MQTT for Events and Commands

Rationale

Communication between software systems and programmable logic controllers requires a transport and protocol that guarantees behaviour so we can reduce additional logic for each solution. MQTT provides persistent subscriptions and guaranteed delivery, OPCUA provides on demand streaming. Both transport layers are applied depending on solution requirements.

MQTT	Discrete events. Guaranteed delivery. Activity coordination. Send commands to other systems. Receive results or failures.
OPCUA	Realtime streaming of system state. Ad-hoc activity statuses. Alerts and estops.

MQTT is a general purpose transport layer. In this document we define a protocol that runs on top of MQTT for the purpose of sending commands and publishing events between software systems and programmable logic controllers. The protocol is general purpose and provides implementation details to govern the exchanging of information and behaviour between any systems. It’s possible to have many systems interacting using this protocol within a single MQTT broker.

Topic format

MQTT messages are published with a topic. Applying principles from AWS MQTT design best practices we propose the following topic formats for events and commands. Explicit parameters are defined within a solution.

Client Solution Environment	A unique identifier for a client's solution and environment so all systems are grouped together. Unique per environment to reduce accidental cross environment messages.
System Name	A unique identifier for a specific system that is capable of generating events or executing commands.
Event / Command Name	A unique identifier of the event or command offered by a system.

Event

`<client-solution-environment>/<systemname>/evt/<eventname>`

Example:

`SpaceX-SLC-40-UAT/LaunchController1/evt/VesselLaunchSuccessful`

Command

`<client-solution-environment>/<systemname>/cmd/<commandname>/<action>`

Example:

`SpaceX-SLC-40-UAT/LaunchController1/cmd/LaunchVessel/request`

`SpaceX-SLC-40-UAT/LaunchController1/cmd/LaunchVessel/response`

`SpaceX-SLC-40-UAT/LaunchController1/cmd/LaunchVessel/reject`

`SpaceX-SLC-40-UAT/LaunchController1/cmd/LaunchVessel/cancel`

Events

An event is a discrete notification of an important piece of information. One system publishes an event and many systems can subscribe. Events are the closest to a standard MQTT messages but require the following configuration to adhere to the protocol defined by this document. This same configuration is required for command messages.

QoS 1 or 2	QoS 1 At least once delivery QoS 2 Exactly once delivery This protocol requires more packets to validate successful delivery The MQTT broker will retain the message until it verifies a successful delivery to the subscriber.
Expect persistent session on reconnect	When a system subscribes to a topic the MQTT broker will persist this subscription when the system is offline, queuing messages to send when a system reconnects. To support receiving messages delivered while offline a system needs to reconnect with the persistent session mode.
UTF-8 encoded JSON payload format	JSON provides structure and schema without requiring in-depth parsing.

Commands

A command is offered by a server system and can be requested by a client system. The server subscribes to the /request and optionally /cancel topics, the client subscribes to the /resolve and /reject topics.

Request	**/request**	Client initiates a command by publishing to the /request topic and passes a messageId alongside optional information in the JSON payload for tracking the client's unique request.
Reject	**/reject**	Server rejects a command by publishing to the /reject topic and passes the messageId from the request alongside optional information in the JSON payload to communicate the reason for rejection. The client will receive this message. If a client receives a messageId that they have not seen before, they can ignore the message.
Resolve	**/resolve**	Server completes a command by publishing to the /resolve topic and passes the messageId from the request alongside optional additional information in the JSON payload to communicate successful results. The client will receive this message. If a client receives a messageId that they have not seen before, they can ignore the message.
Cancel	**/cancel**	Optionally a client could attempt to cancel a request by publishing to the /cancel topic and pass the messageId from the request alongside optional information in the JSON payload to communicate the reason for cancellation. If implemented at all it's recommended this is on a best effort basis. A successful cancel could be detected in the reject payload, but this document's protocol does not provide guidance on how this occurs.

Example command:

`SpaceX-SLC-40-UAT/LaunchController1/cmd/LaunchVessel/request`	Client wants to launch the vessel. Payload may include countdown information.
`SpaceX-SLC-40-UAT/LaunchController1/cmd/LaunchVessel/resolve`	LaunchController1 publishes the result of successfully receiving the launch command. This may mean the launch has happened or that the command has been received but further communication will occur later.
`SpaceX-SLC-40-UAT/LaunchController1/cmd/LaunchVessel/reject`	LaunchController1 rejects the launch command. Could be incorrect parameters or the result of other state.
`SpaceX-SLC-40-UAT/LaunchController1/cmd/LaunchVessel/cancel`	Client wants to cancel. The LaunchController1 may be able to cancel the launch or it may be too late.