Using Communication Protocols With MachineLogic

Contents

Overview

In this document you will learn how to set up your MachineMotion controller and use MachineLogic to communicate with external devices using Ethernet/IP, MQTT or HTTP

Typical use cases of bi-directional communication with MachineMotion:

- **Robot-MachineMotion** communication (e.g. 7th axis sensors sending command to a robot)
- **PLC-MachineMotion** communication

Figure 1: MachineMotion

Introduction to Ethernet/IP

Ethernet/IP (Industrial Protocol) is a communication protocol used in industrial automation. It is one of the application layer protocols in the Common Industrial Protocol (CIP) suite, which is managed by the Open DeviceNet Vendor Association (ODVA). For more information and access to the official documentation, please go to odva.org

Definitions

- **EDS file**: The Electronic Data Sheet (EDS) is the standardized file format used in industrial automation to describe the communication capabilities and parameters of devices on a network.
- **Device**: Any piece of equipment or machinery that has the capability to communicate over the Ethernet/IP network.
- **Scanner**: The scanner is the device that initiates and controls the data exchange over the Ethernet/IP network. It polls "Adapter" devices to gather data.
- **Adapter**: Adapter devices responds to the Scanner's request to provide data or services. Adapters can be sensors, actuators, drives or any other component capable of Ethernet/IP communication.

Programming

MachineLogic exposes Ethernet/IP communication through the **Set Output** and **Wait** commands. MachineLogic gives access to 8 user inputs and 8 outputs to enable communication with scanner devices such as a PLC. For more information about configuring an Ethernet/IP network using the MachineMotion controller, please consult this [User Manual](#)

Figure 2: Ethernet/IP

Introduction to MQTT

Definitions

- **Client**: Device that can send (publish) and receive (subscribe) data
- **Packet**: Data sent by a client
- **Topic**: Subject line through which packets are sent
- **Broker**: Piece of software running on a computer which acts as the transit between another device or another broker

Message Queuing Telemetry Transport (MQTT) is a bi-directional lightweight message protocol which consists of a set of rules that defines how Internet of things (IoT) devices can publish and subscribe to data over the Internet. MQTT is used for messaging and data exchange between IoT and industrial IoT (IIoT) devices, such as embedded devices, sensors, industrial PLCs, and now, MachineMotion.

Each client can produce and/or receive data by publishing and/or subscribing. A client can publish a packet for a given topic, and anyone who subscribes to it can receive a copy of all messages for that topic. Multiple clients can subscribe to a topic from a single broker, and a single client can register subscriptions to topics with multiple brokers. This helps in both sharing data and managing and controlling devices. A client cannot broadcast the same data to a range of topics and must publish multiple messages to the broker, each with a single topic given. With MQTT broker architecture, the client devices and server application become decoupled. This allows clients to communicate with a single common recipient, and therefore funnel all information from the same place.

Example:

Here is a simple example to illustrate a common situation in which the user wants MachineMotion to communicate with a robot.

- **Packet:** Sensor's message
- **Payload:** Sensor message's data (e.g. 0 or 1)

- **Topic:** Sensor-Topic

- **Client 1:** MachineMotion controller
- **Broker 1:** MachineMotion's MQTT broker
- **Broker 2:** Robot's MQTT broker
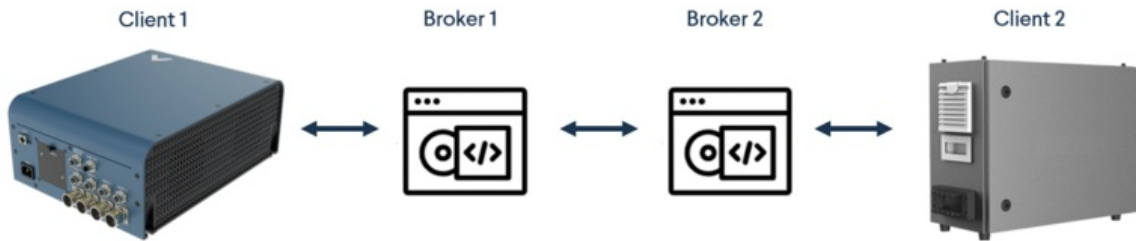- **Client 2:** Robot

Figure 3: MQTT example 1

If a sensor communicating with MachineMotion has "publish" capabilities, Broker 1 (MachineMotion's broker) has subscribe and publish capabilities, and the robot's broker has subscribed to MachineMotion's broker, a change of state of the sensor will automatically be received by the robot. In this specific case, the sensor status will send the signal to MachineMotion, which will then send the message to the robot.

