Industrial-strength MQTT/Sparkplug B

Building industrial MQTT networks at scale with edge computing

OPTO 22

The Edge of Automation.™

Opto 22

43044 Business Park Drive • Temecula • CA 92590-3614
Phone: 800-321-6786 or 951-695-3000
Pre-sales Engineering is free.
Product Support is free.
www.opto22.com

Form 2357-210114

INDUSTRIAL-STRENGTH MQTT/SPARKPLUG B

Building industrial MQTT networks at scale with edge computing

INTRODUCTION

Since 2015, MQTT has consistently ranked as the most popular internet of things (IoT)-specific messaging protocol in the Eclipse Foundation's annual IoT Developer Survey. An open-source OASIS/ISO standard, MQTT is used in many consumer applications, like mobile chat, home automation, and car-sharing. It is supported by major cloud computing and IoT platforms for enterprise applications in smart energy, health, and banking services as well. And more recently, MQTT has gained traction within the manufacturing and processing industries.

However, there are several obstacles to bringing the power of MQTT to bear in an industrial environment. MQTT's innate flexibility is also a potential drawback, requiring stronger guarantees of interoperability and state management to meet the needs of a diverse industrial network. Likewise, the integration of disparate device protocols cannot be addressed purely through MQTT, given its current level of support in field devices and the long lifespan of legacy systems. And while MQTT goes a long way to addressing fundamental cybersecurity issues, MQTT by itself isn't sufficient to create a secure industrial IoT (IIoT) infrastructure.

For system integrators, developers, engineers, and managers wondering what MQTT can do for them, this paper explains the fundamentals of the protocol, demonstrates how the Sparkplug B specification adapts MQTT to industrial applications, and shows how to establish and scale MQTT networks using industrial edge computing. This approach creates a secure foundation for IIoT networks that scale up smoothly and operate reliably in support of mission-critical applications.

FUNDAMENTALS OF MQTT V3.1.1

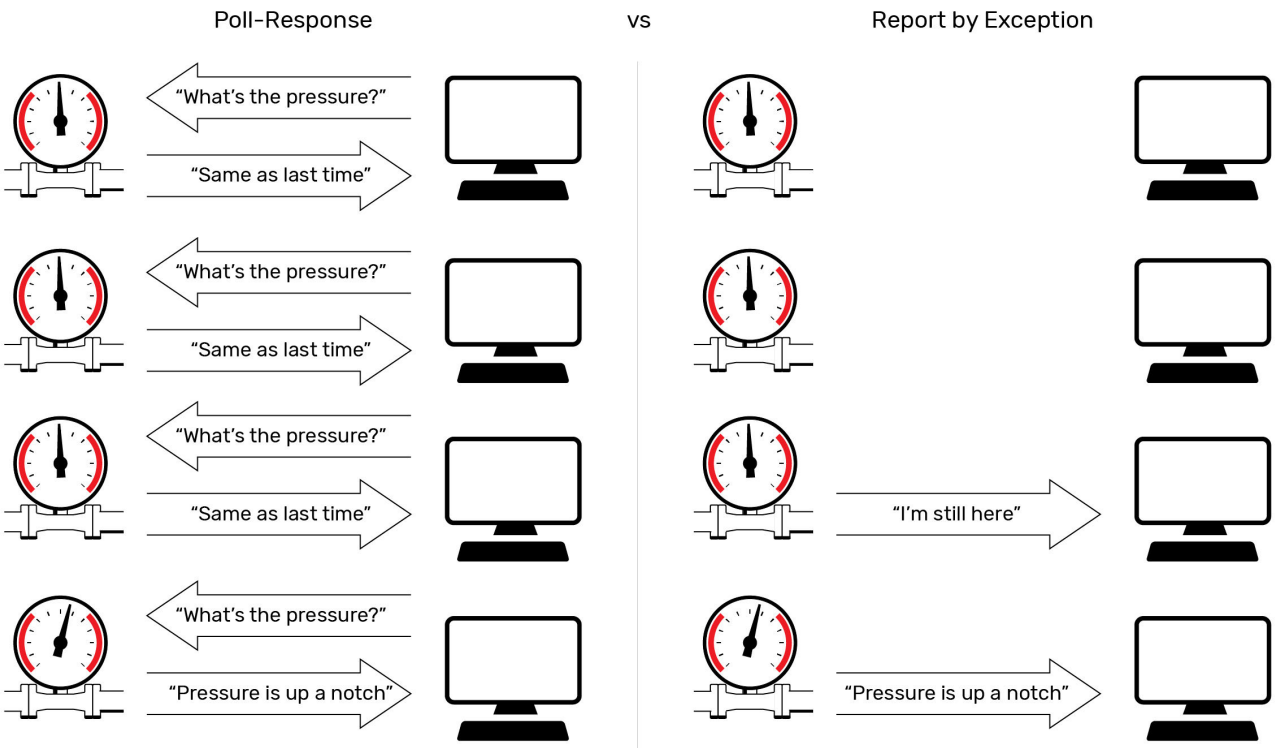
MQTT (formerly MQ Telemetry Transport) was designed for industrial networks. In the 1990s, ConocoPhillips (now Phillips 66) needed a way to improve telemetry reporting over its costly, low-bandwidth dial-up and satellite SCADA infrastructure. IBM partnered with system integrator Arcom Control Systems (now Cirrus Link Solutions) to develop a minimalist communication protocol that gracefully handled intermittent network outages and high latency among distributed devices over TCP/IP.

With efficiency and stability as principal goals, IBM and Arcom made a few critical decisions in designing the MQTT protocol, including the use of a publish-subscribe architecture, a message payload with very low overhead, and mechanisms for dealing with client disconnections.

