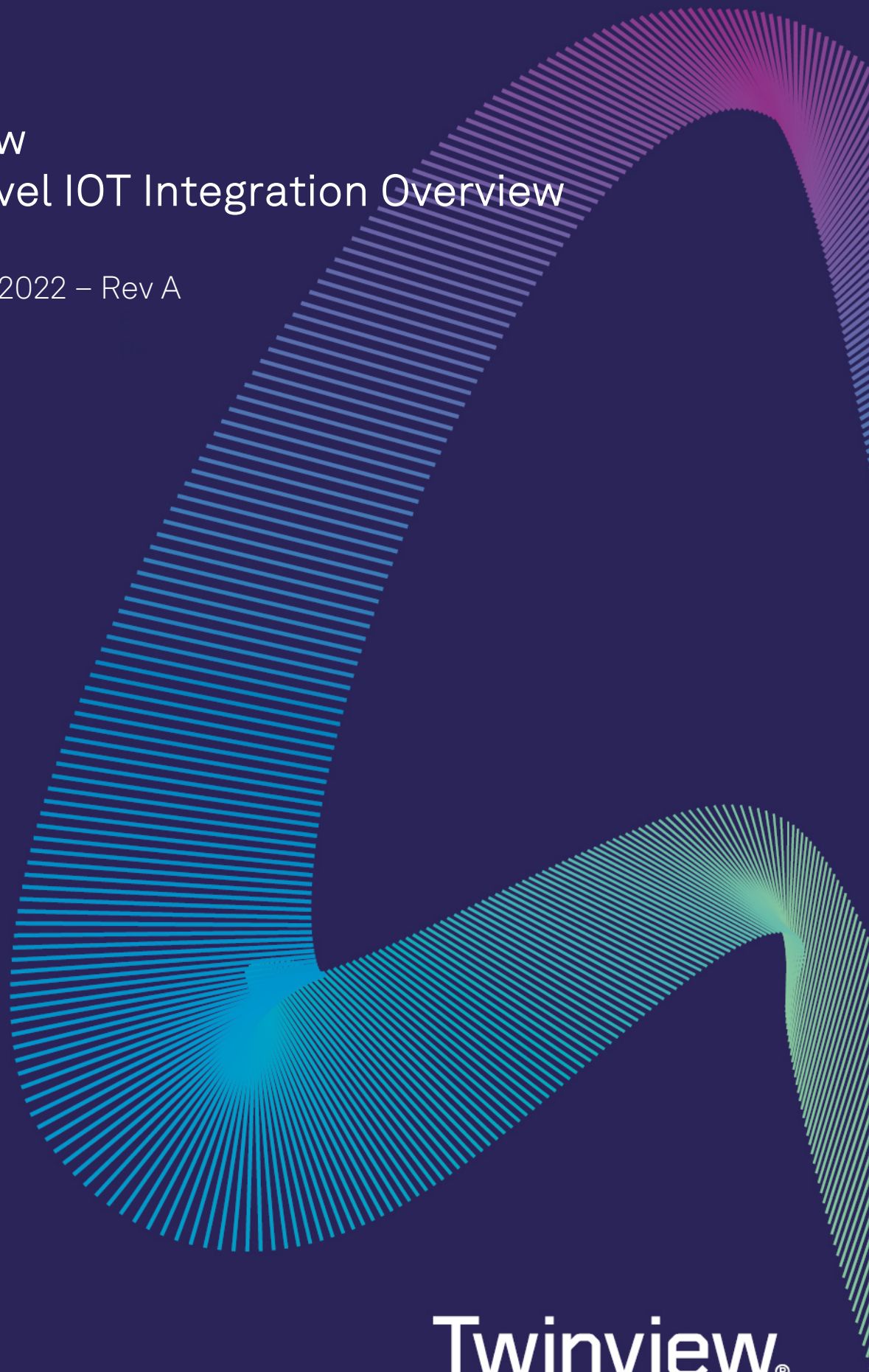
Twinview
High Level IOT Integration Overview

