ADR-20 Naming, UUIDs, SSOT, and Information Locus for System Points

ADR-20

Date: 2025-05-22
Time: 1:46 PM PDT

Version: 2.5 (Refined command topic target derivation; Added "Manual" component concept; Restored original v2.3 phrasing and incorporated discussion updates)

Context:

The Mushroom Automation System V2 relies heavily on a Single Source of Truth (SSOT) for all configurations. This includes the definition of all data points within the system. Clear, consistent, and adaptable strategies for naming, uniquely identifying (UUID), and defining the locus of information for these points are crucial for system integrity, maintainability, and scalability. This ADR aims to precisely define where attributes are defined, naming conventions, and the flow of information to runtime artifacts, emphasizing an explicit and generative approach for MQTT topic construction based on structured fields in `system_definition.yaml`.

Core Principles for Information Distribution in SSOT (Aligned with SSOT Plan PDF & Topic Naming Proposal PDF):

1. `system_definition.yaml` **as the Master Point List & Component Inventory:**
    - This file is the primary SSOT. It contains a comprehensive list of all data points in the system within its `points:` section.
    - For each point, `system_definition.yaml` defines its globally unique `uuid` and its core system-level attributes, including new fields to explicitly support structured topic generation (see Section II.A, e.g., `function_grouping`, `topic_device_slug`, `topic_originator_slug`, `topic_directive_slug`, `topic_status_slug`). The `units` field will directly contribute to sensor topic paths.
    - It also defines all system `components` and their `id`s (`source_component_id`). This includes a conceptual component for manual inputs (e.g., `id: "manual_inputs"`).
    - `points_provided` / `virtual_points_provided` in component definitions link components to the UUIDs they source.
    - `system_definition.yaml` links to component-specific configuration files.
    - A global MQTT topic prefix (e.g., "mush/") must be defined (e.g., `global_settings.mqtt_topic_prefix`).
2. **Component-Specific YAMLs for Implementation Details:** (As per v2.3)
    - These files provide implementation details for points sourced by that component (e.g., pin assignments for microcontrollers).
    - They reference UUIDs from `system_definition.yaml`. MQTT topic construction details are now primarily driven by `system_definition.yaml` fields.
3. **Pydantic Models for Validation:** (As per v2.3, Point Definition in `core_ssot_models.py` will

be updated based on changes in Section II.A of this ADR).

- `core_ssot_models.py` defines models for `system_definition.yaml`, including the enriched `PointDefinition` with new topic-related fields.
- `component_configs.py` defines models for component-specific YAMLs.
- `MicrocontrollerConfig.HardwarePoint` will not contain manual topic suffix fields.

4. **`build.py` as the Consolidator, Validator, and Generator (Enhanced Role for Topic Generation):**
   - Reads all YAMLs and validates them.
   - **Constructs Full MQTT Topics:** Uses `GLOBAL_MQTT_PREFIX`, `source_component_id`, and the explicit topic-related fields from `PointDefinition` in `system_definition.yaml` (e.g., `function_grouping`, `topic_device_slug`, etc.) to deterministically generate the full MQTT topic path for each point. It will slugify `units` for sensor topics and derive the target component for command topics using `controls_drivers` / `controls_microcontroller` relationships from the source component's definition.
   - Populates `global_point_registry.json` (this registry can serve as a data source for Grafana metadata).
   - Generates `autogen_config.h` for microcontrollers, including these derived full MQTT topics.
5. **UUID as the Immutable Global Identifier.** (As per v2.3)
6. **Functional Naming (System-Wide):** (As per v2.3) The `name` field in `system_definition.yaml` (master `points:` list) remains the primary human-readable functional identifier. Its role in direct topic generation is reduced in favor of the new explicit topic fields.
7. **Local Naming (Component-Specific & C++ Macros):** (As per v2.3) The local `name` in component YAMLs (for C++ macros) is systematically derived (e.g., ComponentID + functional part of master name).
8. **MQTT Topic Structure Rationale:** (As per ADR-013 v2.1/v2.2, and refined by Topic Naming Proposal PDF, and further refined here in v2.5)
   - The topic structure is now: `GLOBAL_MQTT_PREFIX` + `/` + `source_component_id` + `/` + `function_grouping_slug` + `/` + type_specific_path_elements_from_dedicated_slug_fields...
   - This structure is explicit, robust, and directly derived from SSOT fields.
   - UUID remains key for data analysis continuity if `source_component_id` changes.