Publish-subscribe communication

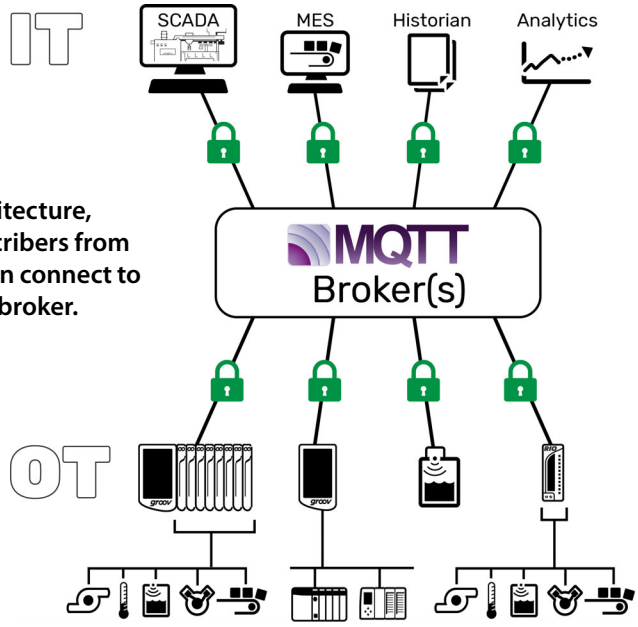
Traditional communication models use a *poll-response* mechanism to interrogate field devices and maintain state awareness. Data is refreshed when a master device or application issues a request for new data, causing field devices to respond with the requested data. Queries are often triggered cyclically based on timing parameters.

Poll-response protocols generate a lot more network traffic than report-by-exception.



IBM abandoned this model in favor of a brokered *publish-subscribe* model that is central to the strength of MQTT. In that scheme, a central server acts as a broker, managing data delivery for the entire network.

In the MQTT architecture, publishers and subscribers from across the organization connect to a common data broker.



Rather than being prompted by command, MQTT-enabled field devices and gateways publish data to the broker only when they detect a change in a monitored value, a behavior known as *report-by-exception*. Other clients, including software and other field devices, can register with the broker as subscribers to any data published on the network.

Since some devices may not need to publish very often, state awareness is maintained by periodically sending a small keep-alive packet to the broker, in addition to other mechanisms that we'll discuss later.

Flexible topic paths

Although each client identifies itself to the broker with a unique *client ID*, with the broker as the one and only endpoint, there is no device addressing scheme. Instead, clients identify published data using individual, hierarchical *topic paths* represented as plain text strings.

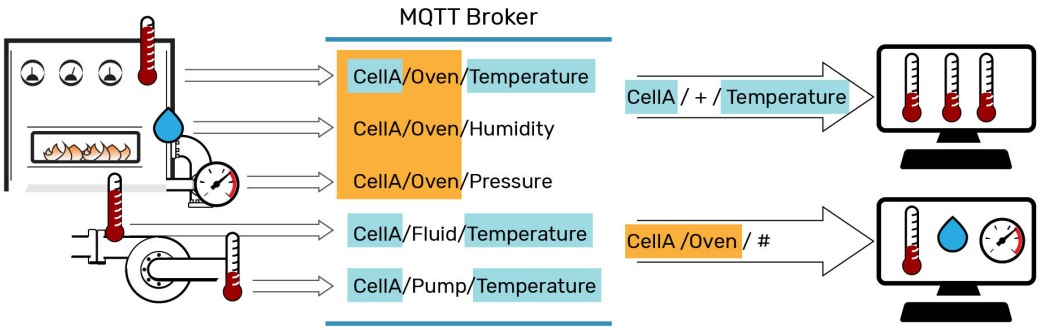
For example, `CellA/Oven/Temperature` could represent the internal temperature of an oven in a particular work cell. A client could subscribe to that topic by using the exact path as a *topic filter* or could enter a more general filter to capture related data as well.

For instance, there might be other devices in that same cell that also publish temperature data as `CellA/Pump/Temperature` and `CellA/Fluid/Temperature`. A subscriber could use the *single-level wildcard* (+) character to register a filter with the broker that would match all three: `CellA/+/Temperature`.

Or maybe that oven in CellA also publishes pressure and humidity data. The subscriber could register those topic filters independently as `CellA/Oven/Pressure` and `CellA/Oven/Humidity` or could capture all subtopics using the *multi-level wildcard* (`#`) character: `CellA/Oven/#`.

The broker stores these topic filters as part of each client's network session and routes matching updates from publishers to any and all matching subscribers: one-to-one or one-to-many. This creates a system that is something like Twitter for machines: a free flow of anonymous, interest-based, event-driven, bi-directional communication.

MQTT's flexible topic filters provide a simple subscription mechanism.

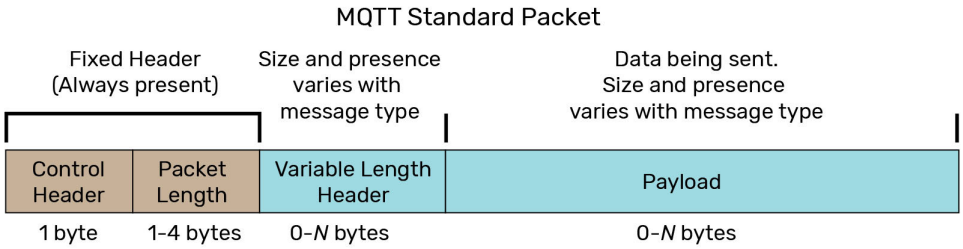


Message payload

MQTT is an application layer protocol like HTTP, meaning that it defines the interface methods that applications and devices use to talk to each other over a network, and both are commonly used as messaging transports. In fact, MQTT has become a contender with HTTP for the most widely used

protocol in IoT applications. But unlike HTTP, MQTT was designed for machine-to-machine communication.

While HTTP is famously heavyweight, with a long list of message headers used to describe and respond to resources, MQTT is data-agnostic, with a streamlined on-the-wire footprint that can be processed efficiently by devices with limited power and processing capabilities. It uses a simple byte array payload with a fixed 2-byte header and variable-length header fields (up to a few additional bytes) to indicate packet length or control codes. A packet can be up to 256 MB in size and can transport anything from process variables to a picture of your favorite pet.



Fault tolerance mechanisms

Reliable messaging is critical for industrial communications because of the need for accurate system state awareness and timely control response. Understanding this, MQTT's designers included features to gracefully handle unintended client disconnections and ensure data delivery to subscribers.

Messages can be transmitted or received with one of three *quality of service levels* (QoS 0,1, or 2). Higher QoS levels carry more overhead but provide a stronger guarantee of delivery.

- QoS 0 publishes a message once with no required acknowledgment.
- QoS 1 publishes a message repeatedly until it is acknowledged.
- QoS 2 publishes a message repeatedly but guarantees that it is received only once.

Published topics can be flagged as *retained messages*. This flag instructs the MQTT broker to store the most recent update to that topic. When a new client subscribes to that topic, it will receive this message immediately, rather than waiting for the next publication.

