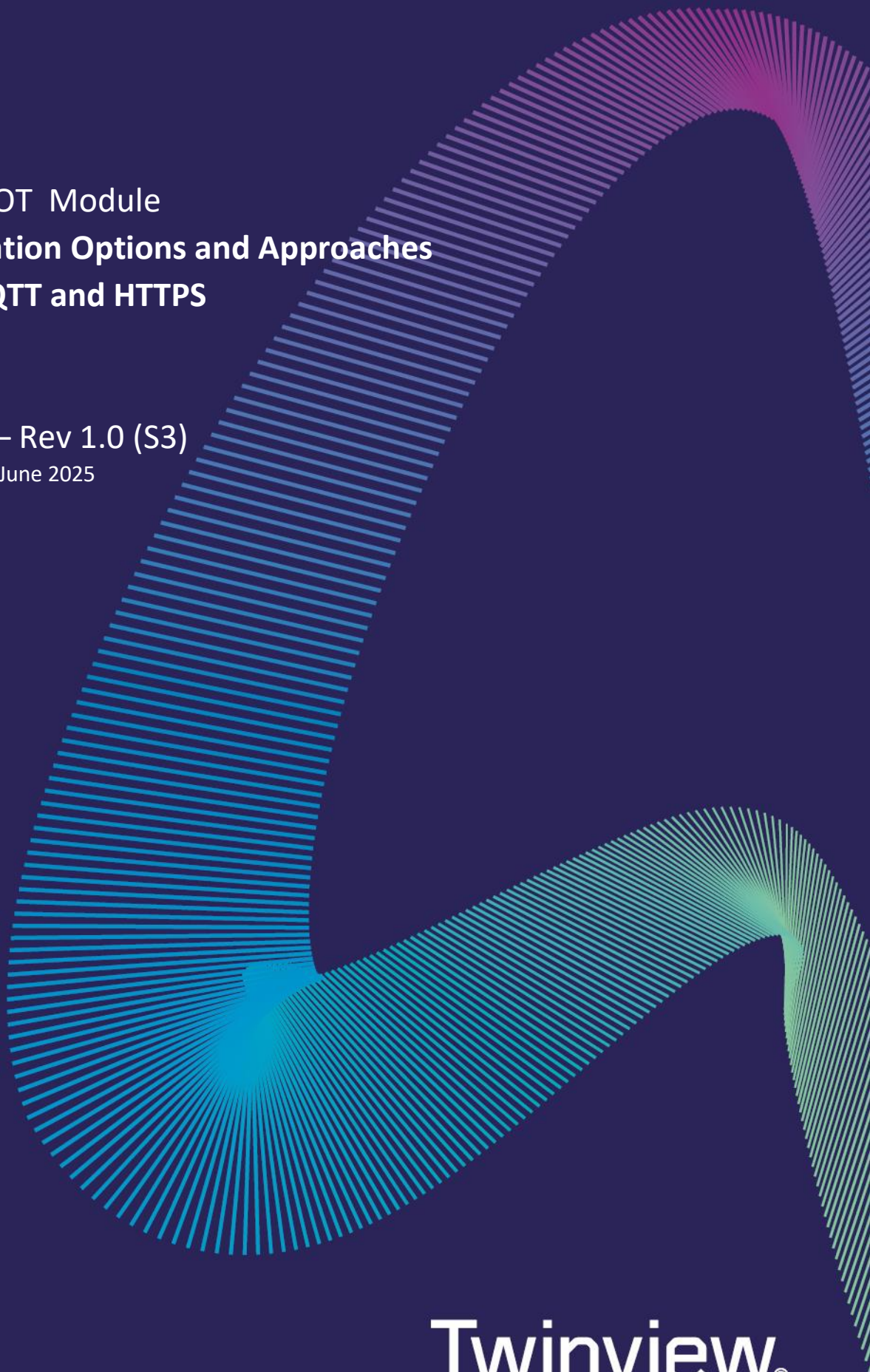
Twinview IOT Module

IOT Integration Options and Approaches
Part 1 : MQTT and HTTPS

June 2025 – Rev 1.0 (S3)

Issue Date: 11th June 2025

Owner: Adam Ward
Status: Approved
Security Level: **PUBLIC**



Twinview®

DOCUMENT REVISION HISTORY

Revision	Date	Author	
Rev 1	11/06/25	Adam Ward	First Issue
Rev 2	27/06/25	Adam Ward	Updated for Security alignment

DOCUMENT STATUS

Revision	Date	Author	
S3	11/06/25	Adam Ward	For Review

SECURITY CLASSIFCATION

Asset Level: PUBLIC
Distribution: Need to Know

This document does not contain sensitive information and may be distributed internally or externally to 3rd parties who need to know the information within.

This document may be a serialised instance for the purpose tracking change and distribution under ISO27001 Clause 7.5 -"Retained Information".

Table of Contents

1. Introduction

1.1. Purpose

This document is designed to give a Systems integrator or the BMS Controls contractor an understanding of Twinview and detailed implementation guidance for ingesting data from Building Management Systems (BMS).

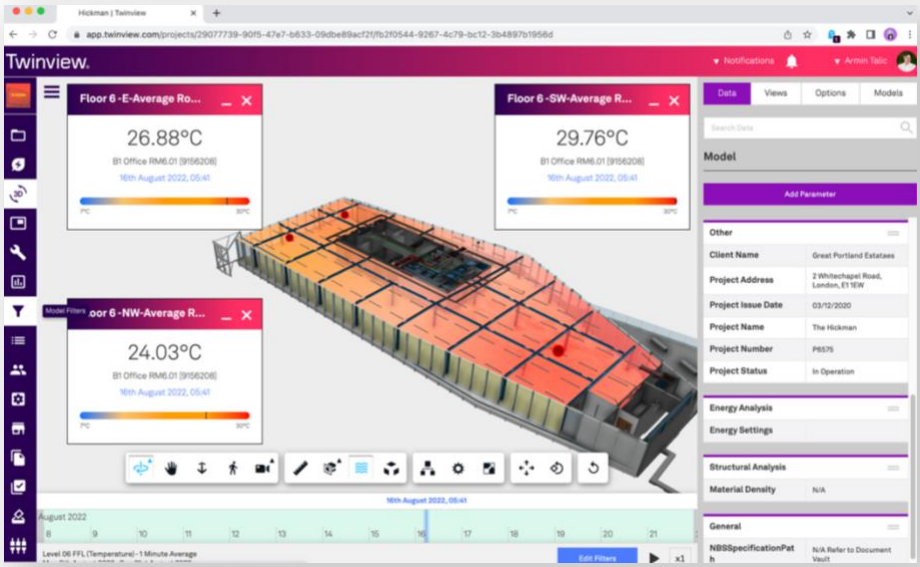
It provides a comprehensive overview of the systems, data, and integration approaches that necessitate the involvement of the Systems Integrator. Additionally, the document covers important aspects such as integration checklists and the Twinview testing requirements to ensure the successful deployment and testing of Twinview platform integrations.

This document will demonstrate several supported approaches to transfer system data from on-prem technologies to the Twinview cloud services in a cyber-safe and efficient manner.

This document is a standard Implementation Guidance for use in early-stage exploration and discussions and once an approach is chosen and defined, it documented in a Project specific Implantation Strategy Document.

