

AN13643

i.MX RT Industrial Drive Development Platform Software Overview

Rev. 1.2 — 17 February 2025

Application note

Document information

Information	Content
Keywords	AN13643, i.MX, RT1170, ISA/IEC 62443, multimotor 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 multimotor control applications, hence reducing the required development effort and time to market of industrial components and solutions.



1 Acronyms

[Table 1](#) lists the acronyms used in this document.

Table 1. List of acronyms

Name	Description
AD	Anomaly Detection
ADC	Analog 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	Lightweight IP
MAD	Monitoring, Analytics, and Damage control
MCU	Microcontroller 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

Table 1. List of acronyms...continued

Name	Description
SNMP	Simple Network Management Protocol
SPI	Serial Peripheral Interface
SSL	Secure Sockets Layer
SSS	Secure Subsystem
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 modular-board kit that speeds up the development, evaluation, and validation of complex multimotor control applications for industrial robots, mobile robotics, multi-axis machinery, 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 applied to simultaneously control up to four Permanent Magnet Synchronous Motors (PMSMs), while handling advanced functionalities, such as data logging, fault detection, deterministic connectivity (Ethernet TSN), and complex user interfaces. By using on the **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 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 start developing multimotor control and other industrial applications quickly:

- The **i.MX RT Industrial Drive Development Platform hardware package** consists of a expansion card integrating the i.MX RT1170 crossover MCU, a digital board to expand the interfaces available to the expansion card, and a power-stage board to transform control commands into power signals for driving up to four servo motors. All the boards can be configured and adapted to meet the specific requirements of the application that is developed. More information on the hardware package are at <http://www.nxp.com/imxrtindustrialdrive>.
- The **i.MX RT Industrial Drive Development Platform software package** consists of a reference demo application and an API that demonstrate how to take advantage of i.MX RT Industrial Drive Development Platform hardware capabilities to develop a secure, robust, and reliable multimotor control system that meets the requirements, standards, and best practices required by industrial products. This significantly reduces the effort required to develop multimotor 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.

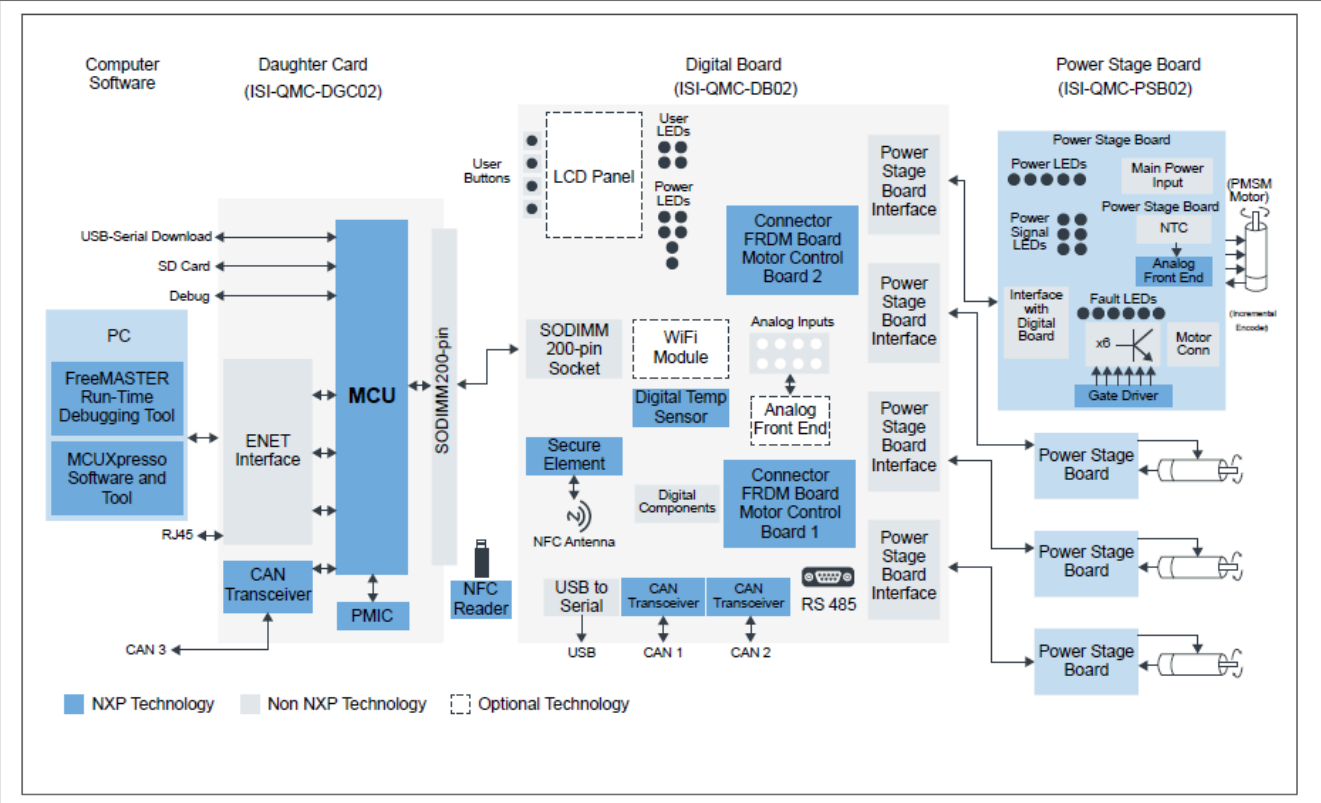


Figure 1. Hardware architecture of i.MX RT Industrial Drive Development Platform

The NXP components used for this platform along with part numbers are listed in [Table 2](#).

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 front-end	NXP NAFE 13388
CAN transceiver	NXPTJA115x (expansion 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 multimotor control applications** that leverage the **i.MX RT1170 MCU** along with 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 various **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 using 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 an **ISA/IEC 62443-4-2 SL3** certified solution that can be used as a starting point to develop certified industrial products and components.

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

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 using 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 an ISA/ IEC 62443-4-2 SL3 certified 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:

- **Nonrestricted:** with this version of the software, users have the possibility to configure 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 the ISA/IEC 62443-4-2 certification.
- **Restricted:** this version of the software is certified to be fully compliant with the 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 restricted software provides a reference implementation for users who want their product certified for ISA/IEC 62443-4-2 SL3.

Note: *The i.MX RT Industrial Drive Development Platform and software do not automatically grant the 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. It 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 the use-case-specific functionality.

- [Section 5](#) provides an overview of compile time options and configurations.
- [Section 5](#) provides an overview of external libraries and dependencies.
- [Section 5](#) provides the relevant legal information.

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 the i.MX RT Industrial Drive Development Platform hardware, and several underlying software resources, to demonstrate how to implement a secure multimotor 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 1](#).

As outlined in the diagram, the demo application relies on the i.MX RT Industrial Drive Development Platform Ethernet TSN support to exchange both the best-effort and time-sensitive data with local PCs and servers and with the cloud through secure TLS connections. Upon a 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 the static webpages that can then be accessed through a standard web browser and used to send motor commands and control the status of the system upon a 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 restart it if 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 verify that the system is still up and running.

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, and so on.

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 applicable. See the `qmc_features_config.h` file in the source folders for both the CM4 and the CM7 cores for more information on available configurations.

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 the 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 ran 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 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 form the i.MX RT Industrial Drive Development Platform software can be logically grouped based on the functionality they implement, as shown in [Figure 2](#):

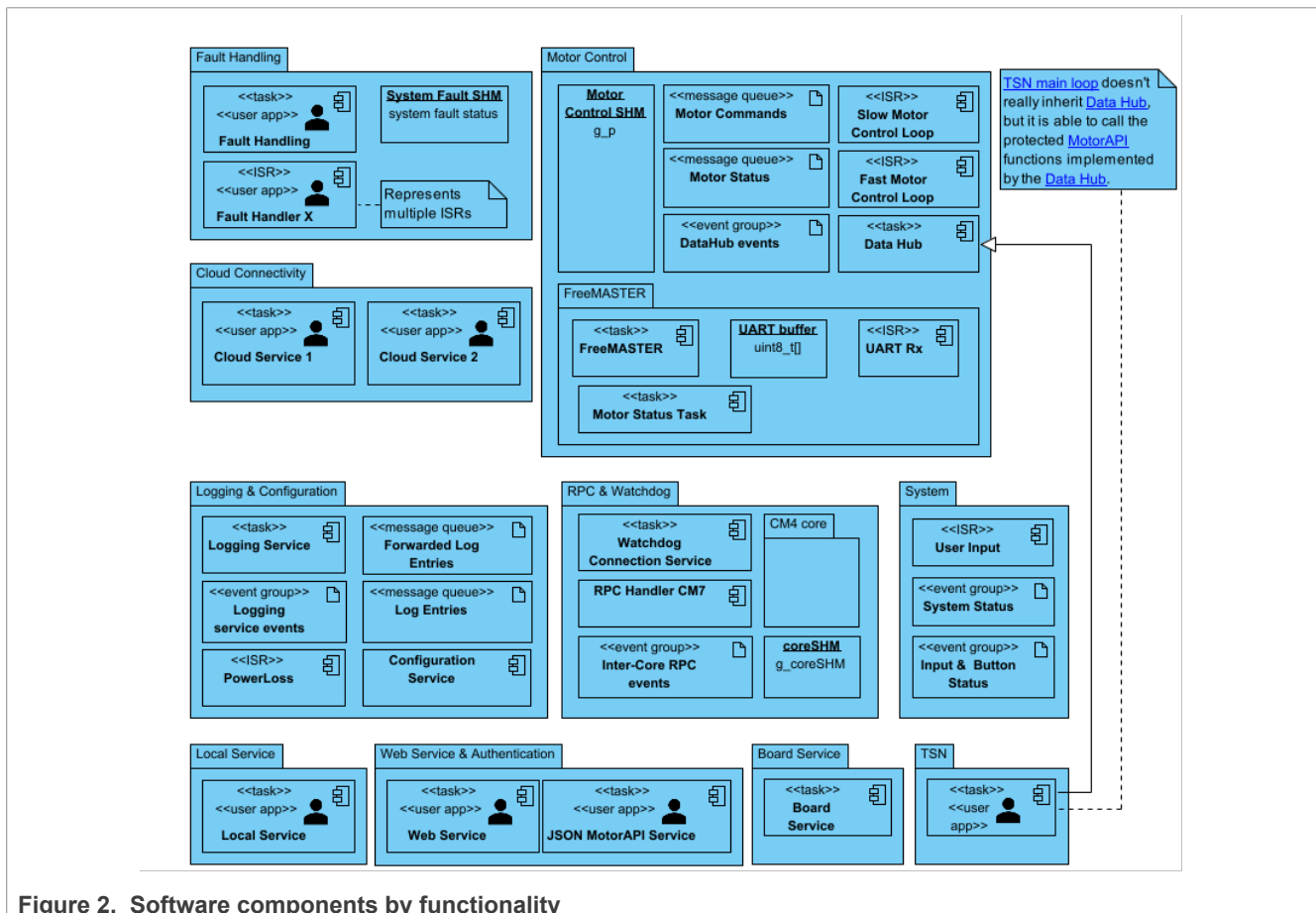


Figure 2. Software components by functionality

- **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 an 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.

- **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 (raised by parameter checks performed by the motor-control components) or hardware faults (such as 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 & Configuration:** these components handle the logs that are generated by other software components and write them in the appropriate location (such as an internal flash memory or an SD card) and set and retrieve the software configurations via a configuration service.
- **RPC & watchdog:** these components are responsible for connecting to the secure watchdog back-end service and forwarding the data received through the network to the watchdog task running in the Cortex-M4 core using Remote Procedure Calls (RPCs).
- **System management:** these components are responsible for keeping the status of the system and updating the user input status (such as buttons pressed or released) based on the 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 to the TSN network at a regular interval.
- **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: *Anomaly detection was not implemented in the final version.*

- **Others:** components are provided to deal with other operations, such as reacting to user inputs and updating the display (local service), implementing the web service functionality of the demo (web 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 the 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.

Note: *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	<p>Provides access to the motor-control algorithm. Functions can be called from regular FreeRTOS tasks only. Calling from the interrupt service routines is not allowed.</p> <p>The implementing task shall create one common message queue for the mc_motor_command_t entries and multiple individual queues for each connected task to send the mc_motor_status_t entries. Queuing the mc_motor_command_t or mc_motor_status_t shall raise the MOTOR_CMD_EVENT or MOTOR_STATUS_EVENT signals, respectively.</p>

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

Name	Description
Motor API ISR	Provides access to the motor-control algorithm. Functions can be called from regular FreeRTOS tasks, as well as from interrupt service routines.
Logging API	Provides access to the logging service. The implementing task shall create one common message queue for the log_record_t messages. Queuing a new entry shall raise the LOG_EVENT signal. Additionally, multiple individual queues for each connected task shall be created to forward log entries. Queuing the log_encrypted_record_t into one of these queues shall raise the corresponding LOG_FORWARD_EVENT signal.
Fault API	Provides access to the fault-handling system. Functions can be called from regular FreeRTOS tasks only. Calling from interrupt service routines is not allowed.
Fault API ISR	Provides access to the fault-handling system. Functions can be called from regular FreeRTOS tasks, as well as from interrupt service routines.
Configuration API	Provides functions to access the application configurations as key-value pairs. It allows to get or set the value of the i.MX RT Industrial Drive Development Platform software configurations.
AnomalyDetectionAPI	Provides access to the anomaly detection model, executed by the anomaly detection service. The detection result is accessible via the system-status event group.
BoardAPI	Provides access to board-specific features implemented on the Cortex-M7 core, like temperature sensors, display, and so on. It also provides functions for initialization, such as pinmux setting, clock setting, and so on.
RemoteProcedureCallAPI	Provides access to the features implemented on the Cortex-M4 core, like SPI slave selection, RTC, and secure watchdog. These functions are called on the Cortex-M7, while the actual function is executed on the Cortex-M4 core. Data is transferred using shared memory (g_coreSHM) and the global interrupt (GINT) of the IOMUX module.
RemoteProcedureCallAPI_boot	Provides access to features implemented on the Cortex-M4 that are required during boot (from the SBL).
LoggingAPI_boot	Logging functionality required by the bootloader. It can be used by the bootloader only.
Web Service API	-

The overview of all software components, and how these components to interact through the available APIs, is shown in [Figure 3](#). [Table 4](#) describes how to read the diagram.