A client can also opt to connect to an MQTT broker with a *persistent session*. When the client disconnects, intentionally or otherwise, the broker keeps these network sessions open and stores the client's subscription filters along with any matching QoS 1 or 2 messages that arrive while it is disconnected. When the client reconnects, messages are issued in the order they were received.

Finally, each client can also designate a special message as a *last will and testament* (LWT), with its own topic path, QoS, and retention settings. This message is registered with the broker upon connecting and is distributed to subscribers should the client connection be interrupted unexpectedly. Each client's *keep-alive timer* value determines when the broker considers this connection lost.

Advantages for industry

Combined with its streamlined payload, the communications model used by MQTT results in an 80-90%

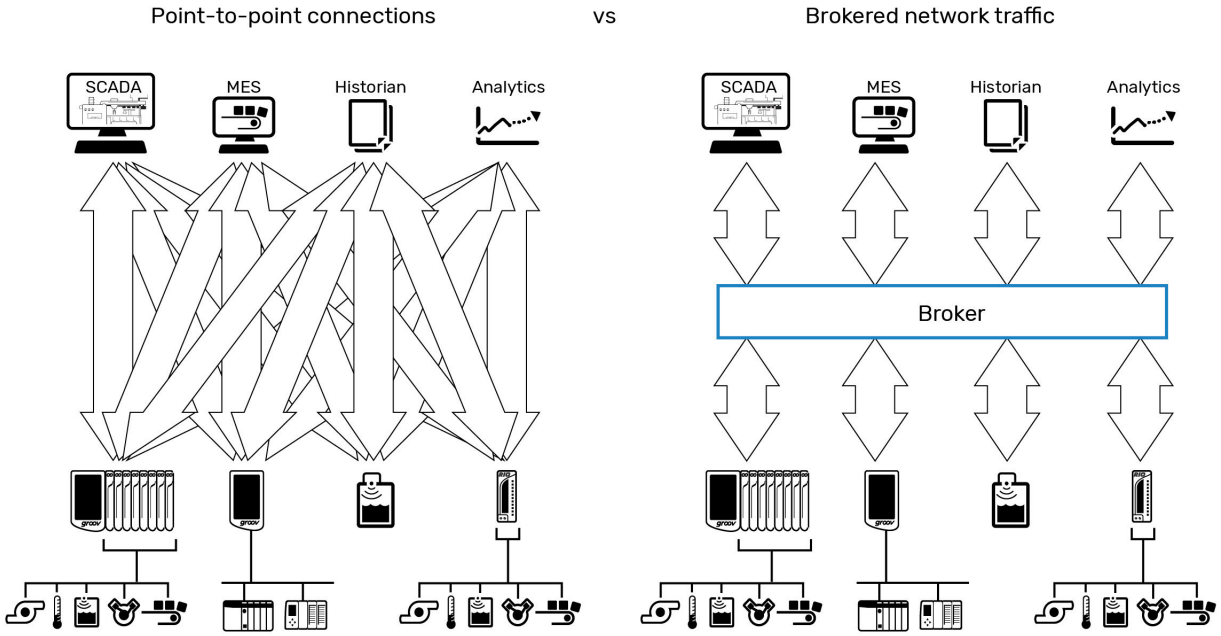
MQTT's designers included features to gracefully handle unintended client disconnections and ensure data delivery to subscribers.

reduction in overall bandwidth consumption compared to poll-response protocols, according to Cirrus Link. This efficiency creates significant room for existing networks to grow—up to millions of connections—with MQTT.

Decoupled data

But it's the decoupled nature of MQTT data exchange that unlocks the scalability of industrial networks. Without the need for point-to-point connections or direct addressing, MQTT networks can grow and share data flexibly.

Any client that wants access to published data—a maintenance database, ERP, or SCADA system, for instance—can simply point to the common MQTT broker and subscribe to any desired topics, without needing the details of the publishing source. Network traffic between publishers and the broker is unaffected by the number of subscribers receiving updates, and subscribers do not require reconfiguration if a field device type or IP address changes.



Data integrity and security

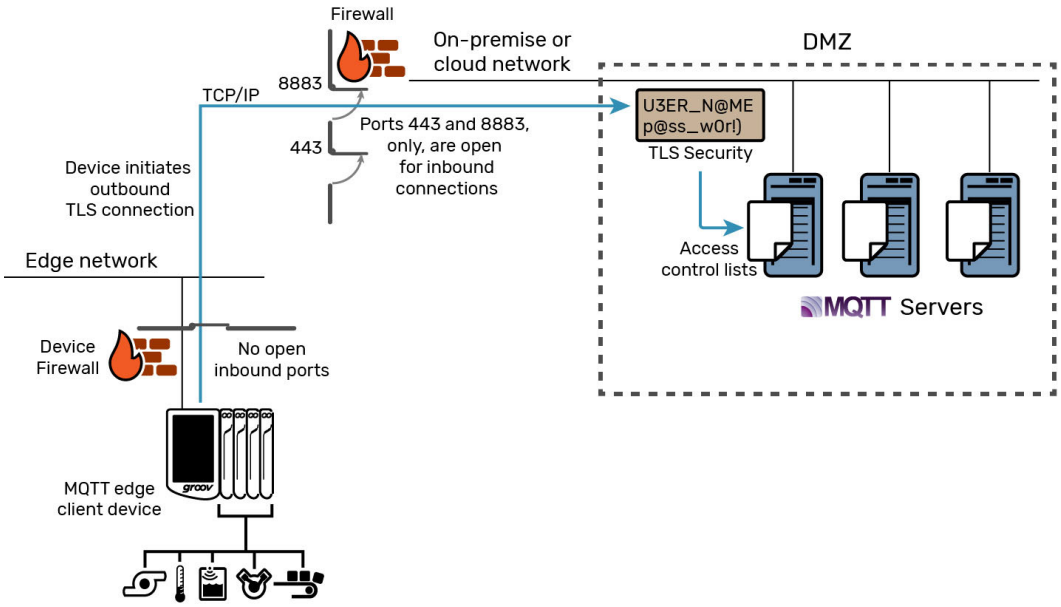
The architecture also has important implications for data integrity and security. Since the MQTT broker doesn't house or modify data, but only distributes it, each publisher is the single source of truth for its respective topics, reducing the potential for discrepancies and data siloing.

MQTT's fault-tolerance features complement the flexible nature of the publish-subscribe model by building in guarantees that ensure that clients are advised of changes in data quality and equipment state without constant polling.

