GetControlParameters	Public	Get the ADE module parameters.

6 Appendix: compile-time options and configurations

The i.MX RT Industrial Drive Development Platform software is highly configurable to meet the requirements of the particular use case and application. Features and configurations can be easily turned on/off or changed by setting the value of the corresponding flags and parameters, so, for example, it is possible to:

- Configure motor control features and parameters such as maximum number of motors (up to 4) and speed and frequency limits;
- Activate / deactivate the anomaly detection feature;
- Activate / deactivate the hardened version of the software which is compliant with *ISA/IEC 62443-4-2 SL3* standard.

The complete and updated list of configurable features and parameters is provided in the *software model* in the *package diagram → features & compile-time configuration options* section. Features and configurations can also be checked and configured in the i.MX RT Industrial Drive Development Platform software source code in *qmc_features_config.h* and *qmc_cm4_features_config.h* header files.

7 Appendix: external libraries and dependencies

The table below lists the main external libraries and components used by the i.MX RT Industrial Drive Development Platform software. A complete and updated list of external libraries can be found in the *software model* in *package diagram → libraries* section.

Table 30. External libraries and dependencies used by i.MX RT Industrial Drive Development Platform software

Name	Description
FreeRTOS	FreeRTOS is an open source real-time operating system kernel for embedded devices and supports a variety of microcontroller platforms.
FatFs	FatFs is a generic FAT/exFAT filesystem module for small embedded systems. It uses the SD MMC SDIO Card middleware to interface with the SD card.
coreMQTT	This library comes as an add-on to FreeRTOS. It provides functionality to establish MQTT connections and push MQTT topic updates to an MQTT broker. It features integration with lwIP and mbedTLS.
mbedTLS	MbedTLS is an open-source library implementing TLS and SSL protocols together with accompanying cryptographic algorithms. It is targeting embedded systems.
SE hostlib	NXPs Secure IoT Middleware provides access to the SE051 secure element and general cryptographic functionality implemented in software via the Secure Sub-System (SSS) API. It provides mbedTLS with an alternative implementation (*mbedTLS_ALT*) that makes use of the cryptography features of the hardware. It integrates with *FreeRTOS* and the *coreMQTT* library to enable cloud onboarding aided by EdgeLock SE05x.
lwIP	Leightweight IP is a widely used open-source TCP/IP stack for embedded systems.
GenAVB TSN Stack	This library provides access to the TSN features of i.MX RT1170. It is bundled with a modified version of the lightweight IP (lwIP) TCP/IP stack, which allows best-effort and TSN connections at the same time. Best-effort connections are accessible through the standard lwIP APIs.
FreeMASTER	FreeMASTER is a standalone application that allows fine tuning of the motor control parameters for each motor.
RTCESL	NXPs Real Time Control Embedded Software Libraries provide highly optimized functions for mathematical computations, digital filters, motor control and power conversions. They will be used by the motor control algorithm of i.MX RT Industrial Drive Development Platform.

8 Legal information

8.1 Definitions

8.2 Disclaimers

8.3 Trademarks

Tables

Figures

Contents