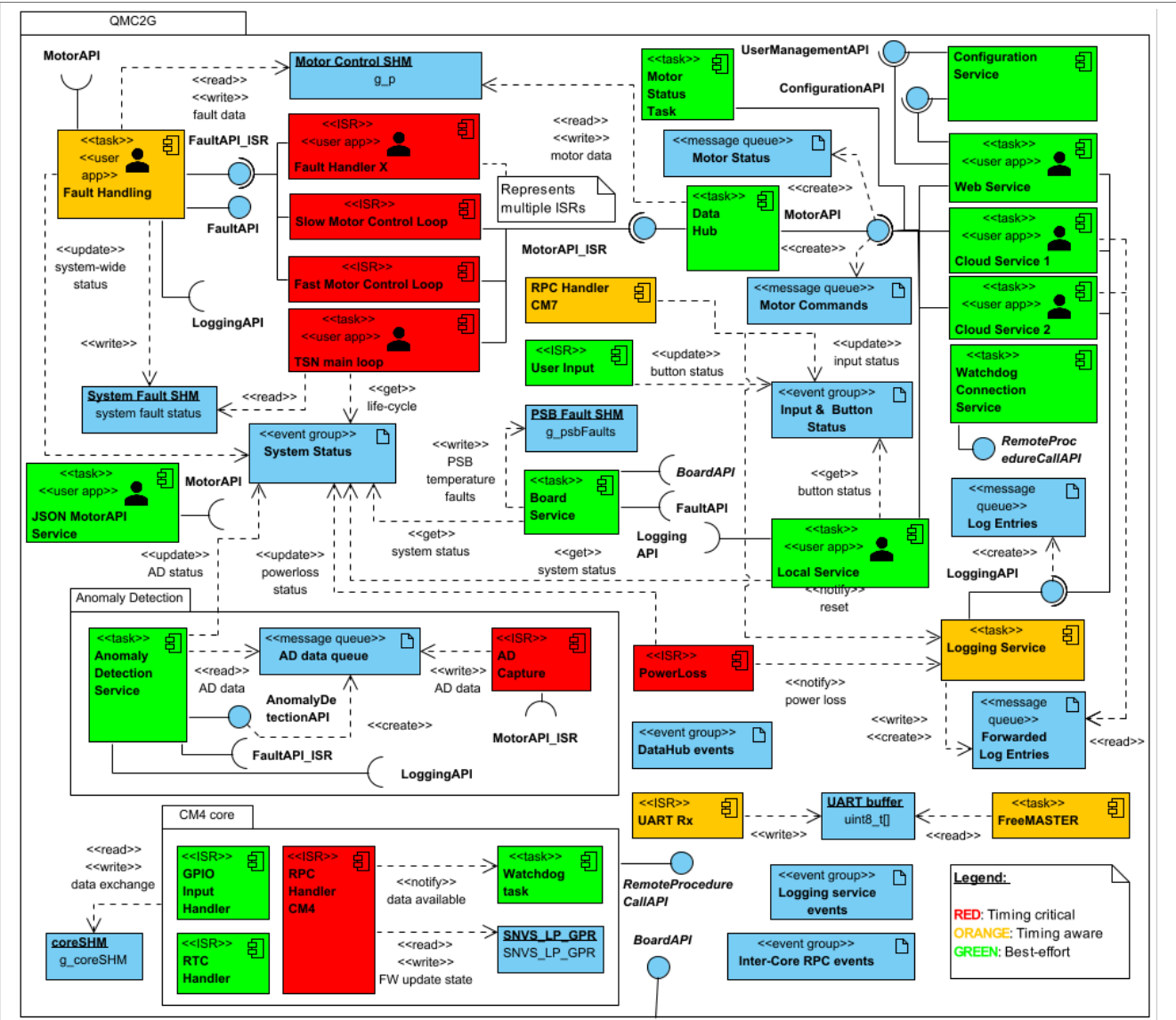


Figure 3. Software components and interaction through APIs

Table 4. Legend

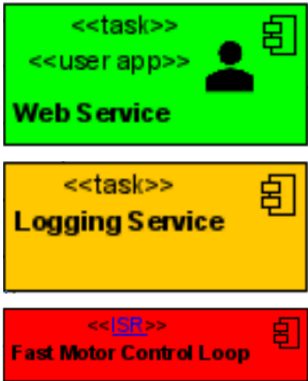
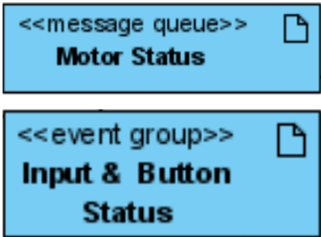
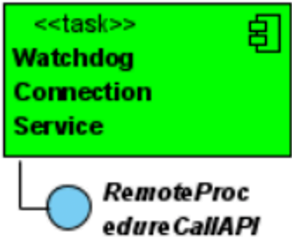
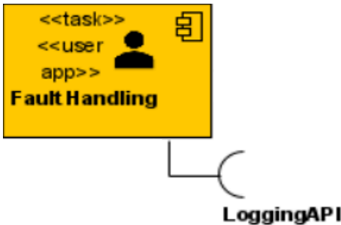
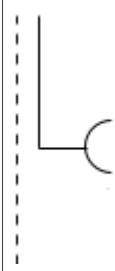
Symbol	Description
<div></div>	<p>Green, red, and yellow rectangles represent the software components (tasks or ISRs). The name of the component is written in bold, while the type of the component (task or ISR) is specified between <<>>.</p> <p>The color of the rectangle defines the priority assigned 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 are executed using a best-effort policy.</p> <p>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 using on the public API that is exposed by the other components.</p>
<div></div>	<p>Blue rectangles represent artifacts (event groups and message queues) or other data structures (such as shared memory).</p>
<div></div>	<p>Blue circles represent the 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.</p>
<div></div>	<p>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.</p>

Table 4. Legend...continued

Symbol	Description
	-

Note: The diagram shown in [Figure 3](#) does not 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 be simplified or removed to improve legibility.

3.3 Overview of main software-security features

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

Note: The security features can be enabled/disabled using software configurations. The **ISA/IEC 62443** certified version of the software (see [Section 2.2](#)) automatically activates all security features required to comply with the **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). The Encrypted XiP is 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 the 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 the i.MX RT Industrial Drive Development Platform using a secure channel. The firmware update image is preauthenticated 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 preauthentication, the control is passed to the SBL to finalize the firmware-update process. A local update using an SD card is also supported and directly handled by the SBL.
- **Logging:** log records are signed and encrypted using the keys stored in EdgeLock SE05x before saving them to the SD card or sending them to an external service (such as a cloud service).
- **TCP/IP connections:** whenever a TCP/IP connection is required (such as 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.

- [Section 4.1](#)
- [Section 4.2](#)

- [Section 4.3](#)
- [Section 4.4](#)
- [Section 4.5](#)
- [Section 4.6](#)
- [Section 4.7](#)
- [Section 4.8](#)

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 4](#).

The **data hub** task implements the **Motor API** and **Motor API ISR** and it acts as an interface for all components to queue motor commands in the **motor commands** message queue. Commands are then sent to the shared memory (**Motor_Control_SHM**) where they are executed by the motor-control algorithm. The **data hub** also pushes the motor status at a 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 the **motor status** updates.

The **slow motor control loop** and **fast motor control loop** components are called repeatedly for each motor and update the **motor status** queues through the **Motor API ISR**. 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 the **UART RX** task and pushed to the **UART buffer**, where they are finally read by the **FreeMASTER** task and queued for execution. The **FreeMASTER** task has access to inner variables of the **slow motor control loop** and **fast motor control loop**.

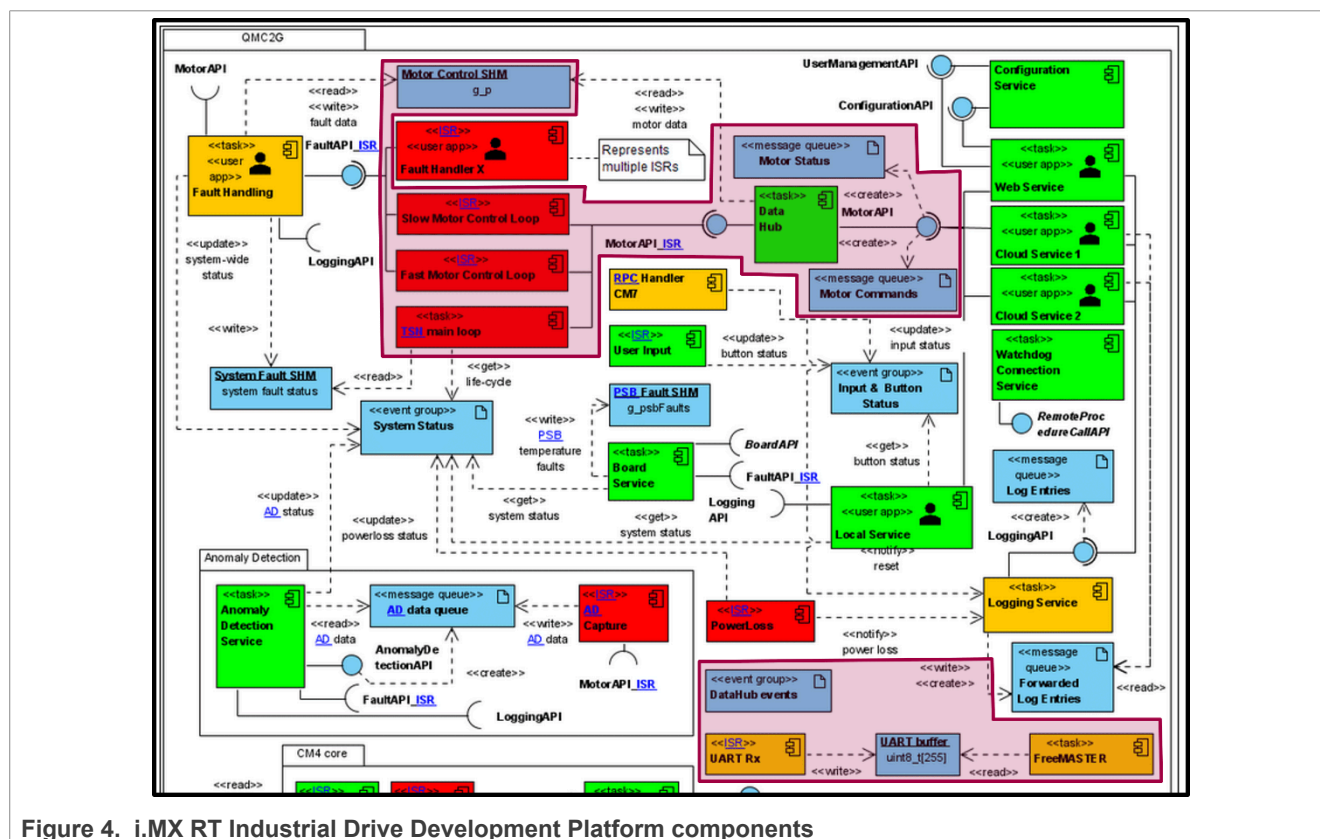


Figure 4. i.MX RT Industrial Drive Development Platform components

Table 5. Motor control and TSN components

Name	Type	Description
Data hub	Task	The data hub task bridges the 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 dequeued when they become available and 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 MOTOR_STATUS_EVENT event.
Slow motor-control loop	ISR	Deals with the slow aspects of motor control, such as the DC-bus voltage measurements. It reads the motor commands from the shared memory and executes them. It also updates part of the motor status values using MotorAPI_ISR. It can update the fault status of a motor and emit FAULT events using FaultAPI_ISR.
Fast motor-control loop	ISR	Deals with the fast aspects of motor control, such as PWM control, phase current measurements, and so on. It updates part of the motor status values using MotorAPI_ISR. It can update the fault status of a motor and emit FAULT events using FaultAPI_ISR.
TSN main loop	ISR	Handles the sending and receiving of TSN packets. Whenever a packet that contains new motor commands is received via TSN, it sets the motor commands for execution using (model element not found) MotorAPI_ISR. It sends the motor and system status to the TSN network at a regular interval.
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 that allows to disable this task is required. FreeMASTER has direct access to the inner variables of the fast motor-control loop and the slow motor-control loop, which is not modeled in detail here.
UART RX	ISR	UART receive handler. Writes the incoming data to the UART buffer for further processing by FreeMASTER. This handler is instantiated and registered only if FreeMASTER is enabled in the build configuration.
JSON MotorAPI service	Task User app	Provides the web service with motor status values. It registers a motor status queue handle via Motor API and forwards the status values to the web service.
Motor status task	Task	If enabled, it provides the backend of the motor-control page in the corresponding FreeMASTER project. It registers a motor status queue handle via Motor API and forwards the status values to the FreeMASTER motor-control page. The motor commands, received via FreeMASTER, are queued for execution. FEATURE_GET_MOTOR_STATUS_FROM_DATA_HUB: boolean enables this task.

Table 6. Motor control and TSN artifacts

Name	Type	Description
Motor commands	Message queue	Message queue for motor commands. Elements get queued using the MC_QueueMotorCommand call of the Motor API. The data hub then dequeues the commands and transfers them to the shared memory for execution.
Motor status	Message queue	Message queues for motor status values. Elements are queued by the data hub. A task can dequeue them using the MC_DequeueMotorStatus call of the Motor API. Each task that wants to receive motor status values must register its own queue with the data hub using the Motor API. The maximum number of queues available can be defined at compile time. Each queue has its own event bit.