November 2022 – Rev A



Twinview®

Table of Contents

DOCUMENT REVISION HISTORY

Revision	Date	Author	
Rev A	29/11/22	Adam Ward	First Issue

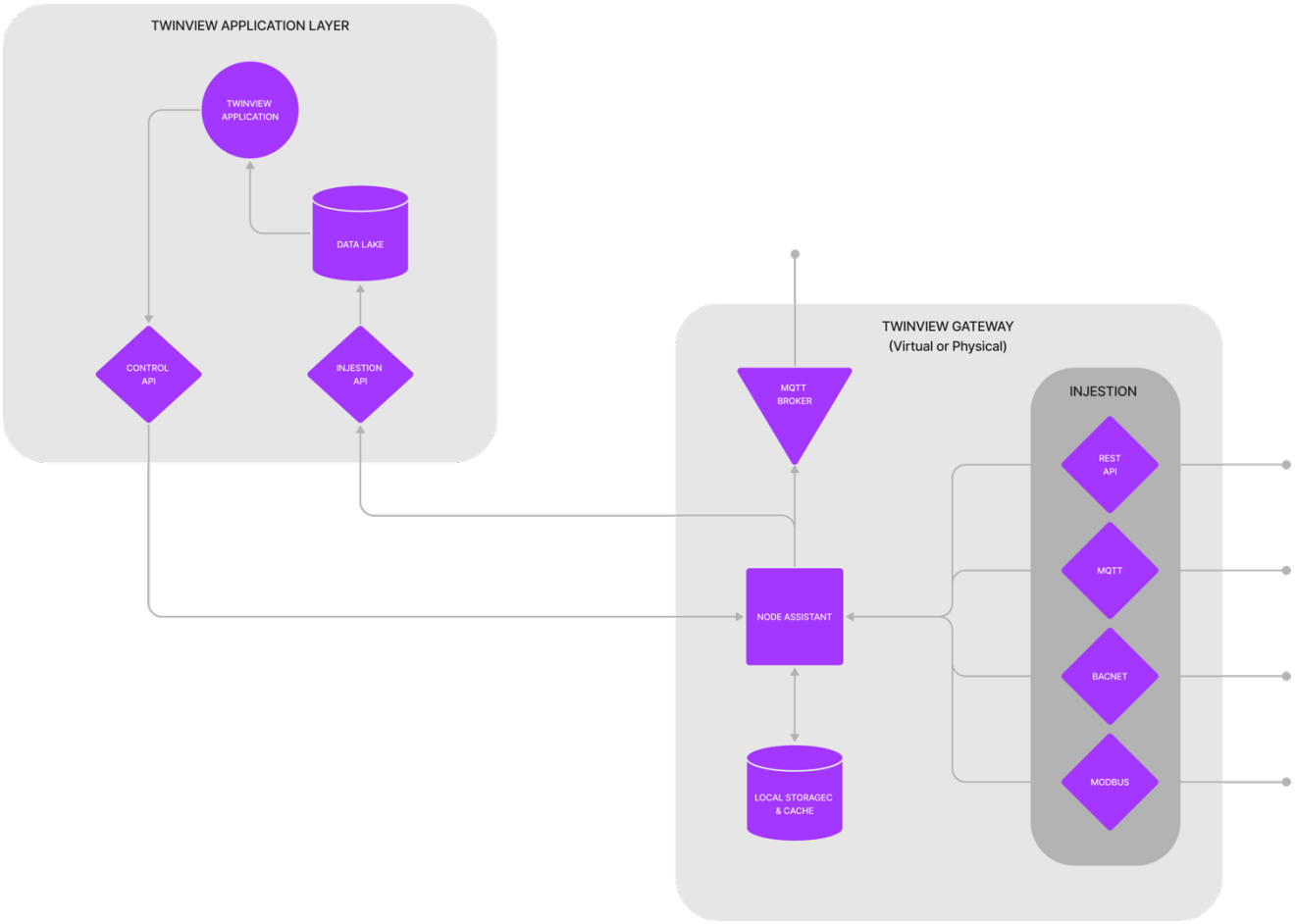
High Level Overview

This document is designed to give a high level understanding of Twinview and how it can integrate into your buildings systems and IOT networks. This is not a how to guide or training on implementation and is intended as an overview only. In depth specific training is available to all customers and partners on implementation.

Twinview is agnostic to any specific hardware and has various ways to integrate into the buildings systems. Twinview is a visualisation layer that can present information from multiple sources, including the buildings BMS. Sensor network, and systems.

Once in Twinview this data is represented as a 'Data Stream' and can be shown on a dashboard, report or used in heatmapping etc.

The data is stored by Twinview in the Twinview internal data lake. This is because we need to retain historic data for the purposes of charting, reporting and analysis.



Supported Protocols & Implementation

Twinview is device/system agnostic and does not require any specific manufacturer sensors or hardware to operate. Twinview integrates into the buildings systems and network at either at a protocol level or through its Rest API.

Twinview has built in support OTB for the following:

- Rest API
- BACnet
- Modbus
- MQTT
- LoRaWAN
- Zigbee
- Zwave

The Twinview Gateways

Twinview uses a gateway device to communicate with your buildings systems and devices. This gateway is specific for your building and for security considerations it is good practice to have a single gateway per building.

It should be noted however that the gateway does not necessarily need to be a physical device, it can be a virtual gateway which is hosted on the cloud. The choice of whether a physical gateway or virtual gateway is needed is dependent upon the building, its systems, and the client specific requirements.