I. Point Types by Architectural Layer

(Content from v2.2 of ADR-20 - unchanged by v2.3 or v2.5)

A. **Layer 0: Physical Hardware (Interfaced by Microcontrollers)**

* Direct Sensor Inputs: Raw values from sensors connected to microcontroller pins (e.g., analog temperature sensor, digital switch).

* Complex Sensor Inputs (Multi-Value): Data from sensors like I2C devices (e.g., SCD41 providing CO2, Temperature, Humidity). Each distinct measured value is treated as a separate point in

`system_definition.yaml`.
* Actuator Outputs: Control signals for physical actuators (e.g., relay on/off, PWM duty cycle for a fan).
* Actuator Readbacks: Feedback from an actuator confirming its state, published by the microcontroller.

B. **Layer 1: Microcontrollers (C1, C2, C3)**
* Published Physical Sensor Values (Publish): Processed or direct values from L0 sensors, formatted and published via MQTT. Each distinct sensor value (e.g., SCD41 CO2, SCD41 Temperature) is a separate point.
* Published Actuator Readbacks (Publish): Confirmation of actuator states, published via MQTT.
* Microcontroller System Information Points (Publish): Internally generated data (e.g., "C1_Uptime", "C2_LastRestartReason", "C3_WiFi_RSSI"), published via MQTT.
* Subscribed Actuator Command (Write) Points (Subscribe): MQTT topics the microcontroller listens to for commands to control L0 actuators.

C. **Layer 2: Drivers (e.g., TemperatureDriver, HumidityDriver)**
* Driver Input Points (Subscribe - Consumed from Microcontrollers/Data Processing):
* Sensor values (e.g., point representing "C1_AmbientTemperature" consumed by TemperatureDriver).
* Actuator readbacks (e.g., point representing "C2_HeaterRelayReadback" consumed by TemperatureDriver).
* Driver Output Points (Publish - Commands to Microcontrollers): Logical commands generated by the driver's FSM/control logic (e.g., point representing "TempDriver_HeaterCommand" which targets the MQTT topic for C2's heater relay write point).
* Driver Internal State Points (Publish - for Monitoring/Governors): Information about the driver's own state (e.g., point representing "TempDriver_FSM_State").
* Driver Configuration/Setpoint Input Points (Subscribe - from Governors/Users): MQTT topics the driver subscribes to for configuration or setpoints (e.g., point representing "TempDriver_TargetTemperatureSetpoint").

D. **Layer 2.5: Data Processing Layer**
* Consumed Input Points (Subscribe): Raw sensor values, actuator readbacks.
* Published Processed/Filtered/Synthetic Sensor Values (Publish): Calibrated data, averages, derived metrics.
* Published Data Quality & Staleness Indicators (Publish).

E. **Layer 3: Governors (e.g., TemperatureGovernor, EnvironmentGovernor)**
* Governor Input Points (Subscribe - Consumed from Drivers/Data Processing/User): Processed sensor values, driver states, user requests.
* Governor Output Points (Publish - Commands/Setpoints to Drivers): Logical commands or setpoints.
* Governor Internal State Points (Publish - for Monitoring): Governor's own state, active strategy.

F. **Layer 4: User / Manual Interaction Layer**
* Manual Input Points (Publish - by User): Points for direct user influence. These can be conceptually sourced by a "Manual" component type (e.g., `id: "manual_inputs"`) defined in `system_definition.yaml`.
* Consumed Display Points (Subscribe - by Visualization Tools): Any point visualized.

