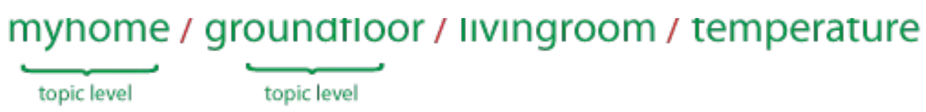
MQTT topics are a fundamental part of how the MQTT protocol routes messages between publishers and subscribers. They act as "addresses" that define where each message should be delivered. Unlike traditional messaging queues, MQTT topics are hierarchical and enable a highly flexible publish/subscribe communication model. Here's Part 5 of **MQTT Essentials**, where we will focus on MQTT topics, MQTT Wildcards, and explore best practices for their usage in detail. In this article, we will also examine SYS-topics, which offer insights into the broker itself.

If you are exploring to understand **publishing, subscribing, and unsubscribing in MQTT**, check out Part 4 of this series. Else, let's dive in.

What Are MQTT Topics and Their Role in MQTT Message Filtering?

In MQTT, `Topic` refers to a UTF-8 string that filters messages for a connected client. A topic consists of one or more levels separated by a forward slash (topic level separator). The UTF-8 strings act as identifiers for messages, enabling a publish/subscribe system where publishers send messages to specific topics and subscribers can subscribe to those topics to receive relevant messages. They are crucial for routing messages between clients, as the broker uses topic subscriptions to filter and deliver messages to the appropriate subscribers.

Topics are often structured hierarchically, with levels separated by forward slashes (/). For example, a topic like "sensors/temperature/livingroom" indicates a temperature reading from a sensor in the living room.



The Basics of MQTT Topics

In comparison to a message queue, MQTT topics are very lightweight. The client does not need to create the desired topic before they publish or subscribe to it. The broker accepts each valid topic without any prior initialization.

MQTT Topic Names vs. Topic Filters:

Publishers use "topic names" to send messages, while subscribers use "topic filters" to specify which topics they want to receive messages from. Topic filters can also include wildcards (e.g., "+" or "#") to match multiple topics.

In comparison to a message queue, MQTT topics are very lightweight. The client does not need to create the desired topic before they publish or subscribe to it. The broker accepts each valid topic without any prior initialization.

Examples of MQTT Topics

Here are some examples of MQTT Topics:

1. `myhome/groundfloor/livingroom/temperature` : This topic represents the temperature in the living room of a home located on the ground floor.
2. `USA/California/San Francisco/Silicon Valley` : This topic hierarchy can track or exchange information about events or data related to the Silicon Valley area in San Francisco, California, within the United States.

3. `5ff4a2ce-e485-40f4-826c-b1a5d81be9b6/status` : This topic could be used to monitor the status of a specific device or system identified by its unique identifier.

4. `Germany/Bavaria/car/2382340923453/latitude` : This topic structure could be utilized to share the latitude coordinates of a particular car in the region of Bavaria, Germany.

By clicking on the image, you interact with a video on YouTube.
Please read our **privacy policy page** to understand how we process data.

MQTT topics are key in establishing communication between MQTT clients and brokers. They enable efficient filtering and routing of messages based on their content. Properly defining and structuring topics is crucial in ensuring effective data exchange and handling within MQTT-based systems.

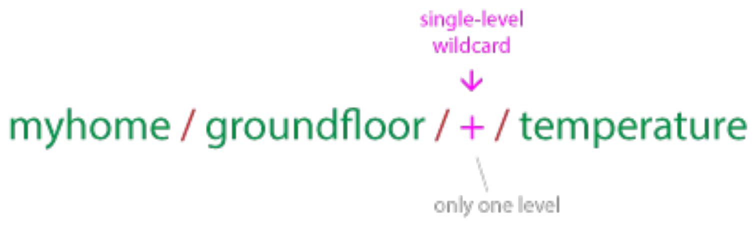
What Are MQTT Wildcards and How to Use Them With

Topic Subscriptions?

In MQTT, wildcards provide a powerful mechanism for subscribing to multiple topics simultaneously. When a client subscribes to a topic, it can either subscribe to the exact topic of a published message or utilize wildcards to broaden its subscription. It's important to note that wildcards can only be used for subscription and not for publishing messages. There are two types of wildcards: `single-level` and `multi-level` .

