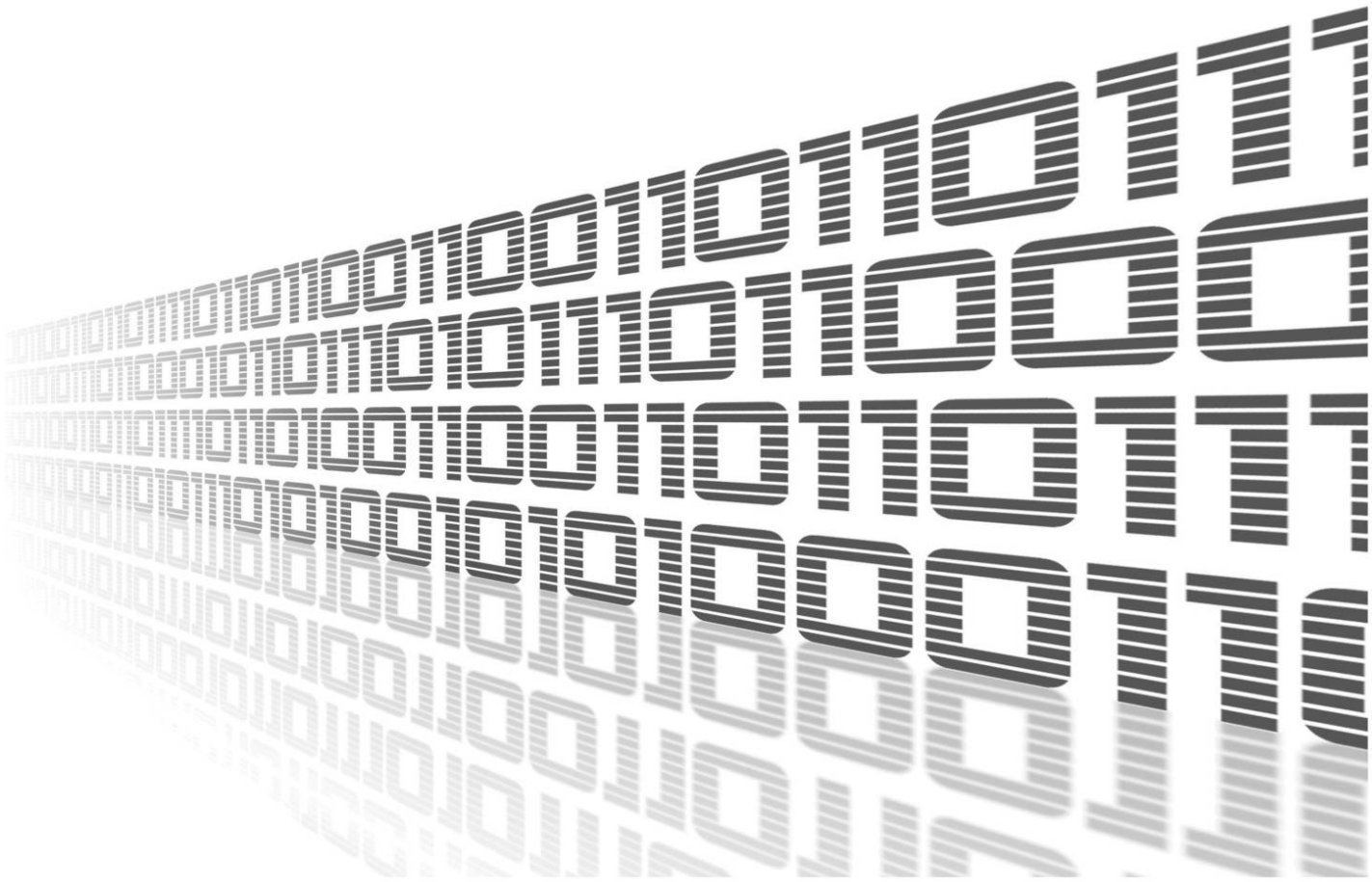
ADVANTECH

RouterApp

MQTT Broker



Advantech Czech s.r.o., Sokolska 71, 562 04 Usti nad Orlici, Czech Republic
Document No. APP-0125-EN, revised on May 13, 2025.

Used symbols

Danger – Information regarding user safety or potential damage to the router.

Attention – Problems that can arise in specific situations.

Information – Useful tips or information of special interest.

Contents

List of Figures

List of Tables

1. MQTT Introduction

1.1 What is MQTT and MQTT Broker

MQTT (Message Queuing Telemetry Transport) is a lightweight **messaging protocol** designed for efficient communication between devices in IoT (Internet of Things) and mobile environments. It enables devices to publish messages to topics and subscribe to receive messages on those topics, facilitating real-time, reliable, and scalable data exchange with minimal bandwidth and power consumption requirements.

1.2 What does MQTT Broker do?

MQTT Broker is a server or middleware that acts as a central hub for facilitating communication between devices and applications in an MQTT-based Internet of Things (IoT) architecture.

1. **Message Routing**: It routes messages between publishers (devices or applications that send data) and subscribers (devices or applications that receive data). When a device publishes a message to a specific topic, the MQTT Broker ensures that the message is delivered to all subscribers interested in that topic.
2. **Protocol Translation**: It handles the MQTT protocol, allowing devices and applications using MQTT to communicate with each other regardless of the underlying network protocols they use. This enables interoperability among various IoT devices and applications.
3. **QoS (Quality of Service) Management**: MQTT supports different levels of QoS for message delivery—QoS 0 (At most once), QoS 1 (At least once), and QoS 2 (Exactly once). The MQTT Broker manages the delivery of messages based on the QoS level specified by publishers and subscribers, ensuring reliable and efficient message delivery according to the desired level of reliability.
4. **Session Management**: It manages client sessions, ensuring that clients remain connected and handling client reconnects seamlessly. This is essential for maintaining continuous communication between devices and applications, especially in unreliable network conditions.
5. **Security**: MQTT Brokers often provide security features such as authentication, access control, and encryption to protect the data exchanged between devices and applications. This ensures that only authorized clients can publish or subscribe to specific topics and that data remains confidential and secure during transmission.