The broker alone manages user authentication, data access rights, and message delivery, simplifying network management and allowing each client to remain anonymous to other network members. And since MQTT connections are device-originating (outgoing), only the broker is required to have an open firewall port. Field devices can be completely locked down while still permitting bi-directional communication.

MQTT also supports optional username and password fields, but in an effort to keep the specification as simple as possible, it primarily relies on security mechanisms in other layers. The most common method is to make use of the transport layer security (TLS) mechanisms already built into the TCP/IP stack (port 8883 is registered for MQTT TLS). In combination with certificates of trust to authenticate the identity of connected endpoints, secure site-to-site MQTT communication is feasible even over public networks.

MQTT's device-originating connections simplify cybersecurity.



FASTER, BETTER WITH SPARKPLUG B

Because of its flexibility, MQTT has been adopted for many IoT solutions. It imposes few naming conventions and accommodates any payload of an appropriate size with equal efficiency. Developers can customize MQTT's basic structure (topic paths) and behaviors (retained messages, persistent sessions, and so on) for their MQTT client as best suits the application, with no extraneous design elements to account for. However, this same flexibility can also be a limitation, particularly for the kind of large, diverse networks characteristic of industrial installations.

With no uniform naming standard, no common data format, and no contextual information on published topics, the details of each publisher's data—which topic paths to subscribe to, how to decode or interpret the payload, the appropriate QoS level, and how to recognize a publisher's LWT—must be known ahead of time in order to be discovered and used by subscribers. Since each device is free to use different conventions, configuring a large network can require significant effort akin to mapping conventional tag data between applications. These factors slow the pace of development and may even inhibit interoperability, adding to the cost of integration and undermining the goals of IIoT.

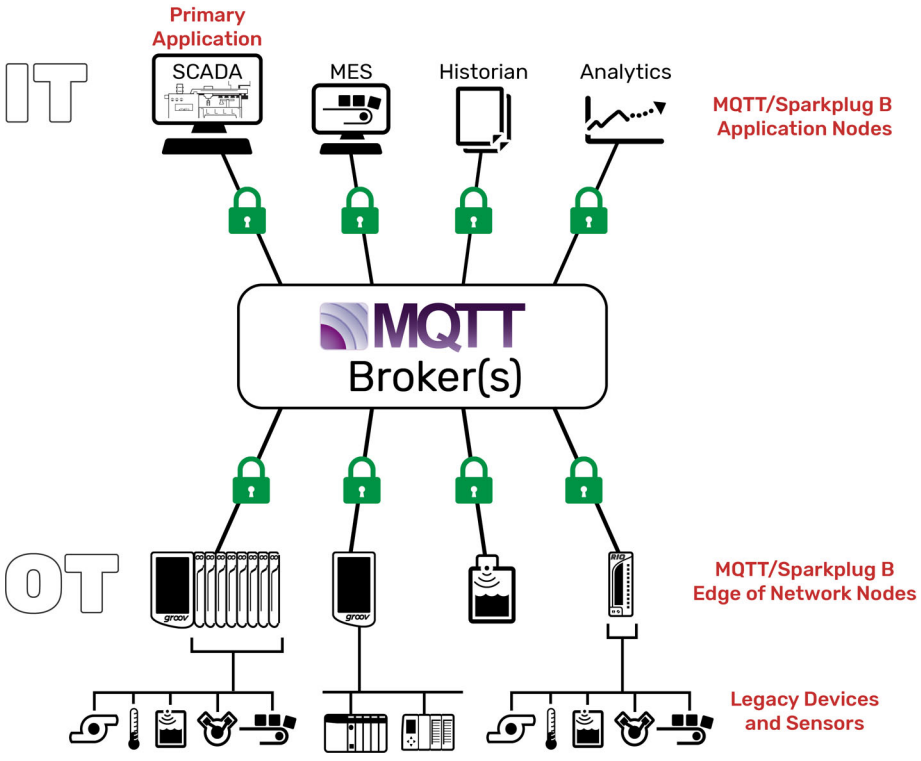
Likewise, while MQTT includes features to monitor client connection status and data quality, there's no guarantee that a vendor will utilize them in a field device or software client.

These weaknesses, and other observations about MQTT's adoption in the marketplace, inspired the development of the open-source Sparkplug MQTT Topic Namespace and Payload Definition, under the leadership of MQTT co-inventor Arlen Nipper and Cirrus Link Solutions. The current version, Sparkplug B (SpB), expands on the basic MQTT architecture to address common industrial use cases and adds a handful of important implementation details to MQTT clients that conform to the specification.

Components of an MQTT/Sparkplug B network

Sparkplug adds to and clarifies the roles of basic MQTT clients, enabling new features and more explicit messaging that are the basis for other enhancements in the specification.

The MQTT/Sparkplug B architecture defines clear roles and behaviors for MQTT clients and their data, adapting the underlying MQTT framework to better support typical SCADA/IIoT use cases.



The specification distinguishes between two types of MQTT clients:

- **MQTT/Sparkplug B Edge of Network (EoN) Nodes:** These clients provide physical and/or logical gateway functions to enable MQTT/Sparkplug B communications for legacy devices and sensors. EoN nodes also include smart devices and sensors capable of publishing their own Sparkplug B data, process variables, or metrics directly to an MQTT broker.
- **MQTT/Sparkplug B Application Nodes:** These are software clients, optionally including one *primary application* responsible for sending commands and receiving historical data (more information below). An MQTT/SpB application node may also be a gateway to legacy software systems.

Sparkplug nodes use predefined message types (see next section) to distinguish between internal data and data originating from connected devices. Nodes are also responsible for reporting on the state of their connected devices, if present.

Any MQTT 3.1.1-compliant broker will support Sparkplug B payloads, and one or more can be used as needed for redundancy, high availability, or scalability.

Topic namespace

Sparkplug defines a standard format for MQTT topic paths, creating a unified namespace for all SpB clients on the network:

`spBv1.0/<Group ID>/<MESSAGE TYPE>/<Edge Node ID>/<Device ID>`

Element	Definition	Source
<Group ID>	A logical identifier for a group of MQTT nodes	Defined by user
<MESSAGE TYPE>	Indicates whether the message contains state information, data, or a command and whether it pertains to a node, device, or the primary application	Predefined by SpB spec; cannot be changed by user
<Edge Node ID>	Identifies a specific MQTT node	Defined by user. The Group ID/Edge Node ID combination must be unique
<Device ID>	Identifies a device attached physically or logically to a node	(Optional) Defined by user, if applicable

Payload definition

Sparkplug B defines a standard, structured, data-rich but efficient payload format.