Table 6. Motor control and TSN artifacts...continued

Name	Type	Description
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. See the i.MX RT Industrial Drive Development Platform software model for a description of all event bits in 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 using Motor API , MotorAPI_ISR , and FaultAPI_ISR . For a detailed description of the shared memory, see its data type definition (mc_control_SHM).
UART buffer	Other	Data buffer for the 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 the i.MX RT Industrial Drive Development Platform. The components are highlighted in [Figure 5](#).

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 the future versions of the i.MX RT Industrial Drive Development Platform software.

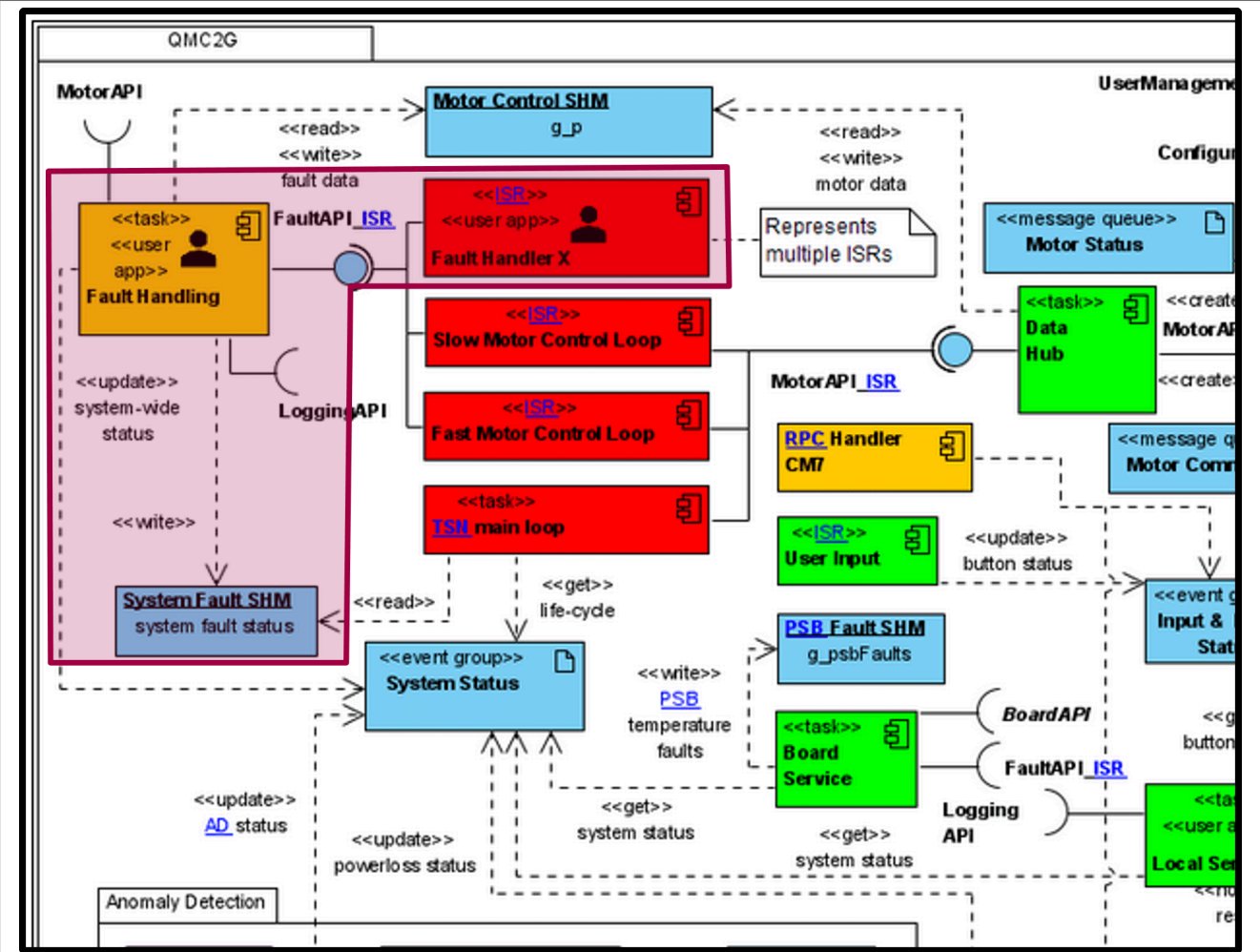


Figure 5. Fault-handling components

Table 7. Fault-handling components

Name	Type	Description
Fault handling	Task (user app)	Reacts on the fault events and decides on an appropriate fault action. The fault action is a motor command (<code>mc_motor_command_t</code>), sent via the Motor API for execution. Faults may be software faults (such as parameter checks performed by the fast motor-control loop) or hardware faults (such as PWM error). This task reacts on the 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 several handlers; each with its own behavior. It shall send a FAULT event indicating the <code>fault_source_t</code> to the fault handling task using <code>FAULT_RaiseFaultEvent_fromISR</code> from <code>FaultAPI_ISR</code> . The demo application uses this feature for the MCU overtemperature and PWM fault events only.

Table 8. Fault-handling artifacts

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

4.3 Cloud connectivity

These components and artifacts implement the cloud connectivity functionality of the i.MX RT Industrial Drive Development Platform demo application. The components are highlighted in [Figure 6](#).

The i.MX RT Industrial Drive Development Platform leverages on the **cloud service 1** and **cloud service 2** tasks to open up to two concurrent cloud connections to either Azure or AWS. 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.

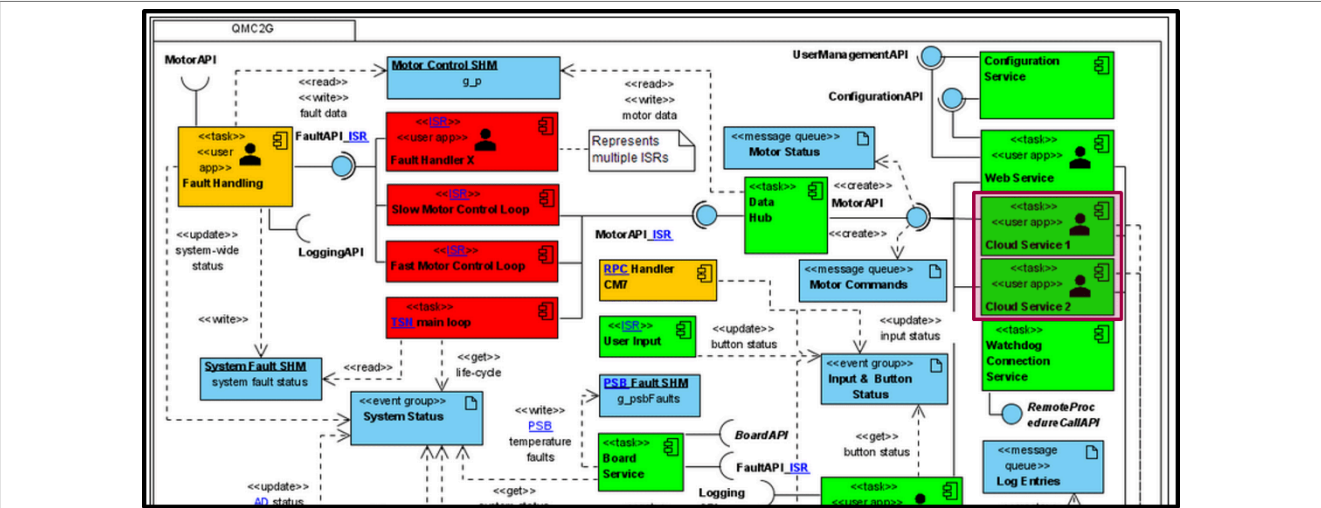


Figure 6. Cloud connectivity components

Table 9. Cloud connectivity components

Name	Type	Description
Cloud service 1	Task (user app)	If the configuration parameters for a cloud service are set, the cloud service 1 task shall establish a TLS secured connection to the configured cloud service. Otherwise, it goes to the 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 broker/cloud responsibility to provide history access to the data where it is required (such as logs). The cloud service task shall reset the QMC_SYSEVENT_LOG_MessageLost event bit after reporting it via MQTT.
Cloud service 2	Task (user app)	Same as the 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 the i.MX RT Industrial Drive Development Platform. The components are highlighted in [Figure 7](#).

i.MX RT Industrial Drive Development Platform Software Overview

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 the subscribed tasks (**forwarded log entries** message queue(s)).

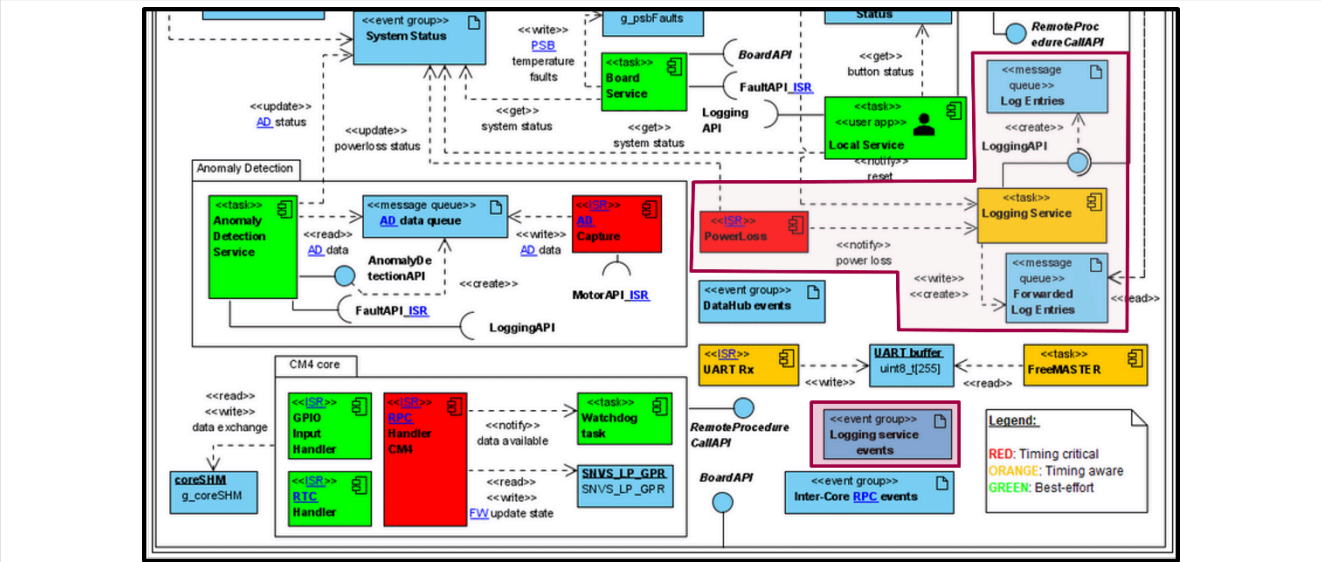


Figure 7. i.MX RT Industrial Drive Development Platform components

Table 10. Logging components

Name	Type	Description
Logging service	Task	<p>The logging service implements the logging API. It manages the audit log stored in the flash and SD card. New log entries are dequeued from the queue and added to the log. Every log message must be written to the dedicated internal flash memory.</p> <p>When a POWER_LOSS event is received, all unnecessary components shall be disabled and the log entries must be 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.</p> <p>If applicable, certain log entries can be forwarded to each registered forwarded log entries queue. Several log entries may be combined into one when forwarding.</p> <p>If the free logging space gets lower than a configurable (compile time) threshold, the logging service shall set an event bit (QMC_SYSEVENT_LOG_LowMemory) in the system status event group. The logging service shall clear the event bit if the free logging space exceeds this threshold.</p> <p>The logging service shall set an event bit (QMC_SYSEVENT_LOG_MessageLost) in the system status event group if a log record that has not been reported to the cloud is overwritten. This event bit may also be set if a general logging error that leads to an event not being recorded occurs.</p>
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 are queued using a call to the logging API. The logging service then dequeues and processes these elements.

Table 11. Logging artifacts...continued

Name	Type	Description						
Forwarded log entries	Message queue	Message queues for encrypted log messages that are forwarded to some external processing, such as cloud storage. Elements are queued by the logging service. A task can dequeue them using the LOG_DequeueEncryptedLogEntry call of the logging API. Each task that wants to receive logging messages for further processing must register its own queue with the logging service using the logging API. The maximum number of queues available can be defined at compile time; each queue has its own event bit.						
Logging service events	Event group	<div>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.<table><tr><th>Bit</th><th>Event</th></tr><tr><td>1</td><td>Log Entries queue event</td></tr><tr><td>2-24</td><td>Pool for Forwarded Log Entries queue events</td></tr></table></div>	Bit	Event	1	Log Entries queue event	2-24	Pool for Forwarded Log Entries queue events
Bit	Event							
1	Log Entries queue event							
2-24	Pool for Forwarded Log Entries queue events							

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. The components are highlighted in [Figure 8](#).

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 the functional watchdogs and the secure watchdog. The **RPC Handler CM4** ISR processes the calls to the **RPC API** and triggers the appropriate action in the Cortex-M4 core, such as kicking the watchdog, setting the output pins on the SNVS domain, or getting 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 core, and triggers the appropriate action in the Cortex-M7 core.