Overall, an MQTT Broker plays a crucial role in enabling scalable, efficient, and secure communication in IoT ecosystems by facilitating the exchange of data between connected devices and applications.

1.3 What does MQTT Bridge do?

An **MQTT bridge** allows two MQTT brokers to be connected together. Messages published on specified topics on one broker can be forwarded to the other broker, and vice versa. This is useful for linking separate MQTT networks, aggregating data from multiple sites, or creating hierarchical MQTT architectures. The direction (in, out, or both) and the topic patterns for message forwarding are configurable for each bridge connection.

2. Router App Description

2.1 Router App Introduction

Router app *MQTT Broker* is not included in the standard router firmware. Uploading of this router app is described in the Configuration manual.

The *MQTT Broker* Router App extends the functionality of Advantech routers by providing an integrated MQTT broker service based on the popular open-source Eclipse Mosquitto™ project. This allows the router itself to act as a central communication hub for MQTT-enabled devices on the local network, facilitating IoT messaging without requiring an external broker server. Key features include support for standard MQTT operations, TLS encryption (PSK and certificate-based), client authentication (username/password and certificate CN), and MQTT bridging to connect with other brokers.

2.2 Web Interface

After installing the Router App, the GUI can be accessed by clicking the Router App name on the *Router Apps* page of your router's web interface.

The left panel of the GUI contains a menu with the following sections: *Information* → *Status*, *Configuration* → *Broker*, *Configuration* → *Bridge 1–3*, and *Administration* → *Return*, see Figure 1.

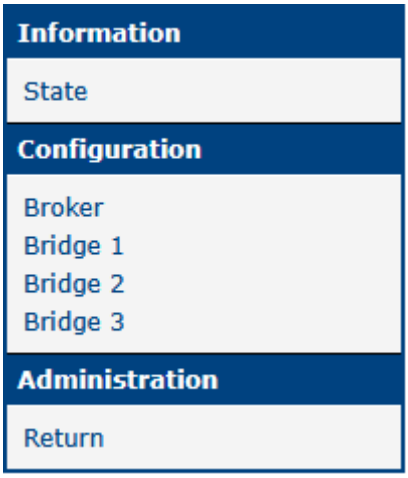


Figure 1: Router App Main Menu

Key functionality:

- The *Information* → *State* page displays current MQTT service state.
- *Configuration* → *Broker* and *Configuration* → *Bridge 1–3* pages contain settings described in subsequent sections.
- *Administration* → *Return* exits the Router App interface and returns to the main Router Apps page.

2.3 Broker Configuration

By clicking the *Broker* menu item, you can configure the broker, configuration options are displayed on Figure 2 and all items are described in Table 1.

