Renesas RA Family

RA AWS MQTT/TLS Cloud Connectivity Solution - Ethernet

Introduction

This application note describes IoT Cloud connectivity solution in general, provides a brief introduction to IoT Cloud providers like Amazon Web Services (AWS), and covers the FSP MQTT/TLS module and its features. The application example provided in the package uses AWS IoT Core. The detailed steps in this document show first-time AWS IoT Core users how to configure the AWS IoT Core platform to run this application example.

This application note enables developers to effectively use the FSP MQTT/TLS modules in end-product design. Upon completion of this guide, developers will be able to add the “AWS Core MQTT”, “Mbed TLS”, and “secure sockets on FreeRTOS plus TCP ” using the Ethernet interface, configure them correctly for the target application, and write code using the included application example code as a reference for an efficient starting point.

References to detailed API descriptions, and other application projects that demonstrate more advanced uses of the module, are in the *FSP User’s Manual* (available at: https://renesas.github.io/fsp/), which serves as a valuable resource in creating more complex designs.

This MQTT/TLS AWS Cloud Connectivity solution is supported on the CK-RA6M5.

Applies to:

- RA6M5 MCU Group

Required Resources

To build and run the MQTT/TLS application example, the following resources are needed.

Development tools and software

- e² studio ISDE v22.10.0 or later (renesas.com/us/en/software-tool/e-studio)
- Flexible Software Package (FSP) v4.2.0 (renesas.com/us/en/software-tool/flexible-software-package-fsp)

Hardware

- Renesas CK-RA6M5 kit (renesas.com/ra/ck-ra6m5)
- PC running Windows® 10 and an installed web browser (Google Chrome, Internet Explorer, Microsoft Edge, Mozilla Firefox, or Safari)
- Micro USB cables (included as part of the kit. See *CK-RA6M5 User’s Manual)*
- Ethernet cable
- A router with at least 1 available 100 Mbps/full duplex Ethernet port

Prerequisites and Intended Audience

This application note assumes that the user is adept at operating the Renesas e² studio IDE with Flexible Software Package (FSP). If not, we recommend reading and following the procedures in the *FSP User's Manual* sections for ‘Starting Development’ including ‘Debug the Blinky Project’. Doing so enables familiarization with e² studio and FSP and validates proper debug connection to the target board. In addition, this application note assumes prior knowledge of MQTT/TLS and its communication protocols.

The intended audience is users who want to develop applications with MQTT/TLS modules using Ethernet modules on Renesas RA6 MCU Series.

Note: If you are a first-time user of e² studio and FSP, we highly recommend you install e² studio and FSP on your system in order to run the Blinky Project and to get familiar with the e² studio and FSP development environment before proceeding to the next sections.

Note: This Application Project and Application Note can only use versions FSP v4.2.0.

Note: If you want to quickly build and run the attached application, please jump to section (2 Running the MQTT/TLS Ethernet Application Example).

Prerequisites

1. Access to online documentation available in the Cloud Connectivity References section
2. Access to latest documentation for identified Renesas Flexible Software Package
3. Prior knowledge of operating e^2 studio and built-in (or standalone) RA Configurator
4. Access to associated hardware documentation such as User Manuals, Schematics, and other relevant kit information (renesas.com/ra/ck-ra6m5)).

Contents

1. Introduction to Components for Cloud Connectivity

1.1 General Overview

The Internet-of-Things (IoT) is a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies. The 'things' in this definition are objects in the physical world (physical objects) or information world (virtual) that can be identified and integrated into communication networks. In the context of the IoT, a 'device' is a piece of equipment with the mandatory capabilities of communication and the optional capabilities of sensing, actuation, data capture, data storage and data processing. Communication is often performed with providers of network-hosted services, infrastructure, and business applications to process/analyze the generated data and manage the devices. Such providers are called Cloud Service Providers. While there are many manufacturers for devices and cloud service providers, for the context of this application note, the device is a Renesas RA Microcontroller (MCU) connecting to services provided by Amazon Web Services (AWS) for IoT.

1.2 Cloud Service Provider

AWS IoT provides the cloud services that connect your IoT devices to other devices and AWS cloud services. As a Cloud Service Provider, AWS IoT provides the ability to:

- Connect and manage devices
- Secure device connections and data
- Process and act upon device data
- Read and set device state at any time

Figure 1 summarizes the features provided by AWS IoT.

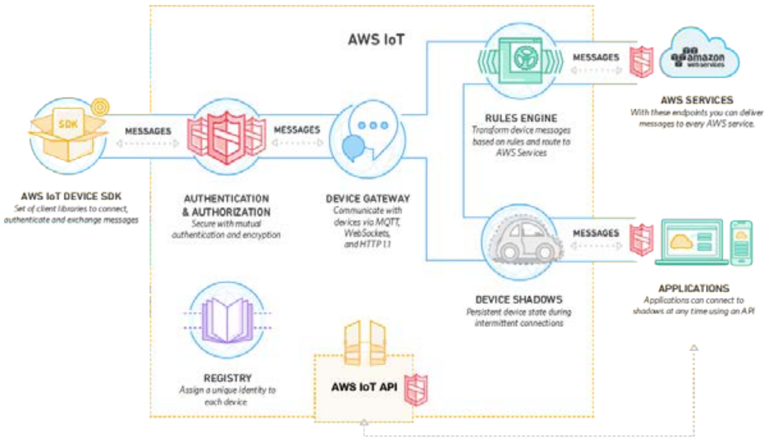

Figure 1. AWS IoT Features, Service Components, and Data Flow Diagram

A key feature provided by AWS is the AWS IoT Software Development Kit (SDK) written in C, which allows devices such as sensors, actuators, embedded micro-controllers, or smart appliances to connect, authenticate, and exchange messages with AWS IoT using the MQTT, HTTP, or WebSocket's protocols. This application note focuses on configuring and using the AWS IoT Device SDK and the included MQTT protocol available through the Renesas Flexible Software Package (FSP) for Renesas RA MCUs.

1.3 Cloud Dashboard

A cloud dashboard is a monitoring and controlling GUI for the multiple services, that you can build and access on a web browser. It has key advantages over on-premises software such as being easier to deploy, requiring little to no IT support and is accessible on multiple devices.

The **Dashboard** provides a high-level view of your entire fleet of devices and allows you to act on individual devices quickly. You can view graphical representations of relevant device information for your fleet, such as device ownership type, compliance statistics, and platform and OS breakdowns. You can access each set of devices in the presented categories by selecting any of the available data views from the **Device Dashboard**.

1.3.1 Data Monitoring

Data monitoring on the dashboard is a cloud data analytics monitoring solution that lets you track your performance metrics and easily visualize your data sets. You will be able to get a high-level view of your metrics or you can drill down and analyze the detail.

For instance, it can be sensor data coming from the device in the form of temperature, pressure, and so forth.

1.3.2 Device Management

Device Management provides high-level control to configure the devices in bulk for the entire fleet of devices or to control the individual devices.

Note: All the Dashboard-specific details for this Application Project are discussed in the later section of the document.

1.4 AWS IoT Core

AWS IoT Core is a managed cloud service that lets connected devices easily and securely interact with cloud applications and other devices. AWS IoT Core can support billions of devices and trillions of messages. It can process and route messages to AWS endpoints and to other devices reliably and securely. With AWS IoT Core, customer applications can keep track of all devices, all the time, even when the devices are not connected.

AWS IoT Core addresses security concerns for the infrastructure by implementing mutual authentication and encryption. AWS IoT Core provides automated configuration and authentication upon a device's first connection to AWS IoT Core, as well as end-to-end encryption throughout all points of connection, so that data is only exchanged between devices and AWS IoT Core with proven identity.

This application note focuses on complementing the security needs of AWS IoT Core through installing a proven identity for the RA MCU by storing a X.509 certificate and asymmetric cryptography keys in Privacy Enhanced Mail (PEM) format in the on-board flash. The RA MCU has on-chip security features, such as Key Wrapping, to protect the private key associated with the public key and the certificate associated with the device[1]. Additionally, RA MCUs can also generate asymmetric keys using features of the Secure Cryptography Engine (SCE) and API available through the FSP. The SCE accelerates symmetric encryption/decryption of data between the connected device and AWS IoT, allowing the ARM Cortex-M processor to perform other application specific computations.

