# SSOT Definition and Configuration Refactor Plan

**Version:** 2.1 (Referenced "Topic Naming Proposal" for examples)
**Date:** 2025-05-22

**Overall Goal:** To transition the entire Mushroom Automation System V2 to a validated Single Source of Truth (SSOT) centered around `system_definition.yaml`. This involves finalizing Pydantic schemas, migrating existing configurations, enhancing `build.py` for comprehensive validation and artifact generation (including `autogen_config.h` for microcontrollers and a `global_point_registry.json`), and ensuring system-wide consistency. This plan directly supports **Phase 1** of the "Mushroom Automation V2 - Master Project Plan & Roadmap".

**Status Legend:**

- ☐ `[ ]` To Do
- `[/]` In Progress
- `[X]` Done
- `[B]` Blocked

## Phase 1 (of this plan): SSOT Core Definitions & Pydantic Schema Finalization

**Goal:** Establish the definitive structure for all configuration YAMLs and the Pydantic models that validate them, fully compliant with ADR-20.

- **Task 1.1: Finalize `system_definition.yaml` Structure and Content Rules (Ref: ADR-20 Sec.II.A, Master Plan Task 1.1)**
  - ☐ `[ ]` Define `global_settings` structure within `system_definition.yaml` (e.g., `mqtt_topic_prefix`, default `mqtt_broker`, default `ntp_server`).
  - ☐ `[ ]` Finalize structure for the `components:` list. For each component:
    - `id`: Unique string identifier (becomes `source_component_id`).
    - `name`: Human-readable name.
    - `component_type`: e.g., "microcontroller", "driver", "governor", "manual".
    - `config_file`: Path to its component-specific YAML configuration (optional for types like "manual").
    - `points_provided` (for microcontrollers) / `virtual_points_provided` (for drivers/governors/manual): List of UUIDs (from the master `points:` list) that this component sources/manages, as per ADR-20.
  - ☐ `[ ]` Finalize structure for the master `points:` list. For each point, ensure all attributes from **ADR-20 Section II.A** are covered:
    - `uuid`, `name` (system-wide functional), `description`, `value_type`, `units`.
    - `data_source_layer` (required), `access`, `writable_by` (for

documentation/authorization as per ADR-20).

- `persist_to_db` (required, with a system default if appropriate).
- `validation_rules`, `initial_value` (logical).
- `readback_point_uuid` (for command/write points).
- **New ADR-20 Topic Fields:** `function_grouping` (required: "actuators", "sensors", "statuses", "commands").
- **Conditional ADR-20 Topic Fields (based on `function_grouping`):**
  - If "actuators": `topic_device_slug`.
  - If "sensors": `topic_originator_slug` (metric part from slugified `units`).
  - If "commands": `topic_directive_slug` (target component derived by `build.py`).
  - If "statuses": `topic_status_slug`.
- ***Note:*** For illustrative examples of how these fields are used to construct MQTT topics for different point types (actuators, sensors, commands, statuses), refer to the uploaded document: **Topic Naming Proposal + config file fields** Remember that ADR-20 specifies the value goes in the payload (ADR-10), not the topic string itself.

☐ `[ ]` Document conventions for choosing system-wide functional `name`s (Principle #6 in ADR-20).

- **Task 1.2: Finalize Pydantic Models in `core_ssot_models.py` for `system_definition.yaml` (Ref: Master Plan Task 1.1.1 - 1.1.3)**

  ☐ `[ ]` Implement/Refine Pydantic models: `SystemDefinition`, `MQTTBrokerConfig` (for global settings), `ComponentDefinition` (with variants `MicrocontrollerComponentDefinition`, `DriverComponentDefinition`, `GovernorComponentDefinition`, `ManualSourceComponentDefinition`), `PointDefinition`.

  ☐ `[ ]` Ensure `PointDefinition` model includes all attributes from Task 1.1 and **ADR-20 Section II.A**, including all new topic-related fields. Enforce `data_source_layer`, `persist_to_db`, and `function_grouping` as required.

- **Task 1.3: Finalize Pydantic Models in `component_configs.py` for all Component Types (Ref: ADR-20 Sec.II.B, Master Plan Task 1.1.4 - 1.1.6)**

  ☐ `[ ]` **`MicrocontrollerConfig`**:

  - Implement/Refine the `MicrocontrollerConfig` Pydantic model as specified in **ADR-20, Section II.B.1**.
  - Focus: `name` (local functional name), `point_kind` (must align with `function_grouping`), `pin` details, `initial_state` (hardware), `write_point_uuid_ref`, `readback_point_uuid_ref`, `data_point_uuid_ref`.
  - **Crucially: Ensure NO `mqtt_topic_suffix_*` fields are present.**
  - Include device-specific settings like `i2c`, `onewire`, `dht_sensors`, `digital_outputs`, `publish_frequency_ms`, etc.