The core payload contains a structured series of key-value pairs defining an array of metrics and associated metadata. The specification defines a variety of optional fields for each metric, such as name, description, datatype, checksum, historical data flag, and many more.

Data type indicators accommodate complex types common in industrial applications, like matrices and user-defined types (UDTs). The payload specification also permits an array of custom properties for each metric, such as engineering units or scaling limits, that are published along with the primary process variable or data object.

The full payload is then timestamped, sequenced, and encoded using Google protocol buffers (protobufs), an efficient, interoperable, binary representation of the structured data, which maintains a small on-the-wire footprint without sacrificing complexity. When decoded, the payload is typically represented in JavaScript object notation (JSON) but is compatible with many formats. Here is an example of a simple SpB payload decoded into JSON:

```
{
      "timestamp": 1486144502122,
      "metrics": [{
            "name": "My Metric",
            "alias": 1,
            "timestamp": 1479123452194,
            "dataType": "String",
            "value": "Test"
      }],
      "seq": 2
}
```

State management

Sparkplug introduces the concept of birth and death certificate messages to define and ensure the use of appropriate state monitoring mechanisms.

The death certificate simply formalizes the use of MQTT's last will and testament message. Triggered when a client's keep-alive timer has expired, the death certificate indicates an unexpected disconnection. The topic path follows the standard Sparkplug format, using the message types NDEATH and DDEATH, indicating that the certificate pertains to a node or to an attached device, respectively.

The birth certificate, on the other hand, is a new addition of the Sparkplug spec. It is a special message, utilizing message types NBIRTH or DBIRTH, that a client is required to publish for itself and each of its attached devices when first coming online. These messages contain all the published topics for that client or device and inform all subscribers that the client or device is online.

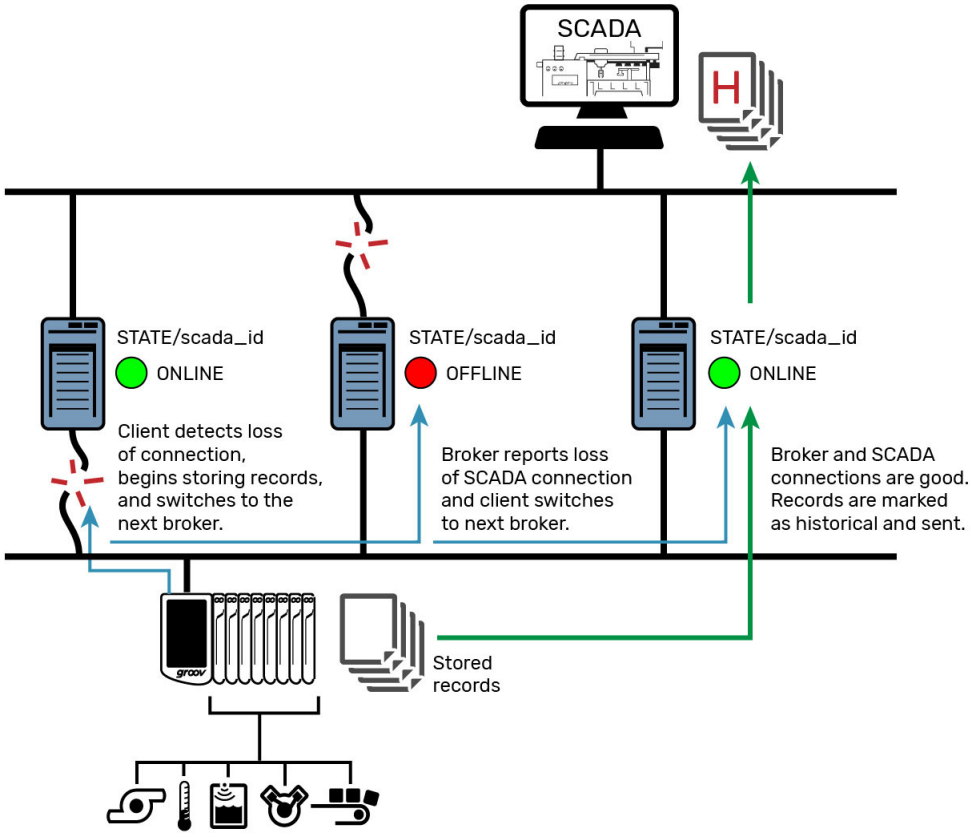
Primary application

While an MQTT network may contain any number of application clients, an industrial setting often has a single critical application used for operations management or process control, like an IIoT or SCADA host. This application is typically the only one permitted to send commands into the network. The Sparkplug specification adapts the MQTT framework to this reality by allowing one application node to be designated as the *primary application*.

An SpB-compliant primary application—like Cirrus Link's MQTT Engine module for Ignition SCADA or Canary Labs's MQTT Data Collector—publishes special birth and death certificates using the topic `STATE/<client ID>`. Unlike regular death certificates, these messages are published with QoS 1 and are retained by the broker to ensure that any MQTT client can identify the primary application's state at any time. This mechanism enables two additional functions that are critical to building fault-tolerant industrial networks.

- **Enhanced failover.** If an MQTT client loses its current broker connection and other connections are available, it can switch automatically. In addition, a Sparkplug B-compatible client can also switch connections based on the value of the primary application's STATE topic. When the primary application's connection is interrupted, its death certificate is published by the broker, indicating that STATE has changed to "OFFLINE." SpB clients will see this and begin searching through their alternate connections for a broker that reflects an "ONLINE" status for the application.
- **Store-and-forward historization.** In combination with the flags built into the Sparkplug B payload, a client can indicate that a message contains historical, rather than real-time, data. If capable, a client can begin storing records any time it or the primary application disconnects, queuing them up until it is assured of safe delivery, then publishing them as historical records. Since these messages are distinguishable from real-time data, store-and-forward provides better continuity for time-series data and time-critical operations than MQTT's standard quality of service levels.

The primary application mechanism enables smart failover and historization.



Advantages for industry

Interoperability

The Sparkplug B specification addresses the potential for inconsistency in MQTT implementations by defining standard client roles and data interfaces designed around industrial applications. By standardizing on Sparkplug B, MQTT clients from different vendors can identify, interpret, and make use of published data without needing to know the details of the originating client. Even brownfield sites with legacy automation networks and devices can leverage Sparkplug because of its explicit support for gateway-attached devices.