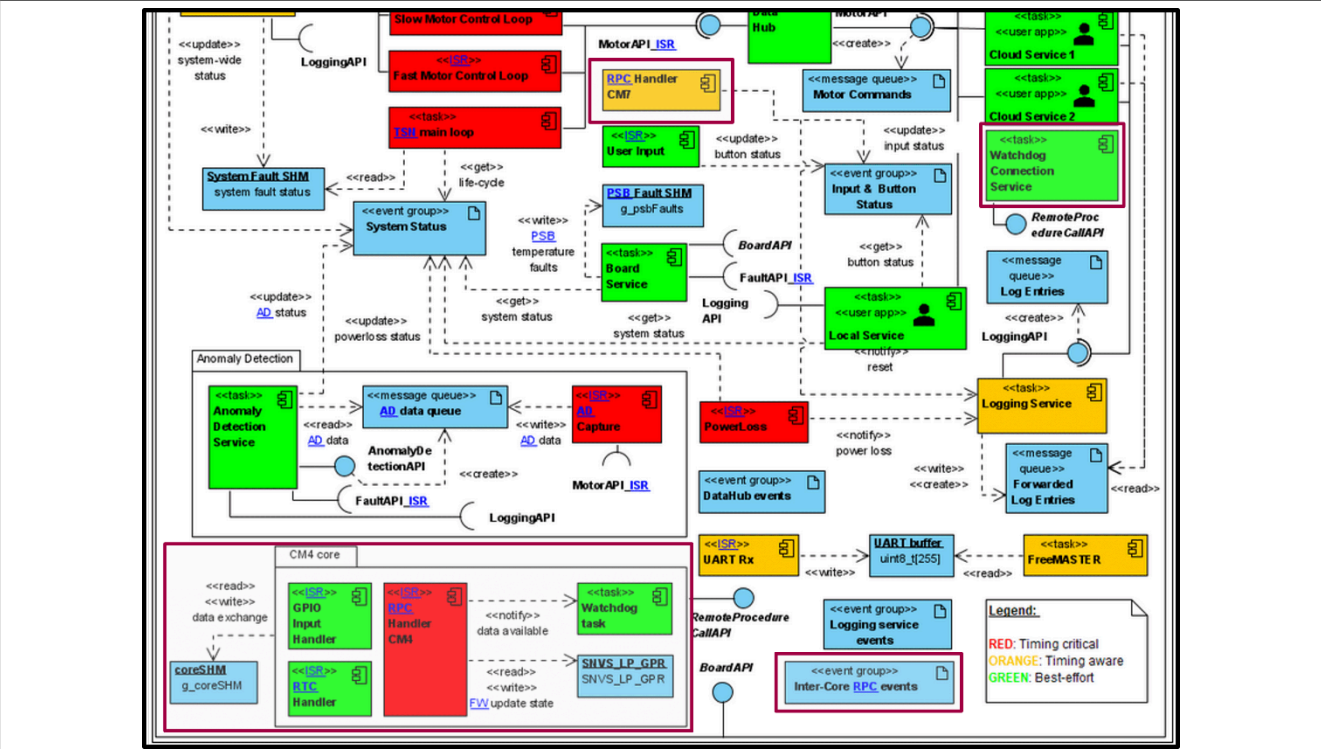


Figure 8. RPC and watchdog components

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 must be configured at compile time. The watchdog task shall also implement an RTC using the secure real-time counter of the i.MX RT1170.
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, such as kicking the watchdog, setting the output pins on the SNVS domain, and so on.
GPIO Input handler	ISR	Triggered by the rising and falling edges on either of the four slow SNVS domain user input pins. It triggers an event via the RPC interface. This is done to prevent a slowdown of the Cortex-M7 core, which occurs when accessing the SNVS domain directly.
RTC handler	ISR	Interrupt handler for the SNVS RTC interrupts. Used for ticking the logical watchdogs by calling <code>LWDGU_Tick(unit_ptr : logical_watchdog_unit_t*) : lwdg_tick_status_t</code> , triggering a notification about an upcoming system reset in case of a logical watchdog expiration and resetting the system. Note that this interrupt handler is shared between the high-power RTC and the low-power Security RTC (SRTC).
RPC handler CM7	Task	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. The data received via the network is forwarded to the watchdog task in the Cortex-M4 core and vice versa. Data forwarding uses the functionality defined by the RemoteProcedureCall API .

Table 13. RPC and watchdog artifacts

Name	Type	Description																																																						
coreSHM	Shared memory	Shared memory of the rpc_shm_t type that holds the necessary data for the RemoteProcedureCallAPI .																																																						
SNVS_LP_GPR	Data structure	<p>Represents the battery backed-up SNVS_LP_GPR registers. The table below lists the 16 bytes and their functions.</p> <table><tr><th>Byte</th><th>Functional group</th><th>Data object</th><th>Byte</th><th>Functional Group</th><th>Data object</th></tr><tr><td>1</td><td>Watchdog</td><td>Timer backup</td><td>9</td><td>Watchdog</td><td>SRTC Offset</td></tr><tr><td>2</td><td>Watchdog</td><td>Timer backup</td><td>10</td><td>Watchdog</td><td>SRTC Offset</td></tr><tr><td>3</td><td>Watchdog</td><td>Status data</td><td>11</td><td>Watchdog</td><td>SRTC Offset</td></tr><tr><td>4</td><td>Bootloader</td><td>FW Update status</td><td>12</td><td>Watchdog</td><td>SRTC Offset</td></tr><tr><td>5</td><td>Watchdog</td><td>Reset reason</td><td>13</td><td>Watchdog</td><td>SRTC Offset</td></tr><tr><td>6</td><td></td><td>Reserved</td><td>14</td><td>Watchdog</td><td>SRTC Offset</td></tr><tr><td>7</td><td></td><td>Reserved</td><td>15</td><td>Watchdog</td><td>SRTC Offset</td></tr><tr><td>8</td><td></td><td>Reserved</td><td>16</td><td>Watchdog</td><td>SRTC Offset</td></tr></table>	Byte	Functional group	Data object	Byte	Functional Group	Data object	1	Watchdog	Timer backup	9	Watchdog	SRTC Offset	2	Watchdog	Timer backup	10	Watchdog	SRTC Offset	3	Watchdog	Status data	11	Watchdog	SRTC Offset	4	Bootloader	FW Update status	12	Watchdog	SRTC Offset	5	Watchdog	Reset reason	13	Watchdog	SRTC Offset	6		Reserved	14	Watchdog	SRTC Offset	7		Reserved	15	Watchdog	SRTC Offset	8		Reserved	16	Watchdog	SRTC Offset
Byte	Functional group	Data object	Byte	Functional Group	Data object																																																			
1	Watchdog	Timer backup	9	Watchdog	SRTC Offset																																																			
2	Watchdog	Timer backup	10	Watchdog	SRTC Offset																																																			
3	Watchdog	Status data	11	Watchdog	SRTC Offset																																																			
4	Bootloader	FW Update status	12	Watchdog	SRTC Offset																																																			
5	Watchdog	Reset reason	13	Watchdog	SRTC Offset																																																			
6		Reserved	14	Watchdog	SRTC Offset																																																			
7		Reserved	15	Watchdog	SRTC Offset																																																			
8		Reserved	16	Watchdog	SRTC Offset																																																			
Intercore RPC events	Event group	<p>This event group represents the status of the remote procedure calls (called on the Cortex-M7 core and executed on the Cortex-M4 core). These events are required to synchronize on the return value and returned data. The user code must not touch them.</p> <p>The table below lists all event bits of the event group.</p>																																																						

Table 13. RPC and watchdog artifacts...continued

Name	Type	Description					
		Bit	Event	Bit	Event	Bit	Event
		1	RPC_SECWD	9	Reserved	17	Reserved
		2	RPC_FUNCWD	10	Reserved	18	Reserved
		3	RPC_GPOUTPUT	11	Reserved	19	Reserved
		4	RPC_RTC	12	Reserved	20	Reserved
		5	RPC_FWUPDATE	13	Reserved	21	Reserved
		6	Reserved	14	Reserved	22	Reserved
		7	Reserved	15	Reserved	23	Reserved
		8	Reserved	16	Reserved	24	Reserved

4.6 System

These components and artifacts deal with system-related functionalities of i.MX RT Industrial Drive Development Platform. The components are highlighted in [Figure 9](#).

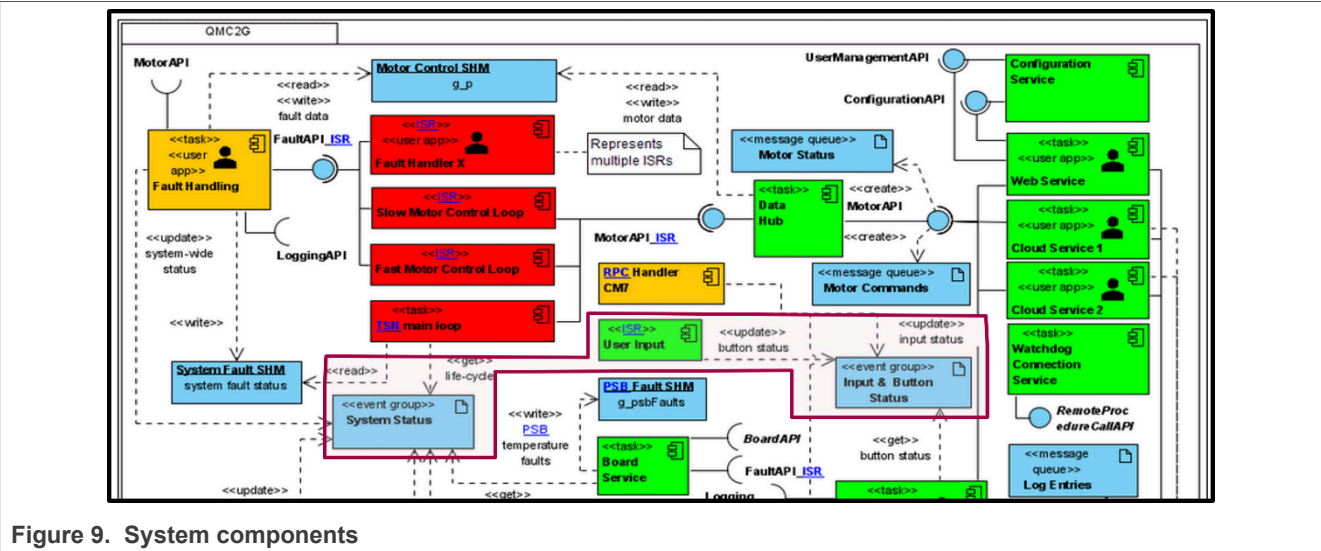


Figure 9. System components

Table 14. System components

Name	Type	Description
User input	ISR	This ISR is triggered by the rising and falling edges on the fast GPIO input pins and the pins used for the 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. The table below lists all event bits of the event group and the functional group they belong to.

