Using WebSocket Client with RAFW Connectivity Devices

This document describes how to use the WebSocket Client Application on the RA6W1/RA6W2, enabling secure WebSocket communication over Wi-Fi with automatic reconnection and keep-alive support.

Contents

1. Introduction **2**
2. System Overview **2**
2.1 Software Requirements 2
3. WebSocket Client Setup **3**
3.1 Set Up the EK-RA6W1/RA6W2 3
3.2 Program the Firmware Image 3
3.3 Set Up WebSocket Server 3
4. WebSocket Client Application **5**
4.1 Configure WebSocket Parameters 5
4.2 Connect Device to WebSocket Server 5
5. Revision History **10**

Figures

Figure 1. OpenSSL command to generate server certificate 3
Figure 2. Example Python code to run a local server 4
Figure 3. Macro to replace IP address 5
Figure 4. Macro to replace the root CA certificate 5
Figure 5. Setting up the device in Station mode 5
Figure 6. Configuring the SNTP Client 6
Figure 7. Console log showing device connected to the AP and SNTP server 6
Figure 8. J-Link terminal connection parameters 7
Figure 9. J-Link terminal log for WebSocket server connection 8
Figure 10. Console log indicating successful connection to the WebSocket server 8
Figure 11. Messages received on the WebSocket server 9
Figure 12. J-Link logs connection management and auto-reconnect 9
Figure 13. Reconnection to the server shown on the CLI 9

© 2026 Renesas Electronics

Terms and Definitions

EVK	Evaluation Kit
RTOS	Real-Time Operating System
MCU	Microcontroller Unit
AP	Access Point
SSID	Service Set Identifier
UART	Universal Asynchronous Receiver/Transmitter
Wi-Fi®	Wireless Fidelity
WLAN	Wireless Local Area Network

1. Introduction

The Renesas RA6W1/RA6W2 module is a Wi-Fi-enabled MCU designed for wireless connectivity applications. In this application, the RA6W1/RA6W2 operates as a WebSocket client that connects to a WebSocket server over a Wi-Fi network with TLS/SSL encryption. The application supports TLS/SSL encrypted WebSocket connections with certificate verification, WebSocket-level ping/pong frames to detect stale connections and TCP keep-alive with configurable parameters to maintain connection health. The application establishes a persistent WebSocket connection over Wi-Fi, enabling real-time data exchange between the RA6W1/RA6W2 device and a remote WebSocket server.

2. System Overview

The WebSocket client application establishes a persistent, bidirectional communication channel between the RA6W1/RA6W2 device and a remote WebSocket server over Wi-Fi. The application manages the complete lifecycle of secure WebSocket connections, from initial network setup through ongoing data exchange and automatic recovery from connection failures.

The operational flow is as follows:

1. A WebSocket server is set up on a system.
2. By default, the RA6W1/RA6W2 boots in Station (STA) mode and attempts to connect using previously stored Wi-Fi credentials.
3. If no valid credentials are available, or after a factory reset, the device must be provisioned with an easy setup on the CLI.
4. The application optionally synchronizes system time using SNTP for TLS certificate validation.
5. WebSocket client initialization begins, loading configuration parameters and TLS certificates.
6. The client establishes a TCP connection to the configured WebSocket server.
7. An HTTP upgrade request is sent to establish the WebSocket protocol over the secure connection and the WebSocket connection enters the connected state.
8. The application begins periodic message transmission to the server.
9. Incoming messages from the server are received and processed asynchronously.
10. Connection health is maintained through TCP keep-alive and WebSocket ping/pong mechanisms.

2.1 Software Requirements

The following software tools and packages are required to complete the project:

- Renesas Electronics Flexible Software Package (FSP): Version 2.2.0 or later.
- e^2 studio IDE: Version 2026-04.3 or later.
- GCC Arm Embedded Toolchain: Version 13.3.1.arm-13-24.
- Serial Terminal Application: Tera Term or similar tools for logging and monitoring output.

3. WebSocket Client Setup

Before running the WebSocket Client application, install the required software, configure the evaluation kit, and set up the WebSocket Server. See the following sections for instructions.