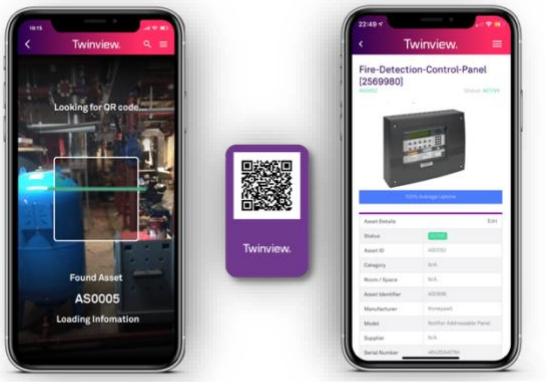
1.2. The Twinview Platform

Twinview, at its core, functions as a federator and aggregator of a property's static, dynamic, and real-time information. This comprehensive data set encompasses a wide range of sources, including IoT data, BMS data, documents, FM jobs, drawings, BIM models, and CAFM, among others. Twinview adeptly comprehends the intricate relationships within this data, offering valuable insights into the operational efficiency of the building through an intuitive and user-friendly interface. This interface is accessible through both web browsers and native mobile applications, providing users with a convenient and accessible way to gain a deeper understanding of their building's performance.



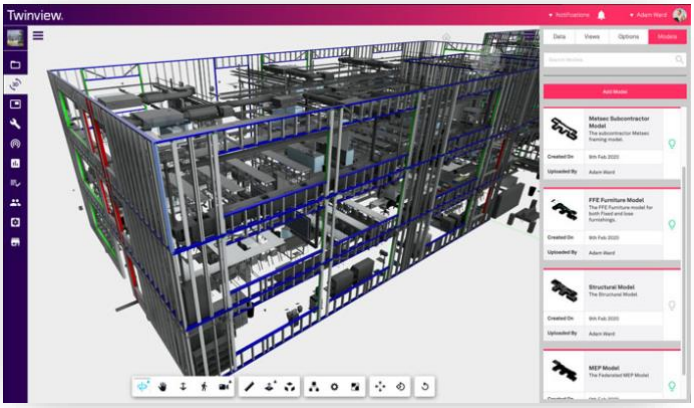
Twinview is modular, and in addition to its core functionality, it may offer additional modules and functionalities. For instance, there are modules for facilities management, document management, and an IoT module that enhances IoT functionality with control and automation capabilities.

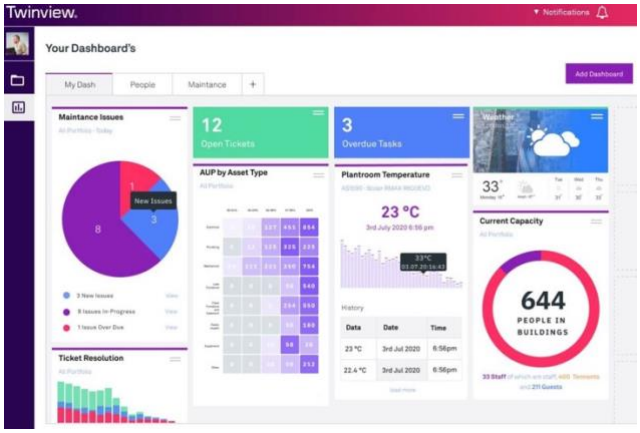
The intended users of Twinview and its modules are diverse, but for the purpose of this document, the primary user of the Twinview platform will be the building owner and the operational teams responsible for managing the building during its operational phase.



It is crucial to understand that the Twinview platform should be viewed as a new layer of digital services built upon existing sub-systems. It should not be considered an inherent dependency of the

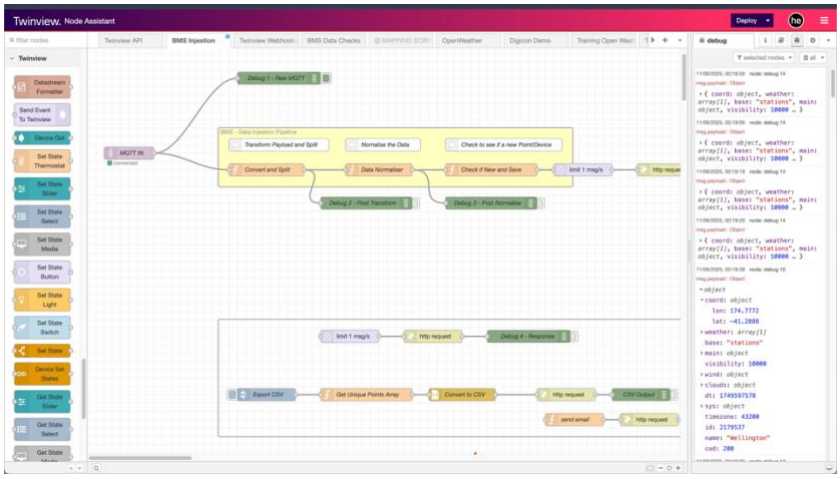
sub-systems, operating autonomously to achieve its specified outcomes, as defined elsewhere. Additionally, Twinview should not be seen as a replacement for these systems.





To facilitate the seamless transfer of system data from on-premises technologies to Twinview cloud services, Twinview will deploy edge gateway devices at the necessary sites to facilitate integration with the sub-systems.

This software layer should be understood as a new layer of digital services built upon existing sub-systems. It should not be considered an inherent dependency of the sub-systems, operating autonomously to achieve its specified outcomes, as defined elsewhere.