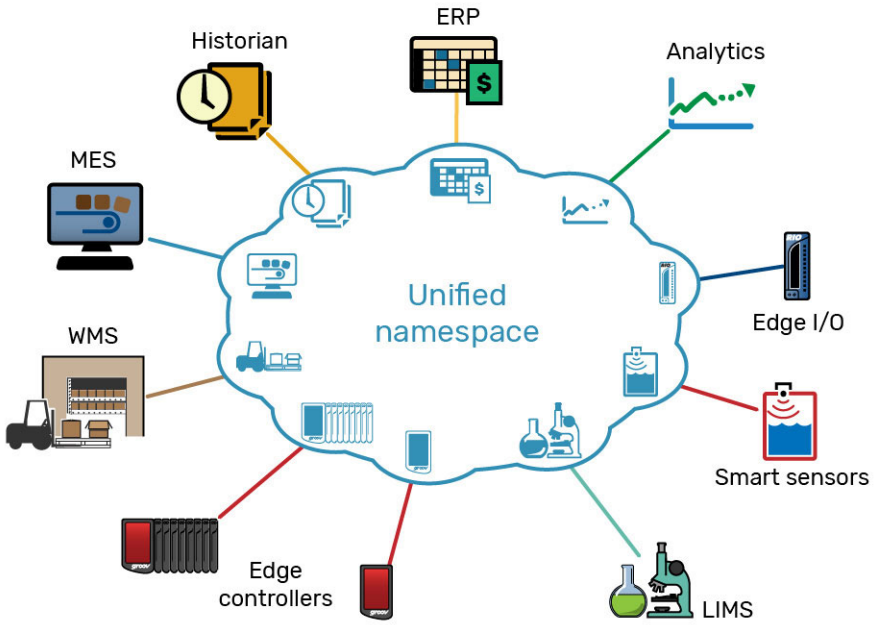
Unified namespace

This level of interoperability satisfies a critical requirement for digital transformation by making possible an enterprise-spanning unified namespace, a common data source defining all business data. Instead of hunting for operations data spread throughout a hierarchy of applications (PLC > SCADA > MES > ERP, for example), each with its own schema and data structures, MQTT devices and applications all contribute to a single data endpoint: the MQTT broker.

Enterprise clients can consume data from the field via the broker, using Sparkplug to provide a common exchange format and uniform context. Their results can be published back into the network to be detected, interpreted, and acted on by other clients. In this way, data can be shared

seamlessly throughout the organization, eliminating data silos, reducing the potential for discrepancies, and allowing a connected system to scale up smoothly.

With Sparkplug, data from many domains can be shared interoperably across an organization.



Enhanced data integrity

Sparkplug also improves data integrity and consistency with its combination of reliability features. By enforcing the last will and testament mechanism, all subscribers are notified when pertinent data becomes stale, and the birth certificate lets them know when data is fresh again. And for applications like those in the regulated industries that require historical data for auditing or root cause analysis, store-and-forward historization provides protection against critical data loss in the face of network or client instability.

Low administration

The combination of Sparkplug's enhanced state management and interoperability unlocks an additional contributor to network scalability by reducing the administrative overhead required to integrate each node and device.

As mentioned, integrating traditional industrial data involves time-consuming tag/variable mapping between applications. But with Sparkplug's birth certificate mechanism, this process is sped up by orders of magnitude. Subscribers, the primary application in particular, can use the information published in the birth certificate to map out available topics in a matter of moments. Since a birth certificate is required from each client whenever it connects to the network or updates its topic structure, the tag hierarchy can be discovered automatically. And thanks to Sparkplug B's payload structure, it comes complete with all metadata and UDT definitions.

BUILDING MISSION-CRITICAL MQTT NETWORKS

Sparkplug B fits MQTT to the needs of industrial applications by defining an open, interoperable standard for industrial data exchange that supports common use cases and the complex data they require, without compromising efficiency. By explicitly defining client roles and message types, Sparkplug B also enhances MQTT's ability to reliably deliver data to the right people at the right time, all of which can give users confidence as they scale up the network.

However, MQTT/Sparkplug B is of little value without robust support from all network clients. Specific features that Sparkplug provides, like legacy device integration and store-and-forward historization, can't be fully realized without field device clients that implement the necessary gateway and storage functions.

And fundamental network reliability is undermined unless every network participant implements complementary cybersecurity and connectivity features. Even the sheer volume of data generated by a large Sparkplug B IIoT network can be overwhelming unless clients are able to filter and sanitize data locally before publishing. Unfortunately, traditional industrial devices were designed with a narrow scope of operation in mind and lack much in the way of general-purpose processing, communication, and storage.

To address these general gaps in operations technology (OT), many industrial automation vendors are drawing inspiration from information technology (IT), turning to an approach known as edge computing—a form of distributed computing that addresses bandwidth constraints in global and regional networks by moving computing resources geographically closer to areas of high demand on the edge of the network.

Industrial edge computing

Industrial edge computing mimics this approach by embedding data processing, networking, and storage at the local process level, rather than strictly in the network core. With more resources at the edge, data can be prepared locally before broad distribution, modern security measures can be layered onto legacy systems, and advanced functionality can be embedded in the process in support of a resilient network.

Industrial edge computing devices come in many varieties, including the *edge input/output (I/O) modules* and *edge programmable industrial controllers (EPICs)* in Opto 22's *groov* product line.

- Edge I/O modules like *groov* RIO scale horizontally, rapidly integrating a variety of traditional field signals (V, mA, ICTD, TC, thermistor, discrete I/O, dry contacts, and so on) and making their data available directly to Ethernet networks.
- Edge controllers like the *groov* EPIC offer more computing and networking power alongside a traditional PLC/PAC control engine,

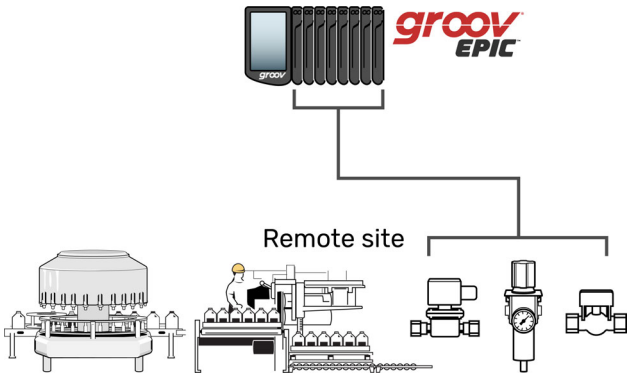
Opto 22's *groov* EPIC and *groov* RIO provide modern security and data processing abilities, combined with traditional control options and a fully compatible MQTT/Sparkplug B client.

providing a broader array of integration options and allowing disparate automation networks to be combined into a single system.

