Azure RTOS sample projects using e² studio or IAR EW

R01AN6455EJ0102
Rev.1.02
2023.1.20

Introduction

Azure RTOS sample projects for each component (ThreadX, FileX, GUIX, NetX Duo, and USBX) can be created using Renesas e² studio or IAR Embedded Workbench (EW) with the on-board emulator. All samples are designed to run on RX family.

This document guides how to create and use these sample projects.

Supported Sample Projects

- **ThreadX sample project**
  Contains ThreadX source code
- **FileX RAM Disk sample project**
  Contains FileX source code
- **NetX Duo Ping sample project**
  Contains NetX Duo ping sample project
- **NetX Duo Iperf sample project**
  Contains NetX Duo iPerf sample project
- **IoT Embedded SDK sample project**
  Sample project to connect to Azure IoT Hub using Azure IoT Middleware for Azure RTOS
- **IoT Embedded SDK PnP sample project**
  Sample project to connect to Azure IoT Hub using Azure IoT Middleware for Azure RTOS via IoT Plug and Play
- **IoT Embedded SDK with IoT Plug and Play sample project**
  Sample project with IoT Plug and Play using multiple components
- **GUIX 8bpp sample project**
  Contains sample for GUIX 8BPP
- **GUIX 16bpp sample project**
  Contains sample for GUIX 16BPP
- **GUIX 16bpp draw 2d sample project**
  Contains sample for GUIX 16BPP with 2D Draw
- **USBX device CDC-ACM Class sample project**
  Contains USBX source code
- **USBX Host Mass Storage Class sample project**
  Contains USBX source code
- **ThreadX Low Power sample project**
  Contains ThreadX & low power utility source code
- **Azure Device Update (ADU) sample project**
  Sample project for OTA firmware update via Microsoft Azure
- **Secure bootloader sample project**
  Used together with ADU sample project to provide a secure boot

Supported Devices

- RX130
- RX140
- RX65N
- RX651
- RX660
- RX66T
- RX671
- RX72N

Supported sample projects are different by each device. For details, please refer to the following URL.

https://github.com/renesas/azure-rtos

Download Links for Development Environment

・e² studio：2022-10 or later

https://www.renesas.com/software-tool/e-studio

- Renesas C/C++ Compiler for RX Family CC-RX：V3.04.00 or later
  https://www.renesas.com/software-tool/cc-compiler-package-rx-family
- GCC for Renesas RX：8.3.0.202104 or later
  https://gcc-renesas.com/rx-download-toolchains/

・IAR Embedded Workbench for RX：4.20.1 or later

https://www.iar.com/products/architectures/renesas/iar-embedded-workbench-for-renesas-rx/

- RX Smart Configurator：V2.13.0 and later
  https://www.renesas.com/software-tool/smart-configurator

・Azure IoT Explorer

https://github.com/azure/azure-iot-explorer/releases

Contents

1. Getting Started

To create new Azure RTOS project, the procedure is different between e² studio and IAR EW.

1.1 Creating project using e² studio

1. Launch e² studio, create new project: [**File**] > [**New**] > [**Renesas C/C++ Project**] and select **Renesas RX** and create a new workspace.

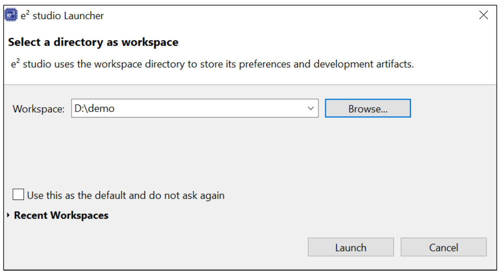


Figure 1.1 Workspace Creation Window

2. Select **GCC for Renesas RX C/C++ Executable Project** or **Renesas CC-RX C/C++ Executable Project**.

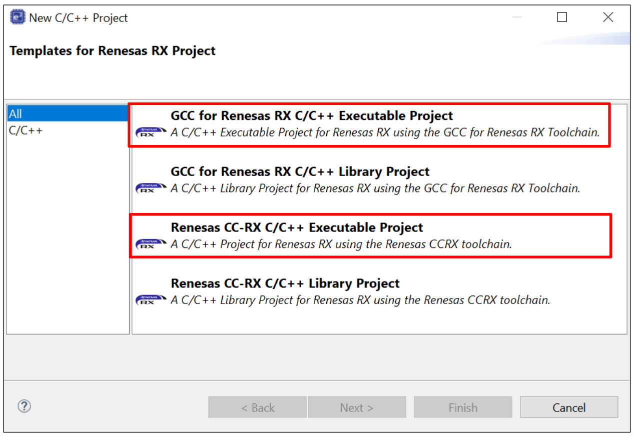


Figure 1.2 Toolchain Setting Window

3. Input the project name.
4. Click [**Next**].

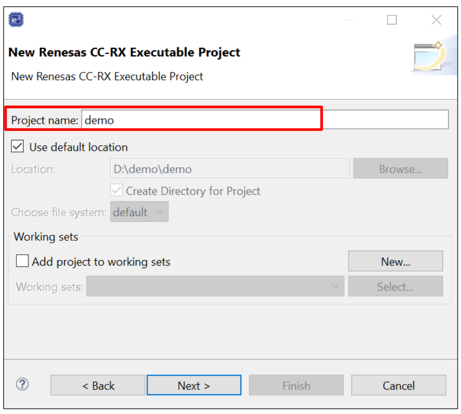


Figure 1.3 Project Creation Window

5. At **RTOS**, select “**Azure RTOS**”.
6. Click **Manage RTOS Versions…** to download software package.
7. At **RTOS Version**, select a version that downloaded at step 6.
8. At Target Board, select a board that you are working on.
9. Click [**Next**].

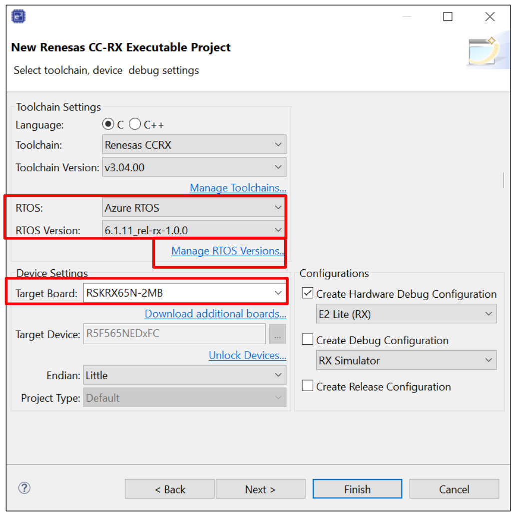


Figure 1.4 RTOS and Target Board Setting Window

10. Click [**Next**].

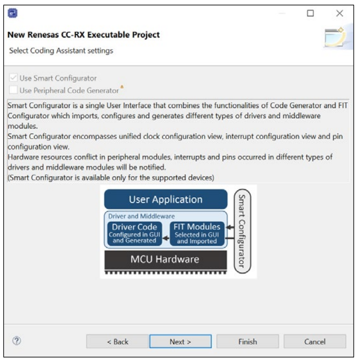


Figure 1.5 Coding Assistant Setting Window

11. Select an application.
12. Click [**Finish**].

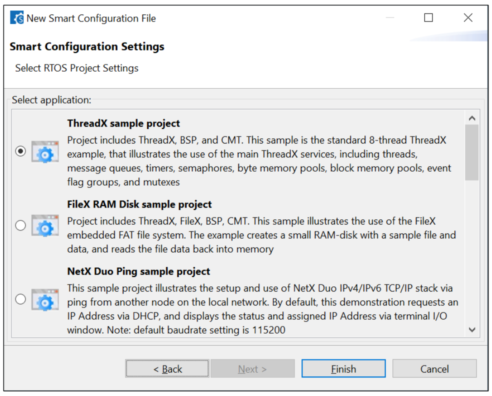


Figure 1.6 Select Application Window

13. Azure RTOS sample project including each component is created.

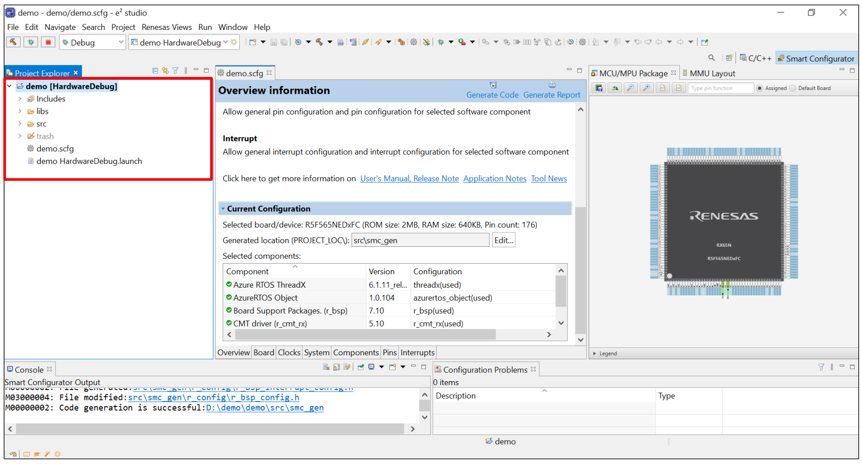


Figure 1.7 Created Sample Project Window

14. Build project: Select the sample project in the e^2 studio workspace and right click and select build to build the sample project.
15. Select Download and Debug to download and start execution of the project. By default, execution stops at a breakpoint set at main.

    Note: Other debugger settings may be required depending on the board type you specify.

    In the case of Renesas Starter Kit+ for RX65N-2MB: click **Debugger** > **Connection Settings** > **Power Target From The Emulator**, and set **No**.
16. Please review the sample descriptions later in this guide for additional setup and expected behavior.