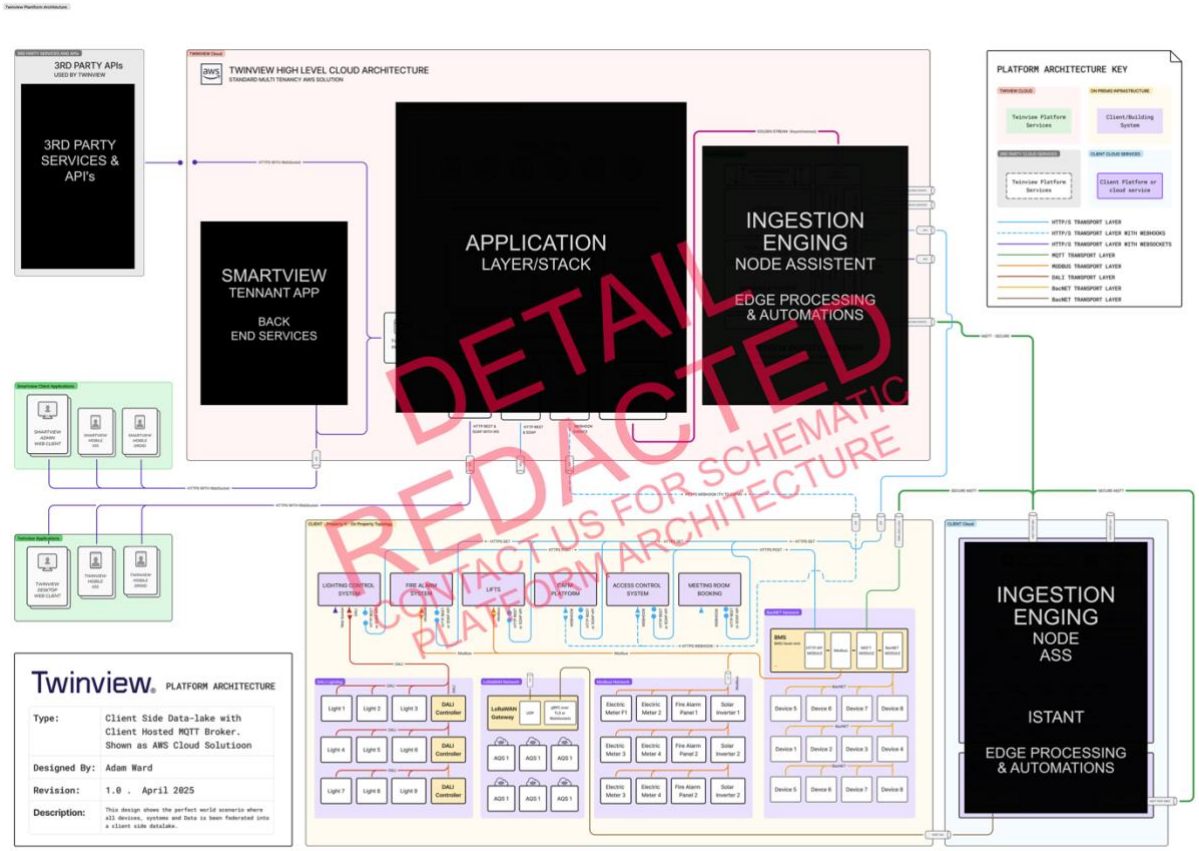


Unlike many digital Twiinview has been designed from the ground up as a SAAS platform,with onboarding as quick as a couple of sdays and not months. Despite this the platform remains highly flexible and hardware agnostic and most of the time not requiring any on site hardware or enforcement specific standards or technologies to get started..

Because of this unique approach Twinviews biggest channel is our partners who use our technology to deliver additional services and ‘in operation’ value to their clients, This allows us to focus on developing the platform and functionality while our partners undertake the implementation.

If you want to learn more about twinview or have a live deminstratuin of our platform you can learn more here: www.Twinview.com or at our partner page here: https://www.twinview.com/partner-program

1.2.3. TWINVIEW SOLUTION ARCHITECTURE



1.2.3. THE MAIN TWINVIEW APPLICATION

For the purpose of this document (System Integrations Overview), the Twinview platform can be thought of as a three-tier cloud-hosted application, with a separate edge Ingestion server handling data ingestion, transformation and normalisation. The 3 tiers are a) front endsvisualisation layer, b) “backend” – application logic, and c) “Database & Storage “. All 3 combined are what we refer to as the Twinview application which is hosted on the Amazon AWS infrastructure.

It’s Main Application a multi-tenancy application that encompasses the front end, back end, databases, but also numerous application controllers, que servers and other and technologies which together for the intellectual property of Twinview.

As a BMS implementation partner, systems integrator, or consultant, you should never directly interact with the primary Twinview application and always via a Twinview approved ingestion engine which has been deployed specifically for the project. The Twinview ingestion server, which can be cloud-hosted or hybrid. This approach moves the processing, transformation, and ingestion of real-time data to the edge of the project.

1.2.4. INGESTION & AUTOMATION ENGINE (AKA. THE NODE ASSISTANT)

Twinview provide a project specific dedicated ingestion server on every project*, and handles the receiving, processing, transformation and ingestion of data from 3rd party systems and sources into Twinview.

The Ingestion Server supports the many the many technologies and protocols used in IOT and building automation and control now and in the future, including HTTP POST/ GET for both REST and SOAP APIs, BACnet IP, Modbus, and LoRawan. In addition, it also visual scripting interface incorporates a logic rule engine for Automations and control.

It also incorporated a Eclipse MQTT server, BACnet Interface, , signage controller, media controller and a security gateway if hosted locally.

The ingestion server runs a special software called 'Node Assistant' which provides a graphics interface and tools to assist you in efficient integration, and ongoing maintance of the integration.

As a 3rd party BMS implementation partner, systems integrator, or consultant the ingestion engine is the touch point for all communications when iy comes to IOT and data ingestion and device control.

The ingestion server does not store of hold any information and is primevally an edge data ingestion server, that receives, processes (transforms, normalises and validates) the data in Realtime before its sent on sent securely onto the mothership (Twinview). This approach is highly horizontally scalable as additional ingestion servers can be deployed if required.

The ingestions server is provided and maintained by Twinview and it could be either cloud hosted (EC3 on AWS) or where necessary a physical device located on premiss (referred to as the Twinview Gateway). Both the hardware and software have been configured and hardened from a security standpoint. The device comes pre-configured with all certificates installed to securely transmit data to Twinview mothership securely and respectfully

As a 3rd party BMS implementation partner, systems integrator, or consultant the ingestion engine is the touch point for all communications when it comes to IOT and data ingestion and device control on a building project..

2. Communication Protocols

Twinview supports numerous ways to integrate

1. MQTT

2. HTTP/S Webservices or API
3. ~~BacNET and Modbus via Hardware Gateway (special use cases only Contact Us)~~
4. The Twinview API

2.2. MQTT

Type: Asynchronous, event-driven

Pattern: Publish - Subscribe

Transport: TLS over TCP (port 8838 for secure)

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe network protocol that is ideal for scenarios where low bandwidth, high latency, or constrained devices are involved. Originally designed for oil pipeline telemetry, MQTT has become a de facto standard for building automation and IoT systems.

Almost all BMS systems, IOT platforms and telemetry platforms can publish to MQTT out the box natively making it a great solution to standardise the implementation across many systems and services.

Key Benefits:

- Efficient: Small packet size and low overhead.
- Scalable: Easily supports thousands of devices and data points.
- Event-driven: Devices publish data only when needed or at regular intervals.
- Decoupled architecture: Publishers and subscribers are unaware of each other, making systems modular and flexible.
- Unlike HTTP can support 10,000s real-time of messages per second, removing the need for bulking of messages or a heartbeat control stream

Important Considerations:

- Topic design is crucial: MQTT doesn't have inherent discovery or querying capability. Good topic structure is essential for efficient data consumption.
- Wildcard support: MQTT supports + (single level) and # (multi-level) wildcards for flexible subscription filtering.
- Security: Use authentication (e.g. TLS with client certs) and authorization (topic-level access control).
- Retained messages: Can be used to store the latest state of a topic.
- Data format: JSON is recommended for payloads due to its readability and support in most programming environments.
- QoS (Quality of Service) Levels: Supports different reliability levels for message delivery.

This following section outlines the setup and naming conventions for MQTT topics in a commercial office building located in London, although the structure is designed to be scalable, enabling seamless integration of multiple properties in future.

2.2.3. HOSTING OPTIONS FOR AN MQTT BROKER

Choosing the right hosting option for the MQTT broker depends on the client's infrastructure, scalability needs, security policies, and management preferences. Here are the common options:

2.2.3.1. ON-PREMISES HOSTING

Description: Broker is hosted on the clients own internal servers as part of the buildings ICT.