3.1 Set Up the EK-RA6W1/RA6W2

An RA6W1/RA6W2 Evaluation Kit (EK-RA6W1/EK-RA6W2) is required to run the WebSocket Client Application. Verify the following:

- A Wi-Fi Access Point (AP) is available with network connectivity
- Both the device and the host PC are on the same network (or routable)
- Network connectivity allows the device to reach the WebSocket server (port 8765 or a custom port)

3.2 Program the Firmware Image

In this Application, the WebSocket client image is used. The application is located in the directory: `application_projects/r19an0512/WebSocket_Client_Application/`. The firmware image is generated and flashed onto the EK-RA6W1/EK-RA6W2 using e^2 studio with J-Link support. e^2 studio provides an integrated build and flash environment, eliminating the requirement for external flash programming scripts.

3.3 Set Up WebSocket Server

A WebSocket server is required for testing the application. Install the `openssh` on the host PC and use it to generate certificates with the following command:

```
openssl req -x509 -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -keyout key.pem -
out cert.pem -days 3650 -nodes -subj "/CN=<ip_address>"
```

Figure 1. OpenSSL command to generate server certificate

The certificates are generated in the same directory where the command runs. Using a python script, run the WebSocket server on port 8765 using the generated certificates. The following is an example of the script used to run the python server:

```
import asyncio

import ssl
import websockets

# Create SSL context
ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
ssl_context.load_cert_chain(

    certfile="<full_path_to_file>cert.pem",
    keyfile=" <full_path_to_file> key.pem"

)

async def echo(websocket):
    async for message in websocket:
        print(f"Received: {message}")
        await websocket.send(f"Echo: {message}")

async def main():

    async with websockets.serve(

        echo,
        "0.0.0.0",
        8765,
        ssl=ssl_context   # changed context
    ):

        print("WebSocket server running on wss://localhost:8765")
        await asyncio.Future()  # run forever
 asyncio.run(main())
```

Figure 2. Example Python code to run a local server

Ensure the system and device on which the server runs are on the same network for successful connection.

4. WebSocket Client Application

This section describes how to get the WebSocket Client application running on the RA6W1/RA6W2 device.

4.1 Configure WebSocket Parameters

Before building and flashing, update the WebSocket parameters in `websocket_client_app.c` to match your setup:

1. Server URI:

   Replace the IP address with that of the system running the websocket server.

```
#define WS_APP_SERVER_URI "wss://<ip_address>:8765"  /* Secure WebSocket (TLS) */
```

Figure 3. Macro to replace IP address

2. Root CA Certificate:

   Replace the certificate with the contents of `cert.pem` generated from the previously provided `open-ssh` command, and format is as follows:

```
#define WS_APP_ROOT_CA \
"-----BEGIN CERTIFICATE-----\n" \
"MIIBWzCCAQKgAwIBAgIUQai7wFktzIp869JxLI11+XwqV/4wCgYIKoZIzj0EAwIw\n" \
"FDESMBAGA1UEAwwJTXlNUVRUX0NBMB4XDTI2MDQyODEyNTE1NFoXDTM2MDQyNTEy\n" \
"NTE1NFowFDESMBAGA1UEAwwJTXlNUVRUX0NBMFkwEwYHKoZIzj0CAQYIKoZIzj0D\n" \
"AQcDQgAESqzC8ZkZHYl/wUeHCzLKnWF6iShILwlw05FTUHSDL5MFVaIIUa4Z0hE6\n" \
"pYiE4iyQBPRLRuCu68evbwLy/OWL+KMyMDAwHQYDVR0OBBYEFP8EjwaIzTWCzyae\n" \
"M1TRMsfq9w52MA8GA1UdEwEB/wQFMAMBAf8wCgYIKoZIzj0EAwIDRwAwRAIgdpqX\n" \
"2jN5djuohhPRYsYOhLApdJ+KN7KMJj2D7XzMUdsCIFHcmwMk3VSCcaXJWQmK00/X\n" \
"56Cb7r88xVaIATL/RUNC\n" \
"-----END CERTIFICATE-----\n"
```