Table 15. System artifacts...continued

Name	Type	Description																																																																														
		<table><tr><th>Bit</th><th>Functional group</th><th>Event</th><th>Bit</th><th>Functional group</th><th>Event</th></tr><tr><td>1</td><td>Life cycle</td><td>Commissioning</td><td>13</td><td>Configuration</td><td>Configuration changed</td></tr><tr><td>2</td><td>Life cycle</td><td>Operational</td><td>14</td><td>Anomaly detection</td><td>Audio anomaly detected</td></tr><tr><td>3</td><td>Life cycle</td><td>Error</td><td>15</td><td>Anomaly detection</td><td>Current anomaly detected</td></tr><tr><td>4</td><td>Life cycle</td><td>Maintenance</td><td>16</td><td>Shutdown</td><td>Power loss</td></tr><tr><td>5</td><td>Life cycle</td><td>Decommissioning</td><td>17</td><td>Shutdown</td><td>Watchdog reset</td></tr><tr><td>6</td><td>Fault</td><td>Motor 1</td><td>18</td><td>Memory</td><td>SD card available</td></tr><tr><td>7</td><td>Fault</td><td>Motor 2</td><td>19</td><td>Network</td><td>TSN synchronization lost</td></tr><tr><td>8</td><td>Fault</td><td>Motor 3</td><td>20</td><td>Network</td><td>No link</td></tr><tr><td>9</td><td>Fault</td><td>Motor 4</td><td>21</td><td>Log</td><td>Flash Error</td></tr><tr><td>10</td><td>Fault</td><td>System</td><td>22</td><td>Log</td><td>Memory Low</td></tr><tr><td>11</td><td>FW update</td><td>Restart required</td><td>23</td><td>Log</td><td>Message Lost</td></tr><tr><td>12</td><td>FW update</td><td>Verify mode</td><td>24</td><td></td><td></td></tr></table> <p>Macros start with QMC_SYSEVENT_.</p>	Bit	Functional group	Event	Bit	Functional group	Event	1	Life cycle	Commissioning	13	Configuration	Configuration changed	2	Life cycle	Operational	14	Anomaly detection	Audio anomaly detected	3	Life cycle	Error	15	Anomaly detection	Current anomaly detected	4	Life cycle	Maintenance	16	Shutdown	Power loss	5	Life cycle	Decommissioning	17	Shutdown	Watchdog reset	6	Fault	Motor 1	18	Memory	SD card available	7	Fault	Motor 2	19	Network	TSN synchronization lost	8	Fault	Motor 3	20	Network	No link	9	Fault	Motor 4	21	Log	Flash Error	10	Fault	System	22	Log	Memory Low	11	FW update	Restart required	23	Log	Message Lost	12	FW update	Verify mode	24		
Bit	Functional group	Event	Bit	Functional group	Event																																																																											
1	Life cycle	Commissioning	13	Configuration	Configuration changed																																																																											
2	Life cycle	Operational	14	Anomaly detection	Audio anomaly detected																																																																											
3	Life cycle	Error	15	Anomaly detection	Current anomaly detected																																																																											
4	Life cycle	Maintenance	16	Shutdown	Power loss																																																																											
5	Life cycle	Decommissioning	17	Shutdown	Watchdog reset																																																																											
6	Fault	Motor 1	18	Memory	SD card available																																																																											
7	Fault	Motor 2	19	Network	TSN synchronization lost																																																																											
8	Fault	Motor 3	20	Network	No link																																																																											
9	Fault	Motor 4	21	Log	Flash Error																																																																											
10	Fault	System	22	Log	Memory Low																																																																											
11	FW update	Restart required	23	Log	Message Lost																																																																											
12	FW update	Verify mode	24																																																																													
Input & button status	Event group	<p>This event group represents the status (pressed/released) of user input buttons and the protective lid as well as the status (high/low) of the user input pins. It shall feature the event bits for both states (pressed/low and released/high), as FreeRTOS only allows to wait for the event bits getting set.</p> <p>The table below lists all event bits of the event group. The lid is represented by one of the input states (implementation hint: define the macros to link lids to an input pin, such as: QMC_IOEVENT_LID_OPEN_SD, QMC_IOEVENT_LID_CLOSE_SD, QMC_IOEVENT_LID_OPEN_BUTTON, QMC_IOEVENT_LID_CLOSE_BUTTON).</p> <table><tr><th>Bit</th><th>Event</th><th>Bit</th><th>Event</th></tr><tr><td>1</td><td>QMC_IOEVENT_BTN1_PRESSEED</td><td>13</td><td>QMC_IOEVENT_INPUT2_HIGH</td></tr><tr><td>2</td><td>QMC_IOEVENT_BTN1_RELEASED</td><td>14</td><td>QMC_IOEVENT_INPUT2_LOW</td></tr><tr><td>3</td><td>QMC_IOEVENT_BTN2_PRESSEED</td><td>15</td><td>QMC_IOEVENT_INPUT3_HIGH</td></tr><tr><td>4</td><td>QMC_IOEVENT_BTN2_RELEASED</td><td>16</td><td>QMC_IOEVENT_INPUT3_LOW</td></tr><tr><td>5</td><td>QMC_IOEVENT_BTN3_PRESSEED</td><td>17</td><td>QMC_IOEVENT_INPUT4_HIGH</td></tr><tr><td>6</td><td>QMC_IOEVENT_BTN3_RELEASED</td><td>18</td><td>QMC_IOEVENT_INPUT4_LOW</td></tr><tr><td>7</td><td>QMC_IOEVENT_BTN4_PRESSEED</td><td>19</td><td>QMC_IOEVENT_INPUT5_HIGH</td></tr><tr><td>8</td><td>QMC_IOEVENT_BTN4_RELEASED</td><td>20</td><td>QMC_IOEVENT_INPUT5_LOW</td></tr><tr><td>9</td><td>QMC_IOEVENT_INPUT0_HIGH</td><td>21</td><td>QMC_IOEVENT_INPUT6_HIGH</td></tr><tr><td>10</td><td>QMC_IOEVENT_INPUT0_LOW</td><td>22</td><td>QMC_IOEVENT_INPUT6_LOW</td></tr><tr><td>11</td><td>QMC_IOEVENT_INPUT1_HIGH</td><td>23</td><td>QMC_IOEVENT_INPUT7_HIGH</td></tr><tr><td>12</td><td>QMC_IOEVENT_INPUT1_LOW</td><td>24</td><td>QMC_IOEVENT_INPUT7_LOW</td></tr></table>	Bit	Event	Bit	Event	1	QMC_IOEVENT_BTN1_PRESSEED	13	QMC_IOEVENT_INPUT2_HIGH	2	QMC_IOEVENT_BTN1_RELEASED	14	QMC_IOEVENT_INPUT2_LOW	3	QMC_IOEVENT_BTN2_PRESSEED	15	QMC_IOEVENT_INPUT3_HIGH	4	QMC_IOEVENT_BTN2_RELEASED	16	QMC_IOEVENT_INPUT3_LOW	5	QMC_IOEVENT_BTN3_PRESSEED	17	QMC_IOEVENT_INPUT4_HIGH	6	QMC_IOEVENT_BTN3_RELEASED	18	QMC_IOEVENT_INPUT4_LOW	7	QMC_IOEVENT_BTN4_PRESSEED	19	QMC_IOEVENT_INPUT5_HIGH	8	QMC_IOEVENT_BTN4_RELEASED	20	QMC_IOEVENT_INPUT5_LOW	9	QMC_IOEVENT_INPUT0_HIGH	21	QMC_IOEVENT_INPUT6_HIGH	10	QMC_IOEVENT_INPUT0_LOW	22	QMC_IOEVENT_INPUT6_LOW	11	QMC_IOEVENT_INPUT1_HIGH	23	QMC_IOEVENT_INPUT7_HIGH	12	QMC_IOEVENT_INPUT1_LOW	24	QMC_IOEVENT_INPUT7_LOW																										
Bit	Event	Bit	Event																																																																													
1	QMC_IOEVENT_BTN1_PRESSEED	13	QMC_IOEVENT_INPUT2_HIGH																																																																													
2	QMC_IOEVENT_BTN1_RELEASED	14	QMC_IOEVENT_INPUT2_LOW																																																																													
3	QMC_IOEVENT_BTN2_PRESSEED	15	QMC_IOEVENT_INPUT3_HIGH																																																																													
4	QMC_IOEVENT_BTN2_RELEASED	16	QMC_IOEVENT_INPUT3_LOW																																																																													
5	QMC_IOEVENT_BTN3_PRESSEED	17	QMC_IOEVENT_INPUT4_HIGH																																																																													
6	QMC_IOEVENT_BTN3_RELEASED	18	QMC_IOEVENT_INPUT4_LOW																																																																													
7	QMC_IOEVENT_BTN4_PRESSEED	19	QMC_IOEVENT_INPUT5_HIGH																																																																													
8	QMC_IOEVENT_BTN4_RELEASED	20	QMC_IOEVENT_INPUT5_LOW																																																																													
9	QMC_IOEVENT_INPUT0_HIGH	21	QMC_IOEVENT_INPUT6_HIGH																																																																													
10	QMC_IOEVENT_INPUT0_LOW	22	QMC_IOEVENT_INPUT6_LOW																																																																													
11	QMC_IOEVENT_INPUT1_HIGH	23	QMC_IOEVENT_INPUT7_HIGH																																																																													
12	QMC_IOEVENT_INPUT1_LOW	24	QMC_IOEVENT_INPUT7_LOW																																																																													

4.7 Anomaly detection

These components and artifacts deal with the anomaly detection functionality of i.MX RT Industrial Drive Development Platform. The components are highlighted in [Figure 10](#).

The anomaly detection functionality is implemented by the **anomaly detection service**. The service runs an anomaly detection algorithm that feeds on the data captured by the i.MX RT Industrial Drive Development Platform microphone and tries to detect a malfunction 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 the **anomaly detection service** reads the data.

Note: The anomaly detection functionality is NOT part of the i.MX RT Industrial Drive Development Platform ISA/IEC 63442-4-2 certification and was NOT included in the final release.

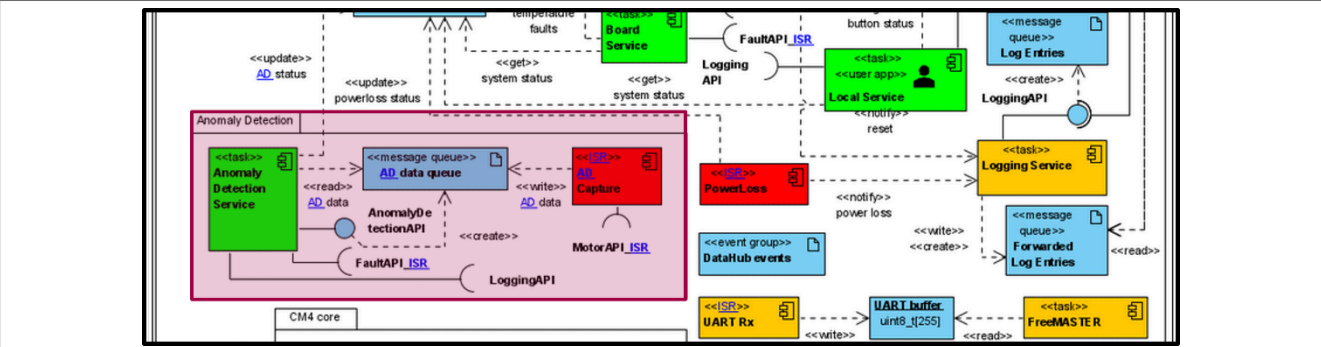


Figure 10. Anomaly detection components

Table 16. Anomaly detection components

Name	Type	Description
AD capture	ISR	<p>The AD capture ISR reads blocks of samples from the digital microphone via the DMA, as well as blocks of mc_motor_status_fast_t values via MC_GetFastMotorStatus Sync_fromISR(motorId : mc_motor_id_t, status : mc_motor_status_fast_t**) : qmc_status_t of the MotorAPI_ISR. It aligns the blocks and queues them into the AD data queue for further processing by the anomaly detection service.</p> <p>The rate of data blocks must be aligned for the microphone data and motor current values. On the motor-control side, it can be controlled by AD_CURRENT_BLK_SIZE. On the audio side, it can be controlled by the PDM microphone sampling rate and the DMA transfer block size. The DMA transfer shall use double buffering.</p> <p>When setting up the DMA for the PDM microphone, make sure it doesn't stall the memory access (wait state due to mapping external memory on FlexSPI). When possible, derive the microphone sampling clock from the same clock source as the timer triggering the fast motor-control loop. This guarantees the synchronicity between motor status values and microphone samples.</p>
Anomaly detection service	Task	<p>This task runs the actual anomaly detection algorithm. The captured data is read from the AD data queue.</p>

Table 17. Anomaly detection artifacts

Name	Type	Description
AD data queue	Message queue	<p>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.</p>

4.8 Others

These components and artifacts deal with other functionalities of the i.MX RT Industrial Drive Development Platform, not covered in the previous sections. The components are highlighted in [Figure 11](#).

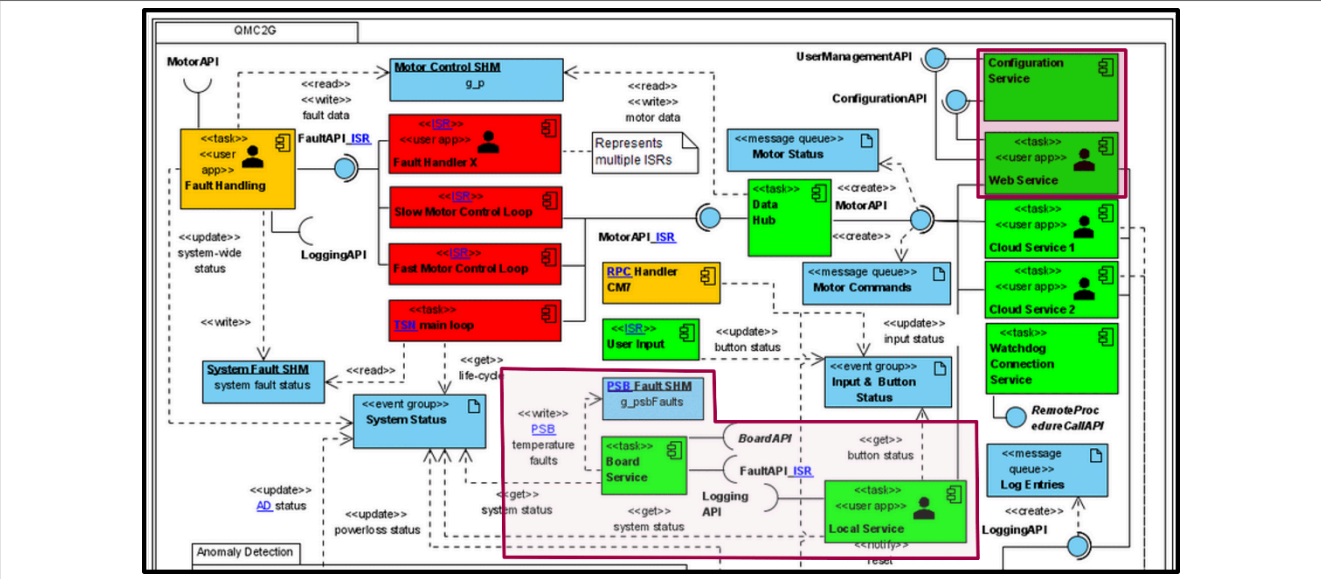


Figure 11. Other components

Table 18. Other components

Name	Type	Description
Configuration service	Task	Implements the Configuration API . It serves the decrypted key-value pairs from the encrypted configuration in the 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 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 themselves to the system. The authentication result is sent to the logging service using the logging API . Upon a failed authentication, the connection is terminated. Upon a successful authentication, the web interface is populated with the information that is appropriate for the user role and the connection is maintained to wait for user requests, such as 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)	The local service can be resumed by changes in the input & button status or system status event groups. It reacts on the 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 (such as due to a fault), a log entry is prepared 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 is read by the MC_SetFastMotorStatus_fromISR function to include the information

Table 19. Other artifacts

Name	Type	Description
		into the motor status when the FEATURE_MC_PSB_TEMPERATURE_FAULTS configuration is enabled.

5 Software APIs

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

- [Section 5.1](#)
- [Section 5.2](#)
- [Section 5.3](#)
- [Section 5.4](#)
- [Section 5.5](#)
- [Section 5.6](#)
- [Section 5.7](#)
- [Section 5.8](#)
- [Section 5.7](#)

5.1 Motor API and Motor API ISR

The **Motor API** and **Motor API ISR** provide access to the motor-control functionality through the **data hub** task. An overview of the available functions is shown in [Figure 12](#). [Table 20](#) provides a brief description of each function.

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 are not accessible to applications by default.