1.2 Creating project using IAR EW

Please refer to following FAQ for the detailed instructions:
https://en-support.renesas.com/knowledgeBase/20533128

In AN ja, same update however changing URL
https://ja-support.renesas.com/knowledgeBase/20533124

2. Sample Project Descriptions

Additional setup and expected behavior of each sample project are described in this section.

2.1 ThreadX sample project

This sample is the standard 8-thread ThreadX example, that illustrates the use of the main ThreadX services, including threads, message queues, timers, semaphores, byte memory pools, block memory pools, event flag groups, and mutexes.

To run this sample, simply follow these steps (assuming the steps described in the previous section were done):

1. Set a breakpoint at any line.
2. Select **Go** to start execution of the sample project.

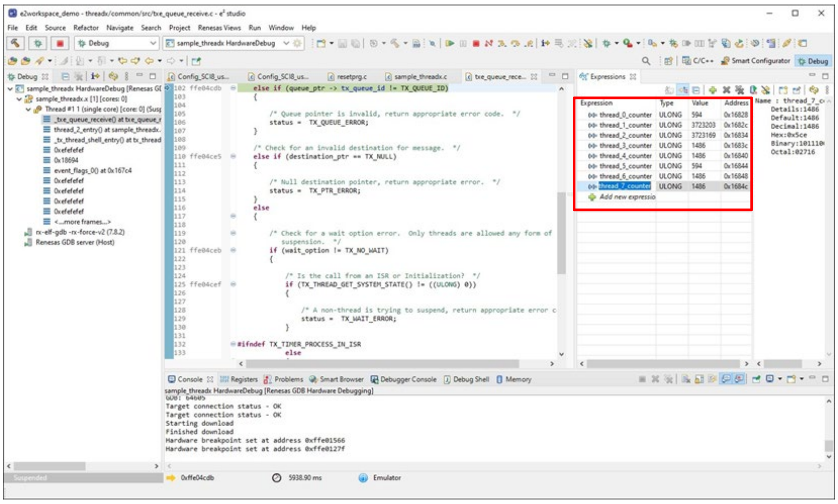


Figure 2.1 e^2 studio Debugger Screen

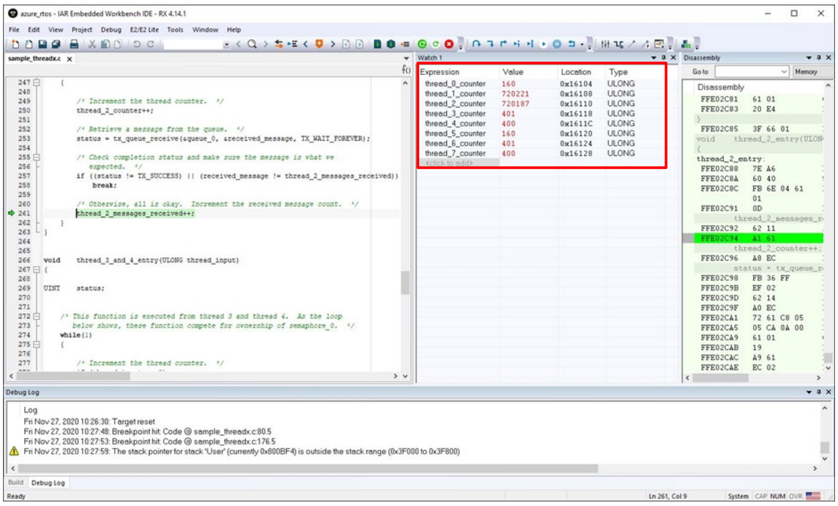


Figure 2.2 IAR EW Debugger Screen

After hitting **Break**, the debugger screen shot above shows various counters incremented by the ThreadX sample as each of the main components of the ThreadX are exercised.

To learn more about Azure RTOS ThreadX, view https://docs.microsoft.com/azure/rtos/threadx/.

2.2 FileX RAM Disk sample project

This sample illustrates the use of the FileX embedded FAT file system. The example creates a small RAM-disk with a sample file and data, and reads the file data back into memory. The debugger can show the data being read.

To run this sample, simply follow these steps (assuming the workspace is already open):

1. Open **sample_filex_ram_disk.c** and set a breakpoint around Line 201 at if (status != FX_SUCCESS)
2. Select **Go** to start execution of the sample project
3. In the **Expression** window for e^2 studio or **Watch** window for IAR EW, ensure you watch the **local_buffer** variable as expression.

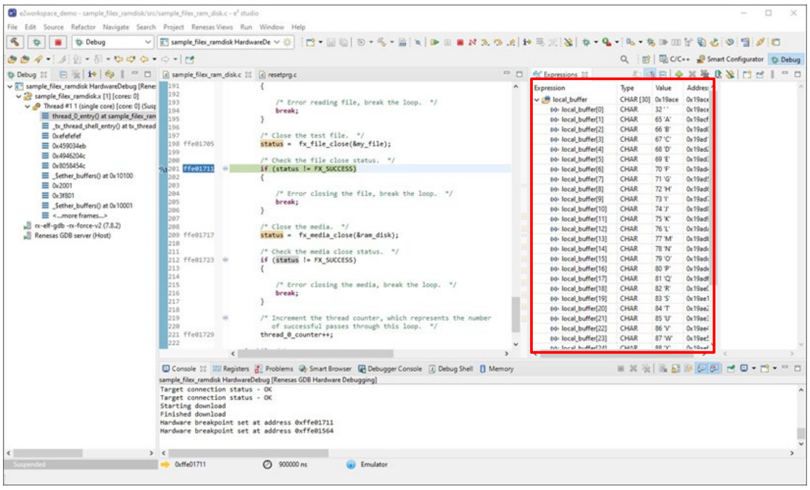


Figure 2.3 e^2 studio Debugger Screen

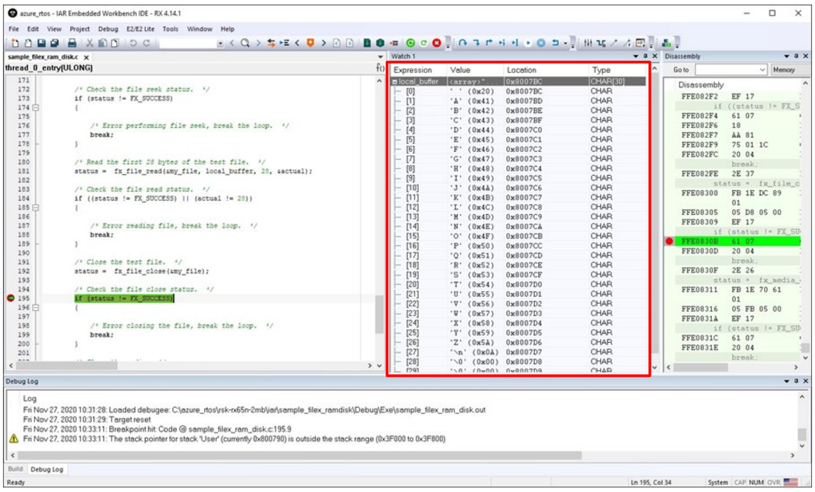


Figure 2.4 IAR EW Debugger Screen

The debugger screen shot above shows the file data read back in the RAM disk sample.

To learn more about Azure RTOS FileX, view https://docs.microsoft.com/azure/rtos/filex/.

2.3 NetX Duo Ping sample project

This sample project illustrates the setup and use of NetX Duo IPv4/IPv6 TCP/IP stack via ping from another node on the local network. By default, this demonstration requests an IP Address via DHCP, and displays the status and assigned IP Address via Terminal program.

To run this sample project, simply follow these steps (assuming the workspace is already open):

1. Verify the serial port in your OS's device manager. It should show up as a COM port

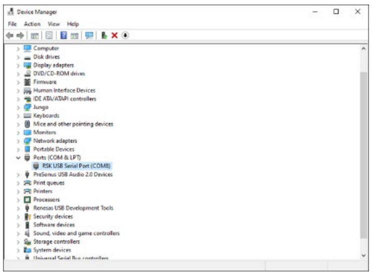


Figure 2.5 Device Manager

2. Open your favorite serial terminal program such as Putty and connect to the COM port discovered above. Configure the following values for the serial port:

   Baud rate: **115200**
   Data bits: **8**
   Parity: **none**
   Stop bits: **1**
   Flow control: **none**

3. Select **Go** to start execution of the sample project

4. As the project runs you should observe the IP address assigned via DHCP in the output window

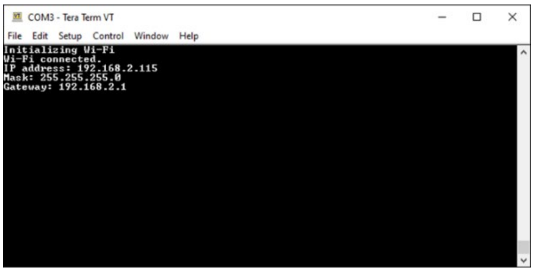


Figure 2.6 IP Address Assigned via DHCP

5. The example above shows that the assigned IP address of the RX MCU is 192.168.2.115. When the demonstration is running it can be pinged by any machine on the network. The following is an example of a ping from a Windows machine on the same local network (using the DOS command window).

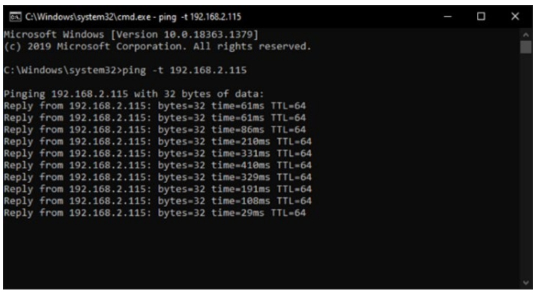


Figure 2.7 Ping Response

To learn more about Azure RTOS NetX Duo, view https://docs.microsoft.com/azure/rtos/netx/.