1.5 MQTT Protocol Overview

This application note features Message Queuing Telemetry Transport (MQTT) as it is a lightweight communication protocol specifically designed to tolerate intermittent connections, minimize the code footprint on devices, and reduce network bandwidth requirements. MQTT uses a publish/subscribe architecture which is designed to be open and easy to implement, with up to thousands of remote clients capable of being supported by a single server. These characteristics make MQTT ideal for use in constrained environments where network bandwidth is low or where there is high latency and with remote devices that might have limited processing capabilities and memory. *The RA MCU device in this application note implements a Core MQTT which communicates with AWS IoT and exchanges example telemetry information, such as temperature, pressure, humidity, accelerometer, magnetometer, and many more types of sensor data.*

1.6 TLS Protocol Overview

The primary goal of the Transport Layer Security (TLS) protocol is to provide privacy and data integrity between two communicating applications or endpoints. AWS IoT mandates use of secure communication. Consequentially, all traffic to and from AWS IoT is sent securely using TLS. TLS protocol version 1.2 or later ensures the confidentiality of the application protocols supported by AWS IoT. A variety of TLS Cipher Suites are supported. This application note configures the RA Flexible Software Package for the MCU based device to provide the following capabilities and AWS IoT negotiates the appropriate TLS Cipher Suite configuration to maximize security.

[1] This application note does not focus on using Key Wrapping for securely storing the private key for devices deployed in a production environment.

Table 1. TLS with Crypto Capabilities in RA FSP

Secure Crypto Hardware Acceleration	Supported
Key Format Supported	AES, ECC, RSA
Hash	SHA-256
Cipher	AES
Public Key Cryptography	ECC, ECDSA, RSA
Message Authentication Code (MAC)	HKDF

On top of these supported features, Mbed Crypto middleware also supports a variety of features which can be enabled through the RA Configurator. Refer to the *FSP User's Manual* section for the Crypto Middleware (rm_psa_crypto).

1.7 Device Certificates, CA, and Keys

Device Certificates, Certificate Authorities (CA), and Asymmetric Key Pairs create the foundation for trust needed for a secure environment. The background information on these commonly used components in AWS is as follows:

A *digital certificate* is a document in a known format that provides information about the identity of a device. The X.509 standard includes the format definition for public-key certificate, attribute certificate, certificate revocation list (CRL), and attribute certificate revocation list (ACRL). X.509-defined certificate formats (X.509 Certificates) are commonly used on the internet and in AWS IoT for authenticating a remote entity/endpoint, that is, a Client and/or Server. In this application note, an X.509 certificate and asymmetric cryptography key pair (public and private keys) are generated from AWS IoT and installed (during binary compilation) into the RA MCU device running the Core MQTT to establish a *known identity*. In addition, a root Certification Authority (CA) certificate is also downloaded and used by the device to authenticate the connection to the AWS IoT gateway.

Certification authority (CA) certificates are certificates that are issued by a CA to itself or to a second CA for the purpose of creating a defined relationship between the two CA. The root CA certificate allows devices to verify that they are communicating with AWS IoT Core and not another server impersonating AWS IoT Core.

The public and private keys downloaded from AWS IoT use RSA algorithms for encryption, decryption, signing and verification[2]. These key pairs, and certificates are used together in the TLS process to:

1. Verify device identity.
2. Exchange symmetric keys, for algorithms such as AES, for encrypting and decrypting data transfers between endpoints.

2. Running the MQTT/TLS Ethernet Application Example

Refer to *RA CK-RA6M5 AWS Ethernet Getting Started Guide* as part of this project bundle for details on running the project and visualizing the sensor data on Renesas AWS dashboard.

3. AWS Core MQTT with Ethernet Interface

3.1 AWS Core MQTT

The AWS MQTT library included in RA FSP can connect to either AWS MQTT or other third party MQTT broker such as Mosquitto. The complete documentation for the library can be found on the AWS IoT Device SDK C: MQTT website. Primary features supported by the library are:

- MQTT connections over TLS to an AWS IoT Endpoint or Mosquitto server or another MQTT broker.

The AWS Core MQTT can be directly imported into a **Thread** stack. It is configured through the RA Configuration Perspective. To add the AWS Core MQTT to a new thread, open `Configuration.xml` with the RA Configuration. While ensuring that the correct thread is selected on the left, use the tab for **Stacks > New Stack > Search** and search for the keyword AWS Core MQTT.

[2] Public Key length used is 2048 bits.

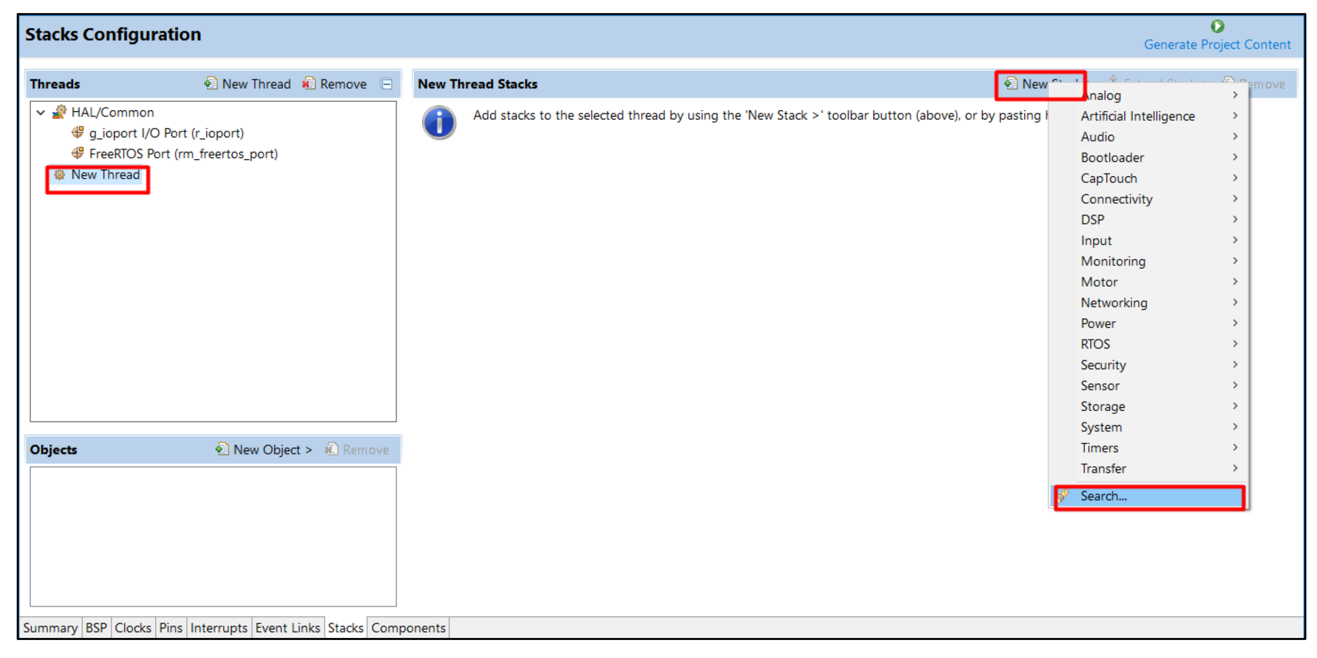

Figure 2. AWS Core MQTT Module Selection

Adding the AWS Core MQTT stack results in the default configuration with *some unmet dependencies*, as shown in the following Figure 3. FSP offers different Transport interfaces to the users. In this application note, we will be covering the Ethernet Interface which uses the *AWS Transport Interface on Secure Socket* as shown in the Figure 4.