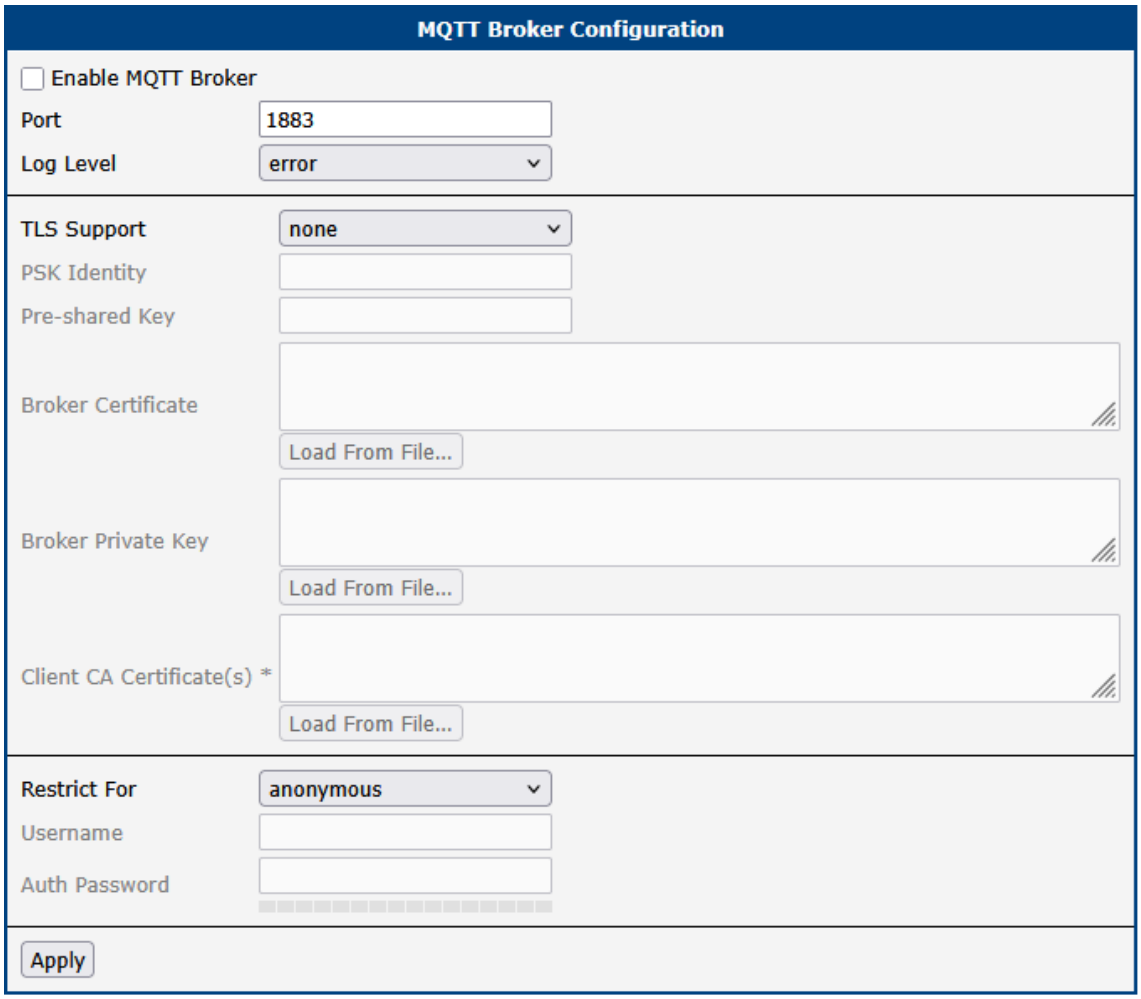


Figure 2: Broker Configuration

Item	Description
Enable MQTT Broker	Enables the MQTT Broker functionality on the router.
Port	Enter the TCP port number where the MQTT Broker will listen for incoming client connections (default is typically 1883 for unencrypted MQTT and 8883 for encrypted MQTT/TLS).
Log Level	Select the level of detail for log information generated by the broker (e.g., None, Error, Warning, Information, Debug).
TLS Support	Options for Transport Layer Security (TLS) encryption are: • **none** – No TLS encryption is used. Communication is unencrypted. • **PSK based** – TLS encryption using a Pre-Shared Key. Requires *PSK Identity* and *Pre-shared Key* fields below. • **certificate based** – TLS encryption using X.509 certificates. Requires *Broker Certificate* and *Broker Private Key* fields below. See Chapter 2.5.1 for more details.

Continued on the next page

Continued from previous page

Item	Description
PSK Identity	When *PSK based* TLS Support is selected, enter the identity string associated with the pre-shared key. This identity is sent by the client to identify which key to use.
Pre-shared Key	When *PSK based* TLS Support is selected, enter the Pre-shared Key as a hexadecimal string. This key must be securely shared with clients.
Broker Certificate	When *certificate based* TLS Support is selected, paste the Broker's public certificate here in PEM format. This certificate is presented to clients to verify the broker's identity.
Broker Private Key	When *certificate based* TLS Support is selected, paste the Broker's corresponding private key here in PEM format. This key must be kept secret.
Client CA Certificate(s)	When *certificate based* TLS Support is selected, optionally paste the Certificate Authority (CA) certificate(s) here in PEM format. If they are specified, the client is forced to provide its certificate and they are used to verify its certificate.
Resctrict For	Options for client authentication are: • **anonymous** – No authentization, no access control. Anybody can connect, pusblish, subscribe. • **username**/**password** – Connecting is limited to user entered in *Username* field and he must authenticate with password entered in *Auth Password* field. • **certificate CN** – Publishing and subscribing is limited to user entered in *Username* filed, client provides username as `subject CN` in it's certificate. Requires *certificate based* TLS support with *Client CA Certificate(s)* to be configured.
Username	Enter the required username to restrict operation to. This field is used for both *username/password* and *certificate CN* options.
Auth Password	When the *username/password* option is selected, enter the required authentication password here.

Table 1: Broker Configuration Items Description

2.3.1 Broker Configuration Example

This example shows how to configure the local MQTT Broker to run on the standard unencrypted port (1883) and require simple username/password authentication for connecting clients.

Scenario:

- Enable the broker service by ticking *Enable MQTT Broker*.
- Listen on TCP port 1883.
- No TLS encryption required.
- Require clients to authenticate with username "advantech" and password "password123".

GUI Configuration:

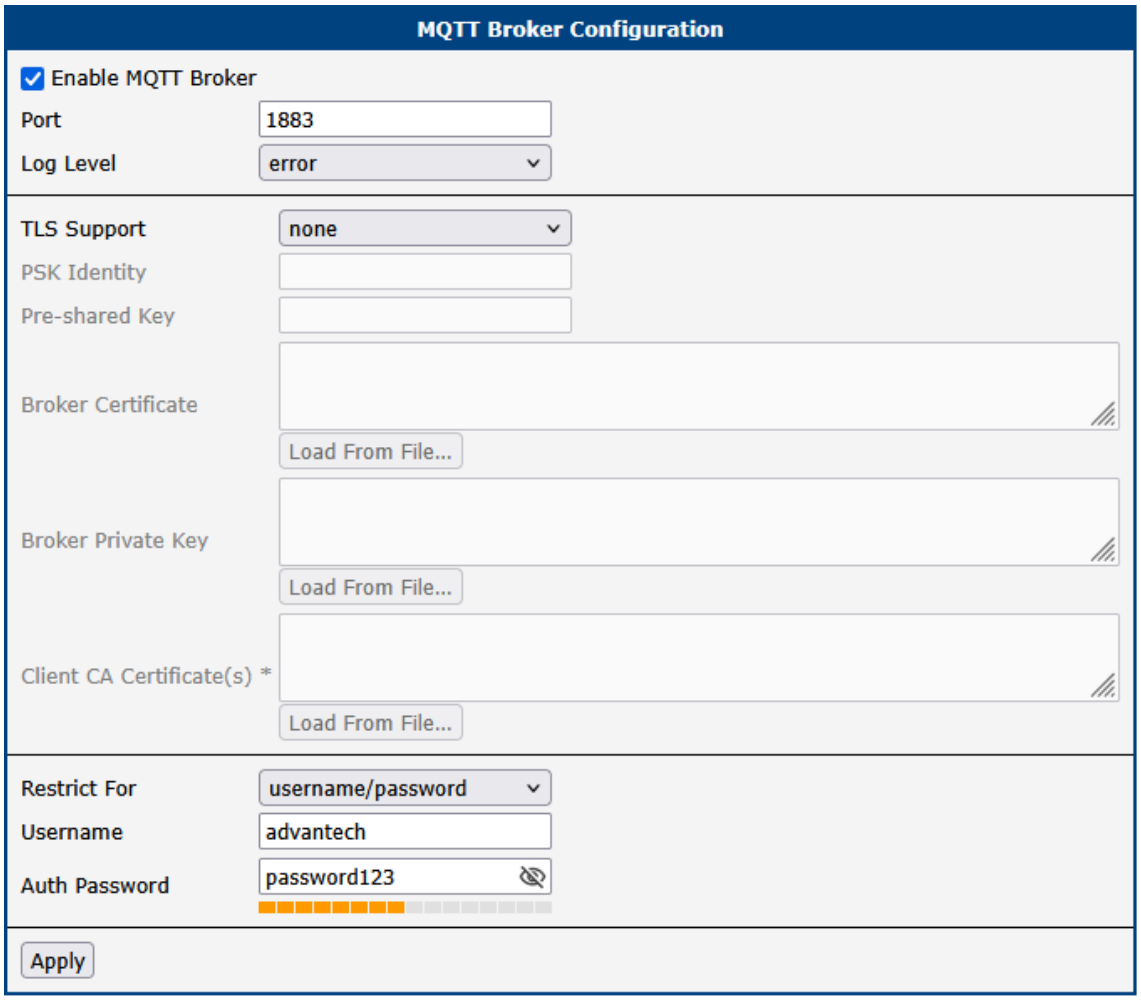


Figure 3: Broker Configuration Example

After applying these settings, MQTT clients can connect to the router's IP address on port 1883, but they must provide the username "advantech" and password "password123" during the connection handshake to be successfully authenticated.

2.4 Bridge Configuration

You can configure up to **three** MQTT bridges (Bridge 1-3). An MQTT bridge connects this local broker to a remote MQTT broker, allowing messages to be shared between them based on specified topic patterns. The configuration page for such a bridge is shown in the Figure 4 and described in Table 2.

Other brokers can connect as a bridge to this broker regardless this configuration. Other brokers must use the connection options defined on the *Broker* page. When *Enable MQTT Broker* on *Broker* page is unchecked, only one useful configuraton is to define two (or three) bridges. Then this router will resend messages between these bridged routers, but will have no its own clients.