2.4 NetX Duo Iperf sample project

This demonstration illustrates TCP and UDP network throughput, using NetX Duo IPv4/IPv6 TCP/IP stack, and the industry-standard Iperf network throughput benchmark, with Jperf GUI. By default, this demonstration requests an IP Address via DHCP, and displays the status and assigned IP Address via Terminal program.

To run the NetX Duo Iperf Sample project, simply follow these steps (assuming the workspace is already open):

Note: This sample is Ethernet based and therefore assumes an Ethernet cable is connected to the Ethernet connector on the board.

1. Verify the serial port in your OS's device manager. It should show up as a COM port.

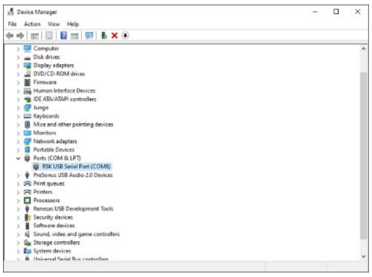


Figure 2.8 Device Manager

2. Open your favorite serial terminal program such as Putty and connect to the COM port discovered above. Configure the following values for the serial port:

   Baud rate: **115200**
   Data bits: **8**
   Parity: **none**
   Stop bits: **1**
   Flow control: **none**

3. Select **Go** to start execution of the sample project.

4. As the project runs you should observe the IP address assigned via DHCP in the output window.

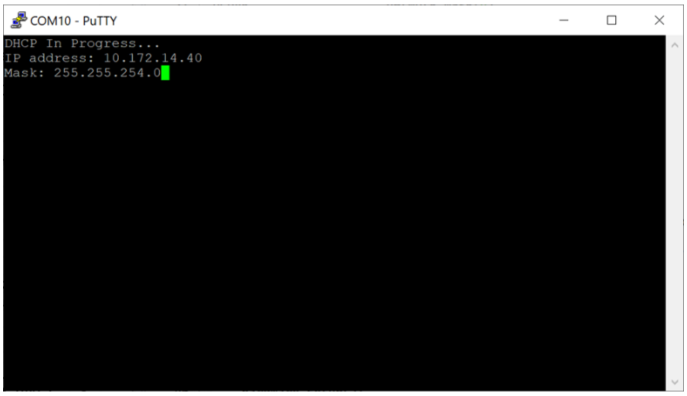


Figure 2.9 IP address assigned via DHCP

5. Once running, simply browse to target IP address (in the screen shot above it is 10.172.14.40) to view the NetX Duo Iperf server page, which provides options for running each Iperf test as well as displays the results of each test. Here is as sample view after browsing 10.172.14.40:

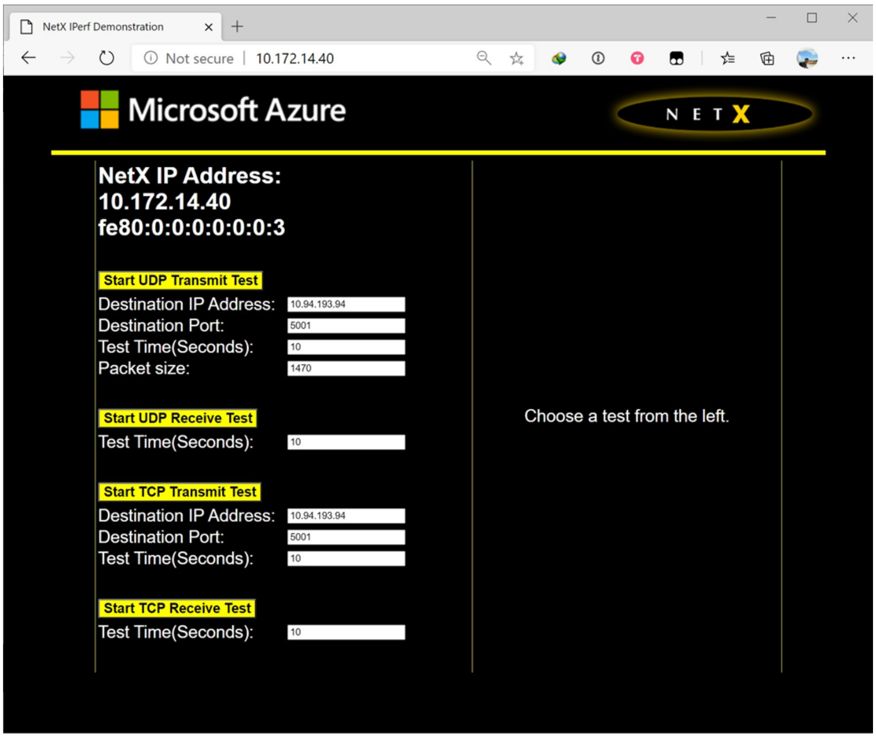


Figure 2.10 NetX Duo Iperf Server Page

Note: Static IP address assignment is also possible by disabling NX_ENABLE_DHCP in the project settings and modifying the default static IP address of 192.168.1.211 in the source file "sample_netx_duo_iperf.c" file.

To learn more about Azure RTOS NetX Duo, view https://docs.microsoft.com/azure/rtos/netx/.

2.5 IoT Embedded SDK sample project

This demonstration connects to Azure IoT Hub using Azure IoT middleware for Azure RTOS. This demonstration also publishes the message to IoT Hub every few seconds.

It is also possible to view device properties, view device telemetry, update device twin, call a direct method on device and send cloud-to-device message using Azure IoT Explorer.

Following videos guide how to set up and run this Azure RTOS sample project in detail.
Azure RTOS Tutorial (1/3) CK-RX65N
Azure RTOS Tutorial (2/3) CK-RX65N: Program Build
Azure RTOS Tutorial (3/3) CK-RX65N: Cloud Operation

1. Prepare Azure resources such as creating an IoT Hub and registering an IoT device by referring Microsoft document.
   For details, please refer to the Application Note (RX65N Group: Visualization of Sensor Data using RX65N Cloud Kit and Azure RTOS).
2. Confirm that you have the copied the following values to use in the next step.

   - **hostname**
   - **deviceID**
   - **primaryKey**
3. Open **sample_config.h** to set the Azure IoT device information constants to the values that you saved in step 2.

Constant name	Value
HOST_NAME	{Your IoT hub hostName value}
DEVICE_ID	{Your deviceID value}
DEVICE_SYMMETRIC_KEY	{Your primaryKey value}

4. Open **main.c** to set the Wi-Fi network parameters when you use the boards of which connectivity is Wi-Fi.

Constant name	Value
WIFI_SSID	{Your Wi-Fi SSID value}
WIFI_PASSWORD	{Your Wi-Fi password}

   You don't need to set specific parameters when you use the boards of which connectivity is ethernet or cellular. Projects with cellular connectivity have "**with EWF**" at the end of the project name on **Select Application Window**.
5. Verify the serial port in your OS's device manager. It should show up as a COM port.

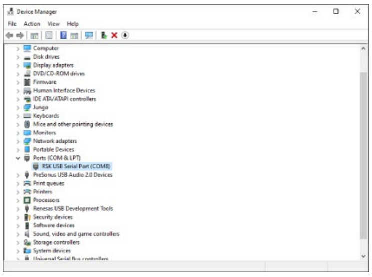


Figure 2.11 Device Manager

6. Open your favorite serial terminal program such as Putty and connect to the COM port discovered above. Configure the following values for the serial port:

   Baud rate: **115200**
   Data bits: **8**
   Parity: **none**
   Stop bits: **1**
   Flow control: **none**

7. Build project

8. Select **Download and Debug** to download and start execution of the project

9. As the project runs, the demo prints out status information to the terminal output window. The demo also publishes the telemetry message to IoT Hub every few seconds. Check the terminal output to verify that messages have been successfully sent to the Azure IoT hub.

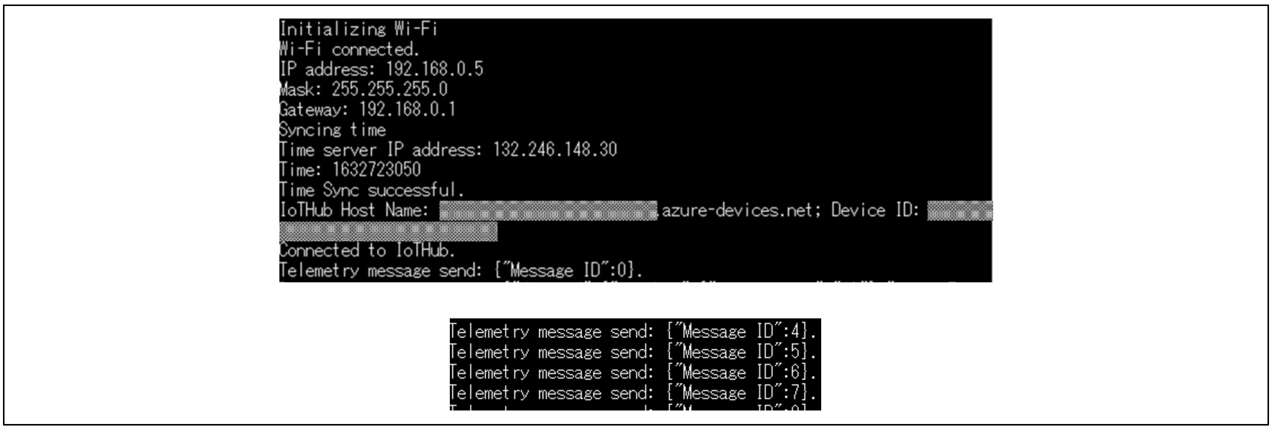


Figure 2.12 Status Information and Telemetry Message

You can use the **Azure IoT Explorer** to view and manage the properties of your devices. In the following steps, you'll add a connection to your IoT hub in IoT Explorer.

