

Protocol Management Software

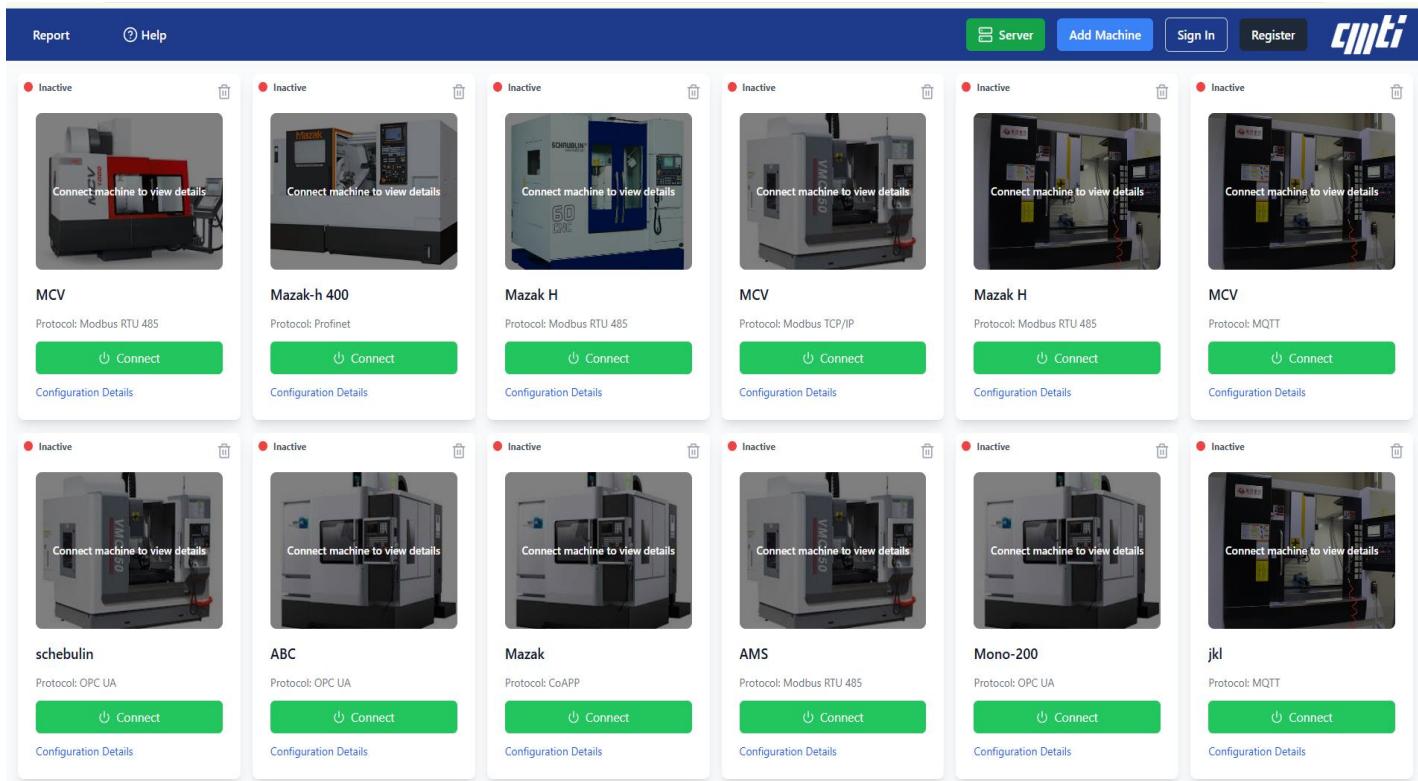
Problem Statement:

This project aims to retrieve and process data from industrial and IoT communication protocols, including Modbus, OPC UA, PROFINET, PROFIBUS, EtherCAT, CAN open, BACnet, HTTP, and MQTT. The objective is to establish seamless connectivity, enable real-time data exchange, and integrate data from various industrial automation and IoT systems for monitoring, analysis, and decision-making. The solution should ensure compatibility, scalability, and secure data handling across diverse protocols and devices.

Main Dashboard:

1. Machine Connectivity Management

- The system displays a list of industrial machines along with their respective communication protocols (e.g., Modbus RTU 485, Modbus TCP/IP, ProfiNet, OPC UA, MQTT).
- Users can connect machines to retrieve real-time data and monitor their status.



The screenshot shows a main dashboard interface for managing machine connectivity. At the top, there is a navigation bar with links for 'Report', 'Help', 'Server' (highlighted), 'Add Machine', 'Sign In', and 'Register'. The logo 'cyti' is also present. Below the navigation bar is a grid of machine cards, each representing a different industrial machine. The grid is organized into two rows of six cards each. Each card includes a thumbnail image of the machine, its name, protocol information, and two buttons: 'Connect' and 'Configuration Details'.

Machine Name	Protocol	Status	Action Buttons
MCV	Modbus RTU 485	Inactive	Connect, Configuration Details
Mazak-h 400	Profinet	Inactive	Connect, Configuration Details
Mazak H	Modbus RTU 485	Inactive	Connect, Configuration Details
MCV	Modbus TCP/IP	Inactive	Connect, Configuration Details
Mazak H	Modbus RTU 485	Inactive	Connect, Configuration Details
MCV	MQTT	Inactive	Connect, Configuration Details
schebulin	OPC UA	Inactive	Connect, Configuration Details
ABC	OPC UA	Inactive	Connect, Configuration Details
Mazak	CoAPP	Inactive	Connect, Configuration Details
AMS	Modbus RTU 485	Inactive	Connect, Configuration Details
Mono-200	OPC UA	Inactive	Connect, Configuration Details
jkl	MQTT	Inactive	Connect, Configuration Details

2. Machine Status Indication

- Each machine card indicates whether the machine is **Active** or **Inactive**, allowing users to identify the connectivity status at a glance.
- Inactive machines display a red status indicator and a prompt to connect the machine for details.

3. User Actions and Controls

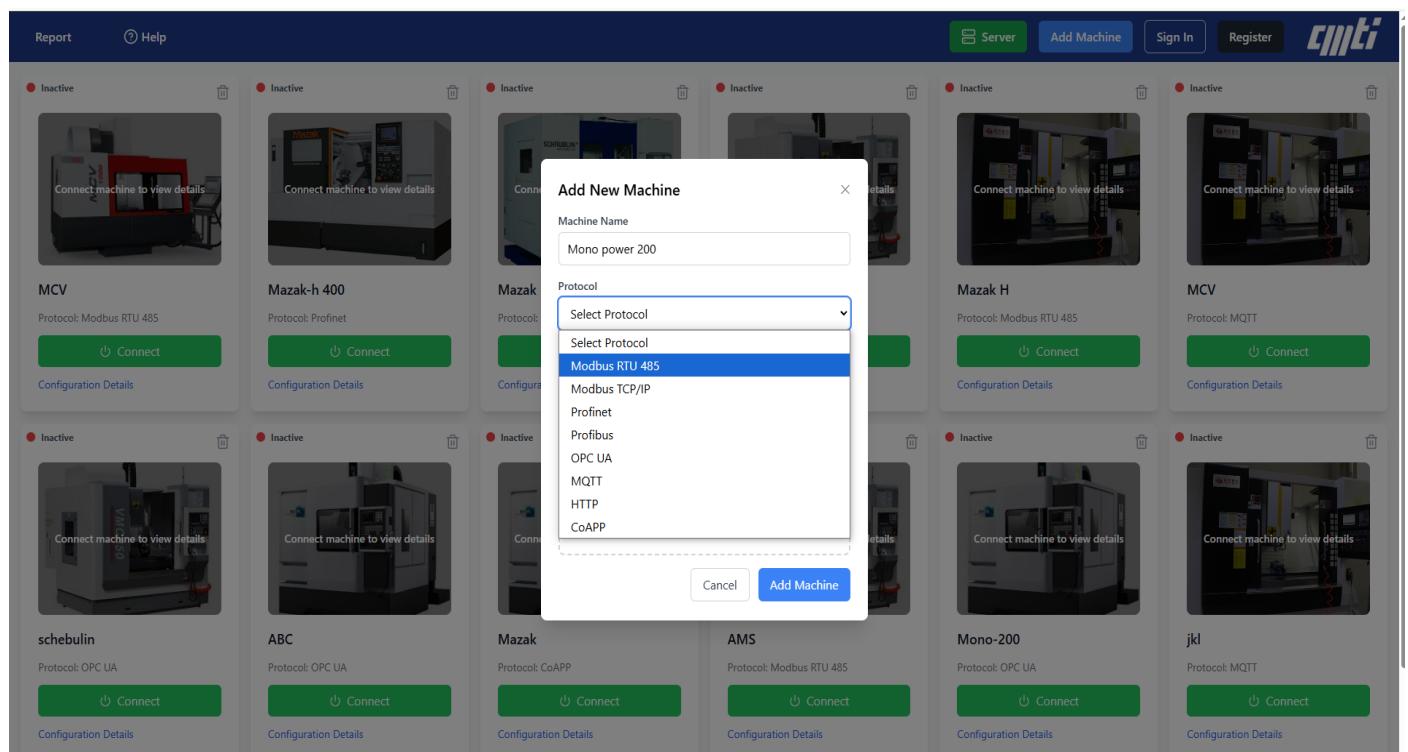
- Users can interact with each machine through the "**Connect**" button to establish a connection.
- Additional actions such as **viewing configuration details** and **deleting a machine** are available for better management.