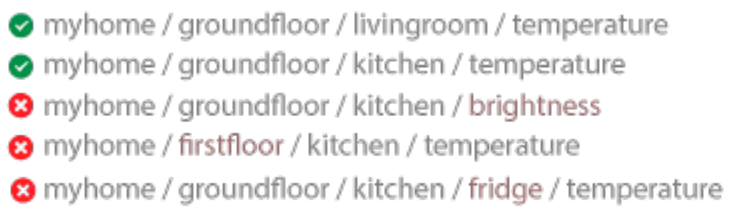
MQTT Wildcard – Single Level: +

The single-level wildcard is represented by the plus symbol (+) and allows the replacement of a single topic level. By subscribing to a topic with a single-level wildcard, any topic that contains an arbitrary string in place of the wildcard will be matched.



Example of how to use MQTT Wildcard Single Level +

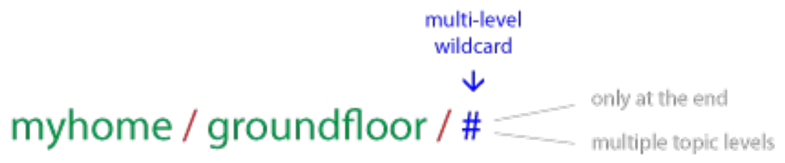
For example, a subscription to `myhome/groundfloor/+/temperature` can produce the following results:



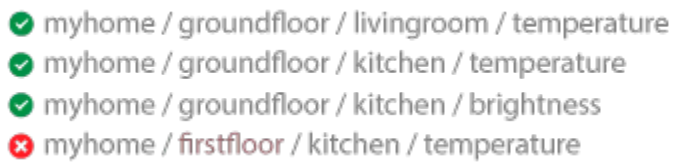
✓ myhome / groundfloor / livingroom / temperature
✓ myhome / groundfloor / kitchen / temperature
✗ myhome / groundfloor / kitchen / brightness
✗ myhome / firstfloor / kitchen / temperature
✗ myhome / groundfloor / kitchen / fridge / temperature

MQTT Wildcard – Multi Level:

The multi-level wildcard covers multiple topic levels. It is represented by the hash symbol (#) and must be placed as the last character in the topic, preceded by a forward slash.



When a client subscribes to a topic with a multi-level wildcard, it receives all messages of a topic that begins with the pattern before the wildcard character, regardless of the length or depth of the topic. If the topic is specified as "#" alone, the client receives all messages sent to the MQTT broker.


myhome / groundfloor / livingroom / temperature
myhome / groundfloor / kitchen / temperature
myhome / groundfloor / kitchen / brightness
myhome / firstfloor / kitchen / temperature

MQTT Topic wildcard hash example

However, it's important to consider that subscribing with a multi-level wildcard alone can be an anti-pattern if high throughput is expected. Subscribing to a broad topic can result in a large volume of messages being delivered to the client, potentially impacting system performance and bandwidth usage. Follow best practices to optimize topic subscriptions and avoid unnecessary message overload.

Why and When to Use MQTT Topics Beginning with $

In MQTT, topic naming flexibility is vast, allowing you to choose names that suit your needs. However, there is one important exception to be aware of: topics that start with a $ symbol have a distinct purpose. These topics are not included in the subscription when using the multi-level wildcard (#) as a topic. Instead, topics beginning with $ are reserved for internal statistics of the MQTT broker, providing valuable insights into its operation.

Publishing messages to topics starting with $ is not permitted, as these topics serve as a means for the MQTT broker to expose internal information and statistics to clients. While there is currently no official standardization for these topics, it is common to use the prefix $SYS/ to denote such information, although specific implementations of brokers may vary.

One recommended resource for understanding $SYS topics is available in the **MQTT GitHub wiki.**

Here are a few examples of $SYS topics and the information they can provide:

1. `$SYS/broker/clients/connected` : Indicates the number of clients currently connected to the MQTT broker.
2. `$SYS/broker/clients/disconnected` : Shows the number of clients that have disconnected from the MQTT broker.
3. `$SYS/broker/clients/total` : Represents the total count of clients, both connected and disconnected, that have interacted with the MQTT broker.
4. `$SYS/broker/messages/sent` : Provides the count of messages sent by the MQTT broker.
5. `$SYS/broker/uptime` : Reflects the duration the MQTT broker has been

running.