1. Download and install latest (above v0.14.5) Azure IoT Explorer from: https://github.com/Azure/azure-iot-explorer/releases

2. Copy the connection string: **[Microsoft Azure Portal](#)** > **sign in** > select your IoT Hub > [**Share access policies**] > [**iothubowner**] > [**Primary connection string**].

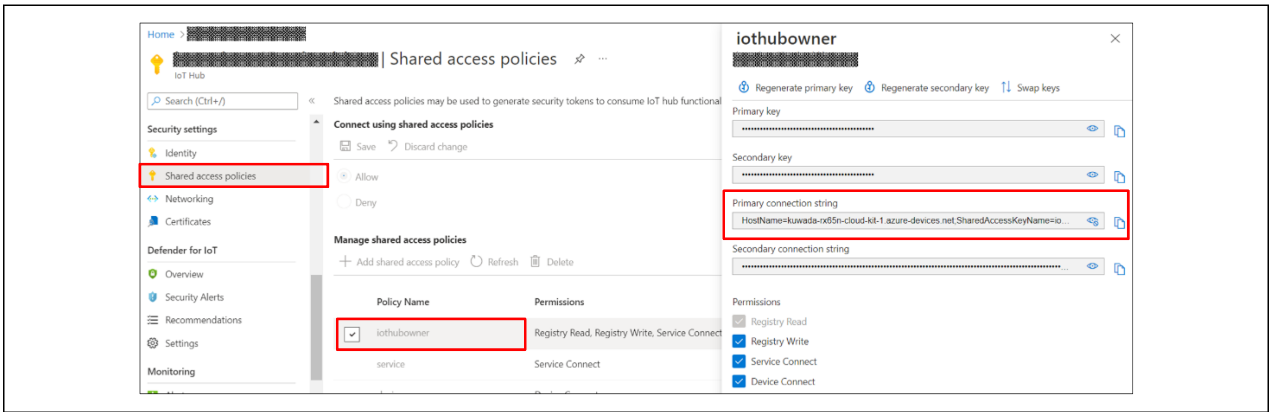


Figure 2.13 Primary Connection String

3. In Azure IoT Explorer, select **IoT hubs** > **Add connection**.

4. Paste the connection string into the **Connection string** box.

5. Select **Save**.

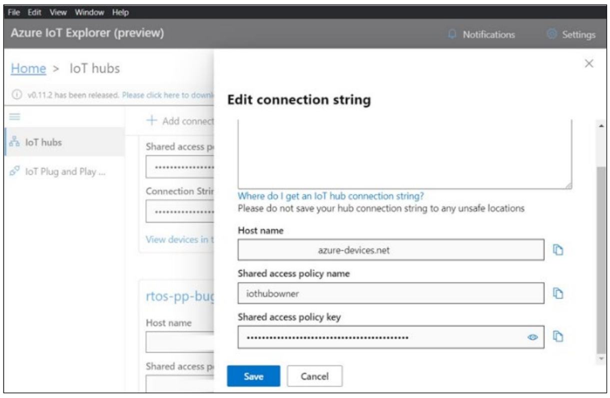


Figure 2.14 Azure IoT Explorer

6. If the connection succeeds, the Azure IoT Explorer switches to a Devices view and lists your device.

To view device properties using Azure IoT Explorer:

1. Select the link for your device identity. IoT Explorer displays details for the device.
2. Inspect the properties for your device in the **Device identity** panel.

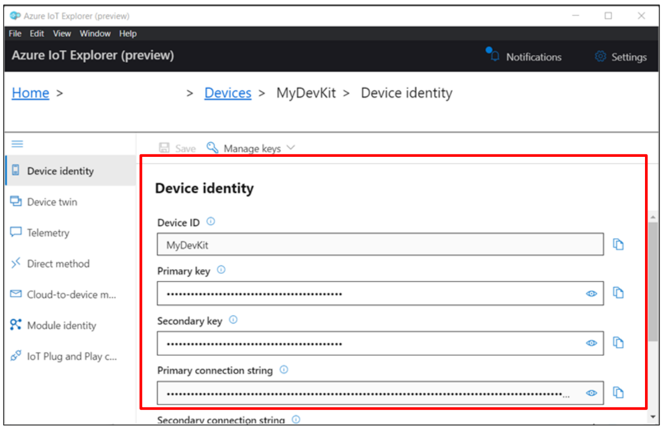


Figure 2.15 Azure IoT Explorer

To view device telemetry using Azure IoT Explorer:

1. In IoT Explorer select **Telemetry**. Confirm that **Use built-in event hub** is set to Yes.
2. Select **Start**.
3. View the telemetry as the device sends messages to the cloud.

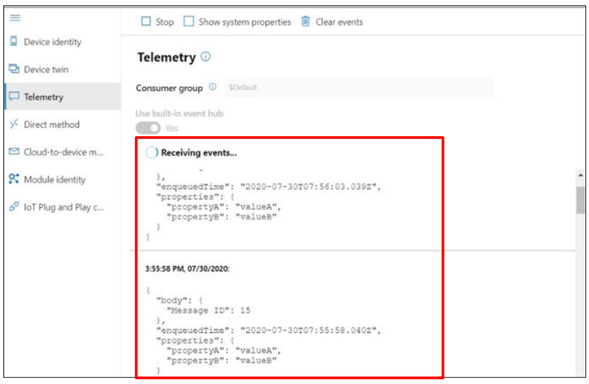


Figure 2.16 Telemetry Message

To update device twin using Azure IoT Explorer:

1. In IoT Explorer select **Device twin**.
2. Modify the **desired** section of the Device twin, you can add a custom twin:

```
"weather": {
        "temperature": "25"
},
```

3. Select **Save**.

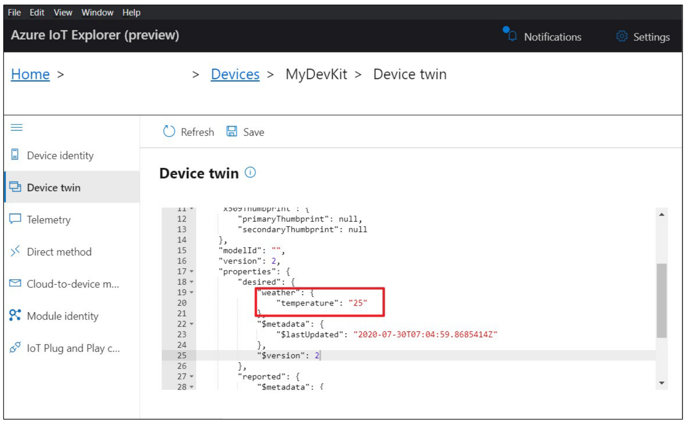


Figure 2.17 Device Twin

4. View the notification for the device twin update status.
5. In the terminal output window, you can view the desired device twin properties are received.

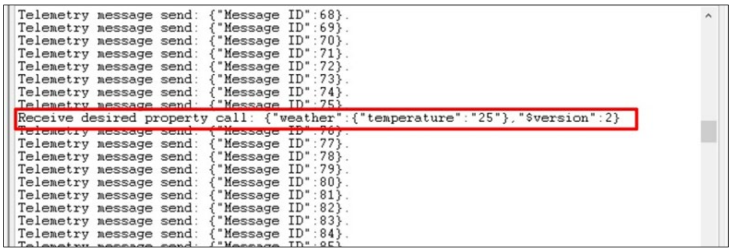


Figure 2.18 Received Desired Device Twin Properties

To call a direct method on device using Azure IoT Explorer:

You can also use Azure IoT Explorer to call a direct method that you have implemented on your device. Direct methods have a name, and can optionally have a JSON payload, configurable connection, and method timeout. To call a direct method in Azure IoT Explorer:

1. In IoT Explorer select **Direct method**.
2. Send a direct method to mimic the device reboot with payload. The device will receive and output the payload as dummy data.

```
- Method name: reboot
- Payload: {"timeout": 500}
```

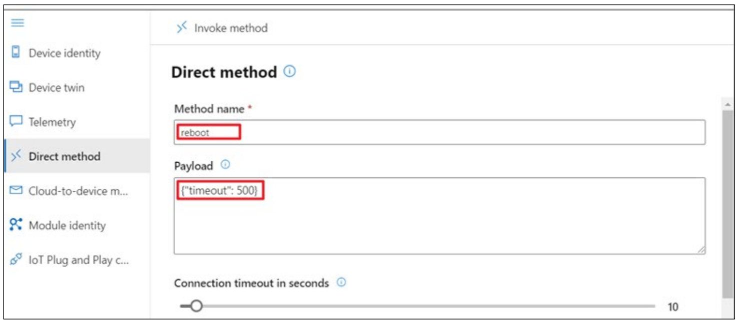


Figure 2.19 Direct Method

3. Select **Invoke method**.
4. In the terminal output window, you can view the method is invoked on the IoT Device.

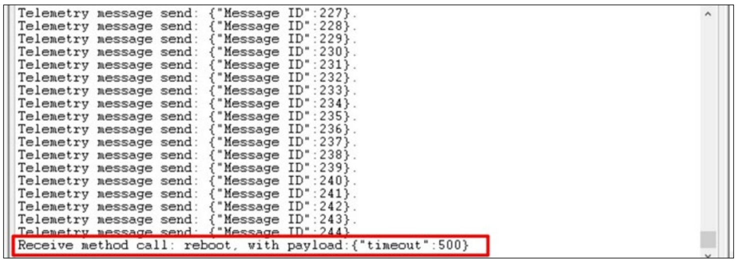


Figure 2.20 Invoked Method

To send cloud-to-device message using Azure IoT Explorer:

1. In IoT Explorer select **Cloud-to-device message**.
2. Enter the message in the Message body:

```
{ "Hello": "Azure RTOS" }
```

3. Check **Add timestamp to message body**.

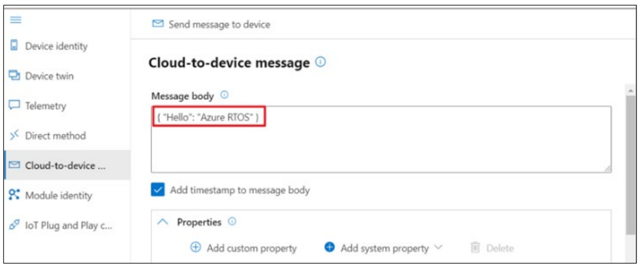


Figure 2.21 Cloud-to-device message

4. Select Send message to device.
5. In the terminal output window, you can view the message is received by the IoT Device.