4. User Authentication & Management

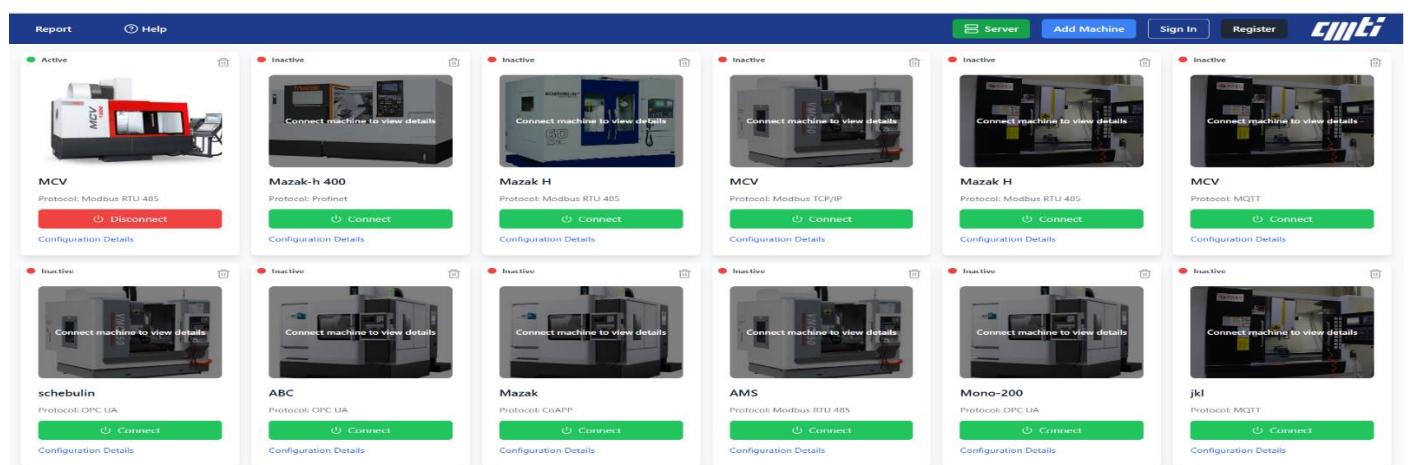
- The interface provides **Sign In**, **Register**, and **Add Machine** options for user authentication and machine management.
- Users need to log in to access full functionality, ensuring controlled access and data security.

Industrial Protocols:

1. Modbus RTU -Rs 485:



The screenshot shows a web-based interface for managing industrial machines. On the left, there's a sidebar with 'Report' and 'Help' buttons. At the top right, there are buttons for 'Server', 'Add Machine', 'Sign In', and 'Register'. The main area displays a grid of machine cards. One card for 'MCV' is active, while others are inactive. Each card shows a thumbnail image of the machine, its name, protocol, and a 'Connect' button. A modal dialog box titled 'Add New Machine' is open in the center. It has a 'Machine Name' input field containing 'Mono power 200' and a 'Protocol' dropdown menu. The dropdown is open, showing a list of protocols: 'Select Protocol', 'Modbus RTU 485' (which is selected), 'Modbus TCP/IP', 'Profinet', 'Profibus', 'OPC UA', 'MQTT', 'HTTP', and 'CoAPP'. There are 'Cancel' and 'Add Machine' buttons at the bottom of the dialog.



This screenshot shows the same interface after the machine has been added. The 'MCV' machine is now listed as 'Active' with a 'Disconnect' button instead of a 'Connect' button. The other machines remain inactive with their original 'Connect' buttons. The 'Add New Machine' dialog is no longer visible.

Live Monitoring

MCV

Connected

Raw WS: Connected Actual WS: Connected

Raw Data

REGISTER	VALUE	STATUS
Register 128	0	ON
Register 133	0	ON
Register 134	0	ON
Register 135	0	ON
Register 136	0	ON
Register 137	0	ON
Register 138	0	ON
Register 139	0	ON
Register 140	48758	ON
Register 141	17264	ON
Register 142	48758	ON
Register 143	17264	ON
Register 144	0	ON
Register 145	0	ON
Register 146	0	ON
Register 147	0	ON

Actual Data

REGISTER	VALUE	STATUS
Register 128	0	ON
Register 130	0	ON
Register 132	0	ON
Register 134	0	ON
Register 136	0	ON
Register 138	0	ON
Register 140	240.7449493408203	ON
Register 142	240.7449493408203	ON
Register 144	0	ON
Register 146	0	ON
Register 148	0	ON
Register 150	0	ON
Register 152	0	ON
Register 154	0	ON
Register 156	49.938255310058594	ON
Register 158	0	ON

Auto Refresh 5 seconds Read Data OPC UA

Specific Registers

← MCV

Dashboard Live Monitoring Specific Registers

Connected

Fetch All Values Save Registers Store Values Auto-Store

Raw Registers

- Register 140
- Register 141
- Register 142
- Register 143
- Register 144
- Register 145
- Register 146
- Register 147

Actual Registers

- Register 142
- Register 144
- Register 146
- Register 148
- Register 150
- Register 152
- Register 154
- Register 156

Raw Register Values

Register	Value	Status	Auto-Refresh	Last Updated
Register 140	48374	SUCCESS	Off	3:37:57 PM
Register 141	17264	SUCCESS	Off	3:37:57 PM
Register 142	48374	SUCCESS	Off	3:37:57 PM

Actual Register Values

Register	Value	Status	Auto-Refresh	Last Updated
Register 142	240.74	SUCCESS	Off	3:37:57 PM
Register 156	49.94	SUCCESS	Off	3:37:57 PM

Last global update: 3:37:57 PM

OPC UA Quick Client

← MCV

Raw WS: Connected Actual WS: Connected

Connected

Raw Data

OPC UA Server URL: opc.tcp://localhost:4841/freeopcua/server/

REGISTER
Register 100
Register 101
Register 102
Register 103
Register 104
Register 105
Register 106
Register 107
Register 108
Register 109
Register 110
Register 111
Register 112
Register 113
Register 114

Connected successfully

Add Server

OPC UA Servers

Server 1

opc.tcp://localhost:4841/freeopcua/server/

Username:

Password:

Browse Results

ID	Description	Value
Parameter_Register_142	Register_142	240.781494140625

All Nodes

Node ID	Type	Description
ns=2;i=67	Parameter	Register_132
ns=2;i=71	Parameter	Register_134
ns=2;i=75	Parameter	Register_136
ns=2;i=79	Parameter	Register_138
ns=2;i=83	Parameter	Register_140
ns=2;i=87	Parameter	Register_142
ns=2;i=91	Parameter	Register_144
ns=2;i=95	Parameter	Register_146