  ☐ `[ ]` **`DriverConfig`**:

- Implement/Refine the `DriverConfig` Pydantic model as detailed in **ADR-20, Section II.B.2**.
- Focus: FSM structures ( `initial_state` , `states` , `transitions` ), `pwm_outputs` . Input/output point UUIDs are referenced within FSM definitions.
- **Minimize/remove explicit `mqtt_topic_suffix_*` fields; topics are derived by `build.py` .**

☐ `[ ]` `GovernorConfig` :

- Implement/Refine the `GovernorConfig` Pydantic model as detailed in **ADR-20, Section II.B.3**.
- Focus: `update_interval_seconds` , `controllers` (list of PID, BangBang, TimeSchedule configs). Input/output point UUIDs are referenced within controller definitions.
- **Minimize/remove explicit `mqtt_topic_suffix_*` fields; topics are derived by `build.py` .**

☐ `[ ]` Ensure local `name` fields in component configs follow derivation conventions (Principle #7 in ADR-20).

- **Task 1.4: Create/Update Sample YAML files (Ref: Master Plan Task 1.1.7)**

☐ `[ ]` Create comprehensive, valid sample YAML files for `system_definition.yaml` and each Component Type ( `MicrocontrollerConfig` , `DriverConfig` , `GovernorConfig` ) that exercise all Pydantic model features and demonstrate the intended configuration structure, consistent with **ADR-20** examples and requirements.

## Phase 2 (of this plan): Configuration Data Migration to New SSOT YAMLs

**Goal:** Translate all existing configuration data into the new, validated SSOT YAML structure, compliant with ADR-20. (Corresponds to Master Plan Task 1.3)

- **Task 2.1: Analyze Old Configs & Define Migration Mapping Strategy (Ref: Master Plan Task 1.3.1)**

☐ `[ ]` Identify all old configuration sources (e.g., `microc_points.json` , `uuid_db.json` , driver `settings.json` , `transitions.json` , `states.json` , hardcoded values).

☐ `[ ]` Document old structures and data meanings.

☐ `[ ]` Finalize UUID migration strategy (generate new UUIDv4s, create old_id -> new_uuid mapping if needed for historical data correlation).

☐ `[ ]` Create a detailed mapping document: old fields/concepts -> new Pydantic schemas ( `PointDefinition` , `MicrocontrollerConfig` , `DriverConfig` , etc.), ensuring alignment with **ADR-20** field definitions (e.g., for `data_source_layer` , `access` , `writable_by` , `units` , `function_grouping` , topic slugs).

- **Task 2.2: Implement Point Migration to `system_definition.yaml` (Ref: Master Plan Task 1.3.2)**

☐ `[ ]` Process old point data based on mapping from Task 2.1.

☐ `[ ]` For each point:

- Assign/migrate UUID.
- Define system-wide functional `name` according to ADR-20, Principle #6.
- Populate all attributes as per `PointDefinition` model (**ADR-20 Section II.A**), including new topic fields. Ensure `units` are standardized.
- Correctly link command and readback points using `readback_point_uuid`.

☐ `[ ]` Populate the `points:` list in the target `system_definition.yaml`.

☐ `[ ]` Validate frequently using `build.py` (Stage 1 validation).

- **Task 2.3: Implement Component Migration to `system_definition.yaml` (Ref: Master Plan Task 1.3.3)**

☐ `[ ]` Identify all distinct component instances.

☐ `[ ]` Assign unique `id` to each (e.g., "c1", "temp_driver_fruiting").

☐ `[ ]` Specify `component_type`, human-readable `name`, and correct `config_file` path.

☐ `[ ]` Populate `points_provided` / `virtual_points_provided` for each component using the new UUIDs from Task 2.2, adhering to ADR-20.

☐ `[ ]` Populate the `components:` list in `system_definition.yaml`.

☐ `[ ]` Validate frequently using `build.py` (Stage 1 validation).