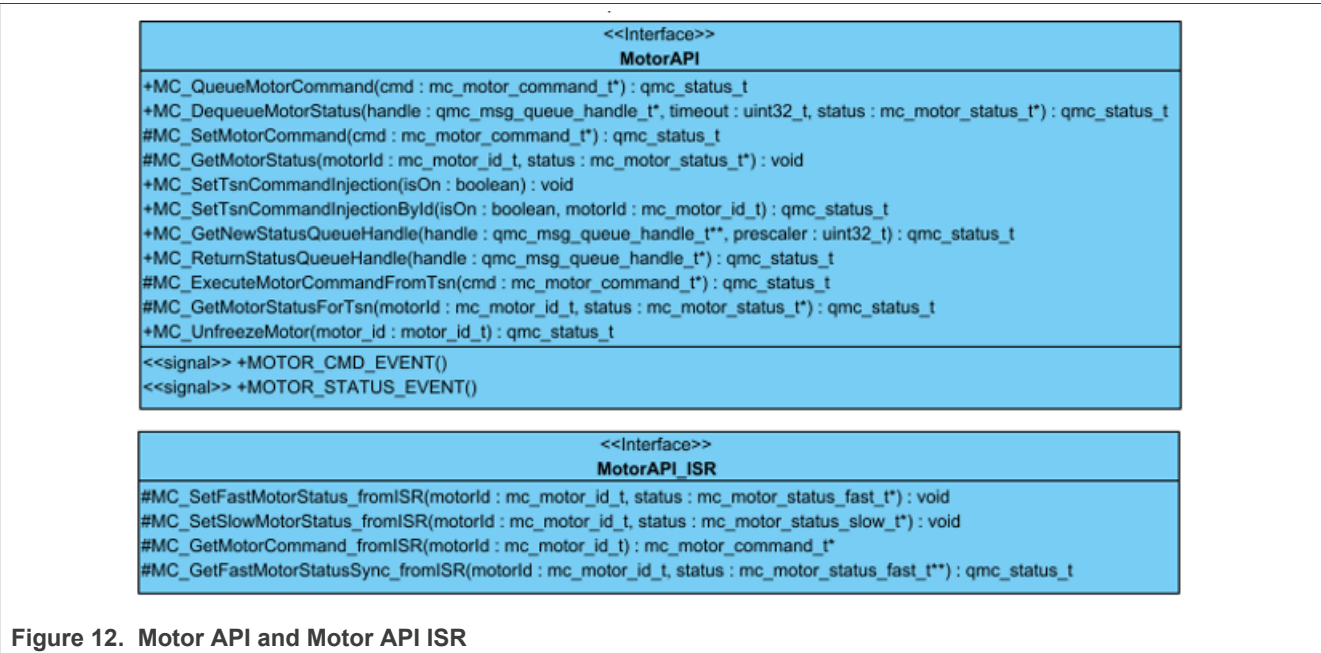


Figure 12. Motor API and Motor API ISR

Table 20. Motor API functions

Name	Visibility	Description
MC_QueueMotorCommand	Public	Puts one mc_motor_command_t (for a single motor) into the message queue for execution and notifies the data hub task.
MC_DequeueMotorStatus	Public	Gets one mc_motor_status_t element from the queue, if available.
MC_SetMotorCommand	Protected	Internal function used by the data hub task to write the mc_motor_command_t to the shared memory.
MC_GetMotorStatus	Protected	Internal function used by the data hub task to read the 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	Requests a new message queue handle to receive motor status messages.
MC_ReturnStatusQueueHandle	Public	Hands back a message queue handle obtained by MC_GetNewStatusQueueHandle .
MC_ExecuteMotorCommandFromTsn	Protected	Function used by the TSN main loop task to set the mc_motor_command_t for execution.
MC_GetMotorStatusForTsn	Protected	Function to be used by the TSN main loop to retrieve the mc_motor_status_t .
MC_UnfreezeMotor	Public	Unfreezes a motor and tells the motor control API that it can stop ignoring further commands to the motor.
MC_SetTsnCommandInjectionById	Public	Enables or disables the execution of motor commands sent via the TSN connection for a specific motor.

5.2 Fault API and Fault API ISR

The **Fault API** and **Fault API ISR** provide access to the fault handling system to tasks and ISRs, respectively. An overview of the available functions is shown in [Figure 13](#). [Table 21](#) provides a brief description of each function.

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 are not accessible to user applications by default.

<<Interface>> FaultAPI
+FAULT_RaiseFaultEvent(src : fault_source_t) : void +FAULT_InitFaultQueue() : qmc_status_t +FAULT_ReadBuffer() : fault_source_t +FAULT_WriteBuffer(fault : fault_source_t) : qmc_status_t +FAULT_BufferEmpty() : boolean

<<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 13. Fault API and Fault API ISR

Table 21. Fault API functions

Name	Visibility	Description
FAULT_RaiseFaultEvent	Public	Sends a FAULT signal to the fault-handling service, indicating the given fault_source_t. For valid fault sources, see the referenced enumeration type.
FAULT_BufferEmpty	Public	Checks if the circular fault buffer is empty.
FAULT_InitFaultQueue	Public	Initializes the fault queue.
FAULT_ReadBuffer	Public	Reads from the circular fault buffer.
FAULT_WriteBuffer	Public	Writes a fault into the circular fault buffer. This should only be used by FAULT_RaiseFaultEvent_fromISR(src : fault_source_t) : void.

Table 22. Fault API ISR functions

Name	Visibility	Description
FAULT_RaiseFaultEvent_fromISR	Public	Sends a FAULT signal to the fault-handling service, indicating the given fault_source_t. For valid fault sources, see the referenced enumeration type. This function should be used within ISRs only. To call it from a task, use FAULT_RaiseFaultEvent(src : fault_source_t) : void from FaultAPI.
FAULT_SetImmediateStopConfiguration_fromISR	Public	Sets the immediate motor stop behavior for the motor defined by stopMotorId that should be applied if the motor, defined by faultMotorId, experiences a fault. If stopMotorId equals faultMotorId, the input is ignored and the configuration remains unchanged.
FAULT_GetImmediateStopConfiguration_fromISR	Public	Gets the immediate motor stop behavior for the motor, defined by stopMotorId, that should be applied if the motor, defined by faultMotorId, experiences a fault. If stopMotorId equals faultMotorId, true is returned, regardless of the configuration.

Table 22. Fault API ISR functions...continued

Name	Visibility	Description
FAULT_GetMotorFault_fromISR	Public	Gets the current motor fault status for the given motor ID. If the motor ID is invalid, kMC_NoFaultMC shall be returned.
FAULT_GetSystemFault_fromISR	Public	Gets the current system fault status.

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 14](#). [Table 23](#) provides a brief description of each function.

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 are not accessible to user applications by default.

<<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 +LOG_InitDatalogger() : qmc_status_t +LOG_FormatDatalogger() : qmc_status_t
<<signal>> +LOG_EVENT() <<signal>> +POWER_LOSS() <<signal>> +WD_RESET()

<<Interface>> LoggingAPI_boot
+LOG_WriteLogEntry(entry : log_record_t*) : qmc_status_t

Figure 14. Logging API functions

Table 23. Logging API functions

Name	Visibility	Description
LOG_FormatDatalogger	Public	Formats (erases) the space of Flash Recorder.
LOG_GetNewLoggingQueueHandle	Public	Requests a new message queue handle to receive logging messages. Operation may return kStatus_QMC_ErrMem , if the queue cannot be created or all statically created queues are in use.
LOG_ReturnLoggingQueueHandle	Public	Hands back a message queue handle, obtained by LOG_GetNewLoggingQueueHandle (handle : qmc_msg_queue_handle_t**): qmc_status_t.
LOG_DequeueEncryptedLogEntry	Public	Gets one log_encrypted_record_t element from the previously registered queue, if available.
LOG_GetLogRecord	Public	Gets the log_record_t with the given ID.
LOG_GetLogRecordEncrypted	Public	Gets the log record with the given ID and encrypts it for the external log reader.
LOG_GetLastLogId	Public	Returns the ID of the latest log entry.

Table 23. Logging API functions...continued

Name	Visibility	Description
LOG_InitDatalogger	Public	Initializes the Flash Recorder. Must be executed before the first use.
LOG_QueueLogEntry	Public	Puts one log_record_t into the message queue and notifies the logging-service task. If the hasPriority flag is set, the log record is put at the start of the queue instead of its end.
LOG_WriteLogEntry	Public	Writes a log entry to the flash and, if an SD card is available, also to the SD card.

5.4 User management API

The **user management API** provides functionalities to change the user access configurations of the i.MX RT Industrial Drive Development Platform software. An overview of the available functions is shown in [Figure 15](#). [Table 24](#) provides a brief description of each function.

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 a "#" symbol are internal (protected) i.MX RT Industrial Drive Development Platform functions that are not accessible to user applications by default.

```

<<Interface>>
UserManagementAPI

+USRMGMT_AddUser(current_session : usrmgmt_session_id, name : unsigned char*, passphrase : unsigned char*, role : usrmgmt_role_t) : qmc_status_t
+USRMGMT_UpdateUser(current_session : usrmgmt_session_id, uid : config_id_t, passphrase : unsigned char*, role : usrmgmt_role_t) : qmc_status_t
+USRMGMT_RemoveUser(current_session : usrmgmt_session_id, name : unsigned char*) : qmc_status_t
+USRMGMT_LockUser(name : unsigned char*, reactivation_timestamp : uint64_t) : qmc_status_t
+USRMGMT_UnlockUser(name : unsigned char*) : qmc_status_t
+USRMGMT_CreateSession(token_buffer : unsigned char*, token_size : size_t*, sid : usrmgmt_session_id*, session : usrmgmt_session_t**, name : unsigned char*, passphrase : unsigned char*) : qmc_status_t
+USRMGMT_EndSession(sid : usrmgmt_session_id) : qmc_status_t
+USRMGMT_ValidateSession(token : unsigned char*, token_size : size_t*, sid : usrmgmt_session_id*, session : usrmgmt_session_t**) : qmc_status_t
+USRMGMT_IterateUsers(pcount : int*, pid : config_id_t*, pconfig : usrmgmt_user_config_t) : qmc_status_t
+USRMGMT_IterateSessions(pcount : int*, psid : usrmgmt_session_id*, session : usrmgmt_session_t**) : qmc_status_t

```

Figure 15. User management API functions

Table 24. User management API functions

Name	Visibility	Description
USRMGMT_AddUser	Public	Adds a new user to the System. May return kStatus_QMC_ErrMem , if no memory for a new user is available, kStatus_QMC_ErrArgInvalid , if the name already exists, and kStatus_QMC_ErrRange , if the username contains invalid characters. If adding the user is successful, the configuration in the flash must be updated.
USRMGMT_CreateSession	Public	Creates a new session. At least one token must be provided to create a session. If not, only the passphrase is checked, and no session is created.
USRMGMT_EndSession	Public	Terminates an active session.
USRMGMT_IterateSessions	Public	Iterates over active sessions.
USRMGMT_IterateUsers	Public	Iterates over users.
USRMGMT_LockUser	Public	Locks a user for a given time. The user credentials cannot be used for login during this time.

Table 24. User management API functions...continued

Name	Visibility	Description
USRMGMT_RemoveUser	Public	Removes a user from the system. May return kStatus_QMC_ErrArgInvalid , if the user indicated by name does not exist. If removing the user is successful, the configuration in the flash must be updated to prevent further access to the system. Also, the corresponding user role entry of the configuration must be set to kUSRMGMT_RoleEmpty to indicate that this slot is available for a new user registration.
USRMGMT_UnlockUser	Public	Unlocks the user credentials that are in the locked state.
USRMGMT_UpdateUser	Public	Updates the user.
USRMGMT_ValidateSession	Public	Validates a session using a token. The pointer to the decoded payload is only valid to the next call of this function.

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 16](#). [Table 25](#) provides a brief description of each function.

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 a "#" symbol, are internal (protected) i.MX RT Industrial Drive Development Platform functions that are not accessible to user applications by default.

<<Interface>> ConfigurationAPI
+CONFIG_GetStrValue(key : unsigned char*, value : unsigned char*) : qmc_status_t +CONFIG_GetStrValueById(id : config_id_t, value : unsigned char*) : qmc_status_t +CONFIG_SetStrValue(key : unsigned char*, value : unsigned char*) : qmc_status_t +CONFIG_SetStrValueById(id : config_id_t, value : unsigned char*) : qmc_status_t +CONFIG_GetBinValue(key : unsigned char*, value : unsigned char*, length : uint16_t) : qmc_status_t +CONFIG_GetBinValueById(id : config_id_t, value : unsigned char*, length : uint16_t) : qmc_status_t +CONFIG_SetBinValue(key : unsigned char*, value : unsigned char*, length : uint16_t) : qmc_status_t +CONFIG_SetBinValueById(id : config_id_t, value : unsigned char*, length : uint16_t) : 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 16. Configuration API functions

Table 25. Configuration API functions

Name	Visibility	Description
CONFIG_GetBinValue	Public	Gets the value indicated by the key. If the key does not exist, it returns kStatus_QMC_ErrRange . If the configuration section does not store the changed value, the default value must be used instead.
CONFIG_GetBinValueById	Public	Gets the value indicated by the id. If the id does not exist, it returns kStatus_QMC_ErrRange . If the configuration section does not store the changed value, the default value must be used instead.