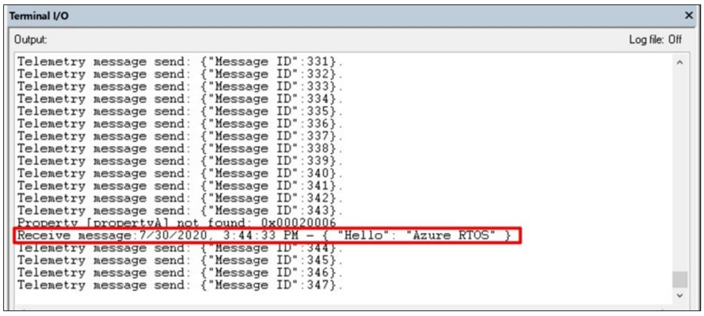


Figure 2.22 Received Message

2.6 IoT Embedded SDK PnP sample project

This demonstration connects to Azure IoT Hub using Azure IoT middleware for Azure RTOS. This demonstration also publishes the message to IoT Hub every few seconds.

It is also possible to view device properties, view device telemetry, update device twin, call a direct method on device and send cloud-to-device message using Azure IoT Explorer.

To run this project, simply follow **2.5 IoT Embedded SDK sample project**.

Moreover, this sample can interact with IoT Plug and Play components using Azure IoT Explorer.

To interact with IoT Plug and Play components using Azure IoT Explorer:

You can use Azure IoT Explorer to interact with IoT Plug and Play components.

Azure IoT explorer needs a local copy of the model file that matches the **Model ID** your device sends. The model file lets Azure IoT explorer display the telemetry, properties, and commands that your device implements.

If you haven't already downloaded the sample model files:

1. Create a folder called **models** on your local machine.
2. Save TemperatureController.json file to the models folder.
3. Save Thermostat.json file to the models folder.

To use the Azure IoT explorer to verify the IoT Plug and Play device application is working:

1. In IoT Explorer, select the **IoT Plug and Play Settings**.
2. Select **Add**.
3. In **Local folder** section and select **Pick a folder** and open the local models folder where you saved your model files. Then select **Save**.

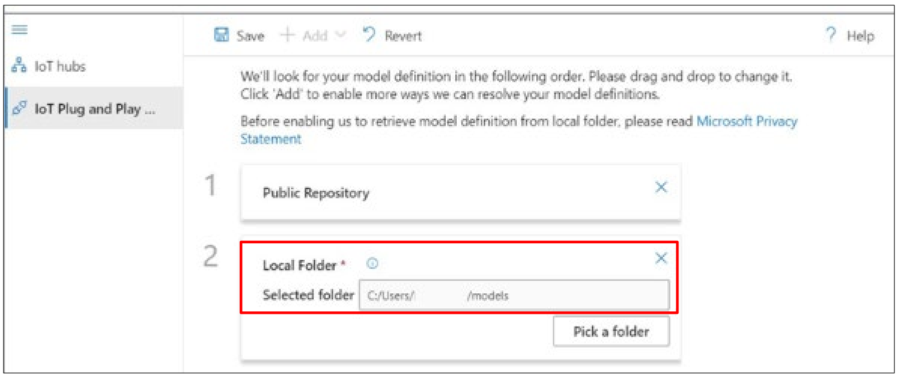


Figure 2.23 IoT Plug and Play Setting

4. On the **IoT hubs** page, click on the name of the hub you want to work with. You see a list of devices registered to the IoT hub.
5. Click on the **Device ID** of the device you created previously.
6. The menu on the left shows the different types of information available for the device.
7. Select **IoT Plug and Play components** to view the model information for your device.

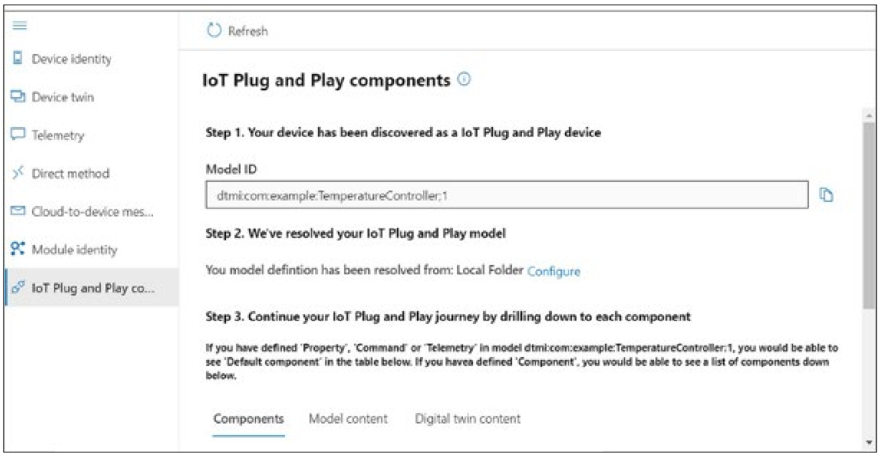


Figure 2.24 Model Information

8. You can view the different components of the device. The default component and any additional ones. Select a component to work with.
9. Select the **Telemetry** page and then select Start to view the telemetry data the device is sending for this component.
10. Select the **Properties (read-only)** page to view the read-only properties reported for this component.
11. Select the **Properties (writable)** page to view the writable properties you can update for this component.
12. Select a property by it's **name**, enter a new value for it, and select **Update desired value**.
13. To see the new value show up select the **Refresh** button.
14. Select the **Commands** page to view all the commands for this component.
15. Select the command you want to test set the parameter if any. Select **Send command** to call the command on the device. You can see your device respond to the command in the command prompt window where the sample code is running.

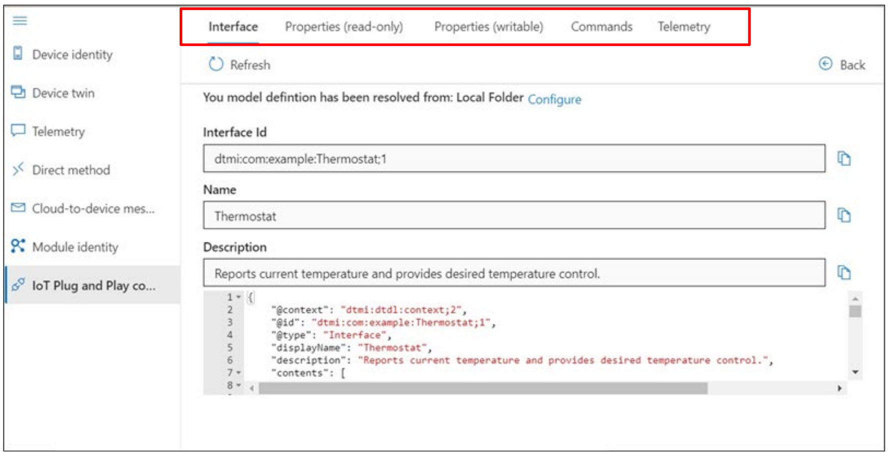


Figure 2.25 IoT Plug and Play Components

2.7 IoT Embedded SDK with IoT Plug and Play sample project

This demonstration connects to Azure IoT Hub using Azure IoT middleware for Azure RTOS. This demonstration also publishes the message to IoT Hub every few seconds.

It is also possible to view device properties, view device telemetry, update device twin, call a direct method on device and send cloud-to-device message using Azure IoT Explorer.

Moreover, this sample can interact with IoT Plug and Play components using Azure IoT Explorer.

To run this project, simply follow **2.6 IoT Embedded SDK PnP sample project**.

2.8 GUIX 8bpp/16bpp/16bpp_draw2d sample project

This demonstration illustrates Washing Machine application using advanced GUIX features such as:

- Widget creation
- Creating multiple screens inside the main screen
- Attaching and detaching the child screen when you switch screens
- Double-buffer toggle control for screen transition without tearing
- Radial slider, vertical and horizontal slider creation
- Running animation

It also illustrates 2 kind of color depth and use of 2D drawing engine (DRW2D) on RX family.

- **sample_guix_8bpp:**
  sample for display of size 480 * 272 with 8 bits color look-up table (CLUT8).
- **sample_guix_16bpp:**
  sample for display of size 480 * 272 with 16 bits RGB 565.
- **sample_guix_16bpp_draw2d:**
  sample for display of size 480 * 272 with 16 bits RGB 565 with 2D drawing engine.

To run each GUIX Sample project, simply follow these steps (assuming the steps described in the previous section were done):

1. Select **Go** to start execution of the demonstration. As the project runs you should observe Washing Machine GUI on board TFT panel. The four different screens are demonstrated as:

Figure 2.26 Main Screen

Figure 2.27 Garments selection screen

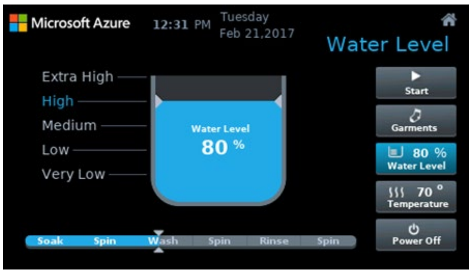


Figure 2.28 Water level selection screen

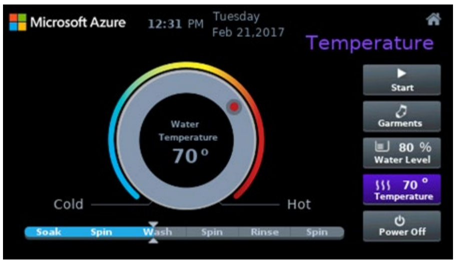


Figure 2.29 Temperature selection screen

The application demonstrates the simulation of the Washing Machine controller from the GUI perspective.