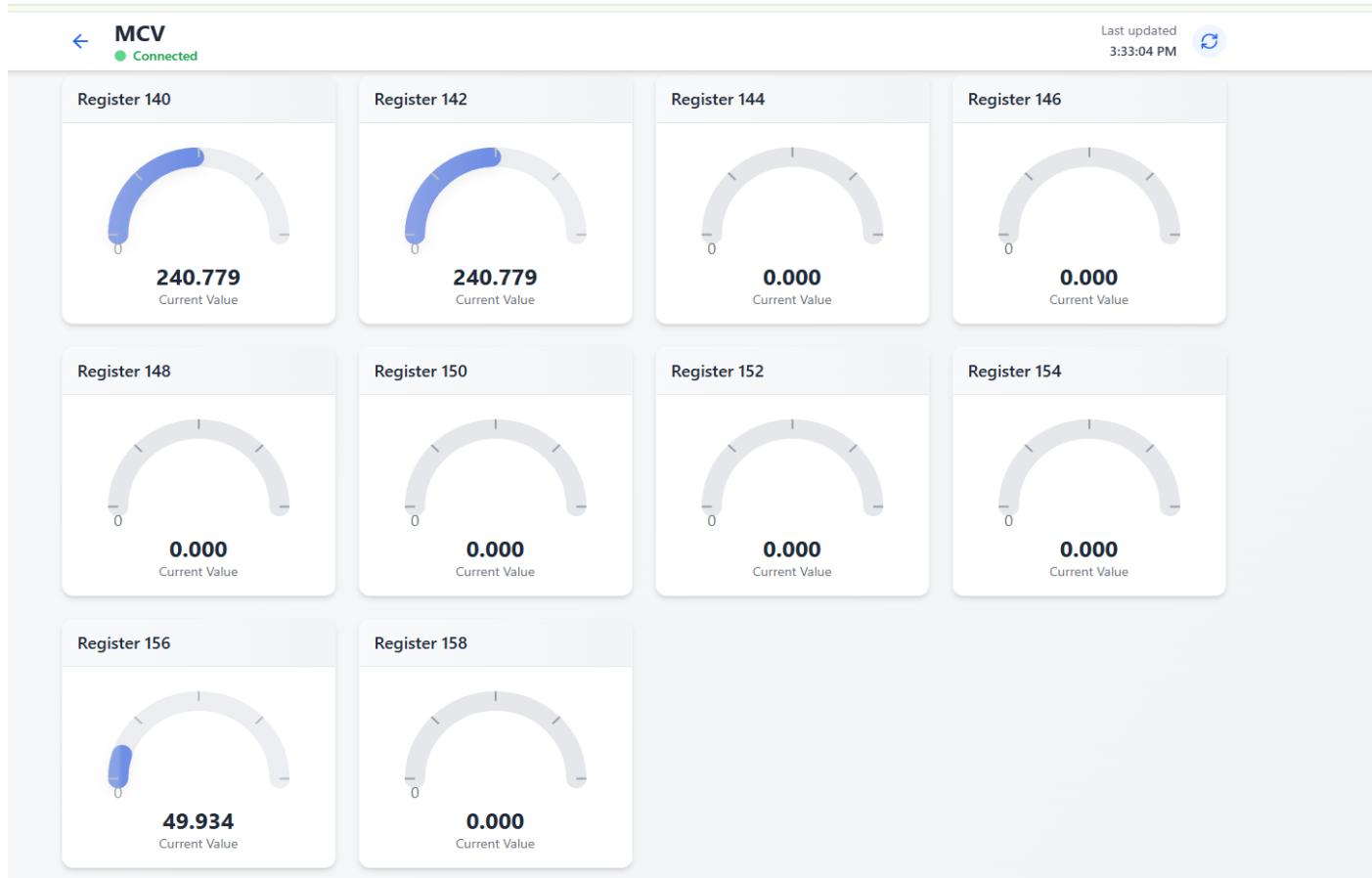
Retrieved 294 nodes in 0.54 seconds

OPC UA Servers

Dashboard Live Monitoring Specific Registers

Auto Refresh 5 seconds Read Data OPC UA

Dashboard of Registers data



Overview

Modbus RTU (Remote Terminal Unit) is a serial communication protocol used for industrial automation systems, allowing communication between devices connected via an **RS-485** network. It is widely used for connecting PLCs, sensors, and controllers to SCADA and monitoring systems.

➤ Modbus RTU Communication Basics

- **Protocol Type:** Request-Response (Master-Slave)
- **Physical Layer:** RS-485
- **Data Transmission:** Half-Duplex
- **Error Checking:** CRC-16 (Cyclic Redundancy Check)
- **Message Structure:** Address, Function Code, Data, CRC

➤ RS-485 Communication Parameters

When configuring **Modbus RTU over RS-485**, the following communication parameters need to be set:

Connection Parameters

Parameters	Description
COM Port	Serial port used for communication (e.g., COM4)
Baud Rate	Speed of data transmission (e.g., 9600, 19200, 115200 bps)
Data Bits	Number of bits in each data byte (typically 8)
Stop Bits	Defines the end of a character (1 or 2)
Parity	Error-checking mechanism (None, Even, Odd)
Slave ID	Address of the device (1-247)

Reading Parameters

Parameters	Description
Function Code	Defines the type of request (e.g., 3 for Read Holding Registers)
Address Selection Type	Defines whether to read a range or a specific address
Start Address	First register to read (e.g., 100)
End Address	Last register to read (e.g., 159)
Data Format	Big Endian / Little Endian
Combine Registers	Whether to combine multiple registers for 32-bit or floating-point data

Configuration Details for Modbus RTU-RS 485

Protocol Configuration

Modbus RTU 485 Configuration

Connection Parameters:

- COM Port: COM4
- Baud Rate: 9600
- Data Bits: 8
- Stop Bits: 1
- Parity: Even
- Slave ID (1-247): 1

Reading Parameters:

- Function Code: 3
- Address Selection Type:
 - Address Range
 - Specific Address
- Start Address: 100
- End Address: 159
- Data Format: Big Endian
- Combine Registers: True

Create OPC UA Server
 Store Data in Database

Protocol Configuration

Modbus RTU 485 Configuration

Review Machine Configuration Details:

PARAMETER	VALUE
Machine ID	27
Machine Name	Mazak
Machine Protocol	Modbus RTU 485
Slave ID	1
Function Code	3
Start Address	100
End Address	159
Specific Address	N/A
Data Format	
Combine Register	true
Store in OPC UA Server	true
Store in Database	true
COM Port	COM4
Baud Rate	9600
Data Bits	8
Stop Bits	1
Parity	even

Generate Report

➤ Data Handling & Storage

- **Create OPC UA Server:** Enables seamless integration with SCADA and IoT platforms.
- **Store Data in Database:** Logs real-time machine parameters for historical analysis.

➤ Stat Card Generation (Based on Marked Section)

- The "Generate Report" button is used to create a **Stat Card**, summarizing key configuration parameters such as:
 - Machine ID
 - Machine Name
 - Communication Settings
 - Modbus Addressing
 - Data Storage Preferences

This stat card provides a quick overview of the device configuration and can be used for diagnostics and validation.