Table 25. Configuration API functions...continued

Name	Visibility	Description
CONFIG_GetBooleanFromValue	Public	Tries to parse a boolean from the configuration value retrieved by CONFIG_GetStrValue(key : unsigned char*, value : unsigned char*): qmc_status_t or CONFIG_GetStrValueById(id : config_id_t, value : unsigned char*): qmc_status_t. Values, such as "true"/"false", "yes"/"no", "on"/"off", and "1"/"0", shall be recognized by the function.
CONFIG_GetIdfromKey	Public	Returns the config_id_t that matches the given key. If the key does not exist, it returns kCONFIG_KeyNone .
CONFIG_GetIntegerFromValue	Public	Tries to parse an integer from the configuration value retrieved by CONFIG_GetStrValue(key : unsigned char*, value : unsigned char*): qmc_status_t or CONFIG_GetStrValueById(id : config_id_t, value : unsigned char*): qmc_status_t.
CONFIG_GetStrValue	Public	Gets the value indicated by a key. If the key does not exist, it returns kStatus_QMC_ErrRange . If the configuration section does not store the changed value, the default value must be used instead.
CONFIG_GetStrValueById	Public	Gets the value indicated by the id. If the id does not exist, it returns kStatus_QMC_ErrRange . If the configuration section does not store the changed value, the default value must be used instead.
CONFIG_SetBinValue	Public	Sets the new value for the given key. This shall not trigger CONFIG_UpdateFlash(): qmc_status_t automatically. If key doesn't exist, return kStatus_QMC_ErrRange . If value is NULL, return kStatus_QMC_ErrArgInvalid .
CONFIG_SetBinValueById	Public	Sets the new value for the given id. This shall not trigger CONFIG_UpdateFlash(): qmc_status_t automatically. If id doesn't exist, return kStatus_QMC_ErrRange . If id equals kCONFIG_KeyNone or value is NULL, return kStatus_QMC_ErrArgInvalid .
CONFIG_SetBooleanAsValue	Public	Converts a boolean to a string to store it as a configuration value. This value is then stored in the configuration by calling CONFIG_SetStrValue(key : unsigned char*, value : unsigned char*): qmc_status_t or CONFIG_SetStrValueById(id : config_id_t, value : unsigned char*): qmc_status_t.
CONFIG_SetIntegerAsValue	Public	Converts an integer to a string to store it as a configuration value. This value is then stored in the configuration by calling CONFIG_SetStrValue(key : unsigned char*, value : unsigned char*): qmc_status_t or CONFIG_SetStrValueById(id : config_id_t, value : unsigned char*): qmc_status_t.
CONFIG_SetStrValue	Public	Sets the new value for the given key. This shall not trigger CONFIG_UpdateFlash(): qmc_status_t

Table 25. Configuration API functions...continued

Name	Visibility	Description
		t automatically. If key doesn't exist, return kStatus_QMC_ErrRange. If value is NULL, return kStatus_QMC_ErrArgInvalid.
CONFIG_SetStrValueById	Public	Sets the new value for the given id. This shall not trigger CONFIG_UpdateFlash(): qmc_status_t automatically. If id doesn't exist, return kStatus_QMC_ErrRange. If id equals kCONFIG_KeyNone or value is NULL, return kStatus_QMC_ErrArgInvalid.
CONFIG_UpdateFlash	Public	Encrypts the current configuration and writes it to the flash.
CONFIG_WriteFwUpdateChunk	Public	Writes a chunk of data to the firmware update location. Implementation hint: the bounds and alignment to the flash pages must be checked.

5.6 Board API

The **board API** provides access to board-specific features, implemented on the Cortex-M7 core, such as temperature sensors and display. It also provides functions for initialization. An overview of the available functions is shown in [Figure 17](#). [Table 26](#) provides a brief description of each function.

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 a "#" symbol are internal (protected) i.MX RT Industrial Drive Development Platform functions that are not accessible to user applications by default.

<<Interface>> BoardAPI
+BOARD_Init(): qmc_status_t +BOARD_GetDbTemperature(temperature : float*) : qmc_status_t +BOARD_SetDbTemperatureAlarm(threshold : float, hysteresis : float) : qmc_status_t +BOARD_GetPsbTemperature(temperature : double*, psb : qmc_psb_temperature_id_t) : 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_ConvertTimestamp2Datetime(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_SetTimeChangedCallback(callback : void (*)(void)) : void +BOARD_GetFwVersion() : qmc_fw_version_t +BOARD_GetDeviceIdentifier() : char* +BOARD_SetLifecycle(lc : qmc_lifecycle_t) : qmc_status_t +BOARD_GetLifecycle() : qmc_lifecycle_t

Figure 17. Board API functions

Table 26. Board API functions

Name	Visibility	Description
BOARD_ConvertDatetime2Timestamp	Public	Convert the qmc_datetime_t structure to qmc_timestamp_t .
BOARD_ConvertTimestamp2Datetime	Public	Converts the qmc_timestamp_t structure to qmc_datetime_t .
BOARD_GetDbTemperature	Public	Gets the current temperature in degree Celsius from the temperature sensor on the digital board.
BOARD_GetDeviceIdentifier	Public	Gets a unique identifier for the system.

Table 26. Board API functions...continued

Name	Visibility	Description
BOARD_GetFwVersion	Public	Gets the version of the currently running firmware. It is retrieved from the firmware manifest section in the flash.
BOARD_GetLifecycle	Public	Gets the life cycle of the QMC system.
BOARD_GetPsbTemperature	Public	Gets the temperature in degree Celsius from the selected temperature sensor on one of the power-stage boards. This function may return cached values read by the board service task instead of reading the sensor directly.
BOARD_GetTime	Public	Gets a timestamp derived from the internal systick counter. The function shall add the offset to the wall-clock time, retrieved by <code>RPC_GetTimeFromRTC(timestamp : qmc_timestamp_t*) : qmc_status_t</code> during system startup.
BOARD_Init	Public	Initializes the BoardAPI. The scheduler must be already running.
BOARD_SetDbTemperatureAlarm	Public	Sets the 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 the threshold. If the temperature drops below the value indicated by hysteresis, 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 the slow output pins on the SNVS domain, the call is forwarded to the internal function <code>RPC_SetSnvsOutput(gpioState : uint16_t) : qmc_status_t</code> .
BOARD_SetLifecycle	Public	Sets the life cycle of the QMC system. The function shall set the system event bits, so that only the specified life cycle is set and all others are cleared.
BOARD_SetTimeChangedCallback	Public	Sets a callback function that shall be called whenever <code>RPC_SetTimeToRTC(timestamp : qmc_timestamp_t*) : qmc_status_t</code> successfully changes the system time.
BOARD_SetUserLed	Public	Sets the LED state indicated by the <code>qmc_led_cmd_t</code> of the given user LED.

5.7 Remote procedure call API

The remote procedure call API provides functionalities to execute operations on the Cortex-M4 core of the platform, such as SPI slave selection, RTC, and secure watchdog. The remote procedure call API overview of the available functions is shown in [Figure 18](#).

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 a "#" symbol, are internal (protected) i.MX RT Industrial Drive Development Platform functions that are not accessible to user applications by default.

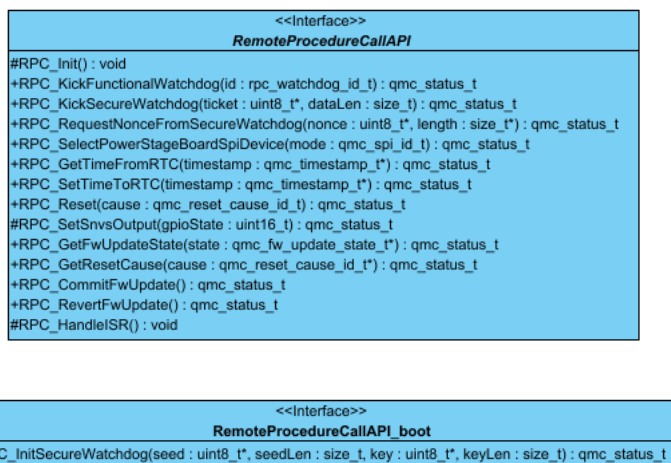


Figure 18. Remote procedure call API functions

Table 27. Remote procedure call API functions

Name	Visibility	Description
RPC_CommitFwUpdate	Public	Communicates a "commit" request to the bootloader. During the next boot, the new firmware is committed and a recovery image is created. May return kStatus_QMC_Timeout , if the CM4 core cannot handle the request in time.
RPC_GetFwUpdateState	Public	Retrieve the reset-proof firmware update status from the battery backed-up SNVS_LP_GPR register. May return kStatus_QMC_Timeout , if the CM4 core cannot handle the request in time.
RPC_GetResetCause	Public	Gets the reason why a reset was triggered by the watchdog implementation of the Cortex-M4 core. May return kStatus_QMC_Timeout , if the CM4 core cannot handle the request in time.
RPC_GetTimeFromRTC	Public	Requests a timestamp from the RTC, implemented in the CM4 core. May return kStatus_QMC_Timeout , if the CM4 core cannot handle the request in time.
RPC_HandleISR	Protected	This function must be called from the GPR_IRQ SW interrupt.
RPC_Init	Protected	Registers the IRQ handlers and initializes the event groups, mutexes, and data structures used by the RemoteProcedureCallAPI . This function is meant to be called by the main function/startup task.
RPC_KickFunctionalWatchdog	Public	Kicks the functional watchdog referenced by the id. May return kStatus_QMC_Timeout , if the CM4 core cannot handle the request in time.
RPC_KickSecureWatchdog	Public	Kicks the secure watchdog by giving it a signed update message (ticket), received from the watchdog connection service. May return kStatus_QMC_

Table 27. Remote procedure call API functions...continued

Name	Visibility	Description
		Timeout , if the CM4 core cannot handle the request in time.
RPC_RequestNonceFromSecureWatchdog	Public	Requests a nonce from the secure watchdog. May return kStatus_QMC_Timeout , if the CM4 core cannot handle the request in time. This function is meant to be called by the watchdog connection service.
RPC_Reset	Public	Triggers a reset. This function shall create a log entry and send it to the logging service. Implementation hint: delay the calling task to allow for some time to process the log entry.
RPC_RevertFwUpdate	Public	Communicates a "revert" request to the bootloader. During the next boot, the old firmware is restored. May return kStatus_QMC_Timeout , if the CM4 core cannot handle the request in time.
RPC_SelectPowerStageBoardSpiDevice	Public	Sets the output state of the SPI selection pins in the SNVS domain, such that the SPI device, referenced by the mode, is selected. This function makes a call to the internal function <code>RPC_SetSnvsOutput(gpioState : uint16_t) : qmc_status_t</code> .
RPC_SetSnvsOutput	Protected	Sets the four output pins/two SPI select pins on the SNVS domain to the desired state. For the construction of the input parameter gpioState , see <code>gpioState : uint16_t</code> . May return kStatus_QMC_Timeout , if the CM4 core cannot handle the request in time.
RPC_SetTimeToRTC	Public	Tries to set the RTC to the given timestamp. May return kStatus_QMC_Timeout , if the CM4 core cannot handle the request in time.
RPC_InitSecureWatchdog	Public	Provides the RNG seed and public key to the secure watchdog implementation on the CM4. Other secure watchdog methods are not allowed until this function is completed. The secure watchdog implementation validates the inputs and triggers a reboot into the recovery mode, if they are invalid.

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 the 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 19](#). [Table 28](#) provides a brief description of each function.

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 a "#" symbol, are internal (protected) i.MX RT Industrial Drive Development Platform functions that are not accessible to user applications by default.

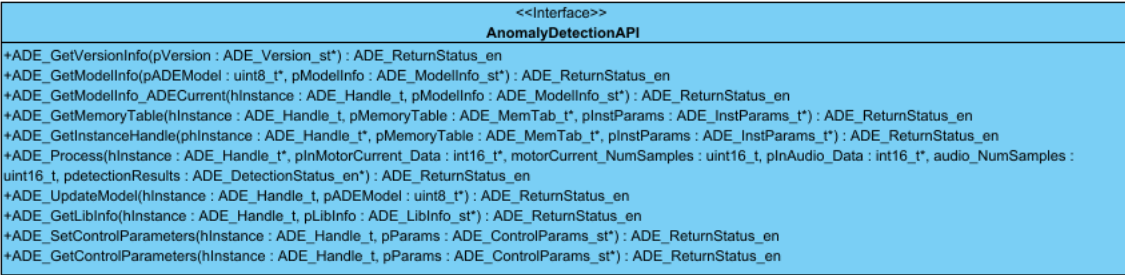


Figure 19. Anomaly detection API functions

Table 28. Anomaly detection API functions

Name	Visibility	Description				
ADE_GetControlParameters		Gets the ADE module parameters.				
ADE_GetInstanceHandle		This function is used to create a bundle instance. It returns the created instance handle through ph Instance . All parameters are set to their default values.				
ADE_GetLibInfo		This function is used to get the library and model information.				
ADE_GetMemoryTable		<div>This function is used for memory allocation and it is free. It can be called in two ways:<table><tr><td>hInstance = NULL</td><td>Returns the memory requirements.</td></tr><tr><td>hInstance = Instance handle</td><td>Returns the memory requirements and allocated base addresses for the instance.</td></tr></table><p>When this function is called for a memory allocation (h Instance=NULL), the memory base address pointers are NULL on return.</p><p>When the function is called for free (hInstance = Instance Handle), the memory table returns the allocated memory and base addresses used during the initialization.</p></div>	hInstance = NULL	Returns the memory requirements.	hInstance = Instance handle	Returns the memory requirements and allocated base addresses for the instance.
hInstance = NULL	Returns the memory requirements.					
hInstance = Instance handle	Returns the memory requirements and allocated base addresses for the instance.					
ADE_GetModelInfo		This function is used to retrieve information about an ADE model.				
ADE_GetModelInfo_ADECurrent		This function is used to retrieve information about the model currently used by the ADE engine.				
ADE_GetVersionInfo		This function is used to retrieve information about the library version.				
ADE_Process		Process function for the ADE module. Note: The input and output buffers must be 32-bit aligned. The number of buffer samples must match the model requirements.				
ADE_SetControlParameters		Sets the ADE module parameters.				
ADE_UpdateModel		This function is used to update the multimodal machine-learning model.				