This project initializes the GUIX system, configures the GUIX drivers, initializes Canvas, creates screens using widget creation APIs, starts the GUIX and handles the Touch Events from the Touch driver. All these are done from the Application Thread.

To learn more about Azure RTOS GUIX, view https://docs.microsoft.com/azure/rtos/guix/.

2.9 USBX device CDC-ACM Class sample project

This demonstration illustrates the setup and use of USBX device CDC-ACM Class to communicate with the host as a serial device. This project initializes the USBX system and device stack, set the parameters for callback when insertion/extraction of a CDC device, read from the CDC class and write to the CDC instance using device CDC-ACM APIs.

Before build the sample and run, you need to connect the USB0 Function on Renesas Starter Kit+ for RX65N-2MB to your computer using the USB-MiniB cable: (assuming Renesas Starter Kit+ for RX65N-2MB is specified as Target Board)

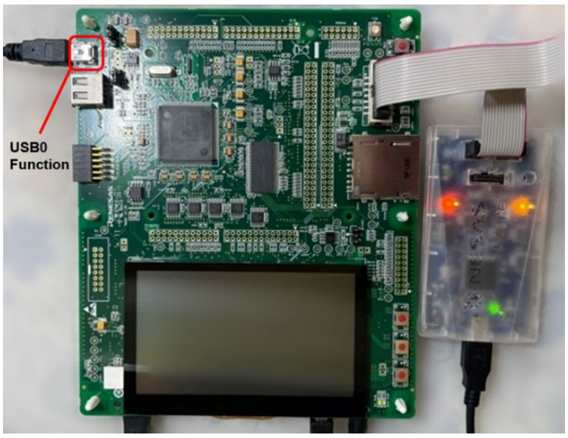


Figure 2.30 USB0 Function on Renesas Starter Kit+ for RX65N-2MB

To run the device CDC-ACM Sample project, simply follow these steps (assuming the steps described in the previous section were done):

1. Select **Go** to start execution of the demonstration.
2. Verify the serial port in your OS's device manager. It should show up as a COM port for the CDC-ACM device.

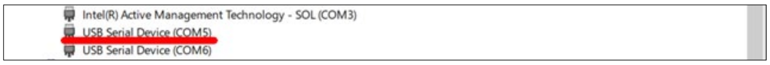


Figure 2.31 Device Manager

3. Open your favorite serial terminal program such as Putty and connect to the COM port discovered above. In this sample project, it is not necessary to set any other settings on the terminal program.
4. As the project runs, you should be able to observe "abcdef" returned from the CDC-ACM device when you input **enter** key to the CDC-ACM device via the terminal.

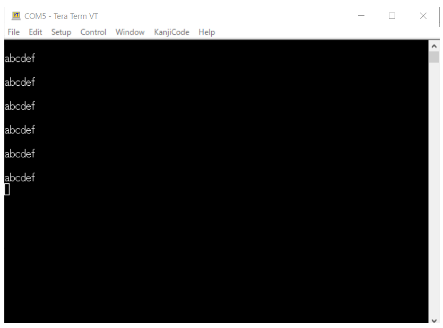


Figure 2.32 Serial Terminal Window

To learn more about Azure RTOS USBX, view https://docs.microsoft.com/azure/rtos/usbx/.

2.10 USBX Host Mass Storage Class sample project

This demonstration illustrates the setup and communication with MSC device (USB flash drive) using USBX HMSC. The sample program initializes the FileX, USBX system and USB driver stack. When a MSC device is inserted, it reads and writes a file to MSC device using device FileX APIs.

1. Change the jumper pins (J7 and J16) on Renesas Start Kit+(RSK) for RX65N-2MB to set to USB Host mode. (assuming Renesas Starter Kit+ for RX65N-2MB is specified as Target Board)

   Note: Jumper pin numbers are different for each RSK.

2. Build USBX HMSC sample project and run.

3. Connect MSC device to USB Standard A connector (red frame) on RSK.

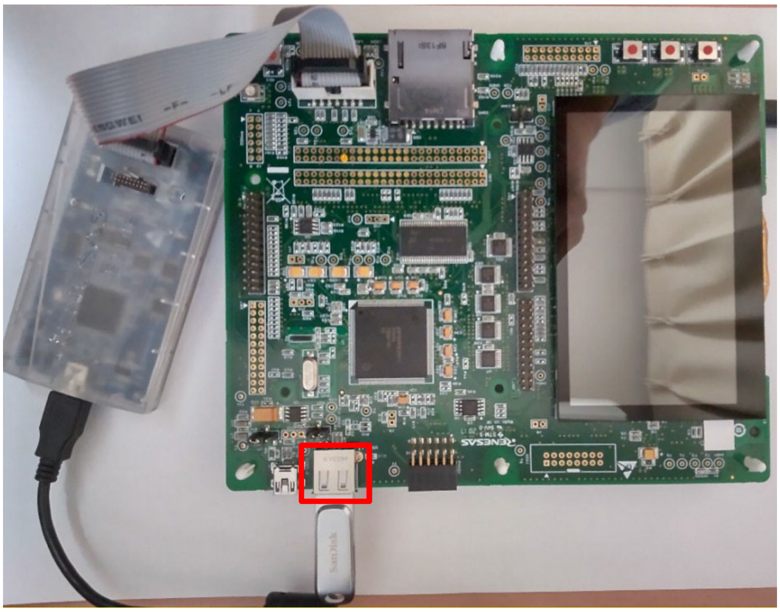


Figure 2.33 USB Standard A Connector on Renesas Starter Kit+ for RX65N-2MB

When the USBX HMSC driver recognizes that MSC device is connected, the sample application program creates a ”counter.txt” file to MSC device using FileX API.

4. Disconnect MSC device from RSK and connect MSC drive to PC.
5. Confirm that “counter.txt” file is generated at the root folder in the MSC device.

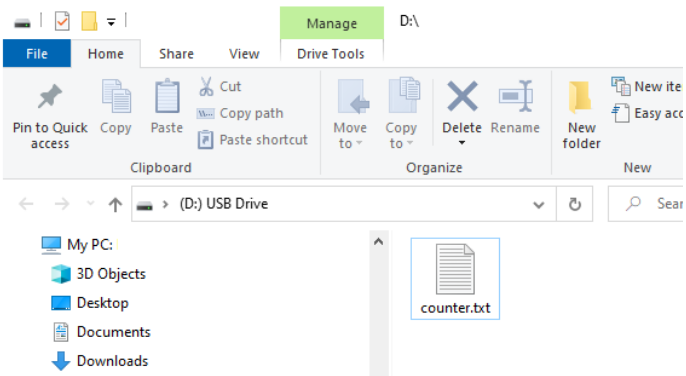


Figure 2.34 Root Folder in MSC Device

6. Open “counter.txt” file using the binary editor on PC. It contains count up numbers from 0x0000 to 0x00FF from the address 0x00000000 as following.

```
ADDRESS   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   0123456789ABCDEF
00000000  00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00   ................
00000010  08 00 09 00 0A 00 0B 00 0C 00 0D 00 0E 00 0F 00   ................
00000020  10 00 11 00 12 00 13 00 14 00 15 00 16 00 17 00   ................
00000030  18 00 19 00 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00   ................
00000040  20 00 21 00 22 00 23 00 24 00 25 00 26 00 27 00    .!.".#.$.%.&.'.
00000050  28 00 29 00 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00   (.).*.+.,.-.../.
00000060  30 00 31 00 32 00 33 00 34 00 35 00 36 00 37 00   0.1.2.3.4.5.6.7.
00000070  38 00 39 00 3A 00 3B 00 3C 00 3D 00 3E 00 3F 00   8.9.:.;.<.=.>.?.
00000080  40 00 41 00 42 00 43 00 44 00 45 00 46 00 47 00   @.A.B.C.D.E.F.G.
00000090  48 00 49 00 4A 00 4B 00 4C 00 4D 00 4E 00 4F 00   H.I.J.K.L.M.N.O.
000000A0  50 00 51 00 52 00 53 00 54 00 55 00 56 00 57 00   P.Q.R.S.T.U.V.W.
000000B0  58 00 59 00 5A 00 5B 00 5C 00 5D 00 5E 00 5F 00   X.Y.Z.[.¥.].^._.
000000C0  60 00 61 00 62 00 63 00 64 00 65 00 66 00 67 00   `.a.b.c.d.e.f.g.
000000D0  68 00 69 00 6A 00 6B 00 6C 00 6D 00 6E 00 6F 00   h.i.j.k.l.m.n.o.
000000E0  70 00 71 00 72 00 73 00 74 00 75 00 76 00 77 00   p.q.r.s.t.u.v.w.
000000F0  78 00 79 00 7A 00 7B 00 7C 00 7D 00 7E 00 7F 00   x.y.z.{.|.}.~...
00000100  80 00 81 00 82 00 83 00 84 00 85 00 86 00 87 00   ................
00000110  88 00 89 00 8A 00 8B 00 8C 00 8D 00 8E 00 8F 00   ................
00000120  90 00 91 00 92 00 93 00 94 00 95 00 96 00 97 00   ................
00000130  98 00 99 00 9A 00 9B 00 9C 00 9D 00 9E 00 9F 00   ................
00000140  A0 00 A1 00 A2 00 A3 00 A4 00 A5 00 A6 00 A7 00   ..｡.｢.｣.､.･.ｦ.ｧ.
```

Figure 2.35 Content of “counter.txt”