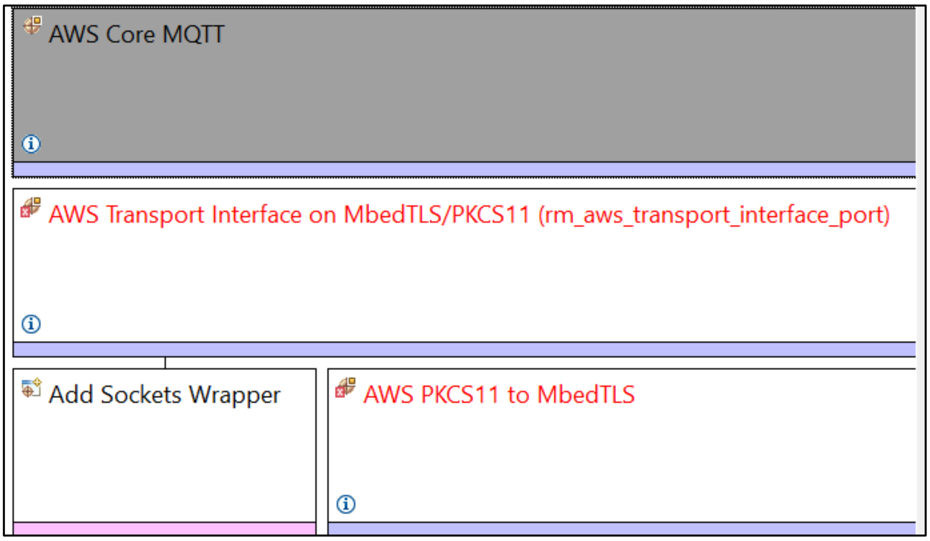

Figure 3. AWS Core MQTT Stack View

While the AWS Core MQTT stack shown contains a lot of dependencies and configurable properties, most default settings can be used as-is. The following change is needed to meet all unmet dependencies (marked in red) for the AWS Core MQTT stack added to a new project (as shown above):

- Enable Mutex and Recursive Mutex usage support as needed by IoT SDK and FreeRTOS in the created Thread properties.

Upon completion of the above step, the AWS Core MQTT is ready to accept a socket implementation, which has dependencies on using a TLS Session and an underlying TCP/IP implementation.

Additional documentation on the AWS Core MQTT is available in the *FSP User's Manual* under *RA Flexible Software Package Documentation > API Reference > Modules > AWS Core MQTT.*

3.2 Transport Layer Implementation

The FSP provided AWS Transport Interface provides options for Cellular, Ethernet and Wi-Fi. **AWS Transport Interface on MbedTLS** module is used for the Ethernet Interface. While the RA FSP contains a Transport interface on MbedTLS Implementation for Wi-Fi and cellular, this application and application note focuses on the use of the Ethernet Interface.

Ethernet Sockets can be added to the Thread Stack by clicking on **Add Sockets Wrapper > New > AWS TCP Wrapper**.

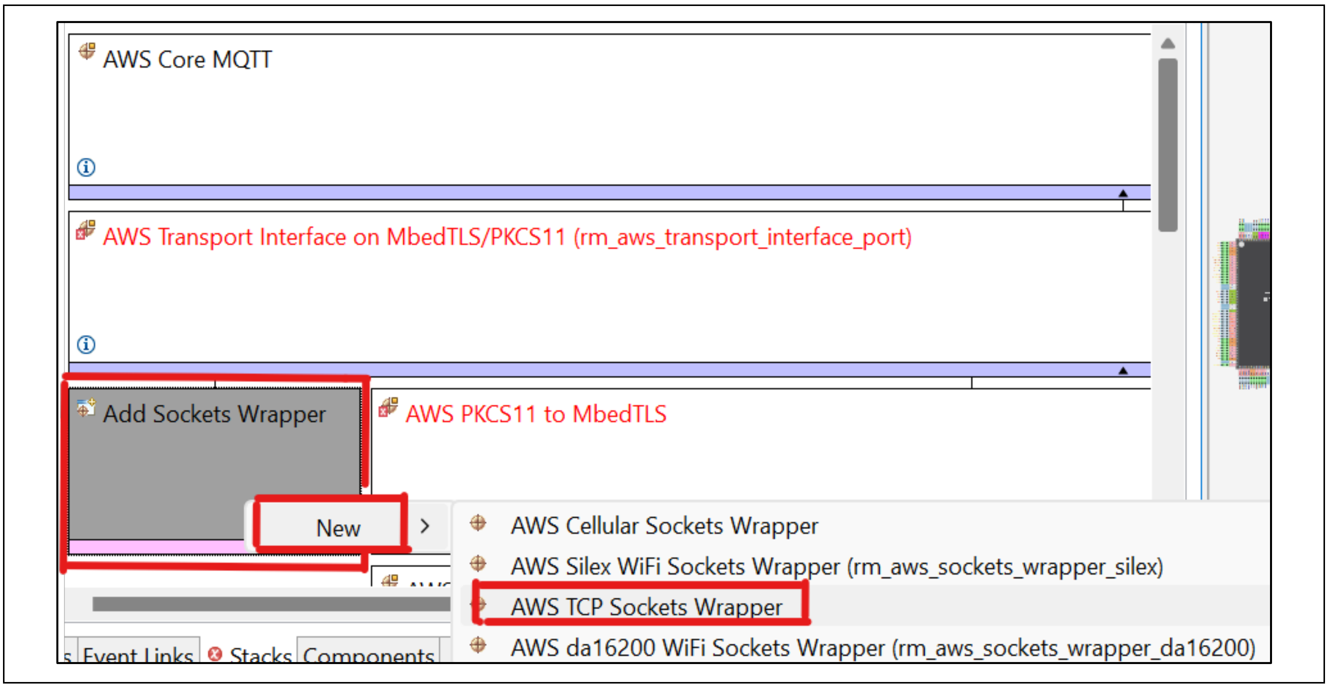

Figure 4. Adding Ethernet Interface to the Core MQTT Module

Upon addition, the needed stack is complete and has unmet dependencies for the dependent modules.

Now hover the cursor over the red blocks and the error will pop up. Make the appropriate settings.

- For the **Buffer Allocation error**: Choose the heap implementation using **New Stack > FreeRTOS > Memory Management > Heap 4**. Also, set **Dynamic Memory allocation** using **Application Thread > Properties > Memory Allocation > Support Dynamic Allocation > Enabled**.
- For AWS PKCS11 to MbedTLS error: MBEDTLS_CMAC_C must be defined Using **MbedTLS(Crypto Only) > Common > Message Authentication Code > MBEDTLS_CMAC_C**.
- For Crypto, MBEDTLS_ECDH_C in MbedTLS must be defined when using MbedTLS. **Using MbedTLS(Crypto Only) > Public Key Cryptography(PKC|ECC| MBEDTLS_ECDH_C**.
- For Crypto: **MBEDTLS_FS_IO, MBEDTLS_PSA_CRYPTO_STORAGE_C, MBEDTLS_PSA_ITS_FILE_C** in MbedTLS must be defined when using MbedTLS. **Using MbedTLS(Crypto Only) > Common > Storage.**
- For RTOS Heap memory error, set Heap Memory allocation using **Application Thread > Properties > Memory Allocation > Total Heap Size > 0x20000**
- For resolving the error: A heap is required to use with malloc with LittleFS, Add heap under **BSP | RA Common | Heap Size of 0x1000 is required to do malloc with LittleFS** and other standard library functions.
- Mutexes must be enabled using **Application Thread > Common > General > Use Mutexes > Enabled**
- Mutexes must be enabled using **Application Thread > Common > General > Use Recursive Mutexes > Enabled**
- UART specific errors can be resolved by enabling the Flow control and selecting the appropriate RTS and CTS pin selection.

  Note: These are the basic settings required to remove the error from the configurator. More specific configurations are listed in the specific module and its usage.

After all the appropriate settings have taken care of the errors due to the missing configuration, the new configurator screenshot looks clean with no errors as shown below in the Figure 5.