Programming

MachineLogic allows you to easily program your machine through its graphical interface and its low-code infrastructure. When it comes to MQTT communication, the same simple programming approach applies. To create a MachineLogic program using MQTT communication, here are the main commands:

To subscribe to a given topic, the *State Machine* or *Wait For* command can be used.
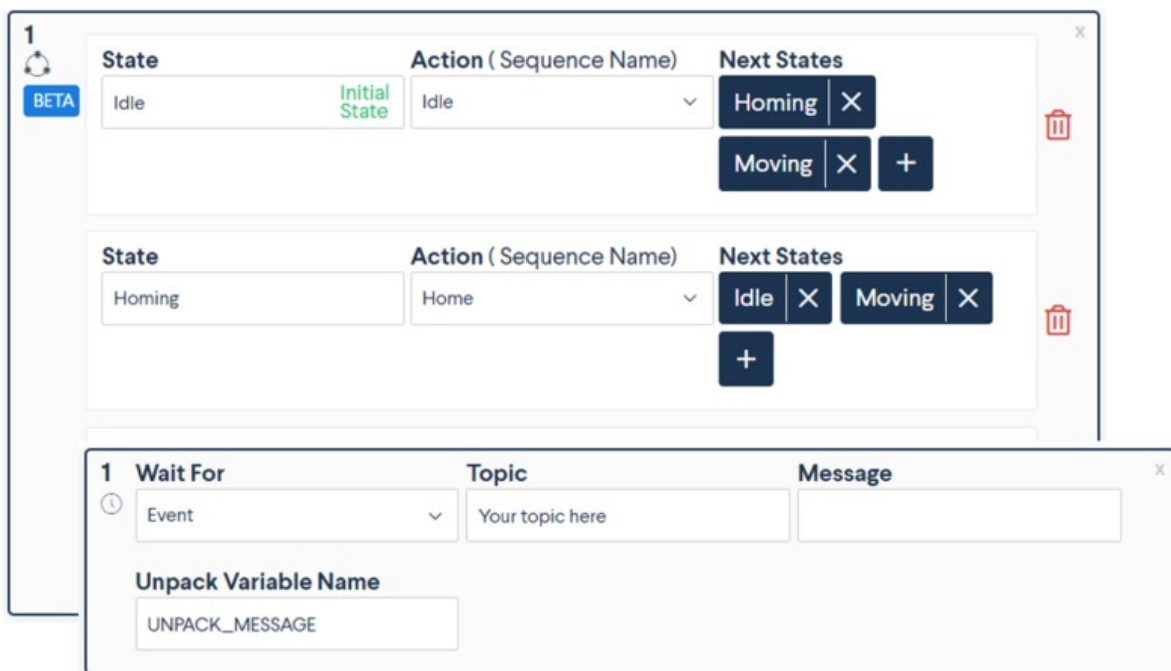

Figure 4: Subscribe to a topic in MachineLogic

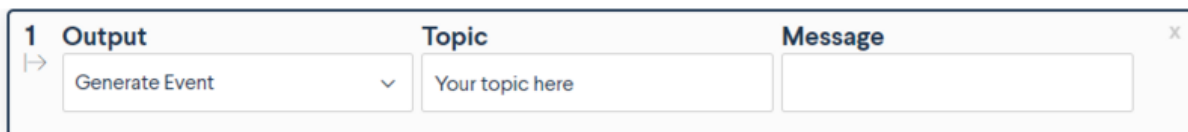To publish to a given topic, the *General Event* output command can be used.