II. Information Locus: Where Point Attributes are Defined (Incorporating Topic Naming Proposal)

A. Master Point Definition (in `system_definition.yaml` `points:` list, validated by `PointDefinition` Pydantic model from `core_ssot_models.py`)

```
For every point in the system, the following attributes are defined here:
* `uuid`: (String) Globally unique, immutable identifier.
* `name`: (String) Primary human-readable functional name (e.g., "FruitingChamber
_HeatingPad_Write", "FruitingChamber_SHT85-0_Temperature_Raw").
* `description`: (String, Optional).
* `value_type`: (Enum/Literal).
* `units`: (String, Optional) e.g., "degF", "on/off", "ppm". `build.py` will slug
ify this for sensor topic paths.
* `data_source_layer`: (Enum/Literal, e.g., microcontroller, driver, governor, ma
nual_input). Required. (For manual inputs, `build.py` links this to a conceptual
"manual_inputs" component ID).
* `access`: (Enum/Literal).
* `writable_by`: (List of Strings, Optional). This field primarily serves as an a
uthorization check and for documentation. The target component for command topic
s is derived differently (see `build.py` logic).
* `readback_point_uuid`: (String, Optional) For write/command points, links to th
e UUID of the corresponding readback/status point.
* `persist_to_db`: (Boolean).
* `validation_rules`: (Dict, Optional).
* `initial_value`: (Any, Optional).
* `linked_points`: (Dict, Optional).
* **New Fields for Explicit Topic Generation (based on "Topic Naming Proposal PD
F" and refined in v2.5):**
    * `function_grouping`: Literal["actuators", "sensors", "statuses", "command
s"] (Required. `build.py` slugs this for the topic path).
    * **Conditional Fields (based on `function_grouping`. Values should be pre-sl
ugified or `build.py` must slugify them):**
        * If `function_grouping == "actuators"`:
            * `topic_device_slug`: str (e.g., "fruiting_chamber_heating_pad").
        * If `function_grouping == "sensors"`:
            * `topic_originator_slug`: str (e.g., "sht85_0_fc", "ds18b20_1_substr
ate").
            * (The metric part of the topic will be the slugified `units` field,
e.g., "degf", "percent_rh").
        * If `function_grouping == "commands"`:
            * `topic_directive_slug`: str (e.g., "heating_pad", "mode", "target_s
etpoint"). `build.py` might append a fixed segment like `/write` or `/set` based
on convention.
            * (The target component for the command topic path is derived by `bui
ld.py` from the source component's `controls_drivers` or `controls_microcontrolle
r` list, see Section II.C).
        * If `function_grouping == "statuses"`:
            * `topic_status_slug`: str (e.g., "fsm_state", "wifi_uptime", "last_r
estart_reason").
```

B. **Component-Specific Implementation Details (in Component YAMLs)**

```
B.1. **For Microcontroller Points (defined in `MicrocontrollerConfig` YAML, `hard
ware_points` list):** (As per v2.3)
* `name`: (String) Local functional name for C++ macro generation (e.g., "C1_SHT8
5_0_Temperature").
* `point_kind`: (Enum/Literal: actuator, sensor_data, system_info). Must align wi
th the `function_grouping` of the linked UUID(s) from `system_definition.yaml`.
* `pin`: (String/Int, Optional).
* `pin_mode`: (Enum/Literal, Optional).
* `initial_state`: (Enum/Literal, Optional) e.g., LOW, HIGH, 0.
* For `point_kind: "actuator"`:
    * `write_point_uuid_ref`: (String) UUID of the master write point (from `syst
em_definition.yaml`) that this physical actuator is commanded by.
    * `readback_point_uuid_ref`: (String) UUID of the master readback point (fro
m `system_definition.yaml`) that this physical actuator publishes.
* For `point_kind: "sensor_data"` or `"system_info"`:
    * `data_point_uuid_ref`: (String) UUID of the master data point (from `system
_definition.yaml`).
* `attributes`: (Dict, Optional).
* **NO `mqtt_topic_suffix_*` fields.** (As per v2.3)

B.2. **For Driver Points & B.3. For Governor Points:** (As per v2.3, refined in v
2.5)
* These will similarly reference UUIDs from `system_definition.yaml`.
* **Topic Suffixes Removed:** The `mqtt_topic_suffix_*` fields in their `output_p
oints` definitions (if any existed in older plans) are removed. `build.py` will d
erive their full publish topics using the new explicit topic-related fields from
the corresponding `PointDefinition` in `system_definition.yaml`.
* Their `input_points` will be lists of UUIDs, and `build.py` will resolve these
to full topics from the `global_point_registry.json`.
```