Benefits:

- Full control over security, data, and infrastructure
- Low latency for LAN-based devices
- No reliance on internet connectivity

Disadvantages:

- Requires in-house expertise to manage and secure
- Higher setup and maintenance costs
- Less scalable without additional investment

2.2.3.2. CLOUD-BASED HOSTING (SELF-MANAGED)

Description: Deployed on cloud infrastructure like AWS, Azure, or Google Cloud, managed by clients own team.

Benefits:

- Scalable infrastructure
- Easier to deploy geographically distributed solutions
- Reduces hardware burden

Disadvantages:

- Still requires internal team for configuration, monitoring, and patching
- Ongoing cloud service costs
- Network latency and bandwidth need to be managed carefully

2.2.3.3. MQTT AS A SERVICE (E.G., HIVEMQ CLOUD, EMQX CLOUD)

Description: Fully managed MQTT brokers provided as a hosted service.

Benefits:

- No infrastructure or maintenance burden
- High availability and reliability built-in
- Built-in dashboards, analytics, and integration features

Disadvantages:

- Recurring subscription fees
- Less control over underlying infrastructure
- Potential data residency and compliance concerns

2.2.3.4. INTEGRATION INTO CLIENT'S EXISTING DATA LAKE OR IOT PLATFORM

Description: The MQTT broker is embedded into or closely integrated with an existing enterprise platform, such as a client's Azure IoT Hub, AWS IoT Core, or a custom ingestion layer in their data lake or pipeline.

Benefits:

- Simplifies data ingestion and analysis - messages flow directly into analytics and storage platforms.
- Leverages existing security, authentication, and governance models.
- Reduces integration complexity with other enterprise systems.
- Can enable end-to-end observability and traceability.

Disadvantages:

- Limited control over broker internals, depending on the platform.
- Potential for higher latency if data lake isn't close to the source.
- May depend on third-party teams to make configuration changes.
- Could introduce tight coupling between systems, reducing flexibility.

The client of property owner should select a hosting option based on the project's scale, required uptime, IT support capacity, and long-term integration strategy. Hybrid approaches are also possible — for example, using a cloud broker for remote buildings and on-prem brokers for local control systems.

Following these basic principles ensures robust and scalable MQTT integration across your building automation systems.

2.2.4. TOPIC STRUCTURE DESIGN

With over 10,000 points from systems such as BMS, energy/gas/water meters, lighting, alarms, and air quality sensors, a consistent and hierarchical topic structure is essential. The goal is to design a topic hierarchy that allow flexibility to subscribe to:

- A single point
- All points for a piece of equipment
- All points for a room, floor, or the entire building
- All points by system type (e.g., HVAC, energy, lighting)

Principles of Topic Design

1. Hierarchical: MQTT topics should follow a clear and hierarchical structure, using forward slashes (/) to denote levels.
2. Scalable: Include placeholders for multiple properties and zones.
3. Readable: Topics should be human-readable and self-explanatory.
4. Consistent: Use lowercase, hyphenated identifiers. Avoid spaces or special characters.
5. Secure: Ensure authentication and access control is enforced per topic structure where needed.

Twinview's Recommended Topic Schema:

```
[building]/[floor]/[room]/[system]/[equipment]/[point_type]/[point_id]
```

Level	Description
building	Unique building ID or name (e.g. bldg01)
floor	Floor identifier (e.g. lvl01, lvl02)
room	Room or zone identifier (e.g. rm103, external, riser_01)
system	System type (e.g. hvac, lighting, energy, water, gas, iaq, alarm)
equipment	Equipment identifier or tag (e.g. ahu1, vav5, meter3)
point_type	sensor, setpoint, status, alarm, limit, etc.
point_id	Unique point ID or short tag (e.g. temp, co2, power, flow)\

Examples:

```
bldg01/lvl01/rm103/hvac/vav5/sensor/temp
 bldg01/lvl02/openplan/lighting/ctrl4/status/state
 bldg01/lvlB1/plantroom/energy/meter1/sensor/power
 bldg01/lvl01/#                // All data from level 1
 bldg01/+/+/hvac/+/sensor/#    // All HVAC sensor data across all floors and rooms
```

To maintain clarity, avoid these common mistakes:

- **Don't use inconsistent naming conventions**: Mixing FirstFloor, 1st_Floor, and Floor1 leads to confusion.
- **Avoid spaces or special characters:** Stick to alphanumeric, _ or -. MQTT topics are case-sensitive and must be URL-safe.
- **Don't overuse wildcards in permanent subscriptions:** This can lead to bandwidth strain or processing overhead.
- **Don't use deep nesting unnecessarily:** Keep the topic hierarchy manageable and meaningful.
- **Avoid including values in topic names:** Use payloads for sensor readings (e.g., temperature = 21.5°C), not in topic names.

2.2.5. PAYLOAD FORMAT

It’s best to use JSON for consistency and extensibility.

Example Payload:

```
{
  "name": "Classroom_01_Temp",
  "value": 21.5,
  "unit": "°C",
  "timestamp": "2025-04-14T12:30:45Z",
  "status": "{ok}"
}
```

All points should send the Point Name, and Vale as a minimum, however it is highly recommended to include several other items if they are available in the BMS.

Key Name	Example	Notes
name	AHU001_Temp	Point Name as string (URL Encoded)
value	25.40	Vale as a number(float), integer, string, Boolean or array
unit	°C	Prefix or suffix, if not applicable use :”n/a” or leave empty.
timestamp	2025-04-14T12:30:45Z	Date & time in ISO 8601 Format. If omitted time of
status	"{ok}"	This is the Point health health from the BMS

2.2.6. SPECIAL TOPIC CATEGORIES

System Health Topics
Publish system diagnostics (e.g., connection status, last seen time, firmware version) under a dedicated topic branch:

```
bldg01/systemstatus/hvac/vav5
```

Metadata Topics

Optionally publish static data like equipment tags, descriptions, and engineering units to a meta branch:

```
bldg01/meta/lvl01/rm103/hvac/vav5/temp
```

2.2.7. MQTT MESSAGE SETTINGS

2.2.7.1. RETAINED TOPICS

Use retained messages for points like setpoints or status so new subscribers get the last value immediately.

2.2.7.2. QUALITY OF SERVICE (QOS)

Choose QoS level based on the importance of data delivery reliability:

- QoS 0: At most once (lowest reliability for non critical points)
- QoS 1: At least once (recommended default)
- QoS 2: Exactly once (highest reliability, more overhead only when critical)

2.2.8. BEST PRACTICE SUMMARY

- **Design topics hierarchically** to support both granular and aggregate subscriptions.
- **Use consistent naming** conventions for floors, rooms, and systems.
- **Use JSON payloads** with clearly defined fields and units.
- **Make use of wildcards** to allow for flexible data consumption patterns.

- **Apply security best practices**: encrypted connections, user authentication, topic-level access control.
- **Use retained messages** for persistent values (e.g., setpoints, statuses).
- **Document your structure**: publish a schema or topic map for integrators and developers.

2.2.9. MQTT WRAP UP

MQTT is the preferred integration approach for integrating real-time Data into Twinview as it is a modern highly scalable solution with the following benefits:

Highly Scalable:

One ingestion server can support 100,000 datapoints with a maximum ingestion/message rate of 10,000 per second.