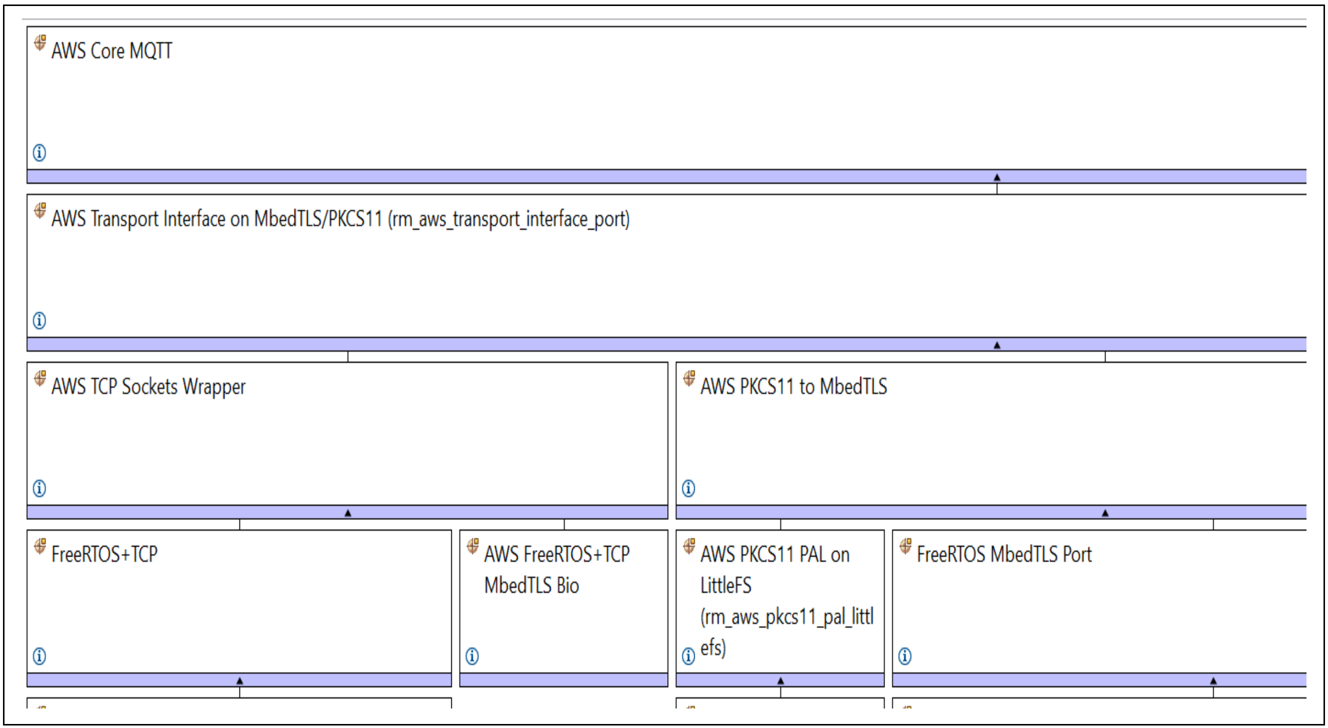

Figure 5. Expanded Socket Interface Module

Additional documentation on AWS Secure Sockets is available in the *FSP User's Manual* under *RA Flexible Software Package Documentation > API Reference > Modules > AWS Secure Sockets.*

3.3 Mbed TLS

Mbed TLS is Arm®'s implementation of the TLS protocols as well as the cryptographic primitives required by those implementations. Mbed TLS is also solely used for its cryptographic features even if the TLS/SSL portions are not used.

TLS Support uses FreeRTOS+Crypto which eventually uses Mbed TLS. The use of Mbed TLS requires configuration and operation of the Mbed Crypto module which in turn operates the SCE on the MCU.

The following underlying mandatory changes are needed to the project using the Secure Sockets on FreeRTOS+Crypto module:

1. Use FreeRTOS heap implementation scheme 4 (first fit algorithm with coalescence algorithm) or scheme 5 (first fit algorithm with coalescence algorithm with heap spanning over multiple non-adjacent/non-contiguous memory regions.
2. Enable support for dynamic memory allocation in FreeRTOS.
3. Enable Mbed TLS platform memory allocation layer.
4. Enable the Mbed TLS generic threading layer that handles default locks and mutexes for the user and abstracts the threading layer to use an alternate thread-library
5. Enable Elliptic Curve Diffie Hellman library.
6. Change FreeRTOS Total Heap Size to a value greater than 0x20000.

Additional documentation on the Mbed TLS is available in the *FSP User's Manual* under *RA Flexible Software Package Documentation > API Reference > Modules > Crypto Middleware (rm_psa_crypto)*.

3.4 MQTT Module APIs Usage

Table 2 lists APIs provided by AWS Core MQTT that are used as a part of the Application Example.

Table 2. MQTT Module APIs

API	Description
MQTT_Init	Initializes an MQTT context
MQTT_Connect	Establishes an MQTT session
MQTT_Subscribe	Sends MQTT SUBSCRIBE for the given list of topic filters to the broker
MQTT_Publish	Publishes a message to the given topic name
MQTT_Ping	Sends an MQTT PINGREQ to broker
MQTT_Unsubscribe	Sends MQTT UNSUBSCRIBE for the given list of topic filters to the broker
MQTT_Disconnect	Disconnect an MQTT session
MQTT_ProcessLoop	Loop to receive packets from the transport interface. Handles keep-alive
MQTT_ReceiveLoop	Loop to receive packets from the transport interface. Does not handle keep-alive
MQTT_GetSubAckStatusCodes	Parses the payload of an MQTT SUBACK packet that contains status codes corresponding to topic filter subscription requests from the original subscribe packet
MQTT_Status_strerror	Error code to string conversion for MQTT statuses.
MQTT_PublishToResend	Get the packet ID of the next pending publish to be resent

4. Cloud Connectivity Application Example

4.1 Overview

This application project demonstrates the use of APIs available through the Renesas FSP-integrated modules for Amazon IoT SDK C, Mbed TLS module, Amazon FreeRTOS, and HAL Drivers operating on Renesas RA MCUs. Network connectivity is established using Ethernet. The application running on a Renesas Cloud Kit also serves as a guide for the operation of Core MQTT, Mbed TLS/Crypto, and Ethernet configuration, using the FSP configurator. The application may be used as a starting point for other customized cloud-based solutions using Renesas RA MCUs. In addition, it simply demonstrates the operation and setup of cloud services available through the cloud service provider.

The upcoming sub-sections show step-by-step creation of a device and security credentials policies as required by the AWS IOT on the cloud side to communicate with the end devices. The example accompanying this documentation demonstrates Subscribe and Publish messaging between Core MQTT and MQTT Broker, on demand publication of sensor data, and asynchronous publication of a “sensor data” event from the MCU to the Cloud. The device is also subscribed to receive actuation events (LED indication) from the Cloud.

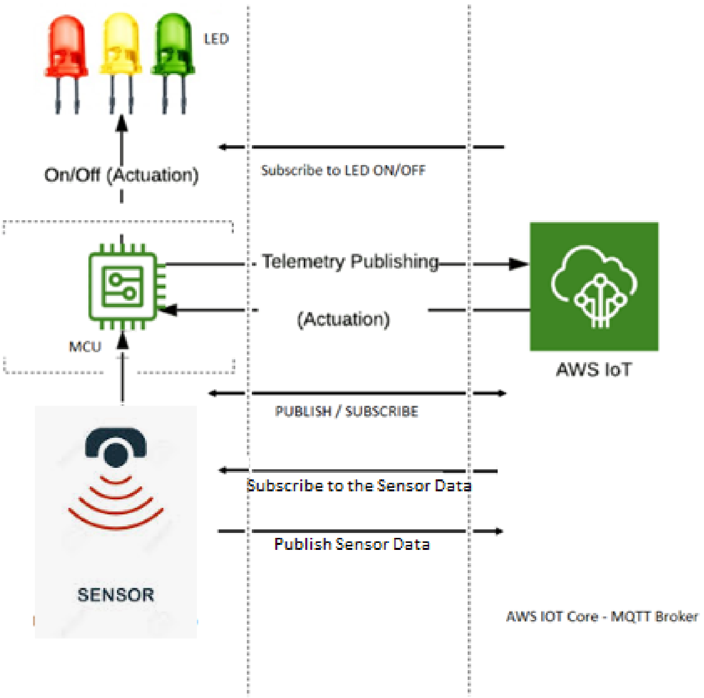

Figure 6. MQTT Publish/Subscribe to/from AWS IoT Core

4.2 MQTT/TLS Application Software Overview

The following files from this application project serve as a reference, as shown in Table 3.