7. Disconnect MSC device from PC and connect the MSC device to RSK. This sample program reads “counter.txt” in MSC device and adds the count up data from the address (0x00000200) in this file.
8. Disconnect MSC device from RSK and connect the MSC drive to PC.
9. Open “counter.txt” file using the binary editor on PC. It contains count up numbers from 0x0000 to 0x00FF from the address 0x00000200 as following.

```
ADDRESS   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   0123456789ABCDEF
00000200  00 01 01 01 02 01 03 01 04 01 05 01 06 01 07 01   ................
00000210  08 01 09 01 0A 01 0B 01 0C 01 0D 01 0E 01 0F 01   ................
00000220  10 01 11 01 12 01 13 01 14 01 15 01 16 01 17 01   ................
00000230  18 01 19 01 1A 01 1B 01 1C 01 1D 01 1E 01 1F 01   ................
00000240  20 01 21 01 22 01 23 01 24 01 25 01 26 01 27 01    .!.".#.$.%.&.'.
00000250  28 01 29 01 2A 01 2B 01 2C 01 2D 01 2E 01 2F 01   (.).*.+.,.-.../.
00000260  30 01 31 01 32 01 33 01 34 01 35 01 36 01 37 01   0.1.2.3.4.5.6.7.
00000270  38 01 39 01 3A 01 3B 01 3C 01 3D 01 3E 01 3F 01   8.9.:.;.<.=.>.?.
00000280  40 01 41 01 42 01 43 01 44 01 45 01 46 01 47 01   @.A.B.C.D.E.F.G.
00000290  48 01 49 01 4A 01 4B 01 4C 01 4D 01 4E 01 4F 01   H.I.J.K.L.M.N.O.
000002A0  50 01 51 01 52 01 53 01 54 01 55 01 56 01 57 01   P.Q.R.S.T.U.V.W.
000002B0  58 01 59 01 5A 01 5B 01 5C 01 5D 01 5E 01 5F 01   X.Y.Z.[.¥.].^._.
000002C0  60 01 61 01 62 01 63 01 64 01 65 01 66 01 67 01   `.a.b.c.d.e.f.g.
000002D0  68 01 69 01 6A 01 6B 01 6C 01 6D 01 6E 01 6F 01   h.i.j.k.l.m.n.o.
000002E0  70 01 71 01 72 01 73 01 74 01 75 01 76 01 77 01   p.q.r.s.t.u.v.w.
000002F0  78 01 79 01 7A 01 7B 01 7C 01 7D 01 7E 01 7F 01   x.y.z.{.|.}.~...
00000300  80 01 81 01 82 01 83 01 84 01 85 01 86 01 87 01   ................
00000310  88 01 89 01 8A 01 8B 01 8C 01 8D 01 8E 01 8F 01   ................
00000320  90 01 91 01 92 01 93 01 94 01 95 01 96 01 97 01   ................
00000330  98 01 99 01 9A 01 9B 01 9C 01 9D 01 9E 01 9F 01   ................
00000340  A0 01 A1 01 A2 01 A3 01 A4 01 A5 01 A6 01 A7 01   ..｡.｢.｣.､.･.ｦ.ｧ.
```

Figure 2.36 Content of “counter.txt”

10. By repeating steps 8 and 9 above, the sample program keeps updating count data to “counter.txt” file in the MSC device.

To learn more about Azure RTOS USBX, view https://docs.microsoft.com/azure/rtos/usbx/.

2.11 ThreadX Low Power sample project

This sample project illustrates how to use ThreadX's Low Power feature. You can confirm the transition to and resume from the following low power modes supported by the device using the Low Power Consumption Device Driver Module (r_lpc_rx).

Device	RX130, RX140	RX65N, RX651, RX660, RX72N, RX671
Supported low power mode	Sleep Mode Deep Sleep Mode Software Standby Mode	Sleep Mode Software Standby Mode Deep Software Standby Mode

2.11.1 Overview of sample project

1. The sample project creates one thread **thread_0**. The **thread_0** turns on the LED when it starts.
2. After executing for about 3 seconds, suspend the own thread by **tx_thread_suspend**.
3. Since there is no other thread to run, **Demo_LowPower_Enter** configured in ThreadX “Enter low power function” configuration is called from **tx_low_power_enter** of ThreadX.
4. **Demo_LowPower_Enter** turns off the LED and transitions to the low power consumption mode.
5. The low power consumption mode is resumed by the interruption of pressing the user switch. The interrupt handler **Demo_callback** is called and **tx_thread_resume** resumes **thread_0**. At this point, **thread_0** does not run.
   If it has transitioned to the deep software standby mode, it will be resumed by the user switch press interrupt or RTC alarm interrupt and reboots from the reset vector.
6. Next, the **Demo_LowPower_Exit** configured in the ThreadX “Exit low power function” configuration is called from **tx_low_power_exit** of ThreadX. **Demo_LowPower_Exit** turns on the LED and returns to ThreadX.
7. The resumed **thread_0** runs.
8. Repeat the transition to the same low power consumption mode in steps 2 to 7 three times in total and execute all low power consumption modes in the following order.

   For RX130 and RX140:

   Sleep Mode (3 times) => Deep Sleep Mode (3 times) => Software Standby Mode (3 times)

   For RX65N, RX651, RX660, RX72N, RX671:

   Sleep Mode (3 times) => Software Standby Mode (3 times) => Deep Software Standby Mode (1 time)

The figure shows the execution flow from suspending the thread_0 with tx_thread_suspend to resuming.

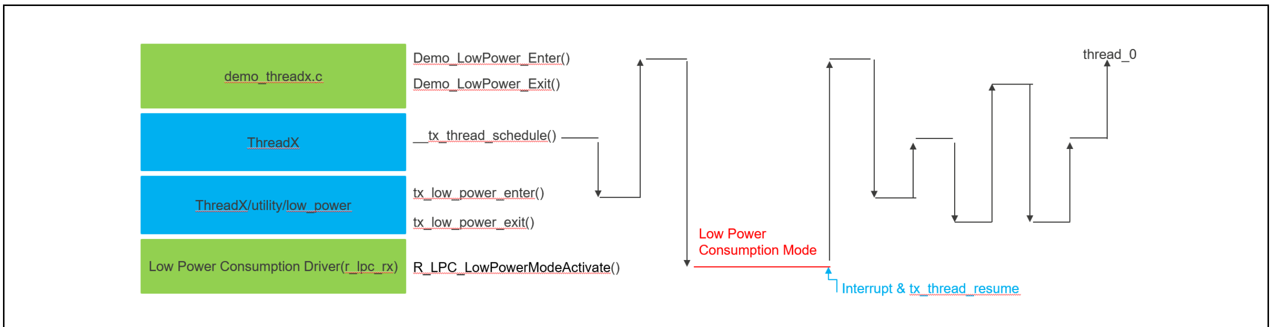


Figure 2.37 Execution Flow after tx_thread_suspend (&thread_0)

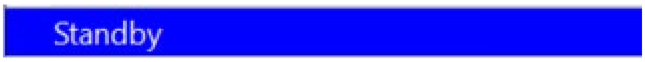
2.11.2 Execute sample project

To run the sample project, simply follow these steps for each board:

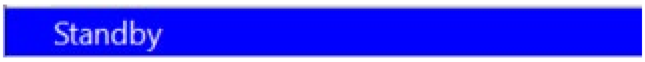
Target Board for RX130 and Renesas Starter Kit for RX140:

1. Select **Launch** to download the program.
2. Select **Resume** to start execution of the project. The program stops at the breakpoint of main function.
3. Select **Resume** to restart.
4. The program turns LED0 on and runs for 3 seconds.
5. The program turns LED0 off and transitions to sleep mode. e^2 studio status bar will change from Running to Sleeping as below:

6. The program is resumed by pressing the user switch (SW1). This cycle is repeated 3 times.
7. Similarly, transitions to deep sleep mode and resume by pressing the user switch is repeated 3 times. e^2 studio status bar will change from Running to Standby as below:



8. Similarly, transitions to software standby mode and resume by pressing the user switch is repeated 3 times. e^2 studio status bar will change from Running to Standby as below:

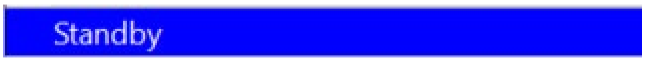


9. Repeat from sleep mode to software standby mode.

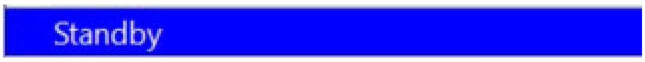
RX65N Cloud Kit:

1. Select **Launch** to download the program.
2. Select **Resume** to start execution of the project. The program stops at the breakpoint of main function.
3. Select **Resume** to restart.
4. The program turns LED1 on and runs for 3 seconds.
5. The program turns LED1 off and transitions to sleep mode. e^2 studio status bar will change from Running to Sleeping as below:

6. The program is resumed by pressing the user switch. This cycle is repeated 3 times.
7. Similarly, transitions to software standby mode and resume by pressing the user switch is repeat 3 times. e^2 studio status bar will change from Running to Standby as below: (*)



8. The program transitions to deep software standby. e^2 studio status bar will change from Running to Standby as below: (*)



9. The program reboots by pressing the user switch.

(*) e2 studio status bar when deep software standby and software standby is the same. So please check SBYCR.SSBY and DPSBYCR.DPSBY register value before executing wait instruction.