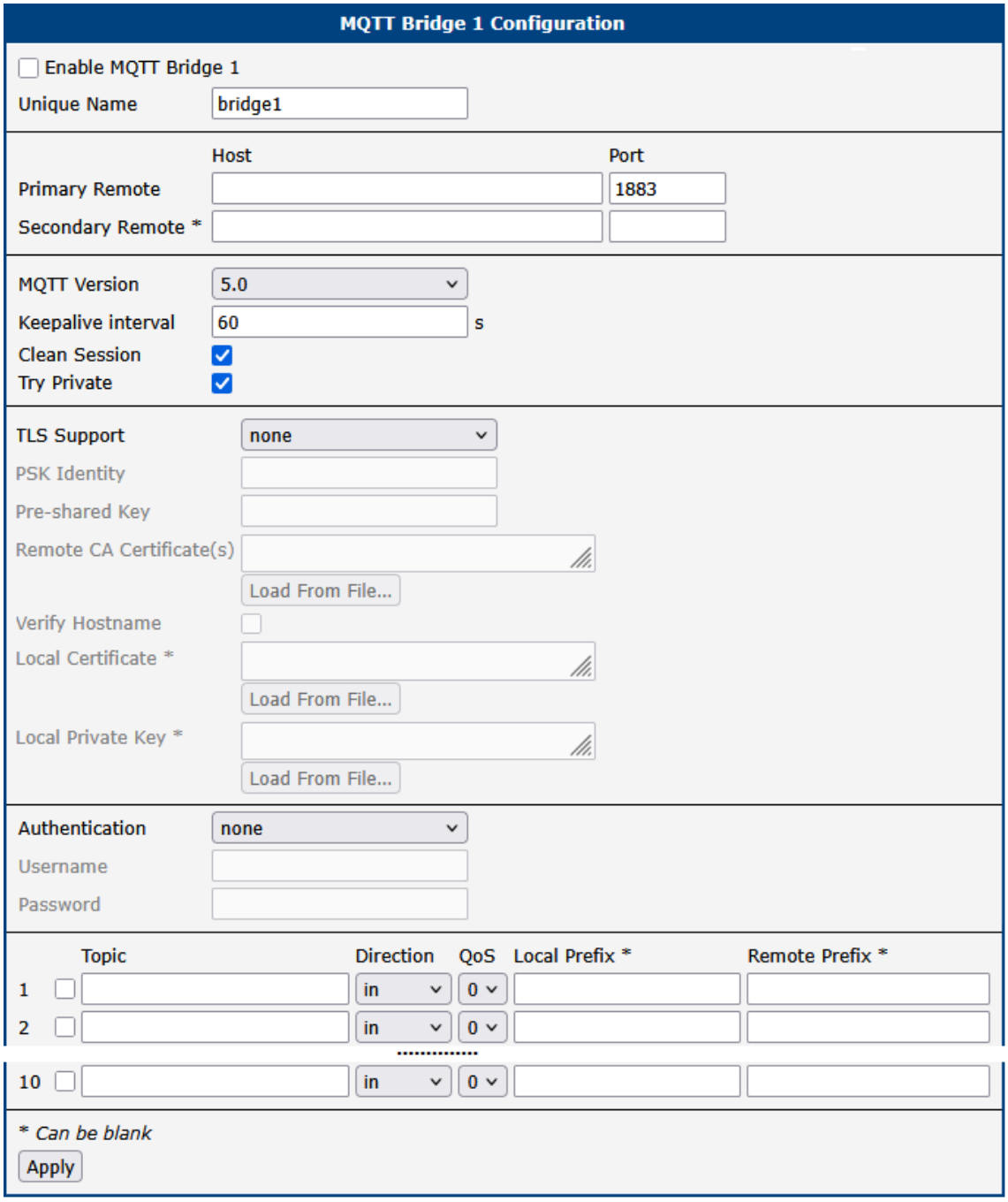


Figure 4: Bridge Configuration

Item	Description
Enable MQTT Bridge *x*	Enables MQTT Bridge number *x* functionality.
Unique Name	A user-defined name to identify this bridge. It is used as connection name and also as client ID to identify this local broker to a remote broker.
Primary Remote Host	The hostname or IP address of the remote MQTT broker to connect to.
Primary Remote Port	The TCP port number of the remote MQTT broker (e.g., 1883 or 8883).
Secondary Remote	Optional second remote MQTT broker to connect to, using the same credentials as the primary remote host. It will be used as backup when the primary broker is no avaiable.
MQTT Version	Choose the MQTT protocol version to use for the connection to the remote broker (3.1.1 or 5.0).
Keepalive Interval	The maximum time interval (in seconds) between messages sent between the local and remote broker. If no other messages are flowing, PING message is sent. When no response, the remote broker is considered disconnected. Defaults typically to 60.
Clean Session	When checked, all messages and subscription will be cleaned up on the remote broker on bridge disconnection. When unchecked, the subscriptions and messages are kept on the remote broker, and delivered on the bridge reconnection.
Try Private	When checked, the local broker try to indicate to remote broker the connection is a bridge not an ordinary client. Bridging then operate more effective. But not all brokers support this feature and then it can cause issues in connection.
TLS Support	Options for TLS encryption for the connection to the remote broker: • **none** – No TLS encryption is used for the bridge connection. • **PSK based** – Use TLS encryption with a Pre-Shared Key for the bridge connection. Requires PSK Identity and Key. • **certificate based** – Use TLS encryption with X.509 certificates for the bridge connection. Requires *Remote CA Certificate(s)*. When the remote broker tries to verify clients via CA, the optional fields *Local Certificate* and *Local Private Key* must be filled.
PSK Identity	A string to identify the pre-shared key to the remote broker (used if TLS Support is PSK based).
Pre-shared Key	When *PSK based* TLS support is selected, enter the pre-shared Key as a hexadecimal string.
Remote CA Certificate(s)	CA certificate(s) (PEM format) used to verify the remote broker's certificate (used if TLS is *certificate based*).
Verify Hostname	Enables/disables checking if hostname of the remote broker and hostname in the it's certifikace match.
Local Certificate	The client certificate (PEM format) for this broker to present to the remote broker (optional, used only if TLS is certificate based and remote broker verifies client certificate).
Local Private Key	The private key (PEM format) corresponding to the Client Certificate (optional, used only if TLS is certificate based and remote broker verifies client certificate).

Continued on the next page

Continued from previous page

Item	Description
Authentication	Options for authenticating this local broker to the remote broker: • **none** – No authentication is sent to the remote broker. • **username**/**password** – Authenticate to the remote broker using a username and password.
Username	Username for authenticating to the remote broker (used if *Authentication* is *username/password*).
Password	Password for authenticating to the remote broker (used if *Authentication* is *username/password*).
Topic Local Prefix Remote Prefix	Filter and rewrite the shared messages. The messages incomming from local broker must have `Local Prefix/Topic` topic to be shared. The messages incomming from remote broker must have `Remote Prefix/Topic` topic to be shared. When the message is forwarded the *Local Prefix* is replaced by *Remote Topic* and vice versa. Be careful to avoid loops when defining multiple rules. You must provide a *Topic* and optionally *Local Prefix* and/or *Remote Prefix*. Prefixes must end with a slash in this case and the *Topic* is concatenated with the prefixes. Or you can leave the *Topic* empty and fill both *Local Prefix* and *Remote Prefix*. Prefixes must not end with the slashin this case and they are used as full topic.
Direction	Filter the shared messages by direction: • **in** – Receive messages matching the topic pattern from the remote broker and publish them locally. • **out** – Send messages matching the topic pattern from the local broker to the remote broker. • **both** – Both send messages to and receive messages from the remote broker based on the topic pattern.
QoS	Specify the Quality of Service level for the shared messages (0: At most once, 1: At least once, 2: Exactly once). It determines the reliability of message delivery across the bridge.

Table 2: Bridge Configuration Items Description

2.4.1 Bridge Configuration Example

This example demonstrates how to configure an MQTT bridge to forward messages published locally on topics starting with `local/sensor/` to a public remote broker (`test.mosquitto.org`) under a different topic structure (`remote/office1/sensor/`).

Scenario:

- Enable bridge 1 by ticking *Enable MQTT Bridge 1*.
- Connect to the public broker `test.mosquitto.org` on port 1883.
- Use MQTT version 3.1.1 without TLS or authentication.
- Forward messages **out** from the local broker to the remote broker.
- Local messages published to `local/sensor/<specific_sensor>` should be published on the remote broker as `remote/office1/sensor/<specific_sensor>`.
- Use Quality of Service (QoS) level 1 for forwarded messages.
- Enable and configure broker to allow sensors to publish messages to it.

GUI Configuration:

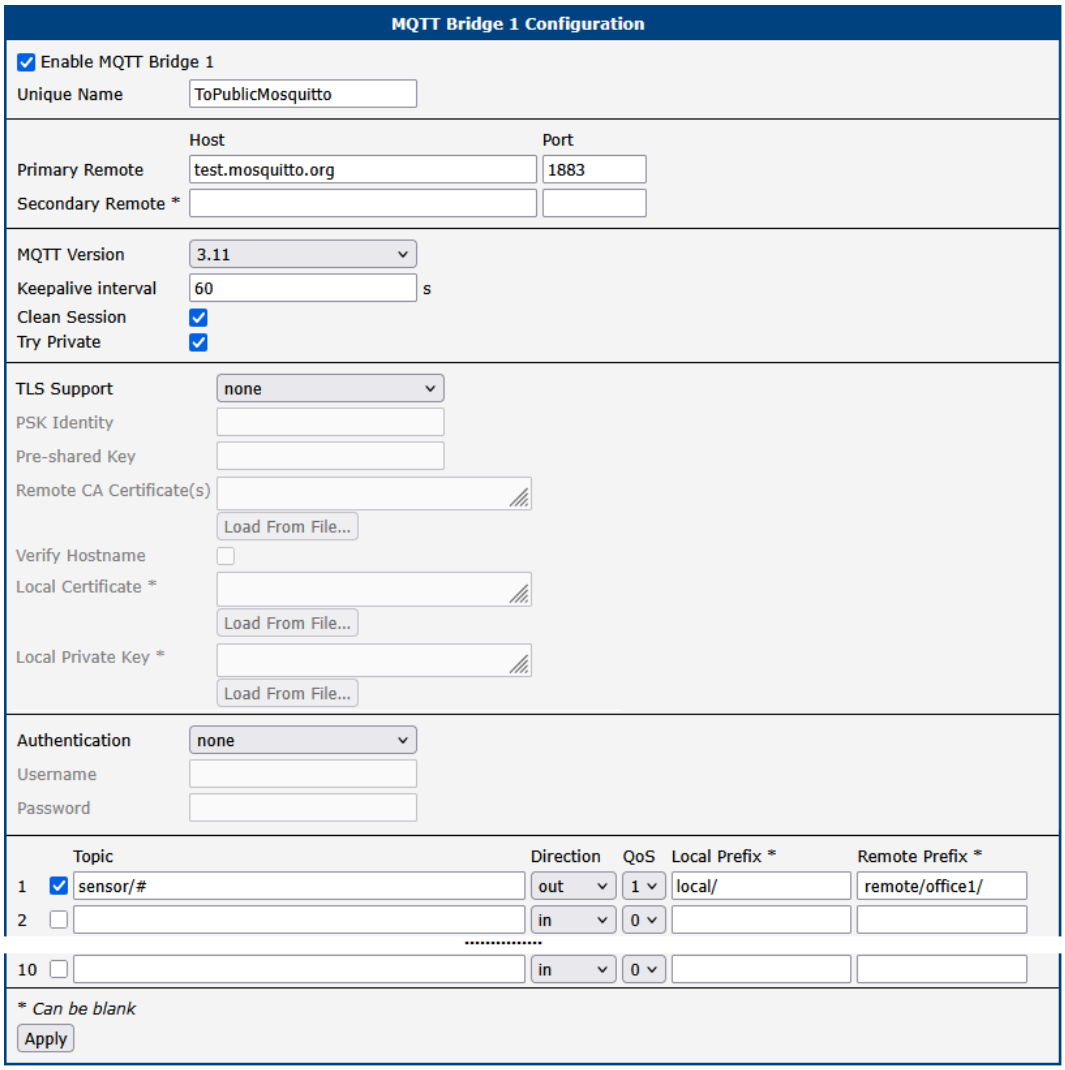


Figure 5: Bridge Configuration Example

Once applied, any message published locally to a topic like `local/sensor/temperature` will be automatically forwarded by the bridge and published to the topic `remote/office1/sensor/temperature` on the `test.mosquitto.org` broker.

How it works:

1. A message with `local/sensor/temperature` topic and a QoS x arrives to the broker.
2. The message is published locally with `local/sensor/temperature` topic and QoS x to all relevant subscribers.
3. The bridging checks its rules with *Direction* set to `out` . It finds the rule configured above.
4. It checks if the topic `local/sensor/temperature` starts with the *Local Prefix* (`local/`). It does.
5. It removes the *Local Prefix*, leaving `sensor/temperature` .
6. It checks if the remaining part (`sensor/temperature`) matches the *Topic* pattern (`sensor/#`). It does.
7. It takes the matched part (`sensor/temperature`) and prepends the *Remote Prefix* (`remote/office1/`), resulting in the topic `remote/office1/sensor/temperature` .
8. It publishes the message to the remote broker with the `remote/office1/sensor/temperature` topic and *QoS* level `1` .

2.5 Security

The MQTT protocol itself does not mandate encryption, but security is typically added using TLS at the transport layer. This section details how to configure encryption and authentication for the MQTT Broker Router App.

2.5.1 Encryption

Encryption of transmission is not performed at the MQTT protocol level itself, but rather at the TCP/IP level using TLS (Transport Layer Security). **Without encryption enabled, all transmitted data, including passwords, is visible in plain text to network listeners!** For encryption, you have two primary options provided by this Router App: *PSK based* and *Certificate based*.

PSK based encryption uses a Pre-Shared Key. This involves symmetric encryption where both the broker and the client must possess the same secret key. This key must be exchanged securely beforehand (out-of-band). A hexadecimal string is typically used as the key. You can generate one using a tool like OpenSSL:
`openssl rand -hex 32`
This command generates a 32-byte (256-bit) key. You must enter this key into the *Pre-shared Key* field in the Router App settings and also configure it on the MQTT client(s)[1]. Additionally, you must provide a *PSK Identity* string in both the Router App settings and the client configuration. This identity tells the broker which key to use for a connecting client, allowing for different keys potentially being used by different clients (although the Router App GUI currently supports configuring only one PSK Identity/Key pair for all PSK clients).

Certificate based encryption uses asymmetric cryptography with X.509 certificates. This is the more common and scalable method for securing MQTT. You can use commercial certificates (from trusted Certificate Authorities, potentially including free options like Let's Encrypt) or generate your own self-signed certificates for private deployments. If you choose self-signed certificates, OpenSSL is a common tool. Here is an example procedure to create a simple CA and a broker certificate:

1. Generate a private key for your Certificate Authority (CA):
   `openssl genpkey -algorithm RSA -out ca.key`
2. Create the self-signed root CA certificate using the CA key:
   `openssl req -new -x509 -days 1826 -key ca.key -out ca.crt`
   • You will be prompted for CA details like Country, Organization Name, etc.
3. Generate a private key for the MQTT Broker:
   `openssl genpkey -algorithm RSA -out broker.key`
4. Create a certificate signing request (CSR) for the Broker. Crucially, set the Common Name (CN) to the hostname or IP address clients will use to connect to the broker:
   `openssl req -new -out broker.csr -key broker.key -subj "/CN=router.local"`
   • Replace `router.local` with the actual hostname or IP address of your router. Use Subject Alternative Names (SANs) if multiple addresses are used.
5. Sign the broker's CSR using your CA key and certificate to create the broker's certificate:
   `openssl x509 -req -in broker.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out broker.crt -days 360`

[1] Client configuration details depend on the specific MQTT client software used and are not covered in this guide; consult your MQTT client documentation.