2. Modbus TCP/IP:

← MCV

IP Address: 172.18.10.15
Port: 503
Unit ID: 1

Function Code: 3
Read Address: 0
Read Count: 10

Auto Off Refresh OPC UA Select Registers

Raw Register Data Last updated: 4:36:41 PM

OPC UA URL: opc.tcp://localhost:4871/freeopcua/server/

Register	Value
Register 0	25
Register 1	8
Register 2	45
Register 3	278
Register 4	2
Register 5	27
Register 6	274
Register 7	1
Register 8	0
Register 9	0

OPC UA Quick Client

← MCV

IP Address: 172.18.10.15
Port: 503
Unit ID: 1

Raw Register Data Last updated: 4:36:41 PM

OPC UA URL: opc.tcp://localhost:4871/freeopcua/server/

Register
Register 0
Register 1
Register 2
Register 3
Register 4
Register 5
Register 6
Register 7
Register 8
Register 9

OPC UA Servers

Add Server Refresh OPC UA Select Registers

Server 1

opc.tcp://localhost:4871/freeopcua/server/

Connected successfully

Username:

Password:

Auto-refresh Values: 5 seconds

Connect Disconnect Browse Node(s)

Retrieve All Nodes

Add Node ID

Browse Results

Type	Description	Value	Time
parameter	Register_7	1	4:37:54 PM

All Nodes

Node ID	Type	Description
ns=2;i=23	Parameter	Register_5
ns=2;i=27	Parameter	Register_6
ns=2;i=31	Parameter	Register_7
ns=2;i=35	Parameter	Register_8
ns=2;i=39	Parameter	Register_9
i=58	Parameter	BaseObjectType
i=62	Parameter	BaseVariableType
i=24	Parameter	BaseDataType

Retrieved 213 nodes in 0.51 seconds

Specific Registers Data

← MCV - Register Selection

IP Address: 172.18.10.15
Port: 503
Unit ID: 1

Function Code: 3
Read Address: 0
Read Count: 10

Register Selection

Register selections saved successfully!

Register 0 Value: 25 <input checked="" type="checkbox"/> Selected <input type="button" value="100ms"/> Auto-refreshing	Register 1 Value: 8 <input checked="" type="checkbox"/> Selected <input type="button" value="100ms"/> Auto-refreshing	Register 2 Value: 45 <input checked="" type="checkbox"/> Selected <input type="button" value="100ms"/> Auto-refreshing
Register 3 Value: 278 <input type="checkbox"/> Select <input type="button" value="Off"/>	Register 4 Value: 2 <input type="checkbox"/> Select <input type="button" value="Off"/>	Register 5 Value: 27 <input type="checkbox"/> Select <input type="button" value="Off"/>
Register 6 Value: 274 <input type="checkbox"/> Select <input type="button" value="Off"/>	Register 7 Value: 1 <input type="checkbox"/> Select <input type="button" value="Off"/>	Register 8 Value: 0 <input type="checkbox"/> Select <input type="button" value="Off"/>
Register 9 Value: 0 <input type="checkbox"/> Select <input type="button" value="Off"/>		

Selected: 3 of 10 registers Auto-refreshing: 3 registers

Overview

The interface allows users to add and manage manufacturing machines, that communicates via Modbus TCP/IP protocol. Modbus TCP/IP is a wireless communication protocol used for industrial automation systems.

➤ Modbus TCP/IP communication Basics:

- **Protocol Type:** Request-Response (Client-Server)
- **Physical Layer:** Ethernet
- **Data Transmission:** Full-Duplex
- **Error Checking:** TCP/IP checksum
- **Message Structure:** MBAP Header (7 bytes), Function Code, Data

➤ TCP/IP Communication Parameters

When configuring Modbus TCP/IP, the following communication parameters need to be set:

Connection Parameters

Parameters	Description
IP Address	Network address of the device (e.g., 172.18.10.15)
Port	Communication port (standard is 502)
Unit ID	Address of the device on internal sub-networks (1-247)

Reading Parameters

Parameters	Description
Function Code	Defines the type of request (e.g., 3 for Read Holding Registers)
Read Address	First register to read (e.g., 0)
Read Count	Number of consecutive registers to read (e.g., 10)
Data Format	Big Endian / Little Endian (implicit)

Configuration Details for Modbus TCP/IP

Review Machine Configuration Details:	
PARAMETER	VALUE
Machine ID	26
Machine Name	AMS
Protocol	Modbus TCP/IP
IP Address	172.18.10.15
Port	502
Unit ID	1
Read Address	0
Read Count	10
Function Code	3
Store in OPC UA Server	false
Store in Database	false

➤ Data Handling & Storage

- **Create OPC UA Server:** Enables seamless integration with SCADA and IoT platforms.
- **Store Data in Database:** Logs real-time machine parameters for historical analysis.

➤ Stat Card Generation (Based on Marked Section)

- The "Generate Report" button is used to create a **Stat Card**, summarizing key configuration parameters such as:
 - Machine ID
 - Machine Name
 - Communication Settings
 - Modbus Addressing
 - Data Storage Preferences

3. Profinet:

← **Read Profinet Data**

Configure fields to read from Mazak-h 400

Connection Status

Machine Name	IP Address	OEM Type
Mazak-h 400	172.18.10.15	Siemens

Connected

Data Field Configuration

+ Add Field

Field 1	DB Number	Offset	Data Type
Data Source	2	0	Int
Count	1		

Field 2	DB Number	Offset	Data Type
Data Source	2	2	Bool
Count	1		

Field 3	DB Number	Offset	Data Type
Data Source	2	3	Char
Count	1		

↓ Read Data

Results

ID	SOURCE	DB	OFFSET	TYPE	VALUE	STATUS
1	db	2	0	int	2542	success
2	db	2	2	bool	false	success
3	db	2	3	char	h	success

Overview

PROFINET is an industrial Ethernet standard for automation that enables real-time communication between controllers and devices in industrial environments. It supports high-speed data exchange and is widely used for connecting PLCs, HMIs, and field devices from various manufacturers, particularly in Siemens' environments.

➤ PROFINET Communication Basics

- **Protocol Type:** Industrial Ethernet (IEC 61158)
- **Physical Layer:** Standard Ethernet
- **Data Transmission:** Full-Duplex
- **Error Detection:** Standard Ethernet CRC

- **Communication Types:** TCP/IP, RT (Real-Time), IRT (Isochronous Real-Time)

➤ ProfiNet Communication Parameters

When configuring **PROFINET**, the following communication parameters need to be set:

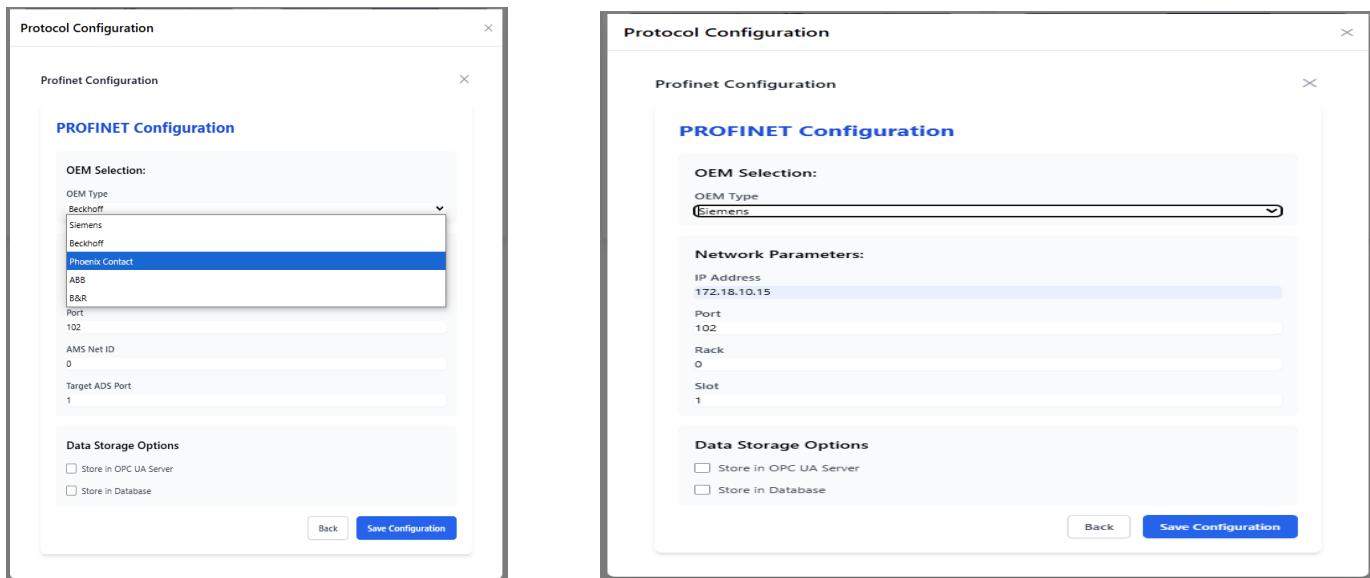
Connection Parameters for Siemens'

Parameters	Description
OEM Type	Device manufacturer (e.g., Siemens, Simatic Comfort, ABB)
IP Address	Network address of the device (e.g., 172.18.10.15)
Port	Communication port (e.g., 102)
Rack	Physical rack number in the hardware configuration (e.g., 0)
Slot	Slot number of the module in the rack (e.g., 1)

Reading Parameters

Parameters	Description
Data Source	Type of data to access (DB, Input, Output, Merker)
DB Number	Number of the data block to read (e.g., 2)
Offset	Address offset within the data block (e.g., 10)
Data Type	Format of the data (String, Bool, Integer, etc.)