C. **Information Derived by `build.py` (and stored in `global_point_registry.json` and `autogen_config.h`)**

```
* `source_component_id`: Determined from `system_definition.yaml`.
* **Full MQTT Topics (Key Change - Based on New Explicit Fields and v2.5 refinements):**
    `build.py` uses `GLOBAL_MQTT_PREFIX`, `source_component_id` (of the publishing component), and the new explicit topic fields from `PointDefinition`.
* **Example Topic Construction by `build.py` (slugification of field values assumed):**
    * Actuator Readback (sourced by uC "c2", `function_grouping: "actuators"`, `topic_device_slug: "fruiting_chamber_heating_pad"`):
        `mush/c2/actuators/fruiting_chamber_heating_pad/readback`
    * Sensor Data (sourced by uC "c1", `function_grouping: "sensors"`, `topic_originator_slug: "sht85_0_fc"`, `units: "degF"` -> slug "degf"):
        `mush/c1/sensors/sht85_0_fc/degf`
    * **Command Topic Derivation (Revised for v2.5):**
        * Source Component (e.g., "temperature_governor", `id: "temp_gov"`) publishes a command.
        * The command point in `system_definition.yaml` has `function_grouping: "commands"` and (e.g.) `topic_directive_slug: "heating_pad"`.
        * The "temperature_governor" component definition in `system_definition.yaml` has `controls_microcontroller: ["c2"]` (or `controls_drivers` if targeting a driver).
        * `build.py` identifies "c2" as the target component slug for the path.
        * `build.py` might append a conventional suffix like `/write` based on the `function_grouping` being "commands".
        * Topic: `mush/temp_gov/commands/c2/heating_pad/write`
    * Status (sourced by driver "temp_drv", `function_grouping: "statuses"`, `topic_status_slug: "fsm_state"`):
        `mush/temp_drv/statuses/fsm_state`
* **Contents of `autogen_config.h`:**
    * `DEVICE_ID`.
    * WiFi, MQTT, NTP configurations.
    * For each `HardwarePoint` entry in `MicrocontrollerConfig`:
        * `#define POINT_NAME_[LocalName] "[LocalName]"`
        * `#define UUID_[LocalName]_[ASPECT] "[Master_UUID]"` (ASPECT is WRITE, READBACK, or DATA)
        * `#define TOPIC_[LocalName]_[ASPECT] "[Full_Derived_MQTT_Topic_String]"`
            * For `WRITE` topics (commands uC subscribes to), the topic string is the full topic of the command point (which will be sourced by a driver/governor). `build.py` resolves this using the `write_point_uuid_ref` and the global registry, ensuring it matches the topic the driver/governor publishes to.
            * For `READBACK` or `DATA` topics (which uC publishes), the topic string will use the uC's `DEVICE_ID` as the `source_component_id` part.
        * `#define PIN_[LocalName] [PinNumber]`
        * `#define MODE_[LocalName] [PinMode]`
        * `#define INITIAL_STATE_[LocalName] [State]`
    * Timing constants.
* **Updated `autogen_config.h` Example Snippet (Illustrative for C2, reflecting v2.5 command topic intent):**

    ```cpp
    // ... (DEVICE_ID "c2", WIFI, MQTT, NTP settings, GLOBAL_MQTT_PREFIX "mush")

    // --- FruitingChamber_HeatingPad (Actuator on C2) ---
```
```