The gateway performs multiple purposes, below are the main ones:

- An edge communication and data acquisition device (DAD)
- Logic engine for control and automation
- Local storage (Cache)
- An MQTT Broker
- Security Gateway
- Digital Display Server

GATWEAY CONSIDERATIONS

For most new buildings first choice should be a Virtual gateway, as this provides the best performance, and is easily managed and maintained. Most modern or new buildings systems will allow integrations supported by the virtual gateway.

For older/existing or specific scenarios a physical gateway may be preferred. The physical gateway is a small device that is physically located in the building and communicates with

the various systems. Although the benefits of a virtual gateway far outweigh the benefits of a physical gateway, there are a few advantages where it may be applicable.

Redundancy – Because the device is located on premiss and has inbuilt storage, it can store up to 30 days cache, allowing redundancy should a lost connection occur.

Wireless Gateway – The device can be installed with optional LoRaWAN, Zigbee or Z-Wave antennas allowing the device to have a secondary role as a wireless gateway, preventing the need for a secondary gateway for a wireless network.

Display Server – The device can be utilised as a display server to power digital signage and dashboarding.

GATEWAY SPECIFICATIONS

The Twinview gateway (physical or virtual) is an embedded Linux machine running ubuntu and a combination of custom applications. The main interface for setting up the gateway and its connection into your buildings systems is through a visual interface called Twinview Node Assistant. This allows integrations through an easy to use visual interface.

The device has inbuilt security and communicates with Twinview through a secure 256 bit encrypted connection. The device comes preconfigured from Twinview with the required SSL certificates and setup for the specific project.

For the physical gateway, it is available in 2 form factors. A small stand-alone device, or a 1U rack mounted chassis and requires a power socket, and an open connection to the Twinview ingestion and control API endpoints. Depending upon configuration, we will require several ports to be opened on the network to allow the device to operate and communicate. This will be undertaken by Twinview and the buildings IT/Security team as part of the onboarding/setup process.

High Level Data Architecture

Every building, its systems and its requirements are unique, and as such the Twinview implementation team will work with you as part of onboarding to understand your buildings current systems and the best way to both integrate and manage the data coming into Twinview.

Although Twinview can integrate into each system using the appropriate protocol/method specific to each sytem we would recommend creating a data layer that standardises your buildings data regardless of system, building or device. Twinview would then communicate with this single data layer providing consistency, ease of implementation and future proofing. This approach can also be rolled out across other projects, providing consistency, and enabling the datawarehouse (a single building) to transform into a datalake (multiple buildings).

To do this we would recommend using a PUBSUB broker (MQTT) to manage all incoming and outgoing data. MQTT is a lightweight, industry proven, and highly scalable solution designed for this purpose. The Twinview gateway, incudes an MQTT broker as standard.

For situations where Twinview has been implemented onto a single building, it's in built MQTT broker can be utilised. For implementation across multiple buildings or where the client has or wants to manage their own datalake, the Twinview can communicate with the clients MQTT broker acting as a bridge.

RECOMMENDED MQTT SCHEMA

If the MQTT approach is adopted, then it is important that the data schema is planned and agreed before implementation. The data schema will be defined by the Twinview implementation team as part of onboarding, after consultation with the client, however the purpose of this section is to define the limitations and best practices adopted by Twinview.

Although there are numerous combinations of IoT communication patterns that share common approaches, there are several best practices that apply to any message pattern irrespective of how a device is publishing or receiving a message. This section articulates several overall best practices for you to review and implement as you design your MQTT topic structures.

The maximum number of forward slashes (/) in the MQTT topic name is seven. You should not prefix the topic with a forward slash as it counts towards the topic levels and may introduce confusion when building AWS IoT policies. This excludes the first three slashes in the mandatory segments for Basic Ingest topics Client/Building/Level/rule-name.

The topic passed when sending a publish request can be no larger than 256 bytes of UTF-8 encoded characters. This excludes the first three mandatory segments for Basic Ingest topics.

Define a consistent naming standard for MQTT topic levels. Since MQTT topics are case sensitive, it is important to use a standard set of naming conventions when designing MQTT topics. For this reason, you should only use lowercase letters, numbers, and dashes when creating each topic level. You should also avoid camel casing and using hard to debug characters such as spaces. Publish Topic names cannot contain wildcards (# , +).

Ensure MQTT topic levels structure follows a general to specific pattern. As topic scheme flows left to right, the topic levels flow general to specific. For example a valve named AS23283 is located in a building belonging toa client named GPE, and is located in the level 2 of the Hickman building. The topic structure begins with the general group, in this case, the name of the client, and ends with the most specific identity, the Thing Name. This example creates the following topic level structure:

gpe/hickman/level2/plantroom/valves/as23283

Include the Thing Name of the device in any MQTT topic the device uses for publishing or subscribing to its data. To track messages destined for a particular device, include the Thing Name as part of any MQTT message that is published by the device or sent to a specific device. The Thing Name should appear near or at the end of the MQTT topic after any routing topic information.