Figure 5: Publish to a topic in MachineLogic

For more complex applications, you can also include payloads via the Variables and Functions features.
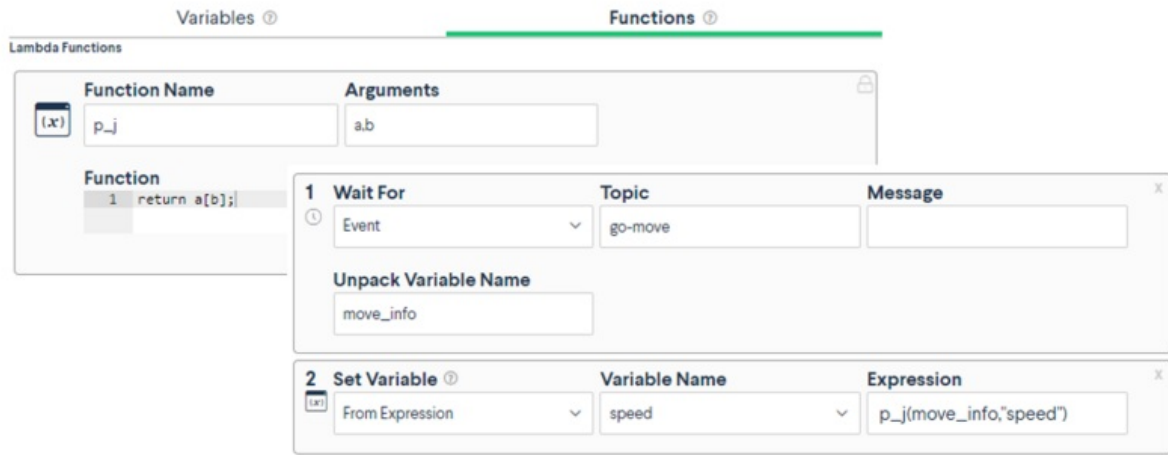
Figure 6: Variables and functions feature on MachineLoigc

Example:

To illustrate what you learned in this document, let's use a design where a robot is mounted on a robotic range extender using a MachineMotion controller.

For a simple use case, the range extender could alternate between 3 different states:

States	Description	Conditions	Next states	Trigger type	Input/Topic	Payload required
Idle	Robot movement along the range extender is stopped	• Range extender must be moving • Brake needs to be activated at the end	Homing	MQTT event	gohome	No
			Moving	MQTT event	gomove	Yes
Homing	Robot moves to the home position of the range extender	• Robot must be ready to move • Brakes are removed	Idle	MQTT event	goidle	No
			Moving	MQTT event	gomove	Yes
Moving	Robot is moving along the range extender	• Robot must be ready to move • Distance requested must be within the stroke limits of the range extender	Idle	MQTT event	goidle	No

Figure 7: MQTT states example

In order to communicate with the robot, the range extender must subscribe and publish different topics using TCP/IP communication:

Publish/Subscribe	Input/Topic	Description	Payload
Subscribe	gohome	Moves the robot to the home position of the range extender	{"speed": X, "accel": Y, "distance": Z}
Subscribe	goidle	Stops the movement of the range extender	N/A
Subscribe	gomove	Move the range extender for a specific distance, at a set speed and acceleration.	{"speed": X, "accel": Y, "distance": Z}
Publish	ishome	Confirmation sent to the robot when at home position on the range extender	N/A
Publish	isidle	Confirmation and exact position sent to the robot when the range extender is stopped	{"position": X }
Publish	ismoving	Signal sent to the robot while the range extender is in movement	N/A

Figure 8: Publish and subscribe MQTT example

In this situation, having a range extender communicating with a robot allows the user to easily program a sequence in which the range extender reacts to a few inputs from the robot and vice versa. Digital I/O communication could have been used in this example, but the transfer of payload for specifications such as distance, speed and acceleration would not have been straightforward. Please keep in mind that this example is for training purposes, and it would probably require adjustments to meet your specific needs.

Wiring & Safety

MachineMotion has a Mosquitto MQTT broker. It can therefore publish and subscribe to MQTT topics to send/receive packets of data. Through a single Local Area Network (LAN) cable connection, you can connect your MachineMotion to the other controller or a router. As a reminder, MachineMotion runs on a server 192.168.7.2
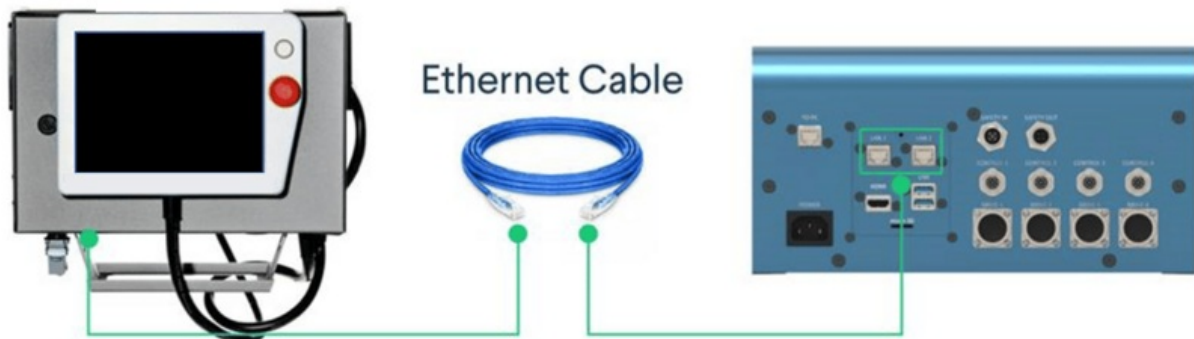