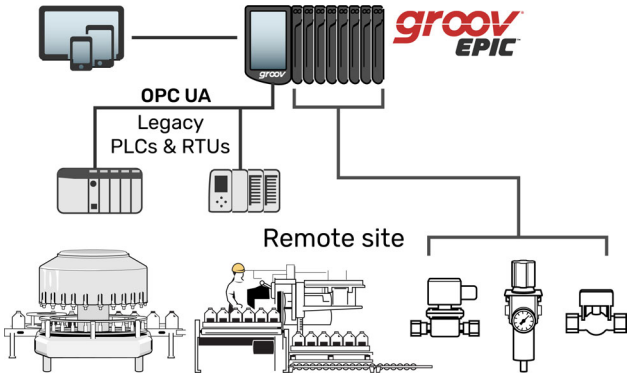
Opto 22's *groov* devices support edge data processing with tools like Inductive Automation's Ignition and the open-source IoT platform, Node-RED, and provide modern IT security options like user authentication, TLS encryption, and device firewalling. They also provide local storage that can support store-and-forward historization. All *groov* devices provide robust MQTT/Sparkplug B client options, designed in collaboration with industry leaders like Cirrus Link and Inductive Automation.

Using industrial edge devices like these in combination with appropriate server hardware, users can build up a secure MQTT/Sparkplug B network following a basic pattern:

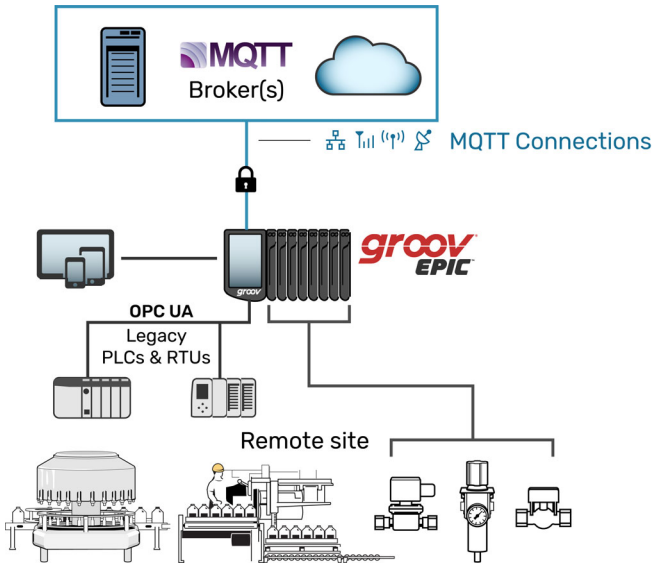
First, instrument assets if needed, and connect the I/O to an edge device like *groov* EPIC.



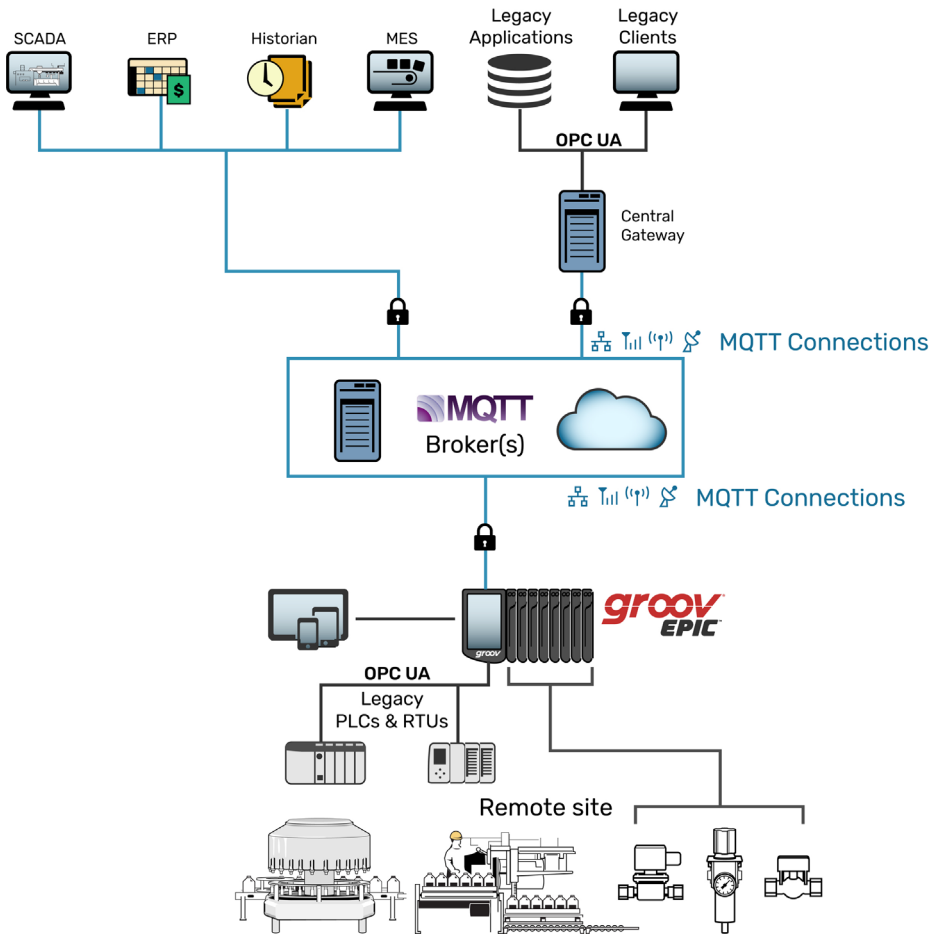
If existing controls are in place already, they can be connected to a segmented network interface on the EPIC, which can use OPC UA drivers to pull the data in.