> ***These $SYS topics offer valuable insights into the internal workings and performance of the MQTT broker, enabling administrators and developers to monitor and analyze crucial statistics.***

By understanding the purpose and significance of topics starting with $, you can effectively leverage this convention to gain deeper visibility into the behavior and performance of their MQTT infrastructure.

Exploring the Dynamic Nature of MQTT Topics

These are the basics of MQTT message topics. As you can see, MQTT topics are dynamic and provide great flexibility. When you use wildcards in real-world applications, there are some challenges you should be aware of. We have collected the best practices that we have learned from working extensively with MQTT in various projects and are always open to suggestions or a discussion about these practices. Use the comments to start a conversation, Let us know your best practices or if you disagree with one of ours!

MQTT Topics Best Practices

Here are a few important best practices for using MQTT Topics:

Have at least one character

Each topic must contain at least one character.

MQTT Topics are Case sensitive

Topics are case-sensitive, meaning " `myhome/temperature` " and " `MyHome/Temperature` " are considered as two different topics.

Avoid Leading Forward Slash

While MQTT allows a leading forward slash in topics (e.g., `/myhome/groundfloor/livingroom`), it introduces an unnecessary topic level with a zero character at the front. This can cause confusion (having a zero character at the front) without providing any benefit. Hence, it’s recommended to exclude the leading forward slash. However, the forward slash alone is a valid topic and can be used to represent a broad topic or serve as a wildcard for subscribing to multiple topics simultaneously.

Never use spaces in an MQTT Topic

A space is the natural enemy of every programmer. Spaces in topics can hinder readability and debugging efforts, particularly during troubleshooting scenarios. Moreover, UTF-8 has many different white space types. We advise against using spaces and uncommon characters altogether in MQTT topics.

Keep MQTT topics short and concise

Remember that each topic is included in every message in which it is used. To optimize network traffic and conserve valuable resources, strive to make your topics concise. This is especially crucial when dealing with resource-constrained devices, where every byte counts.

Use only ASCII characters, and avoid non-printable characters

To ensure consistent and accurate representation of topics, it's advisable to stick to ASCII characters. Non-ASCII UTF-8 characters may display incorrectly, making identifying typos or character set-related issues challenging. Unless essential, refrain from using non-ASCII characters in your MQTT topics.

Embed a unique identifier or the Client Id in topics

To enhance message identification and enforce authorization, consider embedding a unique identifier or the client ID of the publishing client in the topic. This allows you to determine the message sender and control publishing permissions. For example, a client with the client1 ID can publish to `client1/status` but not to `client2/status` .

Avoid Subscribing to Wildcards (#)

Sometimes, it is necessary to subscribe to all messages that are transferred over the broker. For example, to persist all messages into a database. Do not subscribe to all messages on a broker by using an MQTT client and subscribing to a multi-level wildcard. Frequently, the subscribing client is not able to process the load of messages that results from this method (especially if you have a massive throughput). Our recommendation is to implement an extension in the MQTT broker. For example, with the **HiveMQ extensions**, you can hook into the behavior of HiveMQ and add an asynchronous routine to process each incoming message and persist it to a database.

Design for Future Growth

As your system grows, you'll likely add new devices, data points, and capabilities. Design your topic structure to accommodate future expansion without requiring major restructuring. Include version information in your topics when appropriate.

Design your topics to facilitate extensibility without substantially changing the overall topic hierarchy. For example, if your smart-home solution adds new sensors, it should be possible to add these to your topic tree without changing the whole topic hierarchy.

Use specific topics, not general ones

Differentiate your topics to reflect specific data streams or entities. Avoid the temptation to use a single topic for multiple types of messages. For instance, if you have three sensors in your living room, create topics like `myhome/livingroom/temperature` , `myhome/livingroom/brightness` , and `myhome/livingroom/humidity` instead of using a generic topic like `myhome/livingroom` . This practice promotes clarity and enables the utilization of advanced MQTT features such as retained messages. To learn more, read our article on **MQTT retained messages.**

Documentation

Maintain comprehensive documentation detailing your MQTT topics, including their purpose, expected message payload, and any associated conventions or guidelines. This aids in onboarding new team members and fosters better collaboration.

Continuous Improvement

Regularly review and optimize your topic structure based on evolving requirements and feedback from your MQTT ecosystem. Embrace a continuous improvement mindset to ensure efficient and scalable MQTT communication.