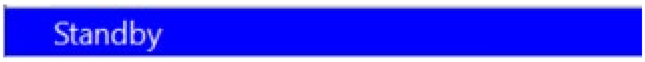
- software standby: SBYCR.SSBY=1, DPSBYCR.DPSBY=0

- deep software standby: SBYCR.SSBY=1, DPSBYCR.DPSBY=1

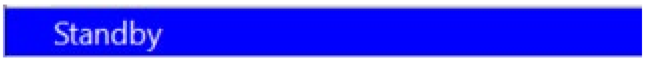
Renesas Starter Kit+ for RX65N-2MB, Renesas Starter Kit for RX660, Renesas Starter Kit for RX671, RX72N Envision Kit and CK-RX65N:

1. Select **Launch** to download the program.
2. Select **Resume** to start execution of the project. The program stops at the breakpoint of main function.
3. Select **Resume** to restart.
4. The program turns LED (usually LED0) on and runs for 3 seconds.
5. The program turns LED off and transitions to sleep mode. e^2 studio status bar will change from Running to Sleeping as below:

6. The program is resumed by pressing the user switch (usually SW1). This cycle is repeated 3 times.
7. Similarly, transitions to software standby mode and resume by pressing the user switch is repeat 3 times. e^2 studio status bar will change from Running to Standby as below: (*)



8. The program transitions to deep software standby. e^2 studio status bar will change from Running to Standby as below: (*)



9. The program reboots by RTC alarm interrupt after about 30 seconds.

(*) e2 studio status bar when deep software standby and software standby is the same. So please check SBYCR.SSBY and DPSBYCR.DPSBY register value before executing wait instruction.

- software standby: SBYCR.SSBY=1, DPSBYCR.DPSBY=0
- deep software standby: SBYCR.SSBY=1, DPSBYCR.DPSBY=1

2.11.3 Configuration of ThreadX Low Power by Smart Configurator

- You can develop own system low power operation for your product referring to this sample project and using Smart Configurator's component configuration feature as below. Each configurable item description is displayed in Macro definition view by clicking the configuration item.

# override byte pool searchs delay	3
# Enable low power mode	☑ Enable
# Enter low power function	Demo_LowPower_Enter()
# Exit low power function	Demo_LowPower_Exit()
# Enable tickless operation in low power mode	☐ Disable
# Enable threadx timer setup	☑ Enable
# Low power timer setup function	Demo_LowPower_Timer_Setup
# Enable threadx user timer adjust	☑ Enable
# Low power user timer adjust function	Demo_LowPower_User_Timer_Adju
# Enable threadx wait	☐ Disable

Macro definition:TX_LOW_POWER
TX_LOW_POWER macro can be used together with TX_ENABLE_WAIT macro
Case 1: TX_LOW_POWER == 1 and TX_ENABLE_WAIT == 0: execute user-defined low power consumption function (call tx_low_power_enter/exit)
Case 2: TX_LOW_POWER == 0 and TX_ENABLE_WAIT == 1: execute only WAIT instruction in ThreadX (tx_low_power_enter/exit are not called)
Case 3: TX_LOW_POWER == 0 and TX_ENABLE_WAIT == 0: no support for low power consumption
Case 4: TX_LOW_POWER == 1 and TX_ENABLE_WAIT == 1: execute tx_low_power_enter, WAIT instruction, and tx_low_power_exit

Figure 2.38 Configuration of ThreadX Low Power

- If the Low Power Consumption Device Driver Module (r_lpc_rx) is used, the module executes "WAIT" instruction inside the r_lpc_rx module. Therefore, please note that "Enable threadx wait" must be disabled.
- If you define your own function for "Enter low power function", "Exit low power function", "Low power timer setup function" and "Low power user timer adjust function", please modify the prototype definition for each function in libs/threadx/tx_user.h manually as well.

```
/* Define Low Power usage */
#define TX_LOW_POWER                                        1

/* Define the Enter low power mode macro*/
#if TX_LOW_POWER
void Demo_LowPower_Enter(void);
#define TX_LOW_POWER_USER_ENTER                             Demo_LowPower_Enter()
#endif

/* Define the Exist low power mode macro*/
#if TX_LOW_POWER
void Demo_LowPower_Exit(void);
#define TX_LOW_POWER_USER_EXIT                              Demo_LowPower_Exit()
#endif

/* Define Low Power tickess usage */
#define USE_TX_LOW_POWER_TICKLESS                           0
#if USE_TX_LOW_POWER_TICKLESS
#define TX_LOW_POWER_TICKLESS
#endif

/* Define the TX_LOW_POWER_TIMER_SETUP macro*/
#if TX_LOW_POWER && !USE_TX_LOW_POWER_TICKLESS
#define USE_TX_LOW_POWER_TIMER_SETUP                        1
#endif
#if USE_TX_LOW_POWER_TIMER_SETUP
void Demo_LowPower_Timer_Setup(unsigned long tx_low_power_next_expiration);  /* can not use ULONG */
#define TX_LOW_POWER_TIMER_SETUP                            Demo_LowPower_Timer_Setup
#endif

/* Define the TX_LOW_POWER_USER_TIMER_ADJUST macro*/
#if TX_LOW_POWER && !USE_TX_LOW_POWER_TICKLESS
#define USE_TX_LOW_POWER_USER_TIMER_ADJUST                  1
#endif
#if USE_TX_LOW_POWER_USER_TIMER_ADJUST
unsigned long Demo_LowPower_User_Timer_Adjust(void);                          /* can not use ULONG */
#define TX_LOW_POWER_USER_TIMER_ADJUST                      Demo_LowPower_User_Timer_Adjust()
#endif

/* Define the TX_ENABLE_WAIT usage*/
#define TX_ENABLE_WAIT                                      0
```

Figure 2.39 libs/threadx/tx_user.h

- The “tx_low_power_next_expiration” parameter is passed to the “TX_LOW_POWER_TIMER_SETUP” function. Since the tx_low_power_next_expiration is the next timer deadline (i.e., the number of ticks before the next wakeup), a low power mode timer must be set so that the low power mode is resumed before this tick number elapses.
  When the tx_low_power_next_expiration is 0xffffffff, there is no next timer expiration date (there is no thread waiting for a timeout), so the user may resume from the low power mode at any time.
  When the tx_low_power_next_expiration is very small value, the transition to the low power consumption mode may be omitted by judging from the transition process time and the resume process time because it depends on the processing time of the user-defined function.
- For the latest information of Low Power APIs, please refer to https://github.com/azure-rtos/threadx/blob/master/utility/low_power/low_power.md .

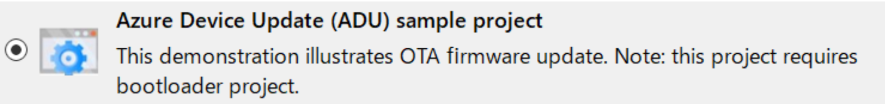
2.12 Azure Device Update (ADU) sample project

This sample project illustrates over-the-air (OTA) firmware update via Microsoft Azure. Azure ADU is a cloud service provided by Microsoft that enables deployment of OTA updating of IoT devices.

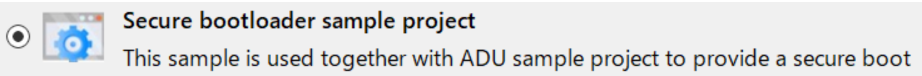
When implementing ADU, secure boot loader sample project must be used together with this project. The secure bootloader function is to verify that firmware to be run is reliable, make sure it has not been tempered, and update it.

To run this sample, simply follow these steps: Please note that this project is not supported by IAR EW.

1. Select **Azure Device Update (ADU) sample project** on **Select Application Window** and create a project.



2. Add new project: [**File**] > [**New**] > [**Renesas C/C++ Project**] and select **Renesas RX**. Then select **Secure bootloader sample project** on **Select Application Window** and create the project specifying the same device and same compiler as specified in step1.



After creating two projects, to setup and build the projects, please refer to [Application Note (Creating a Microsoft ADU Environment)] from "**3.3 File Output Settings**".

Please note that there are some differences in the project structure between the imported projects based on the Application Note and the created projects by e^2 studio.

- Though ThreadX, FileX, NetX Duo will be built as library file using imported project, they will be embedded in **Azure Device Update (ADU) sample project** in created project.
- **"(Board Name)_adu_sample_secure_boot.esi"** written in **"3.9 Section Settings"** does not exist in created project. And the Application Note for imported project assumes that RX65N is used, so the address information may differ on other MCUs. Please refer to the hardware manual of the MCUs used and replace it with the desired value.
- There are some differences in source codes. So please ignore **"3.10 Modifying the Source Code"**.

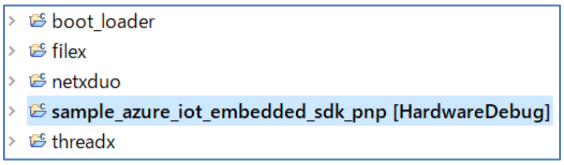


Figure 2.40 imported projects based on Application Note

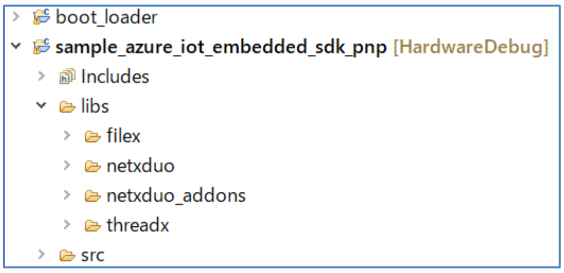


Figure 2.41 created projects by e^2 studio

Where project name is as below
- **Azure Device Update (ADU) sample project**: sample_azure_iot_embedded_sdk_pnp
- **Secure bootloader sample project**: boot_loader

To learn more about Azure ADU, view https://learn.microsoft.com/azure/iot-hub-device-update/.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul. 20, 2022	—	First edition issued
1.01	Oct. 20, 2022	1, 22	Changed project name from “PnP Temperature Control sample project” to “IoT Embedded SDK with IoT Plug and Play sample project”
		2	Added Azure IoT Explorer
1.02	Jan. 20, 2023	6	Improved creation procedure for IAR EW project
		24, 25	Added USBX Host Mass Storage Class sample project
		31	Added Azure Device Update sample project and secure bootloader sample project

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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/.