Figure 4. Macro to replace the root CA certificate

3. Generate the project, build, and flash the firmware using e^2 studio.

4.2 Connect Device to WebSocket Server

1. When prompted by Easy Setup, configure the device in Station mode and connect to an AP. The AP must be the same one connected to the system running the server.

```
COUNTRY CODE ? [Quit] (Default KR, ZZ : for debug) :
Default Country: KR

SYSMODE(WLAN MODE) ?
        1. Station
        2. Soft-AP
        3. WiFi Direct
        4. WiFi Direct P2P GO Fixed
        5. Station & Soft-AP
MODE ? [1/2/3/4/5/Quit] (Default Station) : 1

BAND ?
        1. 2.4 GHz
        2. 5 GHz
        3. AUTO
BAND ? [1/2/3/Quit] (Default 3. AUTO) : 3
```

Figure 5. Setting up the device in Station mode

2. Enable the SNTP client when prompted. The default values are enough for this application to work



Figure 6. Configuring the SNTP Client

3. The device connects first to the AP and next to the SNTP server.



Figure 7. Console log showing device connected to the AP and SNTP server

4. Connect to the J-Link RTT viewer with the following parameters:

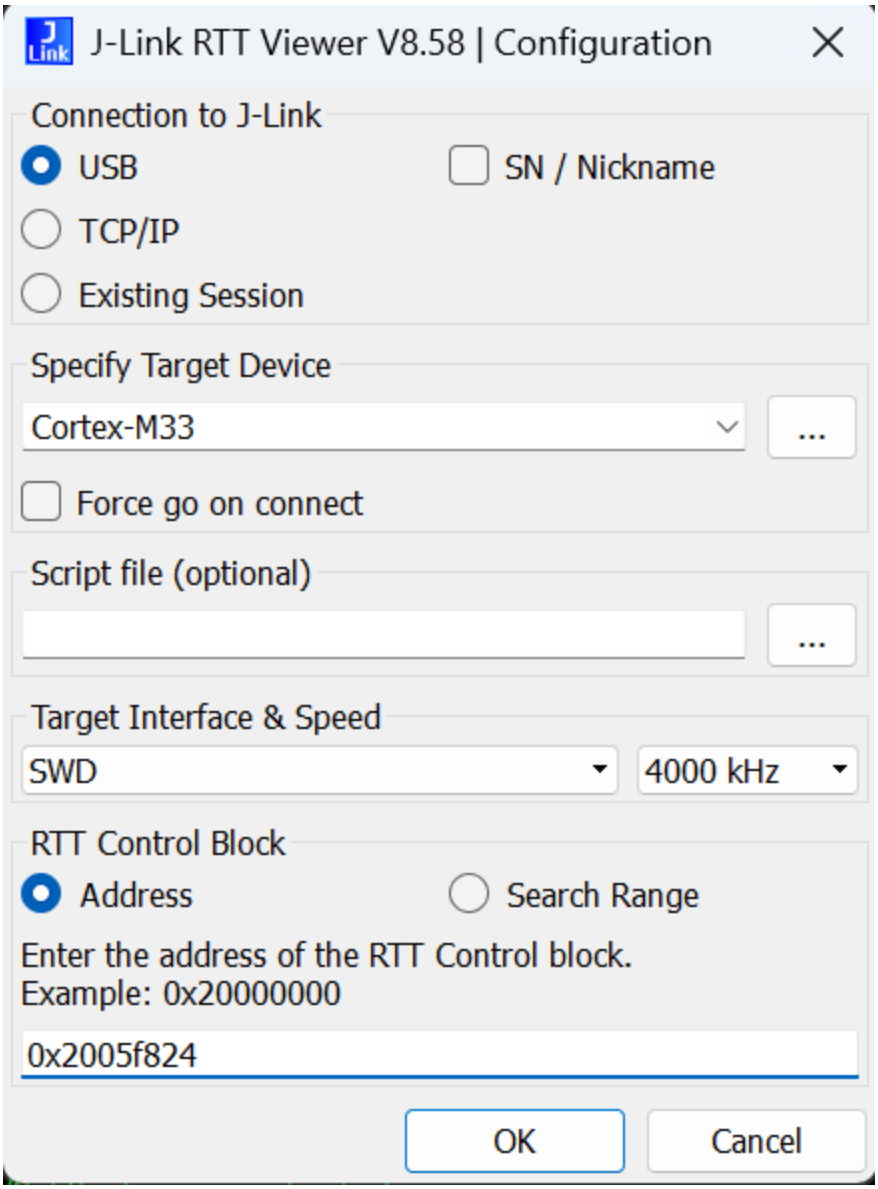


Figure 8. J-Link terminal connection parameters

5. After the SNTP connection is successful, the device attempts to connect to the WebSocket server automatically. This can be seen on the J-Link terminal (see Figure 9).

```
[INFO] [WS_APP] Waiting for WiFi network to be ready...
[INFO] [WS_APP] WiFi interface ready, waiting for network to stabilize...
[INFO] [WS_APP] WiFi network is ready!
[INFO] [WS_APP] Time already synced via SNTP.
[INFO] [WS_APP] Starting WebSocket client app...
[INFO] [WS_APP] WebSocket client started, connecting to wss://10.54.153.175:8765