Security Considerations:

Ensure that your topic structure and naming conventions don't inadvertently expose sensitive information. Implement proper access controls and authentication mechanisms to protect your MQTT communications.

MQTT Topic Design Patterns for Industrial Applications

In industrial settings, well-structured topic design becomes even more critical as the scale and complexity of systems increase. HiveMQ's experience with industrial clients has shown that effective topic design can significantly impact system performance, scalability, and maintainability.

Hierarchical Topic Design for Industrial IoT

Industrial applications benefit from topic designs that follow a hierarchical structure, moving from general to specific information. This approach aligns well with the typical organization of industrial systems.

For example, a manufacturing company might use this structure:

```
company/site/area/line/cell/device/measurement
```

This hierarchical approach makes it easier to:

- Organize data logically across the enterprise
- Subscribe to specific levels of information
- Scale the system as the organization grows
- Apply access control policies

ISA-95 Model for Topic Design

Many manufacturing companies organize their MQTT topics according to the ISA-95 equipment hierarchy model, a common standard in industrial environments. This approach establishes a natural mapping between physical assets and their digital representations.

A typical ISA-95-based topic structure might look like:

```
Enterprise/Site/Area/ProductionLine/WorkCell/Equipment/DataPoint
```

This structured approach ensures consistency throughout the organization and facilitates easier data discovery and interpretation by systems.

The Role of MQTT Topics in Building a Unified Namespace

The **Unified Namespace (UNS)** has emerged as a powerful architectural pattern for industrial IoT, relying heavily on well-designed MQTT topics. A UNS creates a centralized, hierarchical data structure that serves as a single source of truth for an organization. You could design your smart manufacturing system based on an UNS architecture and then use the ISA 95 standard for modeling the data objects that get pushed to your UNS.

In a UNS architecture, **MQTT topics** form the backbone of the data hierarchy. The **MQTT broker** organizes data using a topic hierarchy, which acts as a structured framework for data access within an **UNS**. This enables:

- Real-time data sharing across systems
- Efficient data routing and filtering

- Consistent data representation
- Easy discovery of available data
- Simplified integration of disparate systems

UNS Topic Design Example

Here's an example of how a UNS might organize topics for a manufacturing environment:

```
manufacturing/
  plantA/
    sensors/
      humidity/
        sensor001
        sensor002
    inventory/
      raw_materials/
        current_stock
      finished_goods/
        current_stock
    quality_control/
      inspection/
        results
```

This structure allows participants in the MQTT network to subscribe to specific levels of data as needed. For example, a supervisor might subscribe to `manufacturing/plantA/machines/+/#` to receive all messages related to machines in Plant A, while a quality control analyst might subscribe to `manufacturing/plantB/quality_control/testing/#` to receive data on all tests conducted in Plant B.

Conclusion

That brings us to the end of Part 5 of our MQTT Essentials series. To summarize, MQTT topics serve as the backbone of flexible and efficient message communication in MQTT. By understanding the intricacies and applying best practices, you can optimize your MQTT implementations for maximum performance and scalability.

Throughout this article, we explored the dynamic nature of MQTT topics, delving into wildcard usage, technical considerations, and recommended strategies. We discussed the importance of avoiding leading forward slashes and spaces in topics, keeping topics concise, utilizing ASCII characters, and embedding unique identifiers or client IDs. We also emphasized the significance of not subscribing to all messages using wildcards and the value of extensibility in topic design.

By following these best practices, you can enhance your MQTT infrastructure's readability, maintainability, and security. We hope this article has provided valuable insights and actionable information to elevate your MQTT projects. T hank you for joining us on this journey through the intricacies of MQTT topics. We hope you found this article valuable and informative.

In Part 6 of this series, we will go deeper in **Quality of Service (QoS) levels in MQTT.** We'll explain why this is an essential feature and how you can leverage it.

Looking to understand MQTT control packets and their structure to design and test MQTT-based systems? Read our blog **MQTT Packets: A Comprehensive Guide**.

Stay curious, stay innovative, and together let's shape the future of MQTT.

Are you enjoying our content? Then sign up for our newsletter below. Subscribe to our **RSS feed here** to stay updated. Do check out our **MQTT FAQs** and **MQTT Glossary** to know all the key MQTT terminologies. Watch the video below that complements the concepts discussed in this article.

FAQs About MQTT Topics