Table 3. Application Project Files

No.	Filename	Purpose
1.	`src/app_thread_entry.c`	Contains initialization code and has the main thread used in Cloud Connectivity application.
2.	`src/common_init.c`	Contains code used to initialize common peripherals across the project.
3.	`src/common_init.h`	Contains macros, data structures, and functions prototypes used to initialize common peripherals across the project.
4.	`src/common_utils.c`	Contains code commonly used across the project.
5.	`src/common_utils.h`	Contains macros, data structures, and functions prototypes commonly used across the project.
6.	`src/console_thread_entry.c`	Contains the code for command line interface and flash memory operations.
7.	`src/ICM_20948.c`	Contains the code for the 9-Axis MEMS Motion Tracking™ Sensor
8.	`src/ICM_20948.h`	Contains the Data structure function prototypes for the 9-Axis MEMS Motion Tracking™ Sensor
9.	`src/ICP_10101.c`	Contains the code for Barometric Pressure and Temperature Sensor
10.	`src/ ICP_10101.h`	Contains the Data structure and function prototypes for Barometric Pressure and Temperature Sensor
11.	`src/mqtt_demo_helpers.c`	Contains code and functions used in MQTT interface for Cloud Connectivity.

No.	Filename	Purpose
12.	src/mqtt_demo_helpers.h	Accompanying header for exposing functionality provided by mqtt_demo_helpers.c.
13.	src/oximeter_thread_entry.c	Contains the code for Heart Rate, Blood Oxygen Concentration, Pulse Oximetry, Proximity, Light and Color Sensor
14.	src/Oximeter.c	Contains data structures and functions used for the oximeter sensor
15.	src/Oximeter.h	Contains the Data structure and function prototypes for the oximeter sensor
16.	src/oximstruct.h	Contains the Data structure for the oximeter sensor.
17.	src/r_typedefs.h	Contains the common derived data types
18.	src/RA_HS3001.c	Contains the code and function for Renesas Relative Humidity and Temperature Sensor.
19.	src/RA_HS3001.h	Contains the common data structure's function prototypes for the Renesas Relative Humidity and Temperature sensors.
20.	src/RA_ZMOD4XXX_Common.c	Contains the common code for Renesas ZMOD sensors.
21.	src/RA_ZMOD4XXX_Common.h	Contains the common data structure's function prototypes for the Renesas ZMOD sensors
22.	src/RA_ZMOD4XXX_IAQ1stGen.c	Contains the common code for the Renesas ZMOD Internal Air Quality sensors
23.	src/RA_ZMOD4XXX_OAQ1stGen.c	Contains the common code for the for the Renesas ZMOD Outer Air Quality sensors.
24.	src/RmcI2C.c	Contains the i2c wrapper functions for the third-party sensors not integrated with FSP.
25.	src/RmcI2C.h	Contains the i2c function prototypes for wrapper functions for the third-party sensors not integrated with FSP.
26.	src/sensor_thread_entry.c	Contains the Code to access the Sensor data from the different sensors
27.	src/ user_choice.c	Contains the code for user choice of sensors and user configurations
28.	src/user_choice.h	Contains the Function prototypes for the Sensor and its user configuration for the different sensors and its data accessibility.
29.	src/usr_config.h	To customize the user configuration to run the application.
30.	src/usr_hal.c	Contains data structures and functions used for the Hardware Abstraction Layer (HAL) initialization and associated utilities.
31.	src/usr_hal.h	Accompanying header for exposing functionality provided by usr_hal.c.
32.	src/usr_data.h	Accompanying header file for the application thread.
33.	src/usr_network.c	Contains data structures and functions used to operate the FreeRTOS TCP/IP and Ethernet Module. This file is for Ethernet-specific use.

No.	Filename	Purpose
34.	src/usr_network.h	Accompanying header for exposing functionality provided by usr_network.c This file is for Ethernet-specific use.
35.	zmod_thread_entry.c	Contains the code for indoor air quality sensor
36.	src/SEGGER_RTT/SEGGER_RTT.c	Implementation of SEGGER real-time transfer (RTT) which allows real-time communication on targets which support debugger memory accesses while the CPU is running.
37.	src/SEGGER_RTT/SEGGER_RTT.h	
38.	src/SEGGER_RTT/SEGGER_RTT_Conf.h	
39.	src/SEGGER_RTT/SEGGER_RTT_printf.c	
40.	src/ backoffAlgorithm/backoff_algorithm.c	Retry algorithms with random back off for the next retry attempt
41.	src/ backoffAlgorithm/backoff_algorithm.h	Retry algorithms with random back off for the next retry attempt header file
42.	src/subcription_manager/ mqtt_subscription_manager.c	MQTT Subscription manager, which handles the callback
43.	src/subcription_manager/ mqtt_subscription_manager.h	Associated header file for MQTT Subscription manager, which handles the callback.
44.	src/console_menu/console.c	Contains data structures and functions used to print data on console using UART
45.	src/console_menu/console.h	Contains the Function prototypes used to print data on console using UART
46.	src/console_menu/menu_flash.c	Contains data structures and functions used to provide CLI flash memory related menu
47.	src/console_menu/menu_flash.h	Contains the Function prototypes and macros used to provide CLI flash memory related menu
48.	src/console_menu/menu_kis.c	Contains functions to get the FSP version, get UUID and help option for main menu on CLI
49.	src/console_menu/menu_kis.h	Contains the Function prototypes and macros used to get fsp version, get UUID and help option for main menu on CLI
50.	src/console_menu/menu_main.c	Contains data structures and functions used to provide CLI main menu options
51.	src/console_menu/menu_main.h	Contains the Function prototypes and macros used to provide CLI main menu options
52.	Src/flash/flash_hp.c	Contains data structures and functions used to perform flash memory related operations
53.	src/flash/flash_hp.h	Contains the function prototypes and macros used to perform flash memory related operations
54.	src/i2c.c	Contains data structures and functions used for I2C communication
55.	src/i2c.h	Contains the Function prototypes and macros used for I2C communication
56.	src/ob1203_bio/KALMAN/kalman.c	Contains algorithm for Heart Rate, Blood Oxygen Concentration, Pulse Oximetry, Proximity, Light and Color Sensor sample calculations
57.	src/ob1203_bio/KALMAN/kalman.h	
58.	ob1203_bio/OB1203/OB1203.c	
59.	ob1203_bio/OB1203/OB1203.h	
60.	ob1203_bio/SAVGOL/SAVGOL.c	
61.	ob1203_bio/SAVGOL/SAVGOL.h	
62.	ob1203_bio/SPO2/SPO2.c	
63.	ob1203_bio/SPO2/SPO2.h	