- **Task 2.4: Migrate Microcontroller Hardware Configs to Component YAMLs (Ref: Master Plan Task 1.3.4, ADR-20 Sec.II.B.1)**

☐ `[ ]` For each microcontroller instance defined in Task 2.3:

- Create its specific YAML file (e.g., `control/config/microcontrollers/c1_config.yaml`).
- Populate with `device_id` (matching component `id`), wifi, optional MQTT/NTP overrides, timing_constants.
- Populate `hardware_points` list according to the `MicrocontrollerConfig` Pydantic model (**ADR-20, Section II.B.1**):
  - Use local `name` (derived: component `id` + functional part of master name).
  - Set `point_kind`.
  - Include `write_point_uuid_ref`, `readback_point_uuid_ref`, or `data_point_uuid_ref` linking to UUIDs from `system_definition.yaml`.
  - Define `pin`, `pin_mode`, hardware `initial_state` (e.g., LOW/HIGH).
  - **Ensure no `mqtt_topic_suffix_*` fields.**

☐ `[ ]` Validate each file using `build.py` (Stage 2 validation).

- **Task 2.5: Migrate Driver FSM Configs to Component YAMLs (Ref: Master Plan Task 1.3.5, ADR-20 Sec.II.B.2)**

☐ `[ ]` For each driver instance:

- Translate old FSM logic into the new `DriverConfig` YAML format (`states`, `transitions`, `defining_conditions`, `actions` targeting UUIDs).

    - Define input/output point UUID references within FSM structures as per **ADR-20, Section II.B.2**.
    - **Ensure minimal/no explicit `mqtt_topic_suffix_*` fields.**
  - ☐ `[ ]` Validate using `build.py` (Stage 2 validation).
- **Task 2.6: Migrate Governor Logic Configs to Component YAMLs (Ref: Master Plan Task 1.3.6, ADR-20 Sec.II.B.3)**
  - ☐ `[ ]` For each governor instance:
    - Translate old logic parameters (PID constants, setpoint sources, output targets as UUIDs) into the new `GovernorConfig` YAML format.
    - Define input/output point UUID references as per **ADR-20, Section II.B.3**.
    - **Ensure minimal/no explicit `mqtt_topic_suffix_*` fields.**
  - ☐ `[ ]` Validate using `build.py` (Stage 2 validation).

## Phase 3 (of this plan): `build.py` Implementation & Enhancement

**Goal:** Ensure `build.py` can perform comprehensive validation and generate all necessary runtime artifacts from the SSOT, strictly following ADR-20. (Corresponds to Master Plan Task 1.2)

- **Task 3.1: Implement/Enhance `build.py` Stage 1: `system_definition.yaml` Structural Validation (Ref: Master Plan Task 1.2.1)**
  - `[X]` *Status from Master Plan:* Implemented.
  - ☐ `[ ]` Verify `build.py` correctly loads and validates `system_definition.yaml` against Pydantic models from `core_ssot_models.py` (as updated in Task 1.2 of this plan).
- **Task 3.2: Implement/Enhance `build.py` Stage 2: Component-Specific YAML Validation (Ref: Master Plan Task 1.2.2)**
  - `[X]` *Status from Master Plan:* Implemented.
  - ☐ `[ ]` Verify `build.py` iterates through components in `system_definition.yaml`, loads their `config_file`, and validates it against the correct Pydantic model from `component_configs.py` (as updated in Task 1.3 of this plan), using `COMPONENT_MODEL_MAP`.
- **Task 3.3: Implement/Enhance `build.py` Stage 3: Cross-Validation Logic (Ref: Master Plan Task 1.2.3, ADR-20)**
  - `[/]` *Status from Master Plan:* Partially implemented.
  - ☐ `[ ]` Validate UUID uniqueness across all points in `system_definition.yaml`.
  - ☐ `[ ]` Validate that all UUIDs referenced in `points_provided`, `virtual_points_provided`, FSM actions, FSM conditions, controller configs (input/output points), `write_point_uuid_ref`, `readback_point_uuid_ref`, `data_point_uuid_ref`, and `readback_point_uuid` (on command points) exist in the master `points:` list.
  - ☐ `[ ]` Validate `writable_by` entries against existing component `id`s and `command_hierarchy` levels (as per ADR-20, this is for documentation/authorization).

- [ ] `[ ]` Validate `controls_microcontroller` / `controls_drivers` component ID references and types.
- [ ] `[ ]` Validate consistency between `PointDefinition.data_source_layer` and the `type` of the component providing the point.
- [ ] `[ ]` Check for orphaned points (defined in master list but not provided by any component) and multiply-provided points.
- [ ] `[ ]` Implement checks for consistency between logical `initial_value` (from `system_definition.yaml`) and hardware `initial_state` (from `MicrocontrollerConfig`) for actuators.
- [ ] `[ ]` Implement other internal consistency checks (e.g., state names in driver FSM transitions).