Twinviews Node Assistant:

The Twinviews node assistant can be integrated with ECLYPSE MQTT server or can integrate into client or third-party MQTT server or service.

Topic and Pub/Sub Model:

The topic and pub/sub model is highly efficient and flexible, with authorisation and control managed centrally. This includes certificates, encryption, and hardening techniques.

2.3. Cloud-to-Edge Communication over HTTPS

Type: Synchronous

Pattern: Request-Response

Transport: Runs over HTTP (port 80 or 443 for HTTPS)

With this approach the on-premises Building Management System communicates with the Twinview ingestion server over the internet using webservices and specifically the HTTPS method. With this approach both the BMS and the Twinview Ingestion Server communicate over the internet using a "Request – Response" approach. This is knownas a **client- server** architecture when the BMS and Twinview communicate directly.

The on-premises Building Management System communicates with the Twinview ingestion server (cloud or Physical Server) over HTTPS-based on standard POST/GET REST API calls, this is the same technology used when accessing a website in a browser.

2.3.3. HTTP METHODS (POST AND GET)

2.3.3.1. HTTP POST

The ingestion server supports the ability to receive data directly from the BMS or System via the HTTPS POST method. In HTTPS POST method data is posted from the BMS to Twinview on a regular set 'heartbeat' or 'on-change' when a value changes

2.3.3.2. HTTP POST

The ingestion server also supports the HTTPS GET method to retrieve data from the BMS or System. With this method the Twinview Ingestion server makes HTTP GET request and when received the BMS responds and returns the data in a payload.

Where possible it is always recommended to use HTTP POST to 'post' the data to the Twinview Ingestion Server. This removes the need to configure the buildings network or allow Twinview access inside the network to make the GET request of the BMS or system.

2.3.4. PAYLOAD FORMAT FOR POST REQUEST

Data can be sent to the Twinview ingestion server in either a JSON or XML and must be sent in the BODY of the HTTP message. JSON should always be the first choice as this is more efficient.

When send data via the HTTP approach, it is important to manage the number and frequency of requests been made, and all the required points should data been sent on a regular heartbeat with each point should be defined as a JSON child object, inside a single JSON Payload containing all points. You can see an Example in section 2.3.4.3 of this document.

This example is best practice and to provide guidance only. Twinview does not enforce a schema for the JSON Data or XML schema it expects to receive, although it must be a valid. The Twinview

implementation team or implementation partner have tools in Node Assistant software to allow transformation of the data in real-time as it is ingested.

Handy Tip: *On larger buildings, where the number of points exceed 50,000, the payload size may exceed the maximum payload size. If this is the case, you can chunk the payload down into 2, 3 or 4 separate payloads split equally or by point type etc. If you need to chunk the payload then, it is important that all they are all sent om the same 15 minute heartbeat.*

All point values must be sent with every heartbeat, regardless of whether the value has changed since the last transmission. This ensures that Twinview maintains a complete and up-to-date baseline of all point statuses—including those that rarely change, such as a 'Filter Status'. Relying solely on 'on-change' may result in certain points not reporting their status for extended periods—potentially years—until a change occurs. The heartbeat-based approach guarantees visibility of all points within Twinview as well as allow Twinview to establish and report point health and status of points. This why this 'On-heartbeat' payload is called the 'Control DataStream, as it's the baseline data.

2.3.4.1. FREQUENCY FOR POST REQUESTS ON HEARTBEAT

It is important that the heartbeat frequency is consistent on a REGUALR interval. We recommend every 15 minutes as this is aligned to industry standard reporting methodology for Energy, and Sustainability reporting, including Energy score, DFE K2N (Department of Education) etc.

2.3.4.2. EXAMPLE HEADER REQUEST FOR ON HEARTBEAT

When sending JSON data from a Building Management System (BMS) to an external system (like a cloud digital twin), the **HTTP request header** should include certain key fields to ensure the receiving server understands the format, accepts the source, and can validate or process the request.

Here's a example of HTTP headers you would include in a POST request when sending JSON data:

Key Name	Example	Notes
Content-Type:	application/json	**Required** - Tells the server the body content is JSON
Accept:	Accept: application/json	**Required** - Indicates the client expects a JSON response
x-api-key	d589934e59a764185150c42dff4d74b303b2fda7d681b6039d123f61d9522d8f0073e740c3c98255	**Required** - The API token as provided by Twinview. This token validates who Is sending the data has permission to do so as well as the project.
x-bms-system	Tridium-01	**Optional** - Custom header to identify the source system (e.g., EcoStruxure, Tridium, etc.)
x-data-schema-version	1.0	**Optional** -Custom header to for Versioning for the payload schema (useful for forward compatibility)

This is an example HEADER of the HTTP request :

```
POST <NODE-ASS-BASE-URL>/ingestion/v1/heartbeat HTTP/1.1     //Endpoint to Post Data
Host: 123.45.67.89                          // Added automatically by the host
Content-Type: application/json              // Tells the server the body content is JSON
Accept: application/json                    // Indicates the client expects a JSON response
x-api-key: <your api key>                   // Provides an API token as provided by Twinview.
x-bms-system: EcoStruxure-BMS-01            // Optional Custom header to identify the source system
x-data-schema-version: 1.0                  // Optional Custom header to for Schema Versioning
Content-Length: <calculated when sent>      // Tells the server the body content is JSON
```

2.3.4.3. EXAMPLE BODY REQUEST FOR ON HEARTBEAT

All points should send the Point Name, and Vale as a minimum, however it is highly recommended to include several other items if they are available in the BMS.

Key Name	Example	Notes
name	AHU001_Temp	Point Name as string (URL Encoded)
value	25.40	Vale as a number(float), integer, string, Boolean or array
unit	°C	Prefix or suffix, if not applicable use :”n/a” or leave empty.
timestamp	2025-04-14T12:30:45Z	Date & time in ISO 8601 Format. If omitted time of
status	"{ok}"	This is the Point health health from the BMS

This is an example JSON sent in the BODY of the HTTP request:

```
[
  {
    "name": "Classroom_01_Temp",
    "value": 21.5,
    "unit": "°C",
    "timestamp": "2025-04-14T12:30:45Z",
    "status": "{ok}"
  },
  {
```

```
    "name": "Classroom_01_C02AQI",
    "value": 662,
    "unit": "PPM",
    "timestamp": "2025-04-14T12:30:45Z",
    "status": "{ok}"
  },
  {
    "name": "AHU001_AlarmStatus",
    "value": true,
    "unit": "{false:okay, true:inalarm}",
    "timestamp": "2025-04-14T12:30:45Z",
    "status": "{fault}"
  },
  {
    "name": "AHU001_BoostMode",
    "value": 0,
    "unit": "{0:okay, 1:inalarm}",
    "timestamp": "2025-04-14T12:30:45Z",
    "status": "{ok}"
  },
  {
    "name": "BEU001_Boiler002_Mode",
    "value": 3,
    "unit": "{0:off, 1:normal, 2:ecomode}",
    "timestamp": "2025-04-14T12:30:45Z",
    "status": "{ok}"
  }
]
```


2.3.5. LIMITATIONS AND BEST PRACTICES

The disadvantage of the batching the data to a 15-minute heartbeat is that any alarms or real-time points are no longer real-time and may be a 15 minute delay fromthe alarm been activated to when the alarm or value is received by Twinview.
Although one solution might be to send data on-change, using REST APIs with Request-Response model for near real-time data ingestion can introduce scalability challenges, especially regarding rate limits and maximum concurrent connections. Most API gateways and cloud services, including those hosted on AWS, impose limits on how many requests can be made per second or minute. If the BMS sends high-frequency updates — such as readings from multiple HVAC zones, meters, or occupancy sensors — the number of HTTP requests can quickly exceed these thresholds. This results in throttling, delayed data, or failed requests.