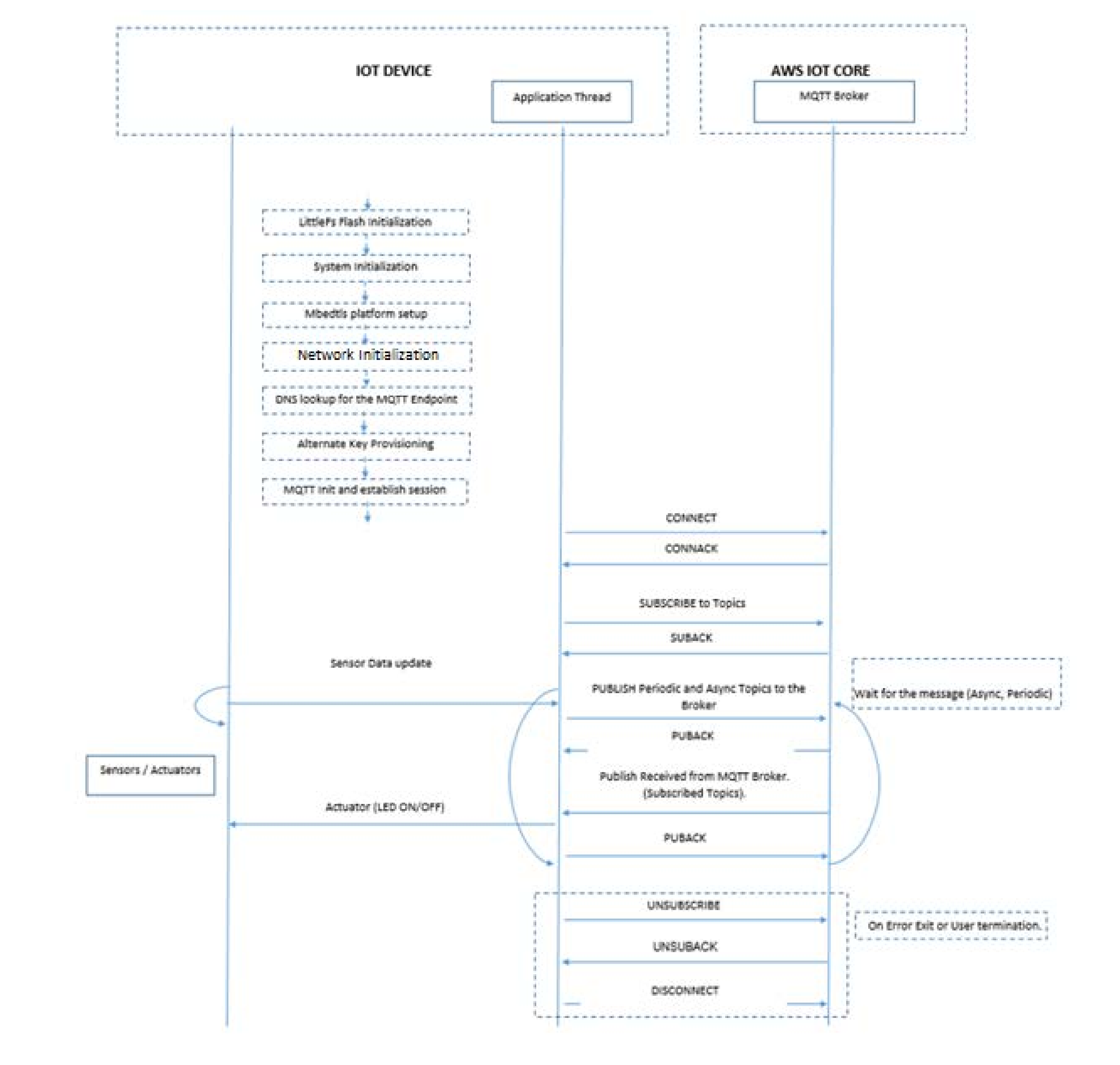

Figure 7. Application Example Implementation Details

4.3 Creating the Application Project using the FSP Configurator

Complete steps to create the project from the start using the e² studio and FSP configurator. The following table shows the step-by-step process in creating the Project. It is assumed that the user is familiar with the e² studio and FSP configurator. Launch the installed e² studio for the FSP.

Table 4. Step-by-step Details for Creating the Application Project for Ethernet

Steps		Intermediate Steps
1	**Project Creation:**	**File → New → C/C++ Project**
2	**Project Template:**	**Templates for New RA C/C++ Project → Renesas RA C/C++ Project → Next**
3	**e² studio - Project Configuration (RA C Executable Project) →**	**Project Name (Name for the Project)** Note: Input your desired name for the project -> **Next**
4	**Device Selection →**	FSP Version: **4.2.0**
		Board: **CK-RA6M5**
		Device: **R7FA6M5BH3CFC**
		Language: C

Steps		Intermediate Steps
5	**Select Tools**	Toolchain: **GNU ARM Embedded** (Default)
		Toolchain version: **(10.3.1.20210824** or newer)
		Debugger: **J-Link ARM → Next**
	Project Type Selection	Flat (Non-TrustZone) Project **→ Next**
6	**Build Artifact and RTOS Selection**	Artifact Selection: **Executable**
		RTOS Selection: **FreeRTOS(v10.4.6+fsp4.2.0) → Next**
	Project Template Selection	**Project Template Selection: FreeRTOS – Minimal – Static Allocation → Finish**
7	**Clock**	**HOCO 20MHz →PLL Src:HOCO → PLL Div/2 →PLL Mul x20.0 → PLL 200MHz**
8	**Stacks Tab** (Part of the FSP Configurator)→	**Threads → New Thread**
9	**Config Thread Properties→**	
		Symbol: **app_thread**
		Name: **App Thread**
		Stack size: **0x12000 Bytes**
		Priority 3
		Thread Context: **NULL**
		Memory Allocation: **Static**
10	Generic RTOS configs under thread (Additional configuration on top of the Default Config provided by FSP)	
	Common → General	Use Mutexs: **Enabled**
		Use Recursive Mutexes: **Enabled**
		Max Task Name Len: **16**
	Common → Memory Allocation	Support Dynamic Allocation: **Enabled**
		Total Heap Size: **0x20000**
	Common->Optional Functions	xTimerPendFunctionCall() Function: **Enabled**
11	Add the Heap Implementation in HAL/Common	
	New Stack →	**RTOS → FreeRTOS Heap 4**
12	**Adding the AWS MQTT Wrapper Module to the Application Thread**	
	Note: Now the Newly created thread (Application thread) is ready to add new stack (Here the AWS Core MQTT is added)	
	New Stack →	**Networking → AWS Core MQTT**
13	Adding the AWS Core MQTT module brings in the AWS Core MQTT module to the configurator. Now the dependency modules must be added.	
13a	Under the **Add sockets wrapper**, add	**New → AWS TCP Sockets Wrapper**
13b	Under the **SCE Compatibility mode**, add	**New → Key Injection for PSA CRYPTO**
13c	Under the **AWS Core MQTT**	**Common → Retry count for reading CONNACK from network → 10**
14	Adding persistent storage support for AWS PKCS11 and resolve the error in the configurator by selecting the Heap size in the BSP Tab. Right-click on pink highlighted stack to:	
	Under the MbedTLS(Crypto only)Add LittleFS module →	**Use → LittleFS**
	BSP Tab → RA Common→	Heap size (bytes) : 0x20000
14a	Under **LittleFS on Flash**	**Block count → (BSP_DATA_FLASH_SIZE_BYTES/256)**
15	Some dependency related to TLS Support are needed to be resolved to remove the error in the FSP configurator by modifying the **MbedTLS(Crypto Only)** property settings.	
	Common → Platform →	MBEDTLS_PLATFORM_MEMORY: **Define**
	Common → General →	MBEDTLS_THREADING_C: **Define**

Steps		Intermediate Steps
	Common → General →	MBEDTLS_THREADING_ALT: **Define**
	Common → Public Key Cryptography (PKC) →	ECC → MBEDTLS_ECDH_C: **Define**
	Common →Hardware acceleration → Public key cryptography	RSA 3072 verify: **Enabled**
	Common →Hardware acceleration → Public key cryptography	RSA 4096 verify: **Enabled**
	Common → Storage →	MBEDTLS_FS_IO: **Define**
	Common → Storage →	MBEDTLS_PSA_CRYPTO_STORAGE_C: **Define**
	Common → Storage →	MBEDTLS_PSA_ITS_FILE_C: **Define**
	Common → Message Authentication Code (MAC)→	MBEDTLS_CMAC_C : **Define**
16	**FreeRTOS + TCP Configuration** Note: This is only applicable for the Ethernet application project. Most of the default settings remain the same, except few of the default configuration needs to be changed	
	Common →	**DHCP callback function → Enable**
		DHCP Send Discover After Auto IP: Enable
		Let TCP use windowing mechanism → Enable
		Stack size in words -> config min stack size *16
		DNS request attempts -> 6
		Set max no. Of events -> ipconfig NUM_NETWORK_BUFFER_DESCRIPTORS + 16
		Size of rx and tx buff for tcp socket -> 8192
17	**Ethernet Driver Configuration**	
	General→	**Flow control functionality → Enable**
	Buffers→	**Number of Tx Buffers → 8**
		Number of Rx Buffers → 8
	Interrupts→	**interrupt priority → 5**
18	**Under Module g_ether_phy0 (r_ether_phy) Ethernet**	
	Common →	**ICS1894 target → Enabled**
	Common →	**Reference clock→ Enabled**
	Ether PHY Driver	
	g_ether_phy0 →	**PHY LSI Address → 5**
19	Adding the HAL Modules as required for the Application Project: External IRQ, GPT Timer0, GPT Timer1, GPT Timer2, for 30 Seconds periodic timer, 1 second Periodic, Heartbeat Monitor Timer, respectively.	
	HAL/Common Stacks → New Stack	**→ Input → External IRQ Driver on r_icu**
	Property Settings for r_icu	Name: **g_sensorIRQ**
		Channel: **14**
		Trigger: **Falling**
		Digital Filtering: **enabled**
		Digital Filtering Sample Clock (PCLK/64)
		Pin Interrupt Priority: **Priority 2**
		Callback: **sensorIRQCallback**
	HAL/Common Stacks → New Stack	**→ Timers → Timer Driver on r_gpt**
	Property Settings for **r_gpt → General**	Name: **g_timer0**
		Channel: **0**
		Mode: **Periodic**
		Period: **10**
		Period Unit: **Milli seconds**
	Interrupts:	Callback: **t_callback**