Include any relevant routing information in the MQTT topic. Relevant routing information includes, but is not limited to, the client & building identifier, any groups the device may be a part of, such as installed location, and the unique identity of your IoT device/asset. To continue with the previous HVAC system example, the MQTT topic gpe/hickman/level2/plantroom/valves/as23283 includes all relevant routing information. Based on this MQTT topic, you can design a system that captures any data related to the entire application using the identifier, gpe, but also can target different areas of interest for subscribing to messages, such as the building location.

If connecting into a clients existing/proposed MQTT broker, ensure that you review their MQTT topic structures as part of your implementation process. The document should include all their topics available for publishing, subscribing, or receiving data, along with the intended producers and consumers of the data. Review the document to ensure it adheres to any limits, internal security requirements, and any application use cases.

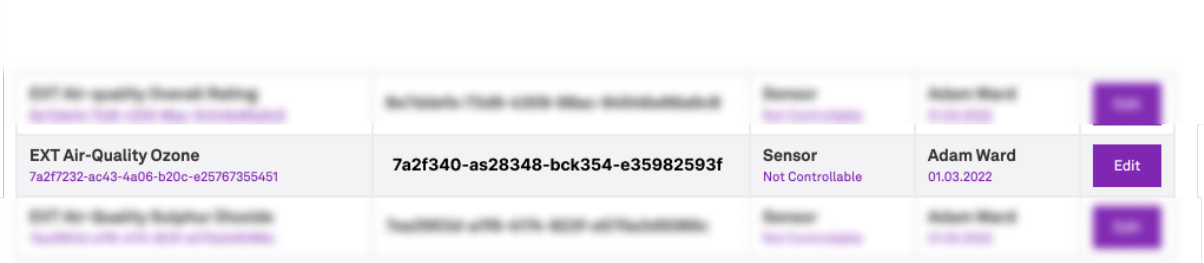
Never allow a device to subscribe to all topics using #, and only use multi-level wildcard subscriptions in IoT rules. By using multi-level wildcards, you can create unintended consequences when you inadvertently add new topics to the hierarchy that may not be intended for that device.

MQTT PAYLOAD SYNTAX

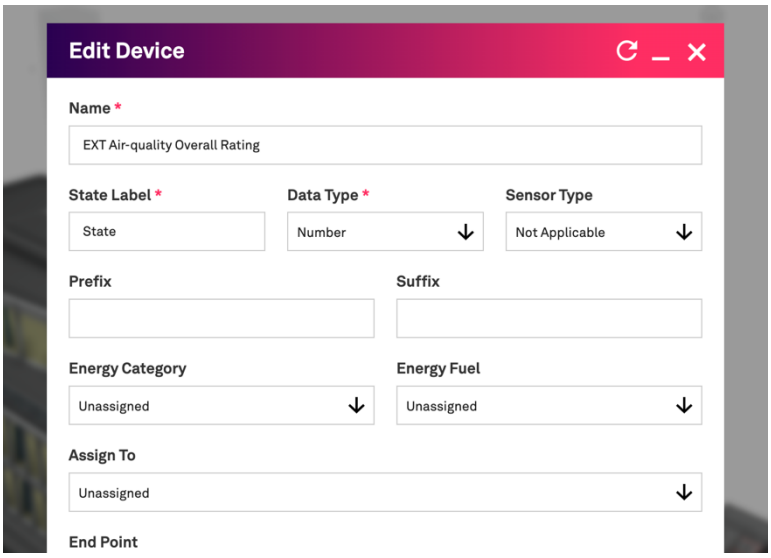
Make sure that you generate a schema for message payloads. MQTT payload information is parsed by the receiving device or IoT application, to inform it of any additional logic it may need to complete its operation. For example, to send data to the Twinview ingest API the data must be formatted in json as below, include the following fields with the command message payload:

```
{
  "uuid": "legacy_d989e52d-136b-40df-834a-f8688890b",
  "state": "22"
}
```

uuid: The unique identifier of the device DataStream as created in Twinviews via the device integration manager.



state: The data payload been sent. The data type should be Text (string), Number (Float), Boolean, or an array. This should be as defied in the device/end point created on Twinview below.



Conclusion

This document is a high level overview of the Twinview approach to IOT integrations and is based on industry standards and best practice.

When it comes to implementing Twinview and connecting your building and its systems the Twinview implementation team will undertake this process on your behalf. For partners, Twinview affiliates and clients who wish to understand and undertake this process themselves training can however be provided.

Any questions please contact us at:

info@twinview.com
www.twinview.com