Configuration Details for Profinet



➤ Data Handling & Storage

- **Create OPC UA Server:** Enables seamless integration with SCADA and IoT platforms.
- **Store Data in Database:** Logs real-time machine parameters for historical analysis.

➤ Stat Card Generation (Based on Marked Section)

- The "Generate Report" button is used to create a **Stat Card**, summarizing key configuration parameters such as:
 - Machine ID
 - Machine Name
 - Communication Settings
 - Modbus Addressing
 - Data Storage Preferences

This stat card provides a quick overview of the device configuration and can be used for diagnostics and validation.

4. OPC UA:

Mono-200 - OPC UA Monitor

Connection Details

Machine ID: 55
Server URL: opc.tcp://172.18.10.15:4840
Security Policy: None
Security Mode: None
Status: **Connected**
Load Time: 32.46 seconds
Real-time Monitoring Active

Statistics

Total Folders: 225
Total Parameters: 2150
Execution Time: 32.43 seconds

Node ID (Drag & Drop from below or enter manually)

ns=3;s=OperatingMode
i=2259

Stop Monitoring WebSocket data streaming active

Monitored Node Values (Updating in real-time)

NODE ID	DESCRIPTION	VALUE
ns=3;s=OperatingMode	OperatingMode	8
i=2259	State	0

Machine Nodes

2375 nodes found (225 folders, 2150 parameters)

NODE ID	BROWSE NAME	DATA TYPE	VALUE	ACTIONS
i=85	Objects	Folder	-	Use
i=2253	Server	Folder	-	Use
i=2268	ServerCapabilities	Folder	-	Use
i=2997	AggregateFunctions	Folder	-	Use
i=2996	ModellingRules	Folder	-	Use
i=83	ExposeItsArray	Folder	-	Use

Overview

OPC UA (OPC Unified Architecture) is a platform-independent, service-oriented architecture that integrates all the functionality of individual OPC Classic specifications into one extensible framework. It provides a secure and reliable way to transfer data between industrial devices, control systems, and enterprise applications.

➤ OPC UA Communication Basics

- **Protocol Type:** Client-Server and Publish-Subscribe
- **Physical Layer:** Ethernet/IP-based
- **Data Transmission:** Full-Duplex
- **Error Handling:** Built-in error recovery mechanisms
- **Security:** Integrated authentication, encryption, and data integrity

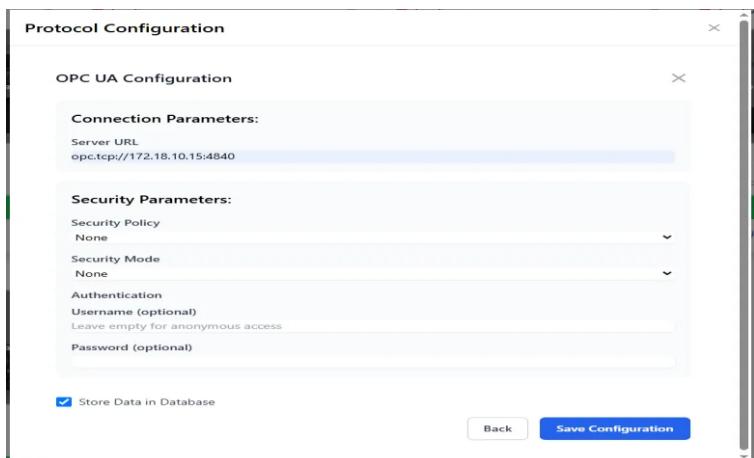
➤ OPC UA Communication Parameters

When configuring OPC UA, the following communication parameters need to be set:

Connection Parameters

Parameters	Description
Server URL	Endpoint URI of the OPC UA server (e.g., opc.tcp://172.18.10.15:4840)
Security Policy	Encryption policy (None, Basic128Rsa15, Basic256, etc.)
Security Mode	Security level (None, Sign, Sign and Encrypt)
Username	User credential for authentication (optional)
Password	Password for authentication (optional)

Configuration Details for OPCUA



➤ Data Handling & Storage

- **Store Data in Database:** Logs real-time machine parameters for historical analysis.
- ### ➤ Stat Card Generation (Based on Marked Section)
- The "Generate Report" button is used to create a **Stat Card**, summarizing key configuration parameters such as:
 - Machine ID
 - Machine Name
 - Communication Settings
 - Modbus Addressing
 - Data Storage Preferences

This stat card provides a quick overview of the device configuration and can be used for diagnostics and validation.

5. BLE Device Manager

Overview

The BLE Device Manager is a comprehensive web-based application designed to scan, connect, and monitor multiple Bluetooth Low Energy (BLE) devices simultaneously. The application provides real-time data streaming, notification management, and device characteristic monitoring capabilities.

The screenshot shows the main dashboard of the BLE Device Manager. At the top, there's a blue header bar with the 'API Client' logo, the title 'BLE Device Manager', and two buttons: '+ New' and 'Back to Dashboard'. Below the header is a purple banner with the text 'Multi-Device BLE Manager' and a subtitle 'Scan, connect, and monitor multiple Bluetooth Low Energy devices simultaneously'. Underneath the banner are four primary action buttons: 'Scan Devices' (blue), '+ Connect Selected (2)' (green), 'Auto Refresh ON' (green), and 'Disconnect All (2)' (red). A section titled 'Select Devices to Connect' contains a dropdown menu showing '2 devices selected'. At the bottom, a green box displays the 'Connected Devices Summary' with the following statistics: Total Connected: 2, Total Notifications: 0, Active Subscriptions: 0, and Data Streams: 5.

This screenshot shows the same dashboard as above, but with a more detailed view of the connected devices. The 'Connected Devices Summary' table remains at the top. Below it, the 'Select Devices to Connect' section is expanded, showing a list of devices with their signal strengths. The list includes:

- Unknown Device (07:3E:35:1B:00:45) with -36 dBm signal strength.
- vivo V50e (Connected, 60:B4:AD:EC:48:C0) with -62 dBm signal strength.
- Mustang BOULT (18:7E:5D:FD:96:F0) with -82 dBm signal strength.
- Unknown Device (0A:69:19:29:5F:20) with -84 dBm signal strength.

vivo V50e
Connected
Disconnect

Address: 60:B4:AD:EC:48:C0

Services (6)

00001801-0000-1000-8000-00805f9b34fb	00001800-0000-1000-8000-00805f9b34fb
0000180d-0000-1000-8000-00805f9b34fb	0000aaa0-0000-1000-8000-aabbccddeeff
0000181c-0000-1000-8000-00805f9b34fb	6e400001-b5a3-f393-e0a9-e50e24dcca9e

Notifiable Characteristics (5)

Service Changed ✗ Failed

UUID: 00002a05-0000-1000-8000-00805f9b34fb
 Service: 00001801-0000-1000-8000-00805f9b34fb
 Error: ("detail": "Error managing subscription: [WinError -2140864509] The attribute cannot be written")