Figure 9: MQTT ethernet wiring

In terms of safety, the Robot Safety Module is the interface between Vention's MachineMotion 2 controller & robots' safety interfaces. The Robot Safety Module manages the safety fault events that happen on the machine to safely stop both the MachineMotion 2 controller and the robot. Please find below the wiring table for a typical use case in which bi-directional safety is required when it comes to a project using MQTT:
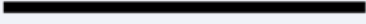
Pins	Wire
From robot	
Pin 1 - Robot Safety Output 0V Contact 1	
Pin 2 - Robot Safety Output 24V Contact 1	
Pin 3 - Robot Safety Output 0V Contact 2	
Pin 4 - Robot Safety Output 24V Contact 2	
To robot	
Pin 3 – Robot Safety Input Channel 1 Contact 1	
Pin 4 – Robot Safety Input Channel 1 Contact 2	
Pin 5 – Robot Safety Input Channel 2 Contact 1	
Pin 6 – Robot Safety Input Channel 2 Contact 2	
Pin 7 – Robot Input Reset Contact 1	
Pin 8 – Robot Input Reset Contact 2	

Figure 10: MQTT safety wiring

The client would have 2 requirements to be compatible with this solution: an MQTT broker and a safety interface (for bi-directional safety). For the latter, here are the minimum requirements:

- 2 x dry contact inputs for STO (to be connected on the TO ROBOT connector of the RSM)
- 2 x 24V safety output (to be connected on the FROM ROBOT connector of the RSM)
- 1 x RJ45 connector (to communicate with MMv2 and pendant via Ethernet and the RSM)

Please refer to the Robot Safety Module technical documentation for more details.

Introduction to HTTP

Definitions

- **Client**: A software application or a program such as a web browser that initiates requests to web servers.
- **Requests**: the request from the client to the server specifying the URL of the resource it wishes to retrieve as well as the request method (GET, POST, PUT, DELETE).
- **Servers**: A software application or a program that listens for incoming requests from clients, processes those requests, and sends back corresponding HTTP responses.
- **Response**: Message sent by the server to a client. An HTTP response contains the status line, the response header and response body.

HTTP stands for Hypertext Transfer Protocol. It is an application layer protocol used for data communication on the World Wide Web. HTTP facilitates the transfer of various resources, such as HTML documents, images, videos, and other types of data, between a client (usually a web browser) and a web server.

The basic concept behind HTTP is the request-response model. When a client wants to access a resource hosted on a web server, it sends an HTTP request to the server. The server processes the request and responds with the requested resource, along with an HTTP response containing the status of the request (e.g., success, error, redirection) and additional metadata about the resource. HTTP operates on top of the TCP/IP (Transmission Control Protocol/Internet Protocol) network stack and typically uses TCP as its transport protocol.
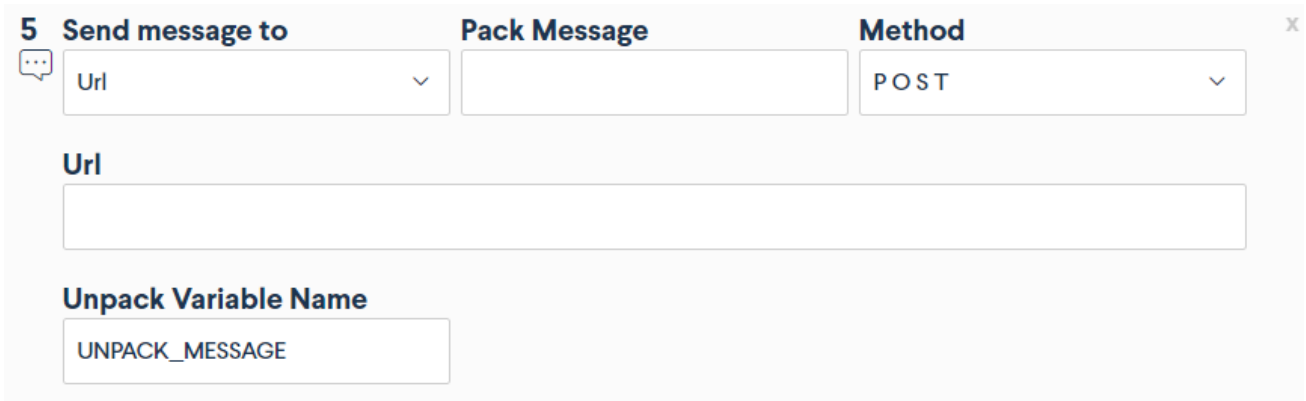
Request Methods

as of firmware version 2.12, MachineLogic supports the HTTP methods listed below:

- **GET**: The GET method is used to request data from a specified resource. It retrieves data from the server without changing anything on the server's side. It is commonly used for fetching web pages, images, or other resources.
- **POST**: The POST method is used to submit data to be processed to a specified resource. It is often used when submitting forms on web pages, sending data to a server to create new resources or update existing ones.
- **PUT**: The PUT method is used to update a resource on the server. It sends the data as a representation of the resource to be updated at a specific URL.
- **DELETE**: The DELETE method is used to remove a resource from the server. It sends a request to delete the specified resource.

Programming

MachineLogic's Code-Free programming allows you to easily program your machine through its graphical interface and its low-code infrastructure. When it comes to HTTP communication, the same simple programming approach applies. To send a HTTP request from MachineLogic, use the **Add Message** command and select URL in the Send message to field:
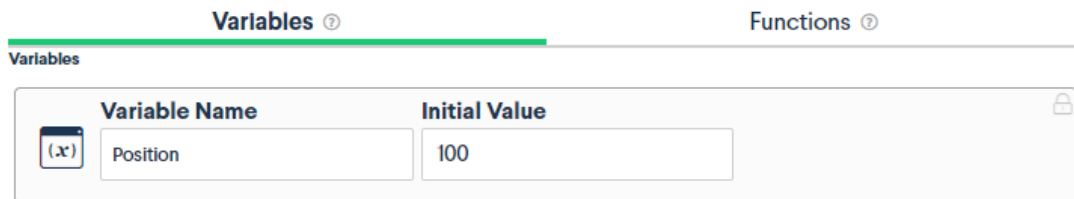
Figure 11: Add Message command

Example:
In the following example, we will show how to format application variables so they can be sent using a POST request from MachineLogic Code-Free programming interface. This can be used to send a log of an actuator's position to an express server:
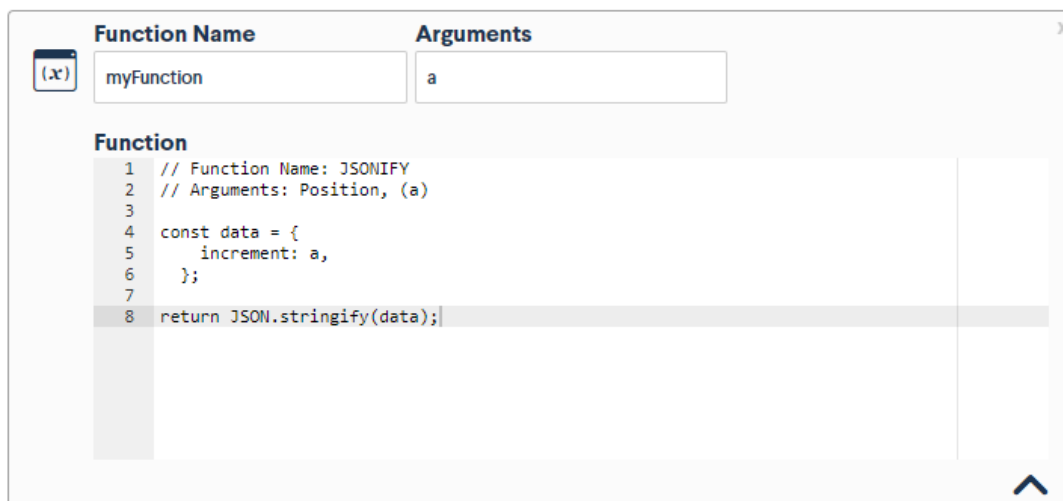
Step 1: create the application variables:

Figure 12: Creating application variables

Step 2: Format the application variables in json format using lambda functions:

Figure 13: Formatting variables using Lambda Functions

Step 3: Send the HTTP request using the Add Message command:

Figure 14: Send application variables as HTTP Request

This examples assumes a server is running on port 3169 of the user's machine listening on the route logPositionData. The server response is encapsulated in the UNPACK_MESSAGE.