Table 28. Anomaly detection API functions...continued

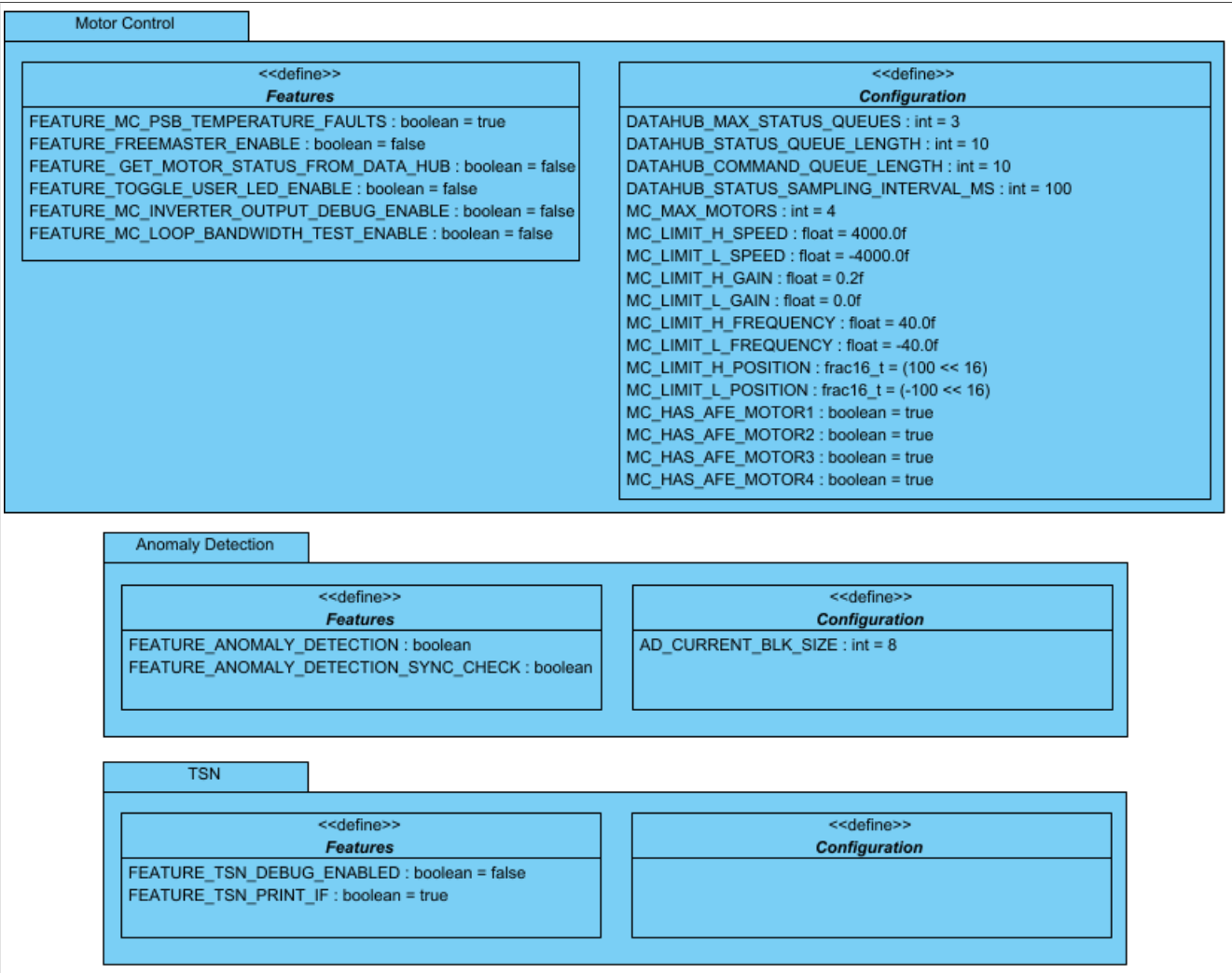
Name	Visibility	Description
		Note: The new model must be compatible with the initial model.

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. For example, it is possible to:

- Configure the motor-control features and parameters, such as the maximum number of motors (up to four) and speed and frequency limits.
- Activate/deactivate the anomaly detection feature.
- Activate/deactivate the restricted version of the software, which has been granted the **ISA/IEC 62443-4-2 SL3** certification.

The complete and updated list of configurable features and parameters is provided in the **software model** in the **package diagram** → **features and compile time configuration options** section.



Logging & Configuration	<div><div><<define>> Features</div><div>FEATURE_CONFIG_DBG_PRINT : boolean = false FEATURE_LCRYPT_DBG_PRINT : boolean = false FEATURE_DATALOGGER_RECORDER_DBG_PRINT : boolean = false FEATURE_DATALOGGER_DISPATCHER_DBG_PRINT : boolean = false FEATURE_DATALOGGER_SDCARD_DBG_PRINT : boolean = false FEATURE_DATALOGGER_SDCARD : boolean = true FEATURE_DATALOGGER_DQUEUE : boolean = true FEATURE_DATALOGGER_DQUEUE_EVENT_BITS : boolean = true</div></div> <div><div><<define>> Configuration</div><div>CONFIG_MUTEX_XDELAYS_MS : uint32_t = 800 CONFIG_MAX_KEY_LEN : uint32_t = 32 CONFIG_MAX_VALUE_LEN : uint32_t = 128 CONFIG_MAX_RECORDS : uint32_t = 128 FLASH_CONFIG_SECTORS : uint32_t = 6 DATALOGGER_MUTEX_XDELAYS_MS : uint32_t = 800 DATALOGGER_RCV_QUEUE_DEPTH : uint32_t = 3 DATALOGGER_DYNAMIC_RCV_QUEUE_DEPTH : uint32_t = 3 DATALOGGER_RCV_QUEUE_CN : uint32_t = 2 DATALOGGER_SDCARD_DIRPATH : char* = "/dat" DATALOGGER_SDCARD_FILEPATH : char* = "/dat/datfile.txt" DATALOGGER_SDCARD_MAX_FILESIZE : uint32_t = 10000000 FLASH_RECORDER_ORIGIN : void* = 0x30002000 FLASH_RECORDER_SECTORS : uint32_t = 32 DATALOGGER_LOW_MEMORY_TRESHOLD : uint8_t = 20</div></div>
BoardAPI	<div><div><<define>> Features</div><div>FEATURE_ENABLE_GPIO_SW_DEBOUNCING : boolean = true FEATURE_BOARD_SANITY_CHECK_DAY_OF_WEEK : boolean = true</div></div> <div><div><<define>> Configuration</div><div>BOARD_GETTIME_REFRESH_INTERVAL_S : int = (24*60*60) /* 1 day */ PSB_TEMP1_THRESHOLD : double = 85.0 PSB_TEMP2_THRESHOLD : double = 75.0 DB_TEMP_THRESHOLD : float = 90. MCU_TEMP_THRESHOLD : float = 100.0 GPIO_SW_DEBOUNCE_MS : int = 5</div></div>
RPC & Watchdog	<div><div><<define>> Features</div><div>FEATURE_SECURE_WATCHDOG : boolean = false</div></div> <div><div><<define>> Configuration</div><div>SECURE_WATCHDOG_HOST : char[] = "api.aedt.server" SECURE_WATCHDOG_KICK_INTERVAL_S : int = 60 SECURE_WATCHDOG_KICK_RETRY_S : int = 30 SECURE_WATCHDOG_SOCKET_TIMEOUT_MS : int = 1000</div></div>
Secure Element Support	<div><div><<define>> Features</div><div>SE051_APPLET_VERSION_06_00 : boolean = false SE051_APPLET_VERSION_07_02 : boolean = true FEATURE_SE_DEBUG_ENABLED : boolean = false</div></div>

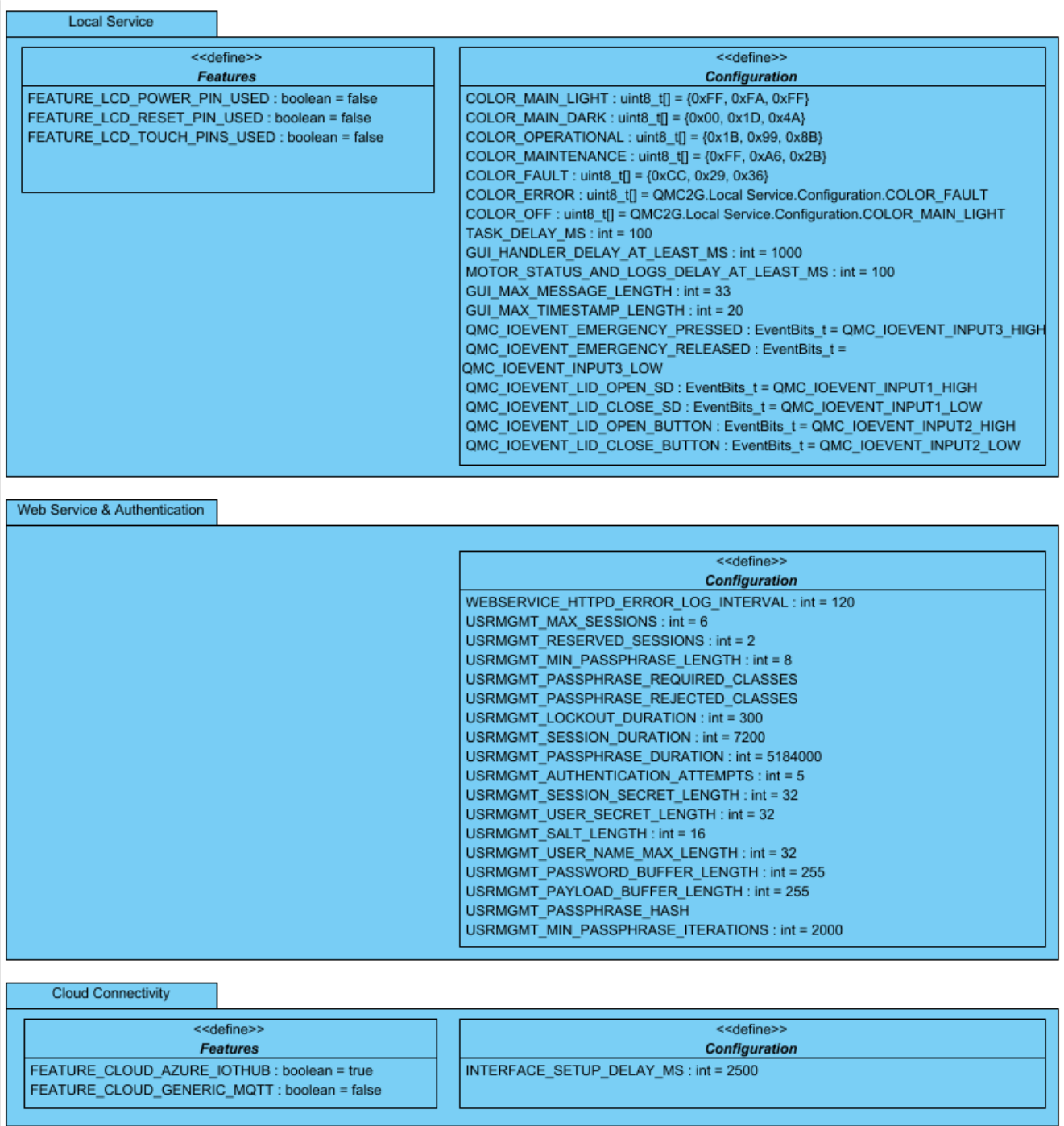


Figure 20. Compile-time options and configurations

7 Appendix: external libraries and dependencies

Table 29 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 is in the **software model** in the **package diagram** → **libraries** section.

Table 29. 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 file system 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 the functionality to establish MQTT connections and push MQTT topic updates to an MQTT broker. It features integration with lwIP and mbedTLS.
mbedTLS	The mbedTLS is an open-source library implementing TLS and SSL protocols with accompanying cryptographic algorithms. It is targeted for 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 Subsystem (SSS) API. It provides the mbedTLS with an alternative implementation (mbedTLS_ALT) that uses the cryptography features of the hardware. It integrates with FreeRTOS and the coreMQTT library to enable cloud onboarding, aided by EdgeLock SE05x.
lwIP	Lightweight 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 enables you to fine-tune the motor-control parameters for each motor.
RTCESL	NXP Real Time Control Embedded Software Libraries provide highly optimized functions for mathematical computations, digital filters, motor control, and power conversions. They are used by the motor-control algorithm of the i.MX RT Industrial Drive Development Platform.

8 Revision history

Table 30. Revision history

Document ID	Release date	Description
AN13643 v.1.2	17 February 2025	<ul style="list-style-type: none"> Updated with latest version of software releases
AN13643 v.1.1	02 February 2022	<ul style="list-style-type: none"> Updated with latest version of software releases
AN13643 v.1.0	01 June 2022	<ul style="list-style-type: none"> Initial version

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at <https://www.nxp.com/profile/terms>, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Amazon Web Services, AWS, the Powered by AWS logo, and FreeRTOS — are trademarks of Amazon.com, Inc. or its affiliates.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μ Vision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

EdgeLock — is a trademark of NXP B.V.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

Contents

1	Acronyms	2
2	i.MX RT Industrial Drive Development Platform	3
2.1	Introducing the i.MX RT Industrial Drive Development Platform software package	4
2.2	Note on ISA/IEC 62443-4-2 certified software	5
2.3	How to use this document	5
3	Architecture overview of i.MX RT Industrial Drive Development Platform software	6
3.1	Functional overview of i.MX RT Industrial Drive Development Platform software components	6
3.2	Overview of software components interaction through APIs	8
3.3	Overview of main software-security features	12
4	Software components and artifacts	12
4.1	Motor control and TSN	13
4.2	Fault handling	15
4.3	Cloud connectivity	17
4.4	Logging	17
4.5	RPC and watchdog	19
4.6	System	21
4.7	Anomaly detection	22
4.8	Others	23
5	Software APIs	25
5.1	Motor API and Motor API ISR	25
5.2	Fault API and Fault API ISR	26
5.3	Logging API	28
5.4	User management API	29
5.5	Configuration API	30
5.6	Board API	32
5.7	Remote procedure call API	33
5.8	Anomaly detection API	35
6	Appendix: compile time options and configurations	37
7	Appendix: external libraries and dependencies	39
8	Revision history	40
	Legal information	41

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.