Steps		**Intermediate Steps**
		Overflow/Crest Interrupt Priority: **Priority 5**
	HAL/Common Stacks → New Stack	**→ Timers → Timer Driver on r_gpt**
	Property Settings for r_gpt → General	Name: **g_timer1**
		Channel: **1**
		Mode: **Periodic**
		Period: **1**
		Period Unit: **Seconds**
	Interrupts:	Callback: **g_user_timer_cb**
		Overflow/Crest Interrupt Priority: **Priority 5**
	HAL/Common Stacks → New Stack	**→ Timers → Timer Driver on r_gpt**
	Property Settings for r_gpt → General	Name: **g_timer2**
		Channel: 2
		Mode: **Periodic**
		Period: **1**
		Period Unit: **Milli Seconds**
	Interrupts:	Callback: **TimerCallback**
		Overflow/Crest Interrupt Priority: **Priority 6**
	HAL/Common Stacks → New Stack	**Clock → Clock Generation Circuit on r_cgc**
	Property Settings for r_cgc	**Name: g_cgc0**
		EXTAL: P212
		XTAL: P213
20	**Modifying the BSP Settings - RA Common for (Main stack and Heap Settings)**	
	Property Settings for RA Common	Main stack size(bytes): **0x2000**
		Heap size (bytes): **0x20000**
21	Adding FreeRTOS Objects for the Application (Topic Queue needs to be created for the application – Message Queue)	
	Stacks Tab → Objects →	**New Object → Queue**
	Property Settings for the Queue	Symbol: **g_topic_queue**
		Item Size (Bytes): **64**
		Queue Length (Items): **16**
		Memory Allocation: **Static**
	Stacks Tab → Objects →	**New Object → Mutex**
	Property Settings for the Mutex	Symbol: **g_sens_data_mutex**
		Type: Mutex
		Memory Allocation: **Static**
	Stacks Tab → Objects →	**New Object → Mutex**
	Property Settings for the Mutex	Symbol: **g_console_out_mutex**
		Type: Mutex
		Memory Allocation: **Static**
	Stacks Tab → Objects →	**New Object → Mutex**
	Property Settings for the Mutex	Symbol: **g_update_console_event**
		Type: Mutex
		Memory Allocation: **Static**
	Stacks Tab → Objects →	**New Object → Binary Semaphore**
	Property Settings for the Semaphore	Symbol: **g_ob1203_semaphore**
		Memory Allocation: **Static**
	Stacks Tab → Objects →	**New Object → Binary Semaphore**
	Property Settings for the Semaphore	Symbol: **g_console_binary_semaphore**
		Memory Allocation: **Static**
22	**Config Thread Properties→**	
		Symbol: **sensor_thread**

Steps		Intermediate Steps
		Name: **Sensor Thread**
		Stack size: 8192 **Bytes**
		Priority: **4**
		Thread Context: **NULL**
		Memory Allocation: **Static**
23	Adding the HS300X Sensor Module and ZMOD4510 OAQ sensor module to the Sensor Thread	
	New Stack →	**Sensor → HS300X Temperature/Humidity Sensor**
	Config HS300X sensor→	Name: g_**hs300x_sensor0**
		Callback: **hs300x_callback**
	New Stack →	**Sensor → ZMOD4510 OAQ Sensor**
	Config ZMOD4510 sensor→	Name: **g_zmod4xxx_sensor1**
		Callback: **zmod4xxx_comms_i2c1_callback**
		IRQ Callback: **zmod4xxx_irq1_callback**
24	Adding ICM-20948 and ICP10101 sensors to the Sensor Thread. Note: FSP doesn't provide an integrated module for ICM-20948 and ICP10101 sensors. This needs to be integrated via the i2c communication device manually. Also, its related sensor driver code needs to be added to the src folder. Add ICM-20948 (9 Axis MEMS) Processing to the Sensor Thread	
	New Stack →	**Connectivity: I2C Communication Device**
	Config I2C Comm Device →	**Name: g_comms_i2c_device5**
		Slave Address: 0x68
		Callback: ICM_comms_i2c_callback
	Add I2C Shared Bus→	**Add I2C Shared Bus→Used→g_comms_i2c_bus0 I2C Shared Bus**
	Module g_i2c_master0 I2C Master	**Rate: Fast Mode**
	Adding I2C Communication Device (for ICP10101) into Sensor Thread	
	New Stack →	**Connectivity: I2C Communication Device**
	Config I2C Comm Device →	**Name: g_comms_i2c_device4**
		Slave Address:0x63
		Callback: ICP_comms_i2c_callback
	Add I2C Shared Bus→	**Add I2C Shared Bus→Used→g_comms_i2c_bus0 I2C Shared Bus**
	Module g_i2c_master0 I2C Master	**Rate: Fast Mode**
25	**Stacks Tab (Part of the FSP Configurator)→**	**Threads → New Thread**
	Config Thread Properties→	
		Symbol: **oximeter_thread**
		Name: **Oximeter Thread**
		Stack size: **2048 Bytes**
		Priority: **4**
		Thread Context: **NULL**
		Memory Allocation: **Static**
26	Adding the OB1203 sensor module to the Oximeter Thread Note : OB1203 sensor code uses non FSP code	
	New Stack →	**Connectivity → I2C Communication Driver**
	Config OB1203 sensor→	Name: **g_comms_i2c_device3**
		Callback: **comms_i2c_callback**
		Semaphore Timeout (RTOS only): **0xFFFFFFFF**

Steps		Intermediate Steps
		Slave Address: **0x53**
		Address Mode: **7-Bit**
		Callback: **comms_i2c_callback**
27	**Stacks Tab** (Part of the FSP Configurator)→	**Threads → New Thread**
	Config Thread Properties→	
		Symbol: **zmod_thread**
		Name: **Zmod Thread**
		Stack size: **1024 Bytes**
		Priority: **3**
		Thread Context: **NULL**
		Memory Allocation: **Static**
28	Adding the ZMOD4XXX sensor module to the Zmod Thread Note: ZMOD4410 IAQ Sensor is configured (part of the FSP configurator)	
	New Stack →	**Sensor → ZMOD4XXX Gas Sensor**
	Config ZMOD4XXX sensor→	Name: **g_zmod4xxx_sensor0**
		Callback: **zmod4xxx_comms_i2c_callback**
		IRQ Callback: **zmod4xxx_irq0_callback**
29	Create and add Console processing Thread	
	Stacks tab (Part of the **FSP Configurator**)	**Threads → New Thread**
	Configure Thread Properties	
	Symbol	**Console_Thread**
	Name	**Console Thread**
	Stack size	**4096 Bytes**
	Priority	**3**
	Auto start	**Enabled**
	Time slicing interval (ticks)	**50**
30	Adding Uart to Console_Thread	
	New Stack →	**Connectivity: UART**
	Config Common →	**FIFO Support: Enable**
		DTC Support: Enable
		Flow Control Support: Enable
	Config General →	**Name: g_console_uart**
		Channel: 5
		Data Bits: 8bits
		Parity: None
		Stop Bits: 1bit
	Config Baud→	**Baudrate: 115200**
	Config Interrupts →	**Callback: user_uart_callback**
	Config Pins →	**TXD: P501**
		RXD: P502
31	Adding Flash to Console_Thread	
	New Stack →	**Storage: Flash**
		Name: user_flash
		Data Flash Background Operation: Disabled
		Callback: flash_callback
		Flash Ready Interrupt Priority: Priority 2
		Flash Error Interrupt Priority: Priority 2

The above configuration is a prerequisite to generate the required stack and features for the cloud connectivity application provided with this application note. Once the **Generate Project Content** button is clicked, it generates the source code for the project. The generated source code contains the required drivers, stack, and middleware. The user application files must be added to the `src` folder.