6. Copy the contents of the generated files `ca.crt` (your CA certificate), `broker.crt` (the broker's certificate), and `broker.key` (the broker's private key) into the corresponding fields (*CA Certificate(s)*, *Broker Certificate*, and *Broker Private Key*) in the Router App's Broker Configuration section. The file contents should be pasted in PEM format (text starting with '—–BEGIN...').
7. Distribute the CA certificate file (`ca.crt`) to all clients so they can verify the broker's certificate.

- Various cryptographic algorithms and parameters can be used during certificate generation. Ensure compatibility with your clients. Importantly, the broker's private key (`broker.key`) must **not** be password protected, otherwise the broker service cannot load it.
- The Common Name (CN) or Subject Alternative Name (SAN) in the broker's certificate (`broker.crt`) must match the address clients use to connect. If there's a mismatch, most MQTT clients will refuse the connection due to failed hostname verification, unless explicitly configured to ignore such errors (e.g., using an `--insecure` flag in tools like `mosquitto_pub` or `mosquitto_sub` , which is generally discouraged for production).

2.5.2 Authentication

Authentication verifies the identity of clients connecting to the broker. This Router App supports authentication via username/password or client certificates.

For the **username/password** option, enter the desired username and password into the *Username* and *Password* fields in the Router App's Broker Configuration. Clients must then provide these exact credentials when connecting. Note that while the underlying Mosquitto broker can support multiple username/password pairs (via configuration files), this Router App GUI currently allows configuring only one pair, which all clients using this method must use.

The **certificate CN** option provides authentication based on the client's X.509 certificate. This method requires *certificate based* TLS encryption to be enabled first. The client must present a valid certificate signed by a Certificate Authority listed in the broker's *CA Certificate(s)* field. The broker then extracts the Common Name (CN) from the subject field of the client's certificate and compares it to the value entered in the *Username* field in the Router App's Broker Configuration. If they match, authentication succeeds.

Here is an example procedure using OpenSSL to create a client certificate signed by the CA created previously:

1. Generate the client's private key:
   `openssl genpkey -algorithm RSA -out client.key`
2. Generate a certificate signing request (CSR) for the client. The Common Name (CN) here will be used for authentication:
   `openssl req -new -out client.csr -key client.key -subj "/CN=client_device_001"`
   • Replace `client_device_001` with the desired unique name for the client.
3. Sign the client's CSR using your CA certificate and key:
   `openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client.crt -days 360`
4. Enter the exact Common Name used in step 2 (e.g., `client_device_001`) into the *Username* field in the Router App's Broker Configuration and select *certificate CN* as the Authentication method.
5. Securely distribute the client's certificate (`client.crt`), the client's private key (`client.key`), and the CA certificate (`ca.crt`) to the client device. The client software must be configured to use these files for TLS connection and authentication.

2.6 Command Line Tools

The MQTT Broker Router App utilizes the open-source Eclipse Mosquitto™ project. After installing the Router App, two useful command-line utilities, `mosquitto_pub` and `mosquitto_sub`, become available via the router's command-line interface (e.g., via SSH). These can be invaluable for testing, debugging, and scripting MQTT interactions directly on the router.

2.6.1 `mosquitto_pub` Command

The `mosquitto_pub` command is used to publish a single message to a specific topic on an MQTT broker.

Command Example

Scenario:

- Publishing the message payload "54" to the topic "SENSOR/ROOM8/TEMPERATURE".
- The target broker is running on the same router at IPv6 address `fc00:202::254`, using port 1883.
- The connection uses PSK-based TLS encryption with the identity "test" and the hexadecimal key.
- Authentication is performed using the username "house" and password "12345".

Command syntax:

```
mosquitto_pub --host fc00:202::254 --port 1883 --psk
  b7e87d9b2074d5889a7969985d9fc3f04e4e5b8cb57956a812b2c52b6411abd8 --psk-identity test
  -u house -P 12345 -t SENSOR/ROOM8/TEMPERATURE -m 54
```

Configuration of the broker on the router:

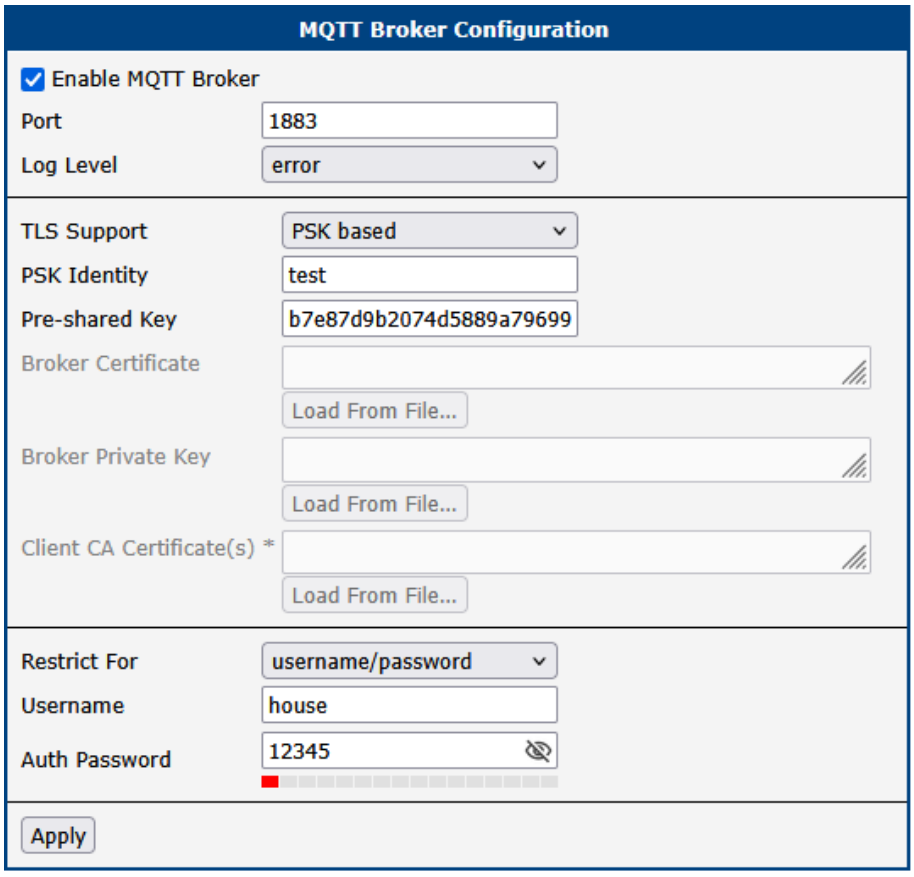


Figure 6: Broker Configuration for the `mosquitto_pub` Command Example

2.6.2 `mosquitto_sub` Command

The `mosquitto_sub` command subscribes to one or more topics (using wildcards if needed) and prints any messages received on those topics.

Command Example

Scenario:

- Subscribing to all topics starting with `factory/sensor/#` (note the wildcard `#`).
- The connection is made to a broker identified by the hostname "router" on the default port 1883.
- The connection uses certificate-based TLS encryption; the client verifies the broker's certificate using `ca.crt` (`--cafile`).
- The client authenticates itself to the broker using its own certificate `client.crt` (`--cert`) and corresponding private key `client.key` (`--key`).
- The command is set to exit after receiving exactly one message (`-C 1`).

Command Syntax:

```
mosquitto_sub -C 1 --host router --port 1883 --cafile ca.crt --cert client.crt --key
  client.key -t factory/sensor/#
```

Configuration of the broker on the router:

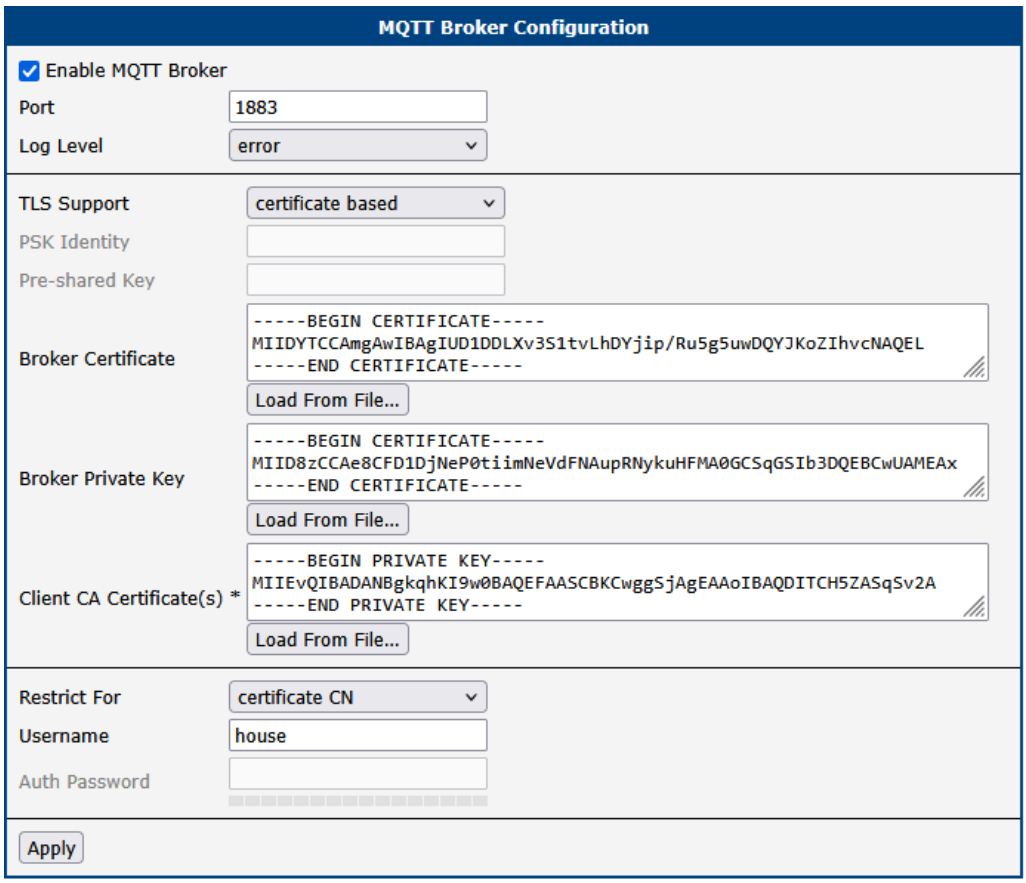


Figure 7: Broker Configuration for the `mosquitto_sub` Command Example

These tools provide powerful ways to interact directly with the MQTT broker running on the router. Consult the Mosquitto documentation (e.g., via `man mosquitto_pub` or `man mosquitto_sub` on the router CLI) or use the `--help` flag for more detailed usage information.

3. Related Documents

[1] MQTT Specifications Version 5.0: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html
[2] MQTT Specifications Version 3.1.1: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
[3] MQTT Broker Mosquitto: https://mosquitto.org/

You can obtain product-related documents on *Engineering Portal* at *icr.advantech.com* address.

To get your router's *Quick Start Guide*, *User Manual*, *Configuration Manual*, or *Firmware* go to the *Router Models* page, find the required model, and switch to the *Manuals* or *Firmware* tab, respectively.

The *Router Apps* installation packages and manuals are available on the *Router Apps* page.

For the *Development Documents*, go to the *Development* page.