Heart Rate Measurement ✓ Subscribed

UUID: 00002a37-0000-1000-8000-00805f9b34fb
 Service: 0000180d-0000-1000-8000-00805f9b34fb

Live Data 50 of 120 entries

0x0475 Len: 2	3:20:49 PM
0x0475 Len: 2	3:20:48 PM
0x0675 Len: 2	3:20:47 PM

Showing last 5 entries

Important value ✓ Subscribed

UUID: 0000aaa1-0000-1000-8000-aabbccddeeff
 Service: 0000aaa0-0000-1000-8000-aabbccddeeff

Unknown ✓ Subscribed

UUID: 0000aaa2-0000-1000-8000-aabbccddeeff
 Service: 0000aaa0-0000-1000-8000-aabbccddeeff

Nordic UART TX ✓ Subscribed

UUID: 6e400003-b5a3-f393-e0a9-e50e24dcca9e
 Service: 6e400001-b5a3-f393-e0a9-e50e24dcca9e

Live Data 1 of 1 entries

6767 Hex: 36373637 Len: 4	3:19:04 PM
--------------------------------	------------

System Architecture

Core Components

1. **Single Device Manager:** Individual device connection and monitoring
2. **Multi-Device Manager:** Simultaneous management of multiple BLE devices
3. **Device Scanner:** BLE device discovery and listing
4. **Characteristic Monitor:** Real-time monitoring of device services and characteristics
5. **Notification Handler:** Management of device notifications and subscriptions

Workflow Description

1. Device Discovery Phase

Process Flow:

1. Click "Scan for Devices" button to initiate BLE device discovery
2. System scans for available BLE devices in range
3. Discovered devices are listed with:
 - o Device name (e.g., "vivo V50e", "Mustang BOULT")
 - o MAC address
 - o Signal strength (RSSI in dBm)
 - o Connection status

2. Device Selection and Connection

Single Device Mode:

- Select individual device from dropdown
- View device-specific information:
 - o MAC Address
 - o Available services
 - o Device characteristics

Multi-Device Mode:

- Select multiple devices using checkboxes
- Displays total selected devices count
- Batch connection capabilities
- Real-time connection status monitoring

3. Service and Characteristic Discovery

Once connected, the system automatically discovers:

Standard BLE Services:

- **Generic Access Profile (GAP):** 00001800-0000-1000-8000-00805f9b34fb
- **Generic Attribute Profile (GATT):** 00001801-0000-1000-8000-00805f9b34fb
- **Device Information Service:** 0000180a-0000-1000-8000-00805f9b34fb
- **Custom Services:** Device-specific UUIDs

Service Characteristics:

- **Heart Rate Measurement:** 00002a37-0000-1000-8000-00805f9b34fb
- **Battery Level:** 00002a19-0000-1000-8000-00805f9b34fb
- **Device Name:** 00002a00-0000-1000-8000-00805f9b34fb
- **Manufacturer Name:** 00002a29-0000-1000-8000-00805f9b34fb

4. Data Monitoring and Notifications

Notification Management:

- Subscribe to characteristic notifications
- Real-time data streaming

- Configurable notification intervals
- Data logging and timestamp recording

Supported Data Types:

- Heart rate measurements
- Battery level monitoring
- Custom sensor data
- Device status information

Technical Features

Device Connection Management

Connection States:

- **Scanning:** Discovering available devices
- **Connected:** Active connection established
- **Disconnected:** Connection terminated
- **Failed:** Connection attempt unsuccessful

Connection Statistics:

- Total Connected Devices: Real-time count
- Total Notifications: Cumulative notification count
- Active Subscriptions: Number of active characteristic subscriptions
- Data Streams: Number of active data streams

Multi-Device Capabilities

Simultaneous Operations:

- Connect up to multiple devices concurrently
- Individual device monitoring
- Batch operations (connect/disconnect all)
- Auto-refresh functionality for device status

Device Management:

- Signal strength monitoring (RSSI)
- Connection quality assessment
- Automatic reconnection handling
- Device filtering and sorting

Data Processing

Real-time Monitoring:

- Live characteristic value updates
- Timestamp-based data logging
- Configurable sampling rates
- Data export capabilities

Notification Handling:

- Automatic subscription management
- Error handling and retry mechanisms
- Data validation and parsing
- Custom notification filters

User Interface Components

Control Buttons

1. **Scan Devices:** Initiates BLE device discovery
2. **Connect Selected:** Establishes connections to selected devices
3. **Auto Refresh ON/OFF:** Toggles automatic device list refresh
4. **Disconnect All:** Terminates all active connections

Information Panels

1. **Device List:** Shows discovered and connected devices
2. **Connection Results:** Displays connection status summary
3. **Characteristic Monitor:** Real-time characteristic values
4. **Notification Panel:** Active notifications and subscriptions

Status Indicators

- **Signal Strength:** RSSI values in dBm
- **Connection Status:** Visual indicators (Connected/Disconnected)
- **Subscription Status:** Active/Inactive notification states
- **Data Stream Status:** Real-time data flow indicators

Error Handling

Common Error Scenarios:

- Device connection timeout
- Service discovery failures
- Characteristic read/write errors
- Notification subscription failures
- Device disconnection handling

Recovery Mechanisms:

- Automatic retry logic
- Connection state management
- Graceful error reporting
- User notification system

Security Considerations

BLE Security Features:

- Device pairing management
- Encrypted connections
- Authentication handling
- Access control mechanisms

Performance Metrics

Key Performance Indicators:

- Connection establishment time
- Data throughput rates
- Notification latency
- Device discovery time
- System resource utilization

API Integration

The system provides API endpoints for:

- Device management
- Data retrieval
- Configuration settings
- Status monitoring
- Batch operations

Future Enhancements

Planned Features:

- Device grouping capabilities
- Advanced filtering options
- Data analytics dashboard
- Export functionality
- Mobile application support
- Cloud data synchronization

Troubleshooting Guide

Common Issues:

1. **Device Not Found:** Check Bluetooth permissions and device proximity
2. **Connection Failed:** Verify device compatibility and power status
3. **No Notifications:** Confirm characteristic subscription settings
4. **Data Loss:** Check connection stability and buffer settings

6. CoAP Protocol Overview

CoAP (Constrained Application Protocol) is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things, designed for machine-to-machine (M2M) applications such as smart energy.

The screenshot shows a monitoring dashboard for a machine named 'Mazak'. At the top, there's a header with a back arrow, the machine name 'Mazak', and a refresh icon. Below the header, a status bar indicates 'Connection Status: Connected' with a green dot and a 'Disconnect' button. A section titled 'Add Sensor Types' allows users to enter sensor types like 'temperature' and adds them to a list of four selected sensors: Temperature, Humidity, Pressure, and Light Intensity. Each sensor is represented by a colored card showing its current reading and status. The 'Temperature' card is pink and shows 22.97°C with a success status. The 'Humidity' card is blue and shows 46.34% with a success status. The 'Pressure' card is purple and shows 1013.54 hPa with a success status. The 'Light Intensity' card is yellow and shows 9.18 lux with a success status. A timestamp at the bottom right indicates the data was last updated at 11:16:08 AM. The bottom section, 'Machine Configuration', displays the machine ID (47), name (Mazak), and protocol (CoAPP).