- **Task 3.4: Implement `build.py` Logic for `autogen_config.h` Generation for Microcontrollers (Ref: Master Plan Task 1.2.5, ADR-20 Sec.II.C)**
  - [ ] `[ ]` For each microcontroller component:
    - Merge global (`system_definition.yaml`) and local (`MicrocontrollerConfig`) settings for WiFi, MQTT, NTP.
    - Iterate through `hardware_points` in `MicrocontrollerConfig`.
    - Use the local `name` to generate C++ macro prefixes.
    - Look up master point details from `system_definition.yaml` using `uuid_ref` fields.
    - **Construct full MQTT topics strictly according to ADR-20, Section II.C**, using `GLOBAL_MQTT_PREFIX`, `source_component_id` (of publishing component, which is the uC for its own data/readbacks, or the controlling component for commands it subscribes to), and the explicit topic fields from `PointDefinition` (`function_grouping`, `topic_device_slug`, `topic_originator_slug`, slugified `units`, `topic_directive_slug`, derived target for commands).
    - Generate all `#define` macros as specified in **ADR-20, Section II.C ("Contents of autogen_config.h")** and its examples.
  - [ ] `[ ]` Ensure output matches the target `autogen_config.h` structure.
- **Task 3.5: Implement `build.py` Logic for `global_point_registry.json` Generation (Ref: Master Plan Task 1.2.4, ADR-20 Sec.II.C)**
  - [ ] `[ ]` Create a JSON object keyed by point `uuid`.
  - [ ] `[ ]` For each point from `system_definition.yaml`'s master `points:` list:
    - Determine its `source_component_id` by checking `points_provided` / `virtual_points_provided` in `system_definition.yaml`'s `components` section.
    - Include all master attributes from `PointDefinition` (name, description, value_type, units, ADR-20 topic fields, etc.).
    - Add the **fully derived MQTT topic(s)** as constructed by `build.py` per **ADR-20, Section II.C**.

- Add implementation details from component-specific YAML where relevant (e.g., `pin` for a microcontroller point).

- [ ] `[ ]` Output the `global_point_registry.json` file.

- **Task 3.6: Implement `build.py` Logic for Other Runtime Artifact Generation (Optional)**
  - [ ] `[ ]` Consider generation of Telegraf input snippets based on `global_point_registry.json`.
  - [ ] `[ ]` Consider generation of ENV files for Dockerized Python services.

## Phase 4 (of this plan): Final Validation, System Integration & Documentation

**Goal:** Ensure the entire SSOT system (configs and `build.py` outputs) is functional, integrated, and well-documented before proceeding to runtime refactoring.

- **Task 4.1: Full Migrated Config Validation with `build.py` (Ref: Master Plan Task 1.3.7)**
  - [ ] `[ ]` Run `build.py` against the complete set of migrated YAML configuration files.
  - [ ] `[ ]` Resolve all validation errors (structural, cross-referential).
  - [ ] `[ ]` Manually review generated artifacts ( `autogen_config.h` files, `global_point_registry.json`) for logical correctness and completeness against ADR-20.
- **Task 4.2: Integrate `build.py` into Development Workflow**
  - [ ] `[ ]` Ensure `build.py` is run automatically (e.g., pre-commit hook, CI pipeline) after any changes to YAML configuration files.
  - [ ] `[ ]` Failed validation should block commits/merges.
- **Task 4.3: Update/Create System Documentation for SSOT (Corresponds to parts of Master Plan Task 5.5)**
  - [ ] `[ ]` Finalize **ADR-20: Naming, UUIDs, SSOT, and Information Locus for System Points**.
  - [ ] `[ ]` Document the structure of `system_definition.yaml` and all component-specific YAML configuration files, referencing ADR-20.
  - [ ] `[ ]` Document all Pydantic models in `core_ssot_models.py` and `component_configs.py`.
  - [ ] `[ ]` Document the usage and stages of `build.py`.
  - [ ] `[ ]` Document the structure and purpose of `global_point_registry.json` and `autogen_config.h`.
  - [ ] `[ ]` Document the configuration migration process and any mappings used.
  - [ ] `[ ]` Update `V2 Architecture and Overview.pdf` to reflect the finalized SSOT data flow.