```
    // FruitingChamber_HeatingPad (Actuator on C2)
    // Master Point Definitions (system_definition.yaml) for UUIDS referenced bel
ow:
    // uuid: "8e43f1a2-b3c4-4d5e-6f70-8192a3b4c5d9" (FruitingChamber_HeatingPad_C
ommand)
    //   name: "FruitingChamber_HeatingPad_Command", function_grouping: "command
s",
    //   topic_directive_slug: "fruiting_chamber_heating_pad", // build.py append
s /write
    //   data_source_layer: driver (publishing source is e.g., "temperature_drive
r_fruiting")
    //
    // uuid: "7d32f1a2-b3c4-4d5e-6f70-8192a3b4c5d8" (FruitingChamber_HeatingPad_R
eadback)
    //   name: "FruitingChamber_HeatingPad_Readback", function_grouping: "actuato
rs",
    //   topic_device_slug: "fruiting_chamber_heating_pad",
    //   data_source_layer: microcontroller (publishing source is "c2")

    // MicrocontrollerConfig hardware_point entry (in c2_config.yaml):
    //   name: "C2_FruitingChamberHeatingPad"
    //   point_kind: "actuator"
    //   write_point_uuid_ref: "8e43f1a2-b3c4-4d5e-6f70-8192a3b4c5d9"
    //   readback_point_uuid_ref: "7d32f1a2-b3c4-4d5e-6f70-8192a3b4c5d8"
    //   pin: 26, pin_mode: "OUTPUT", initial_state: "LOW"

    #define POINT_NAME_C2_FRUITINGCHAMBERHEATINGPAD "C2_FruitingChamberHeatingPa
d"
    #define UUID_C2_FRUITINGCHAMBERHEATINGPAD_WRITE "8e43f1a2-b3c4-4d5e-6f70-8192
a3b4c5d9"
    #define UUID_C2_FRUITINGCHAMBERHEATINGPAD_READBACK "7d32f1a2-b3c4-4d5e-6f70-8
192a3b4c5d8"
    #define PIN_C2_FRUITINGCHAMBERHEATINGPAD 26
    #define MODE_C2_FRUITINGCHAMBERHEATINGPAD OUTPUT
    #define INITIAL_STATE_C2_FRUITINGCHAMBERHEATINGPAD LOW

    // C2 subscribes to the command topic published by the driver.
    // build.py determines the target component for the command path (e.g., "c2")
    // from driver's "controls_microcontroller" list and appends /write.
    #define TOPIC_C2_FRUITINGCHAMBERHEATINGPAD_WRITE "mush/temperature_driver_fru
iting/commands/c2/fruiting_chamber_heating_pad/write"

    // C2 publishes its readback to its own topic:
    #define TOPIC_C2_FRUITINGCHAMBERHEATINGPAD_READBACK "mush/c2/actuators/fruiti
ng_chamber_heating_pad/readback"

    // --- FruitingChamber_SHT85-0_Temperature_Raw (Sensor on C1, example if thi
s was autogen_config_c1.h) ---
    // Master Point Definition (system_definition.yaml):
    // uuid: "f47ac10b-58cc-4372-a567-0e02b2c3d479", name: "FruitingChamber_SHT85
-0_Temperature_Raw",
    //   function_grouping: "sensors", topic_originator_slug: "sht85_0_fc", unit
s: "degF",
    //   data_source_layer: microcontroller (source_component_id will be "c1")
```

```
    // MicrocontrollerConfig hardware_point entry (in c1_config.yaml):
    //   name: "C1_FruitingChamber_SHT85_0_Temperature"
    //   point_kind: "sensor_data"
    //   data_point_uuid_ref: "f47ac10b-58cc-4372-a567-0e02b2c3d479"

    // Example for C1 (if this autogen_config.h was for C1):
    // #define POINT_NAME_C1_FRUITINGCHAMBER_SHT85_0_TEMPERATURE "C1_FruitingCham
ber_SHT85_0_Temperature"
    // #define UUID_C1_FRUITINGCHAMBER_SHT85_0_TEMPERATURE_DATA "f47ac10b-58cc-43
72-a567-0e02b2c3d479"
    // #define TOPIC_C1_FRUITINGCHAMBER_SHT85_0_TEMPERATURE_DATA "mush/c1/sensor
s/sht85_0_fc/degf" // "degf" from slugified units
    ```
```

This version (2.5) of ADR-20 refines how command topic targets are derived, incorporates the "Manual" component concept for better consistency, and clarifies sensor topic generation using slugified units. It restores original phrasing from v2.3 where appropriate while integrating discussed updates.
```