Sensor Type	Current Reading	Status
Temperature	22.97°C	success
Humidity	46.34%	success
Pressure	1013.54 hPa	success
Light Intensity	9.18 lux	success

CoAPP Dashboard Workflow Description

Based on monitoring dashboard, here's the workflow description:

System Overview

Machine Name: Mazak (Machine ID: 47)

Protocol: CoAPP (appears to be a variant or custom implementation of CoAP)

Connection Status: Connected (Active real-time monitoring)

Sensor Configuration

The system monitors four primary environmental and operational parameters:

1. **Temperature Sensor**
 - o Current Reading: 22.97°C
 - o Status: Success
 - o Display: Pink/Rose colored card
2. **Humidity Sensor**
 - o Current Reading: 46.34%
 - o Status: Success
 - o Display: Blue colored card

3. Pressure Sensor

- Current Reading: 1013.54 hPa
- Status: Success
- Display: Purple colored card

4. Light Intensity Sensor

- Current Reading: 9.18 lux
- Status: Success
- Display: Yellow colored card

Workflow Process

Data Collection Flow:

1. **Sensor Data Acquisition:** Four sensor types continuously collect environmental data from the Mazak machine
2. **Protocol Communication:** Data transmission occurs via CoAPP protocol over the network connection
3. **Real-time Processing:** System processes incoming sensor data and updates status indicators
4. **Dashboard Visualization:** Color-coded cards display current readings with success status
5. **Timestamp Management:** Last update timestamp (11:16:08 AM) tracks data freshness

Management Features:

- **Dynamic Sensor Configuration:** "Add Sensor Types" functionality allows runtime addition of new sensor parameters
- **Connection Management:** Real-time connection status monitoring with disconnect capability
- **Status Monitoring:** Individual sensor success/failure status tracking
- **Real-time Updates:** Continuous data refresh with timestamp tracking

This system appears designed for industrial monitoring of the machine environment, providing operators with real-time visibility into critical environmental parameters that could affect machine performance and operations.

7. MQTT Protocol Documentation and Interface Guide

MQTT (Message Queuing Telemetry Transport) is a lightweight, open-source messaging protocol designed for machine-to-machine (M2M) communication and Internet of Things (IoT) applications. It follows a publish-subscribe pattern and is ideal for devices with limited bandwidth, high latency, or unreliable networks.

The screenshot shows a user interface for managing an MQTT connection. On the left, there's a thumbnail of a machine labeled 'Machine 57'. Below it, the text 'Protocol: MQTT' and a red 'Disconnect' button. To the right, the main panel has a title 'jkl'. It displays 'Connected to Machine' and 'WebSocket Connected'. Under 'MQTT Configuration', it lists the Broker IP as '172.18.150.33', Port as '1883', and Protocol as 'MQTT'. It also shows 'Not configured' for Username and 'true' for Store in Database. The 'Available MQTT Topics' section lists 'esp32/line1' and 'esp32/line2'. The 'Topic Data' section shows a JSON object for 'esp32/line1': { "hmt": 0.331884, "vtc": 0.28082, "shaub": 0.231883 }. Buttons for 'Refresh Data' and 'Unsubscribe' are at the bottom.

Key Features of MQTT

- **Lightweight:** Minimal code footprint and bandwidth usage
- **Reliable:** Three levels of Quality of Service (QoS) for message delivery
- **Scalable:** Supports thousands of connected devices
- **Secure:** Built-in security features with SSL/TLS support
- **Bi-directional:** Enables two-way communication between devices and servers

MQTT Architecture

MQTT uses a **publish-subscribe** model with three main components:

1. **Publisher:** Device or application that sends messages
2. **Subscriber:** Device or application that receives messages
3. **Broker:** Central server that routes messages between publishers and subscribers

How MQTT Works

1. Devices connect to an MQTT broker
2. Publishers send messages to specific **topics**
3. The broker receives and distributes messages to all subscribers of that topic
4. Subscribers receive messages from topics they've subscribed to

MQTT Topics

Topics are UTF-8 strings used to filter messages. They use a hierarchical structure with forward slashes as separators:

- Example: home/livingroom/temperature
 - Wildcards: + (single level), # (multi-level)
-

Interface Explanation

Based on the interface shown in your image, here's how each component works:

Connection Status Indicators

-  **Disconnected from Machine:** Shows the physical device connection status
-  **WebSocket Connected:** Indicates successful connection to the web interface

MQTT Configuration Section

Broker Settings

- **Broker:** 172.18.150.33 - The IP address of your MQTT broker server
- **Port:** 1883 - Standard MQTT port (non-encrypted)
- **Username:** Not configured - No authentication username set
- **Password:** Not configured - No authentication password set

Storage and Protocol Settings

- **Store in OPC UA Server:** true - Data is being stored in an OPC UA server
- **Store in Database:** true - Data is being saved to a database
- **Machine ID:** 57 - Unique identifier for this specific machine
- **Protocol:** MQTT - Confirms MQTT protocol is being used

Available MQTT Topics Section

This section shows all the topics your device is publishing to:

- **esp32/line2** - Topic for data from line 2 of the ESP32 device
- **esp32/line1** - Topic for data from line 1 of the ESP32 device

Each topic has an  **eye icon** that allows you to:

- View the topic in real-time
- Subscribe to receive updates
- Monitor data flow

Topic Data Panel

Shows real-time data from the selected topic (esp32/line1):

```
{  
  "nnt": 0.366343,  
  "vtc": 0.280584,
```

```
"shaub": 0.319935  
}
```

This JSON data contains sensor readings or measurements with three parameters:

- **nnt**: Numeric value (possibly a sensor reading)
- **vtc**: Voltage or another measurement
- **shaub**: Another sensor parameter

Data Controls

- **Last updated**: 9:44:03 AM - Timestamp of the most recent data
- **Refresh Data** button - Manually refresh the data display
- **Unsubscribe** button - Stop receiving updates from this topic

How to Use This Interface

1. **Monitor Connection**: Check the status indicators at the top
2. **View Available Topics**: Browse topics in the left panel
3. **Subscribe to Topics**: Click the eye icon next to any topic
4. **View Real-time Data**: Selected topic data appears in the right panel
5. **Refresh Data**: Use the refresh button to get the latest values
6. **Manage Subscriptions**: Use unsubscribe to stop monitoring specific topics

Troubleshooting Tips

- If "Disconnected from Machine" shows red, check physical device connection
- Ensure the broker IP (172.18.150.33) is accessible on your network
- Port 1883 should be open and not blocked by firewalls
- If no data appears, verify the ESP32 device is publishing to the correct topics

Data Storage

Your system is configured to:

- Store data in an OPC UA server for industrial automation integration
- Save data to a database for historical analysis and reporting
- Use Machine ID 57 for data organization and identification

This setup provides a complete IoT data pipeline from sensor collection through MQTT to storage and visualization.

8. HTTP Protocol and API Client Interface Documentation

HTTP (HyperText Transfer Protocol) is the foundation of data communication on the World Wide Web. It's an application-layer protocol that defines how messages are formatted and transmitted between web browsers, servers, and other web services.

API Client HTTP

GET http://172.18.100.214:8006/history_data/1

Body

Response

Status: 200 Time: 322.64 ms Size: 56706 B

Response Body:

```
[{"machine_id": 1, "timestamp": "2025-04-22T16:11:37.325330", "phase_a_voltage": 226.9870147705078, "phase_b_voltage": 231.67901611328125, "phase_c_voltage": 231.2440185546875, "avg_phase_voltage": 229.9700164794922, "line_ab_voltage": 397.531005859375, "line_bc_voltage": 397.5650329589844, "line_ca_voltage": 397.9750061035156, "avg_line_voltage": 397.6860046386719, "phase_a_current": 7.140000343322754, "phase_b_current": 7.4200005531311035, "phase_c_current": 8.520000457763672, "avg_three_phase_current": 7.700000286102295, "power_factor": 0.8169099688529968,}
```

API Client HTTP

GET http://172.18.100.214:8006/history_data/1

POST

PUT

DELETE

PATCH

OPTIONS Body:

HEAD

Response

Status: 200 Time: 322.64 ms Size: 56706 B

Response Body:

```
[{"machine_id": 1, "timestamp": "2025-04-22T16:11:37.325330", "phase_a_voltage": 226.9870147705078, "phase_b_voltage": 231.67901611328125, "phase_c_voltage": 231.2440185546875, "avg_phase_voltage": 229.9700164794922, "line_ab_voltage": 397.531005859375, "line_bc_voltage": 397.5650329589844, "line_ca_voltage": 397.9750061035156, "avg_line_voltage": 397.6860046386719, "phase_a_current": 7.140000343322754, "phase_b_current": 7.4200005531311035, "phase_c_current": 8.520000457763672, "avg_three_phase_current": 7.700000286102295, "power_factor": 0.8169099688529968,}
```