[INFO] [WS_APP] Waiting for connection...
[INFO] [WS_APP] Connected to server
[INFO] [WS_APP] Sent 19 bytes: Hello from RA6W1 #0
[INFO] [WS_APP] Received 25 bytes (opcode=0x01): [INFO] Echo: Hello from RA6W1 #0[INFO]
[INFO] [WS_APP] Sent 19 bytes: Hello from RA6W1 #1
[INFO] [WS_APP] Received 25 bytes (opcode=0x01): [INFO] Echo: Hello from RA6W1 #1[INFO]
[INFO] [WS_APP] Sent 19 bytes: Hello from RA6W1 #2
[INFO] [WS_APP] Received 25 bytes (opcode=0x01): [INFO] Echo: Hello from RA6W1 #2[INFO]
```

Figure 9. J-Link terminal log for WebSocket server connection

When the WebSocket connection is successful, the CLI reflects the same by displaying the WebSocket key on screen.

```
[/RA6W1-RRQ61001] #
Connection COMPLETE to 56:e3:c9:d0:e1:e2
        Assigned addr   : 10.54.153.54
              netmask   : 255.255.255.0
              gateway   : 10.54.153.19
        DHCP Server IP  : 10.54.153.19
        Lease Time      : 00h 59m 59s
        Renewal Time    : 00h 29m 59s

>>> SNTP Server: pool.ntp.org (162.159.200.123)

>>> SNTP Time sync : 2026-05-08 12:37:12
Write upgrade request
GET / HTTP/1.1
Connection: Upgrade
Host: 10.54.153.175:8765
User-Agent: Websocket Client
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: hpbsrcRAktyWQdh4wD56qw==


server key=3JGYNMWPPpUBvK1htFofS9K/T+0=, send_key=hpbsrcRAktyWQdh4wD56qw==, expected_server_key=3JGYN
MWPPpUBvK1htFofS9K/T+0=
```

Figure 10. Console log indicating successful connection to the WebSocket server

The device starts sending packets to the WebSocket server. This process is displayed in the command prompt terminal that is running the server, which indicates that the setup is successful.