Note: `app_thread_entry.c`, `sensor_thread_entry.c`, `oximeter_thread_entry.c`, `zmod_thread_entry.c`, `console_thread_entry.c` are the auto generated file as part of the project creation. Users are required to add code to this file.

Note: FSP generated code must be called/used for the application, while some of the middleware needs to be called exclusively as part of the application for proper initialization. For instance, the `Mbedtls_platform_setup()` call initializes the SCE and TRNG.

For validation of the created project, the same source files listed in section MQTT/TLS Application Software Overview (as shown in Table 3) may be added. Users are required to add the directory path and subdirectory for proper compilation. Refer to the enclosed project for more details.

The details of the configurator from the default settings to changed settings are described in the following sections, including the reason for the change.

4.4 MQTT/TLS Configuration

This section describes the MQTT and TLS module configuration settings that are done as part of this application example.

The following table lists changes made to a default configuration populated by the RA Configurator.

Table 5. Default Configuration for CK-RA6M5

Property	Original Value	Changed Value	Reason for Change
Application Thread			
Common → General → Use Mutexes	Disabled	Enabled	This requirement is set by the AWS IOT SDK C stack
Common → Memory Allocation → Support Dynamic Allocation	Disabled	Enabled	This requirement is set by the AWS IOT SDK C stack
Common → Memory Allocation → Total Heap Size	0	0x20000	Heap required for the FreeRTOS, AWS IOT SDK, Mbed TLS
AWS IoT Common			
Platform Name	Unknown	AWS Cloud Connectivity	This value is user selectable and can be set to any value.
Mbed TLS (Crypto Only)			
Platform → Mbedtls_platform_memory	Undefine	Define	This selection is required in order to support the Mbed_tls.
General → Mbedtls_threading_alt	Undefine	Define	This selection is required in order to support the Mbed_tls to plug in any thread library.
General → Mbedtls_threading_c	Undefine	Define	This selection is required in order to support the Mbed_tls to abstract the threading layer to allow easy plugging in any thread-library.

Property	Original Value	Changed Value	Reason for Change
Public Key Cryptography → ECC → Mbedtls_ecdh_c	Undefine	Define	This selection is required in order to support the Mbed_tls to enable the ECDH module.
LittleFS (Heap Selection)			
BSP → RA Common Heap Size	0x0	0x2000	Heap selection for Heap 3 and below needs to be done here.
FreeRTOS + TCP Configuration			
Common → DHCP Callback function	Disable	Enable	Callback for DHCP handling
Let TCP use windowing mechanism →	Disable	Disable	Let TCP use windowing mechanism → Disabled Enable
Ethernet Driver Configuration			
General → Flow control functionality→	Disable	Disable	Flow control selection for Ethernet
Buffers → Number of Tx Buffers →	1	8	Tx Buffer for Data Transmit
Buffers → Number of Rx Buffers →	1	8	Rx Buffer for Data Reception

5. Sensor Stabilization Time

Sensor Name	When Powered Up First Time	After Soft or Hard Reset
ZMOD4410 IAQ	Up to one hour	Up to one minute
ZMOD4510 OAQ	Up to 24 hours	Up to two hours
OB1203	Up to one minute (after placing the finger on sensor, it may take up to 60 seconds to sense data)	Up to 10 seconds (after placing finger on sensor, it may take up to 60 seconds to sense data)
HS3001	Up to one minute	Up to 10 seconds
ICP	Up to one minute	Up to 10 seconds
ICM	Up to one minute	Up to 10 seconds

Note: Stabilization time of sensor provided above is from the point of sensor initialized

6. MQTT/TLS Module Next Steps

- For setting up a client using a device certificate signed by a preferred CA certificate, refer to the link: https://docs.aws.amazon.com/iot/latest/developerguide/device-certs-your-own.html
- For using a self-signed certificate to configure AWS, refer to the link: https://developer.amazon.com/docs/custom-skills/configure-web-service-self-signed-certificate.html

7. References

[1] International Telecommunication Union, "ITU-T Y.4000/Y.2060 (06/2012)," 15 06 2012. [Online]. Available: http://handle.itu.int/11.1002/1000/11559.

[2] Amazon Web Services, "AWS IoT Core Features," [Online]. Available: https://www.amazonaws.cn/en/iot-core/features/.

[3] Amazon Web Services, "AWS IoT Core," [Online]. Available: https://www.amazonaws.cn/en/iot-core/.

[4] W. T. L. L. O. S. R. N. S. R. X. G. K. N. K. S. F. M. K. D. L. I. R. Valerie Lampkin, Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry, IBM Redbooks, 2012.

[5] I. E. T. Force, "The Transport Layer Security (TLS) Protocol Version 1.2," [Online]. Available: https://tools.ietf.org/html/rfc5246.

[6] Amazon Web Services, "AWS IoT Security," [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/iot-security.html.

[7] Amazon Web Services, "Transport Security in AWS IoT," [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/transport-security.html.

[8] International Telecommunication Union, "X.509 (10/19) Summary," 10 2019. [Online]. Available: https://www.itu.int/dms_pubrec/itu-t/rec/x/T-REC-X.509-201910-I!!SUM-HTM-E.htm.

[9] Eclipse Foundation, "Eclipse Mosquitto™ - An open source MQTT broker," [Online]. Available: https://mosquitto.org/.

[10] Amazon Web Services, "AWS IoT Device SDK C: MQTT," [Online]. Available: https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/mqtt/index.html.

[11] R. Barry, "Mastering the FreeRTOS™ Real Time Kernel," in *A Hands-On Tutorial Guide*, 2016.

[12] A. I. D. S. C. Documentation, "AWS IoT Device SDK C: MQTT Functions," [Online]. Available: https://docs.aws.amazon.com/freertos/latest/lib-ref/c-sdk/mqtt/mqtt_functions.html.

[13] Amazon, "Configuring the FreeRTOS Demos," [Online]. Available: https://docs.aws.amazon.com/freertos/latest/userguide/freertos-configure.html.

[14] "Amazon FreeRTOS mbedTLS," [Online]. Available: https://github.com/aws/amazon-freertos/blob/master/libraries/3rdparty/mbedtls/utils/mbedtls_utils.c.

[15] Renesas Electronics Corporation, "Renesas Flash Programmer (Programming GUI) - Documentation," [Online]. Available: https://www.renesas.com/us/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html#documents.

8. Known Issues

This section talks about the known FSP and tool related issues. More details can be found at the link: https://github.com/renesas/fsp/issues.

Dashboard works properly with Microsoft edge browser, whereas it does not work properly with Google Chrome browser.

9. Debugging

Enable the `USR_LOG_LVL (LOG_DEBUG)` macro in the application project for additional information for debugging.

10. Troubleshooting

To validate the functionality of the sensor data, run the Quick Start Example Project as described in the *CK-RA6M5 Quick Start Guide*.

Website and Support

Visit the following vanity URLs to learn about key elements of the RA family, download components and related documentation, and get support.

CK-RA6M5 Kit Information	renesas.com/ra/ck-ra6m5
RA Cloud Solutions	renesas.com/cloudsolutions
RA Product Information	renesas.com/ra
RA Product Support Forum	renesas.com/ra/forum
RA Flexible Software Package	renesas.com/FSP
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jun.14.22	—	Initial release
1.01	Jun.11.22	—	Minor update
1.02	Mar.15.23	—	Updated to FSP 4.2.0

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.