Key Features of HTTP

- **Stateless:** Each request is independent and contains all necessary information
- **Request-Response Model:** Client sends requests, server sends responses
- **Text-Based:** Uses human-readable text format
- **Port 80/443:** Standard ports for HTTP (80) and HTTPS (443)
- **Connectionless:** Connection is closed after each transaction (HTTP/1.0) or kept alive (HTTP/1.1+)

HTTP Architecture

HTTP follows a **client-server** model:

1. **Client** (browser, app, API client) sends HTTP requests
2. **Server** processes requests and sends HTTP responses
3. **Messages** contain headers, methods, status codes, and optional body data

HTTP Methods:

HTTP defines several methods (also called verbs) that indicate the desired action for a resource:

GET

- **Purpose:** Retrieve data from server
- **Characteristics:**
 - Safe operation (no side effects)
 - Idempotent (same result each time)
 - Data sent via URL parameters
 - Cacheable
- **Use Cases:** Fetching web pages, API data retrieval, downloading files

POST

- **Purpose:** Submit data to server to create new resources
- **Characteristics:**
 - Not safe (has side effects)
 - Not idempotent (multiple calls may create multiple resources)
 - Data sent in request body
 - Not cacheable
- **Use Cases:** Form submissions, creating new records, file uploads

PUT

- **Purpose:** Update or create a resource with complete replacement
- **Characteristics:**
 - Not safe (has side effects)
 - Idempotent (same result each time)
 - Data sent in request body
 - Replaces entire resource
- **Use Cases:** Updating user profiles, replacing configuration settings

DELETE

- **Purpose:** Remove a resource from the server
- **Characteristics:**
 - Not safe (has side effects)
 - Idempotent (deleting same resource multiple times has same effect)
 - Usually, no request body
- **Use Cases:** Deleting records, removing files, user account deletion

PATCH

- **Purpose:** Partial update of a resource
- **Characteristics:**
 - Not safe (has side effects)
 - Not necessarily idempotent
 - Data sent in request body
 - Updates only specified fields

- **Use Cases:** Updating specific fields, incremental changes

OPTIONS

- **Purpose:** Get information about communication options for a resource
- **Characteristics:**
 - Safe operation
 - Used for CORS preflight requests
 - Returns allowed methods and headers
- **Use Cases:** API discovery, CORS handling, checking server capabilities

HEAD

- **Purpose:** Get headers for a resource without the body
 - **Characteristics:**
 - Safe operation
 - Idempotent
 - Same as GET but without response body
 - **Use Cases:** Checking resource existence, getting metadata, cache validation
-

API Client Interface Analysis

Interface Overview

Your API client is a web-based tool for testing and interacting with HTTP APIs. It provides a clean interface for making HTTP requests and viewing responses.

Current Request Configuration

Endpoint: `http://172.18.100.214:8006/history_data/1`

- **Protocol:** HTTP (not HTTPS)
- **Host:** 172.18.100.214 (local network IP)
- **Port:** 8006 (custom port)
- **Path:** `/history_data/1` (endpoint for historical data with ID 1)

Request Details

- **Method:** GET (currently selected)
- **Body:** Empty (appropriate for GET requests)
- **Purpose:** Retrieving historical data for machine/device ID 1

Response Information

Status Indicators

- **Status:** 200 (Success - OK)
- **Time:** 322.64 ms (Response time)
- **Size:** 56706 B (Response size - approximately 56KB)

Response Data Analysis

The API returns electrical/power monitoring data in JSON format:

```
{  
    "machine_id": 1,  
    "timestamp": "2025-04-22T16:11:37.325330",  
    "phase_a_voltage": 226.9870147705078,  
    "phase_b_voltage": 231.67901611328125,  
    "phase_c_voltage": 231.24401855468875,  
    "avg_phase_voltage": 229.9700164794922,  
    "line_ab_voltage": 397.531005859375,  
    "line_bc_voltage": 397.565032958984,  
    "line_ca_voltage": 397.975006103516,  
    "avg_line_voltage": 397.6860046386719,  
    "phase_a_current": 7.140000343322754,  
    "phase_b_current": 7.420000553131035,  
    "phase_c_current": 8.520000457763672,  
    "avg_three_phase_current": 7.700000286102295,  
    "power_factor": 0.8169099688529968  
}
```

Data Parameters Explained:

- **machine_id**: Identifier for the electrical equipment
- **timestamp**: When the measurement was taken
- **phase_*_voltage**: Individual phase voltages (A, B, C)
- **avg_phase_voltage**: Average of all three phase voltages
- **line_*_voltage**: Line-to-line voltages (AB, BC, CA)
- **avg_line_voltage**: Average line voltage
- **phase_*_current**: Current measurements for each phase
- **avg_three_phase_current**: Average current across all phases
- **power_factor**: Efficiency ratio (0-1, closer to 1 is better)

Interface Workflow

Step 1: Method Selection

1. Click the method dropdown (currently showing "GET")
2. Select appropriate HTTP method from the list:
 - GET: Retrieve data
 - POST: Create new data
 - PUT: Update/replace data
 - DELETE: Remove data
 - PATCH: Partial update
 - OPTIONS: Get server capabilities
 - HEAD: Get headers only

Step 2: URL Configuration

1. Enter the complete API endpoint URL
2. Include protocol (http/https), host, port, and path
3. Add query parameters if needed (for GET requests)

Step 3: Request Body (if applicable)

1. For POST, PUT, PATCH requests, add JSON data in the Body section
2. Set appropriate Content-Type headers
3. Format data according to API requirements

Step 4: Send Request

1. Click the blue "Send" button
2. Wait for response (time shown in status)
3. Review response data in the Response Body section

Step 5: Response Analysis

1. Check status code (200 = success, 400s = client error, 500s = server error)
2. Review response time and size
3. Analyze returned data structure and values
4. Use copy button to export response data

Interface Controls

- **+ New:** Create a new API request
- **← Back to Dashboard:** Return to main dashboard
- **Method Dropdown:** Select HTTP method
- **URL Field:** Enter API endpoint
- **Send Button:** Execute the request
- **Copy Button:** Copy response data
- **Body Section:** Add request payload (for POST/PUT/PATCH)

Best Practices for API Testing

1. **Start with GET:** Always test data retrieval first
2. **Check Status Codes:** Ensure requests return expected status
3. **Validate Data:** Verify response structure and values
4. **Test Error Cases:** Try invalid IDs or malformed requests
5. **Monitor Performance:** Check response times for optimization
6. **Document Results:** Save successful request configurations

Troubleshooting Common Issues

- **Connection Refused:** Check if server is running on specified port
- **404 Not Found:** Verify endpoint URL and path
- **401/403 Unauthorized:** Add authentication headers if required
- **500 Server Error:** Check server logs for backend issues
- **Timeout:** Increase timeout settings or check network connectivity

This interface is ideal for testing your electrical monitoring API, allowing you to retrieve historical power data, test different endpoints, and validate your system's API responses.

9. Conclusion

The Protocol Management Software is a comprehensive industrial IoT integration platform that enables seamless connectivity and data collection from diverse manufacturing machines and devices across multiple communication protocols including Modbus, PROFINET, OPC UA, MQTT, BLE, and HTTP. It serves as a universal data gateway that bridges legacy industrial equipment with modern IT infrastructure, providing real-time monitoring, centralized data storage, and unified dashboard management. The software eliminates protocol compatibility barriers by automatically converting and standardizing data from different sources into consistent formats for database storage and OPC UA server integration. Its primary value proposition is enabling digital transformation for industrial operations by unlocking data from previously isolated systems, facilitating predictive maintenance, performance optimization, and data-driven decision making. This solution is particularly valuable for organizations seeking to modernize their industrial data infrastructure while preserving existing equipment investments and ensuring scalable, secure, and reliable industrial data management.