To overcome these issues, a Hybrid approach may be taken:

2.3.5.1. BATCH DATA AND SEND ON SET HEARTBEAT

First, batching data into fewer, larger payloads as described above can reduces the total number of HTTP requests. For example, instead of sending 100 readings per second as individual calls, a single request could send a JSON array of all readings collected within a set time window e.g. every 15 minutes.

2.3.5.2. SEPARATE ENDPOINT FOR ALARM AND REAL-TIME UPDATES

For a select number of devices or points where it is important that any update is reflected in Twinview Realtime a 2nd on-change endpoint exists where the BMS/3rd party system can publish updates to 'on-change' rather than set-heartbeat.

The on-change endpoint should only be used for points that are real-time critical, such as alarms and events. The payload should be sent as a single JSON object (as below) and not a batch multiple as the Heartbeat endpoint.

```
{
   "name": "Classroom_01_Temp",
   "value": 21.5,
   "unit": "°C",
   "timestamp": "2025-04-14T12:30:45Z",
   "status": "{ok}"
 }
```

2.3.6. INGESTION ENDPOINTS FOR POSTING VIA HTTP POST

2.3.6.1. ON-HEARTBEAT (CONTROL ENDPOINT)

The endpoint to send batch values on set heartbeat:

```
{INGESTION_BASE_URL}/ingestion/v1/prod/heartbeat/
```

Key Name	Notes
Base URL	Unique to your Ingestion server, provided by Twinview

Rate Limits	**Maximum –** 1 message every 1 minute **Recommended** – 1 message every 15 minutes.
Maximum Payload. Size	**Default** - 2.5mb
Response Codes	**Optional** – 200 Success, 208 Already Reported, 400 – Bad Request, 401 – Un-authorised, 403 Forbidden, 401 Not Found, 403 – Content too large, 418 – Im a Teapot, 429 – Too Many Requests, 500 – Internal Server Error, 502 – Bad Gateway, 504 – Gateway. Timeout. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status for more info

2.3.6.2. ON-CHANGE (REAL-TIME ENDPOINT)

The endpoint to send select batch values on-change

```
{INGESTION_BASE_URL}/ingestion/v1/prod/onchange/
```

Key Name	Notes
Base URL	Unique to your Ingestion server, provided by Twinview
Rate Limits	**Maximum –** 1 message every 1 second
Maximum Payload. Size	**Default** - 1.5kb
Response Codes	**Optional** – 200 Success, 208 Already Reported, 400 – Bad Request, 401 – Un-authorised, 403 Forbidden, 401 Not Found, 403 – Content too large, 418 – Im a Teapot, 429 – Too Many Requests, 500 – Internal Server Error, 502 – Bad Gateway, 504 – Gateway. Timeout. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status for more info

2.3.7. USING HTTP GET

Rather than the data been sent (POST) to Twinview on a set heartbeat, some 3rd party BMS or systems Where HTTP POST is not possible by the BMS or service, and HTTP GET is available and you can define the frequency to make a request.

Supported Payloads Formats

Format	MIME Type	Description
JSON	application/json	Most common format for APIs
XML	application/xml or text/xml	Used in legacy or enterprise systems
HTML	text/html	When data is rendered as a webpage

Format	MIME Type	Description
CSV	text/csv	Useful for exporting tabular data
Plain Text	text/plain	Debugging, logs, or raw responses

2.3.7.1. NOTES AND BEST PRACTICES

- Use HTTPS to protect GET request URLs, especially when parameters contain sensitive data.
- Avoid sending credentials or tokens in the query string
- Do not use GET for actions that change state (e.g., rebooting, writing configs — use POST or PUT instead).
- Always check API documentation for allowed query parameters and required headers.

3. THE TWINVIEW REST API

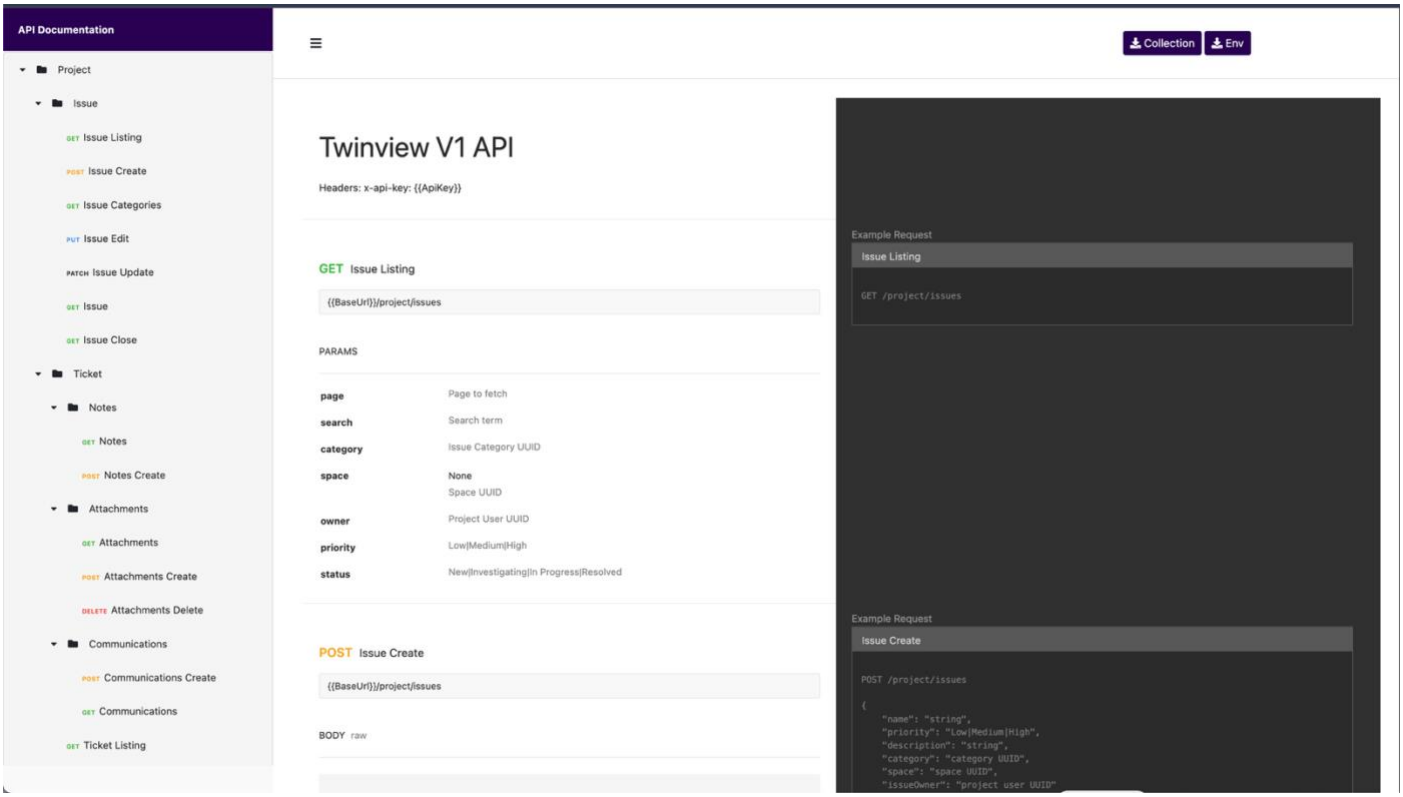
Type: Asynchronous

Pattern: Request-Response

Transport: Runs over HTTP/HTTPS (port 80 or 443 for HTTPS)
Supported: POST, GET, POST, PUT, DELETE
Websocket Support: Ingestion Endpoints
API Version: v2
Architecture: REST
Authentication: API Key