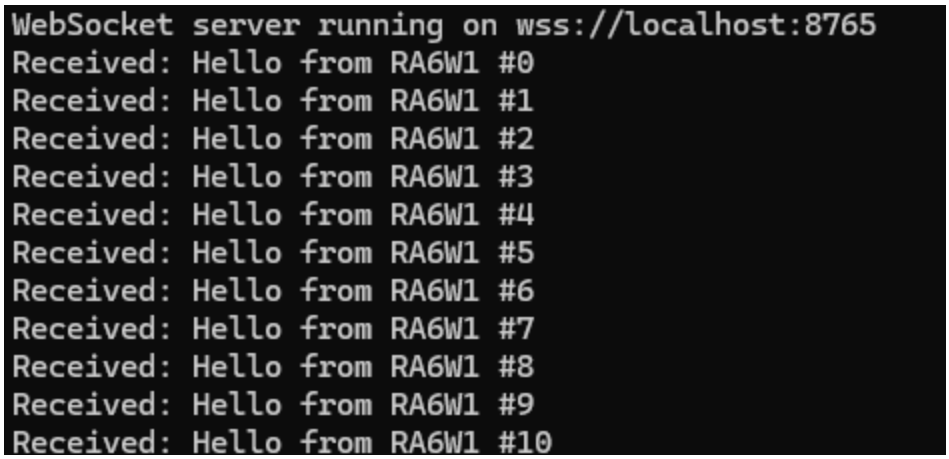


Figure 11. Messages received on the WebSocket server

The same logs can be seen on the J-Link terminal (see Figure 12).

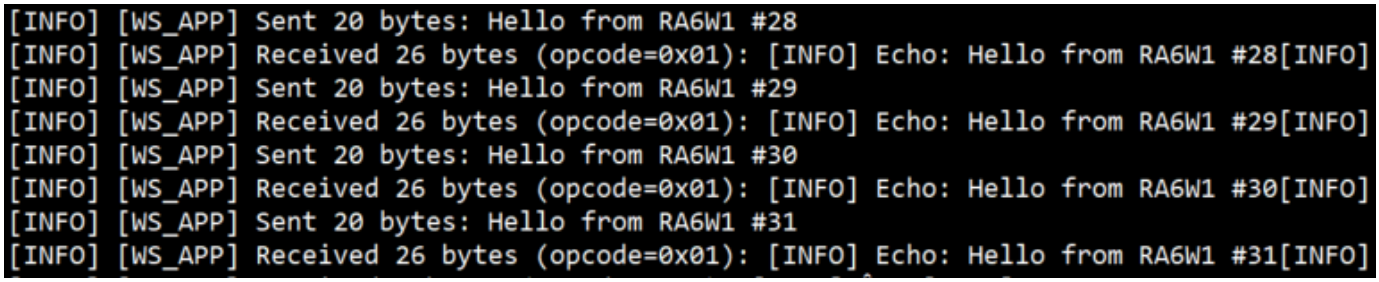


Figure 12. J-Link logs connection management and auto-reconnect

If the device is disconnected from the server, the device attempts to reconnect to the server automatically. The logs are displayed on the CLI and the J-Link terminal.

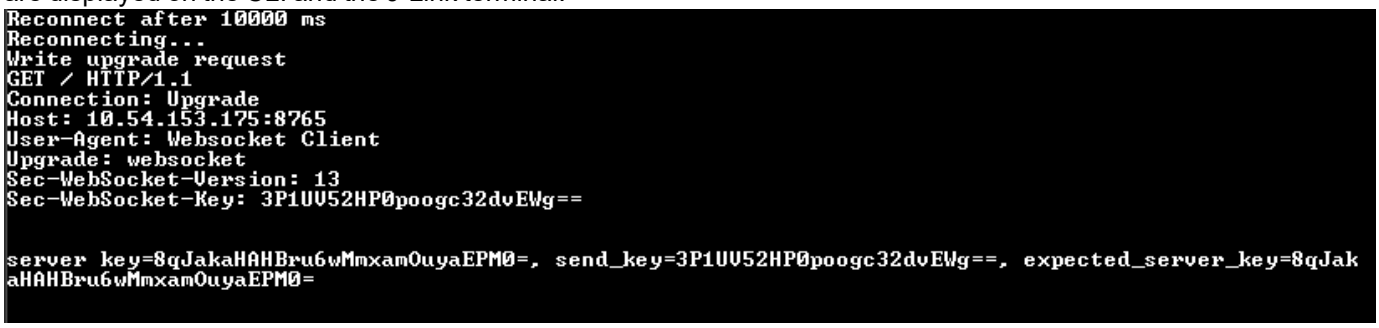


Figure 13. Reconnection to the server shown on the CLI

5. Revision History

Revision	Date	Description
1.02	Sep 21, 2026	Updated the Software requirements section.
1.01	July 21, 2026	Added Software Requirements.
1.00	May 20, 2026	First version.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

CorporateHeadquarters

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-us/

© 2026 Renesas Electronics Corporation. All rights reserved.