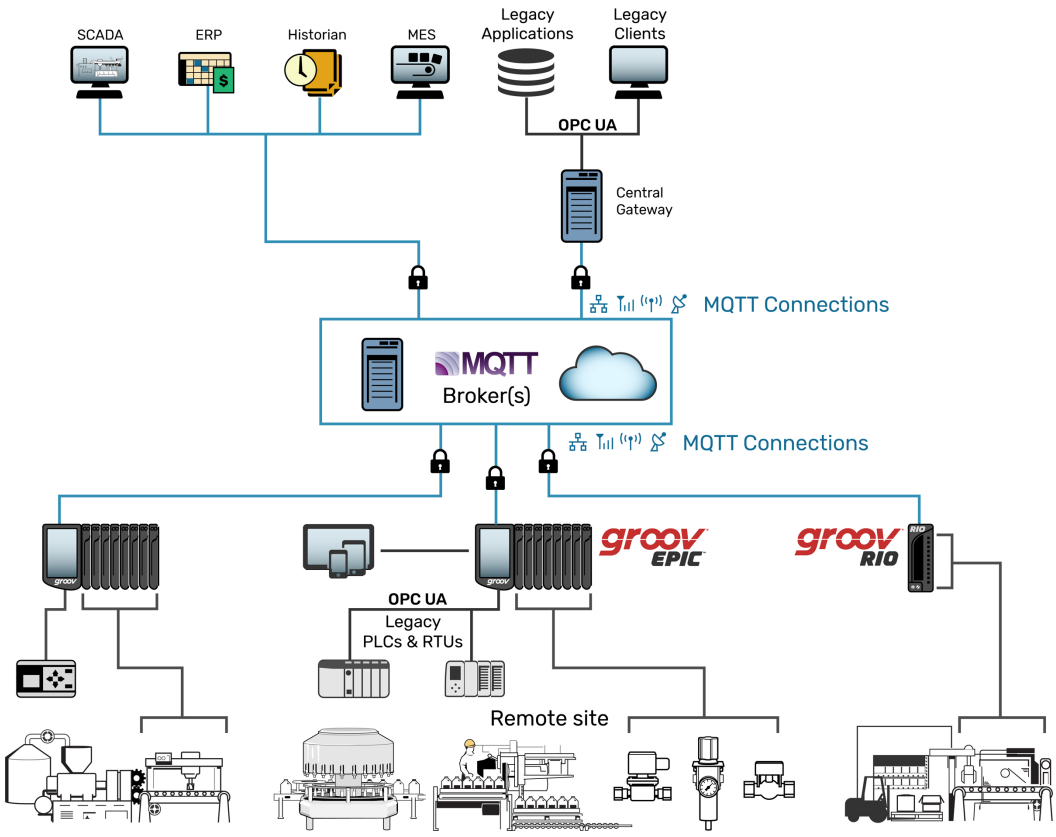
Then, the EPIC can open an encrypted, authenticated connection to an MQTT broker, residing either in the cloud or somewhere on premises, ideally in the corporate network's DMZ (demilitarized zone, an isolated network that bridges internal and external networks).



Now, applications can be plugged into the infrastructure. A gateway application, like Ignition Gateway with Cirrus Link's MQTT Engine, can subscribe to the entire system's tags and make all that data available to other applications, including back into OPC, if needed.



With this foundation, the system can scale up with additional edge devices or additional brokers to build out a highly resilient, high-performance data exchange and command-and-control system that's completely secure.



This approach is open, flexible, and powerful. With MQTT/Sparkplug B as the backbone of the network, up-front licensing costs are significantly limited, and there is no reason to be locked into a particular network configuration.

Users can pick from a variety of industrial edge solutions or MQTT clients; they can implement all or only part of the design, depending on their needs; and the network can scale in a variety of ways. And with edge devices as the physical foundation of the network, there is processing power available in the field to harness in support of the kinds of applications that are driving digital transformation, like database integration, IIoT, AI data modeling, and digital twin development.

CONCLUSION

MQTT is popular, proven, and well supported in enterprise and consumer applications. It makes organization-spanning data exchange possible by decoupling data producers and consumers using a brokered publish-subscribe architecture and by defining a lightweight, data-agnostic communication format that supports millions of connections.

But MQTT's usefulness for industrial applications is challenged by the general lack of interoperability between industrial devices and between operations technology and information technology. Different devices from different vendors may not operate in the same topic namespace and typically lack the support needed to communicate securely with IT systems.

Industrial-strength MQTT/Sparkplug B

Sparkplug B defines an MQTT implementation standard that ensures client interoperability and enhances MQTT with features designed to support the demands of mission-critical industrial systems. The resulting infrastructure delivers data that is fit for use in operations, gracefully handles instability, and helps organizations scale by reducing administrative overhead.

Industrial edge devices complete the architecture by providing complementary reliability in the physical layer, along with a variety of integration options and sufficient computing power to process and transmit field data efficiently and securely.

The combination of MQTT, Sparkplug B, and industrial edge devices forms a complete solution for building and scaling connected IT/OT data networks. It is open, flexible, and powerful enough to support the requirements of any connected application.

ABOUT OPTO 22

Opto 22 was started in 1974 by a co-inventor of the solid-state relay (SSR), who discovered a way to make SSRs more reliable.

Opto 22 has consistently built products on open standards rather than on proprietary technologies. The company developed the red-white-yellow-black color-coding system for input/output (I/O) modules and the open Optomux® protocol, and pioneered Ethernet-based I/O.

Famous worldwide for its reliable industrial I/O, the company in 2018 introduced *groov* EPIC® (edge programmable industrial controller). EPIC has an open-source Linux® OS and provides connectivity to PLCs, software, and online services, plus data handling and visualization, in addition to real-time control. The company's latest product line, *groov* RIO®, provides compact, autonomous edge I/O that is ideal for communicating field data in IIoT applications.

All Opto 22 products are manufactured and supported in the U.S.A. Most solid-state SSRs and I/O modules are guaranteed for life. The company is especially trusted for its continuing policy of providing free product support and free pre-sales engineering assistance.

MADE IN THE USA

For more information, visit opto22.com or contact Opto 22 **Pre-Sales Engineering.** Use the **Contact Us form** on our website or call:

800-321-6786 (toll-free in the U.S. and Canada)
or **951-695-3000**

MADE IN THE USA
OPTO 22 www.opto22.com
43044 Business Park Dr. Temecula, CA 92590-3614

SALES sales@opto22.com
800-321-6786 • 1-951-695-3000

SUPPORT support@opto22.com
800-835-6786 • 1-951-695-3080