API Endpoint Documentation: https://docs.twinview.com/

Type: Twinview exposes a RESTful API you interact via HTTPS endpoints using methods like GET, POST, etc.—and returns data in JSON (primarily).



It gives you access to the data withing Twinview and the ability to read/write this programmatically, providing access to data streams, device states, telemetry, and configuration settings.

4. WEBHOOKS (EVENTS)

Type: Asynchronous

Pattern: Request-Response

Transport: Runs over HTTP/HTTPS (port 80 or 443 for HTTPS)
API Version: v2
Architecture: Event Driven
Authentication: n/a

Twinview offers a range of webhooks that enable direct integration with the Twinview application migrating the need for the complexity or overhead of Node Assistant.

These webhooks allow the platform to send event-driven messages triggered by specific actions or changes—directly to your own systems or services.

While Node Assistant can generate webhooks for virtually any event using Node Assistant, Twinview also provides a set of predefined FM-related webhooks in the main application.

These can be configured by users with the appropriate permissions or roles (typically Project Admins), enabling straightforward integration without needing to set up custom Node Assistant automations.

For example, you might use a webhook to notify your platform when a ticket is created, updated, or closed sending a payload with the data when this happens. Fo example raising a Issue automatically in you rown CAFM system when and issue is created in Twinview.

We're actively expanding this functionality, with new events and triggers being added regularly and as such written documentation is limited.

To view the full list of available webhooks and configure them for your project, go can to your project settings and click the **'Webhook'** button located at the bottom left of the modal, as shown below images.

It should be noted that the Webhooks available may dependant of which Twinview models are activated on your account and also mirror you user permissions and aligned to any restrictions in the permissions/Role.

Please be aware that any webhook URL must be HTTPS by default to ensure data is encrypted in transit and following security best practice.

Below are the current supported Webhook events:

Issues

Issue Create

Issue Updated

Issue Asset Deleted

Issue Attachment Created

Issue Attachemnt Deleted

Issue Note Created

Maintenance Job

Maintenance Job Created

Maintenance Job Updated

Maintenance Job Issue Created

Maintenance Job Issue Deleted

Maintenance Job Attachment Created

Maintenance Job Attachment Deleted

Tasks

Task Created

Task Updated

Task Asset Created

Task Asset Deleted

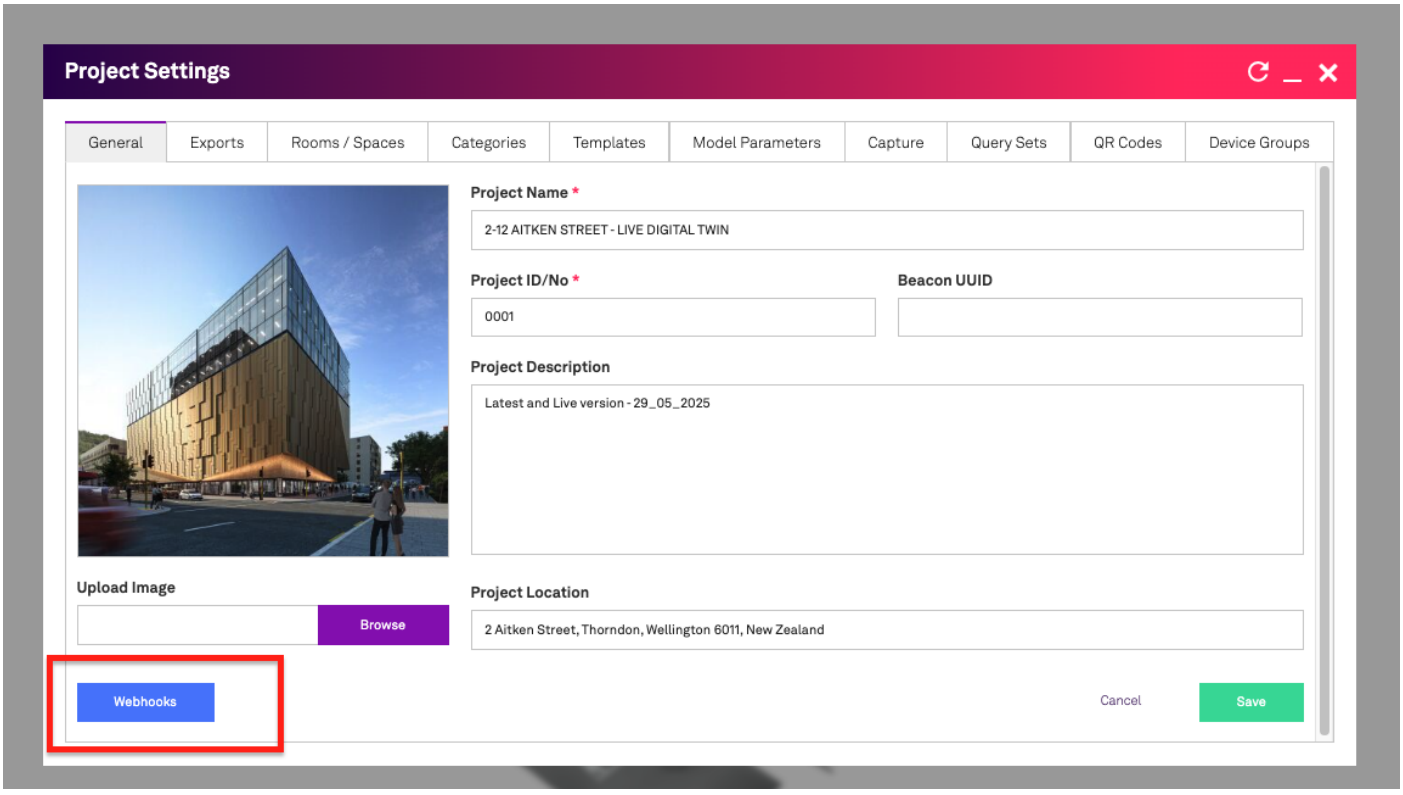
Task Equipment Created

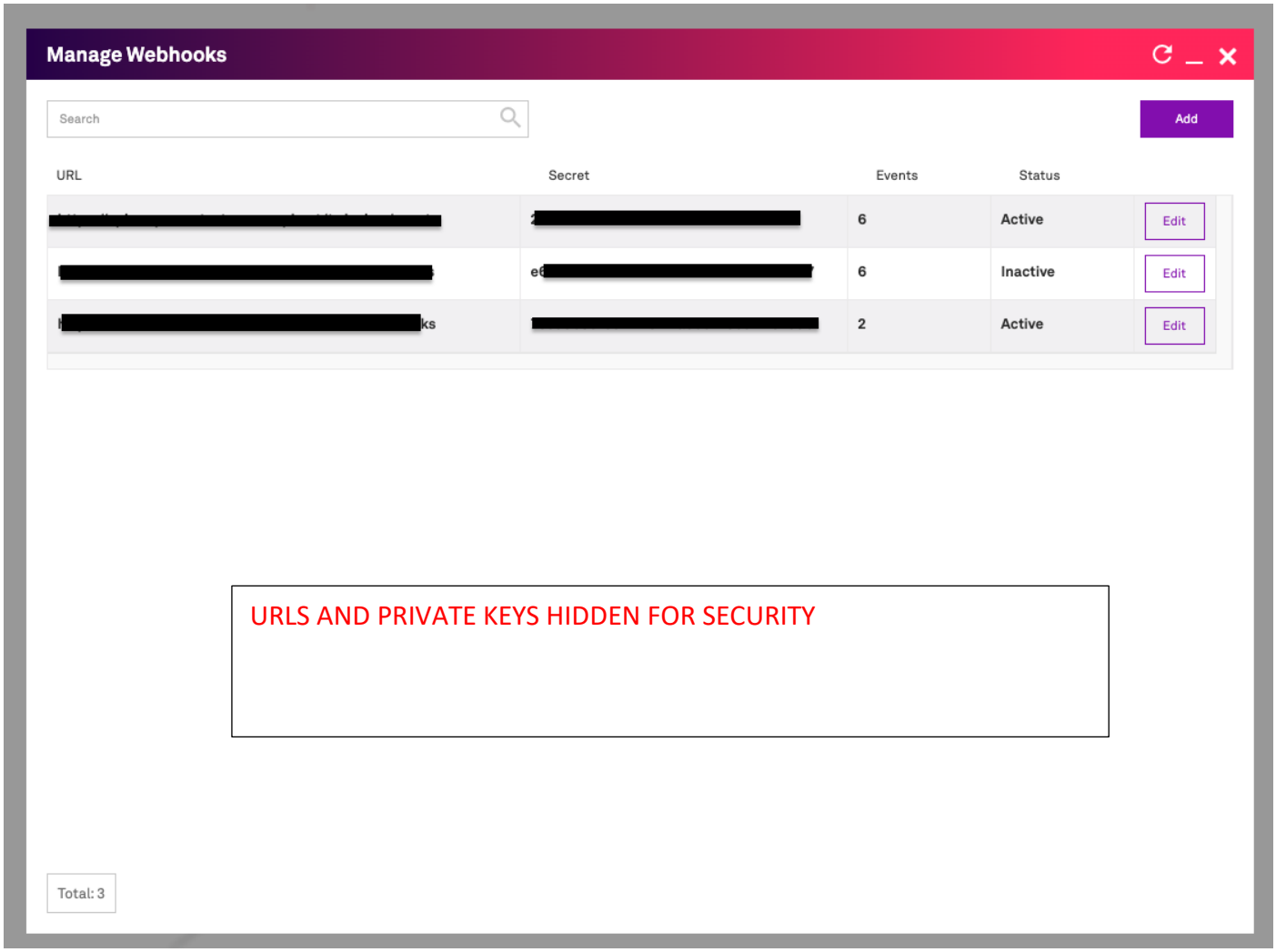
Task Equipment Deleted

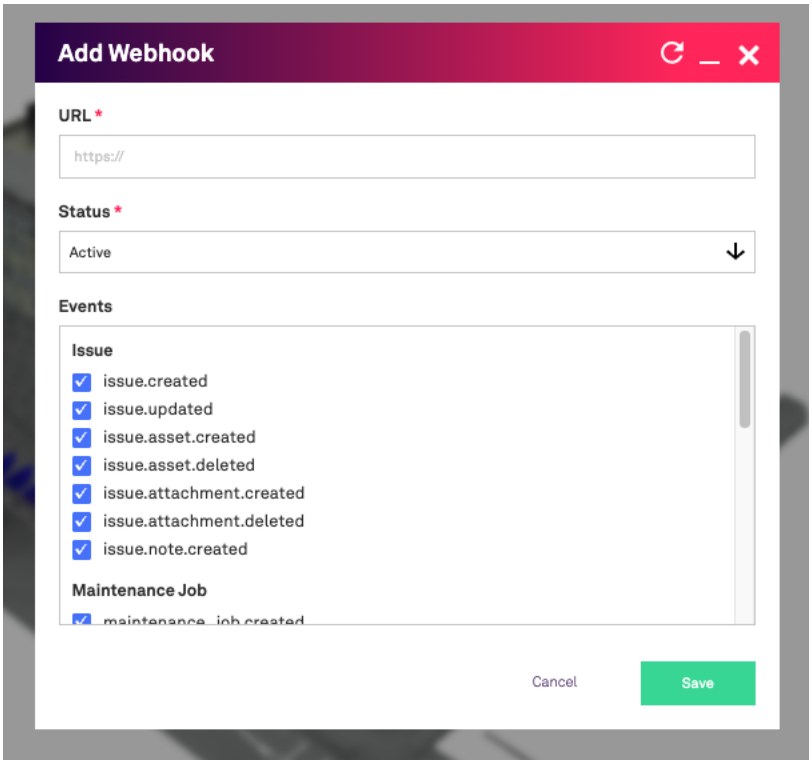
Task Attachment Crated

Task Attachment Deleted

Task Note Created




Manage Webhooks
Search
Add
URL
Secret
Events
Status
6
Active
Edit
6
Inactive
Edit
2
Active
Edit
URLS AND PRIVATE KEYS HIDDEN FOR SECURITY
Total: 3


Add Webhook
URL *
https://
Status *
Active
Events
Issue
issue.created
issue.updated
issue.asset.created
issue.asset.deleted
issue.attachment.created
issue.attachment.deleted
issue.note.created
Maintenance Job
Cancel
Save

6. IMPLEMENTATION AND TRAINING

Twinview have several partner resources and services available and can provide comprehensive training and support on the above integration methods and approaches.

We offer support to suit every partner type, from just providing the software platform, to a turnkey planning, design and implementation service or custom development.

Selection of our support Services:

1. Training
2. Integration Consultation, Workshops and Strategy Planning.
3. Project Specific Implementation Documentation and design.
4. Full Turnkey Implementation service.
5. Custom platform development

if you'd like to explore integration options or simply need some advice; our technical team is here to help.

We're always happy to support our partners—whether you're looking for guidance, exploring potential solutions, or just want to talk through the best approach for your or your client's needs.

Feel free to reach out to any member of our team below for a conversation or tailored advice.

Dan Townsley
Customer Success Manager
dan.townsley@twinview.com

Neil Hancock
Commercial and Partnership Manager
Neil.hancock@twinview.com

Adam Ward
Chief Technology Officer
adam@twinview.com