Plan for Pydantic Models and autogen_config.h Structure

Original Document Source: "Plan for Pydantic Models and autogen_config.h Structure.pdf" (v1.3, 2025-05-20)
This Version: 1.4 (Removed architectural layers section; scope clarified to point to ADR-20 v2.5 for layer/point type definitions) **Date:** 2025-05-22
Time: 2:20 PM PDT

Goal: To define the Pydantic models necessary for validating microcontroller configurations (e.g., `c1_config.yaml`), focusing on global setting overrides and hardware point definitions (excluding MQTT topic specifics). This also specifies the structure of the C++ `autogen_config.h` header files for these settings, as generated by `build.py`. This document serves as a detailed reference for tasks in `ssot_detailed_plan_v2_0` (Task 1.3 for `MicrocontrollerConfig` Pydantic models, Task 3.4 for `autogen_config.h` generation) and `microcontroller_detailed_plan_v2_0`.

Important Note on Scope:

- This document focuses on the Pydantic models for microcontroller-specific configurations (`MicrocontrollerConfig`) and the parts of `autogen_config.h` related to device ID, WiFi, MQTT/NTP overrides, timing constants, and hardware pin/UUID mapping.
- **For definitive information on Architectural Layers and Point Types by Layer, please refer to ADR-20 v2.5, Section I.**
- System-wide attributes for all points (including `uuid`, master `name`, and **all topic generation fields like `function_grouping` and `topic_*_slug` fields**) are defined in `system_definition.yaml` and validated by Pydantic models in `core_ssot_models.py`, as detailed in **ADR-20 v2.5**.
- **All MQTT topics (including those in `autogen_config.h`) are fully derived by `build.py` as per ADR-20 v2.5, Section II.C, based on fields in `system_definition.yaml`. This document does NOT define how topics are constructed.**

1. Pydantic Models for `MicrocontrollerConfig` (to be part of `component_configs.py`)

These Python Pydantic models are used by `build.py` to validate the YAML configuration files for each microcontroller (e.g., `c1_config.yaml`).

```
from pydantic import BaseModel, Field, validator
from typing import List, Optional, Literal, Union, Dict, Any
# Assuming PointUUID is a str-like type defined in core_ssot_models.py

# Basic Configuration Sub-Models (Shared within MicrocontrollerConfig)
class WiFiConfig(BaseModel):
    ssid: str
    password: str

class MQTTBrokerConfigOptional(BaseModel):
    """Defines MQTT Broker fields that can optionally override global settings from system_definition.yaml."""
    address: Optional[str] = None
    port: Optional[int] = None
    username: Optional[str] = None
    password: Optional[str] = None

class NTPServerConfigOptional(BaseModel):
    """Defines NTP Server fields that can optionally override global settings from system_definition.yaml."""
    address: Optional[str] = None
    utc_offset_seconds: Optional[int] = None
    update_interval_ms: Optional[int] = None

# Hardware Point Base and Specialized Sub-Models for MicrocontrollerConfig.hardware_points
# These define implementation details. MQTT topic suffixes are REMOVED (obsolete per ADR-20 v2.5).

class HardwarePointBase_MicrocontrollerImpl(BaseModel):
    name: str = Field(..., description="Local descriptive name for this hardware implementation on the microcontroller (e.g., 'C1_AmbientTemp', 'C2_HeatingElement'). Used to generate C++ define prefixes. Should be systematically derived from the master point name in system_definition.yaml and the device_id.")
    point_kind: str = Field(..., description="Discriminator field: 'actuator', 'sensor_data', 'system_info'. Must align with 'function_grouping' in the master PointDefinition (ADR-20 v2.5).")
    description: Optional[str] = Field(None, description="Optional local description for this implementation detail.")
    attributes: Optional[Dict[str, Any]] = Field(None, description="Key-value pairs for additional type-specific configuration for this microcontroller implementation.")

class ActuatorHardwarePoint_MicrocontrollerImpl(HardwarePointBase_MicrocontrollerImpl):
    point_kind: Literal["actuator"] = "actuator"
    write_point_uuid_ref: str # UUID of the master command point definition in system_definition.yaml.
    readback_point_uuid_ref: str # UUID of the master readback point definition in system_definition.yaml.
    pin: Union[int, str] = Field(..., description="Microcontroller pin for the actuator.")
    pin_mode: Literal["OUTPUT"] = "OUTPUT"
    pin_mode: Literal["OUTPUT"] = "OUTPUT"
    initial_state: Literal["LOW", "HIGH", "0"] = Field(..., description="Hardware initial state for the actuator ('LOW', 'HIGH', or '0' for PWM 0% duty).")

class SensorDataHardwarePoint_MicrocontrollerImpl(HardwarePointBase_MicrocontrollerImpl):
    point_kind: Literal["sensor_data"] = "sensor_data"
    data_point_uuid_ref: str # UUID of the master data point definition in system_definition.yaml.
    pin: Optional[Union[int, str]] = Field(None, description="Microcontroller pin if directly pin-connected.")
    pin_mode: Optional[Literal["INPUT", "INPUT_PULLUP"]] = Field(None, description="Required if 'pin' is specified.")

    @validator('pin_mode', always=True, pre=False) # pre=False to ensure 'values' is populated
    def check_pin_mode_for_sensor(cls, v, *, values, **kwargs): # Adjusted for Pydantic v2
        pin = values.get('pin')
        if pin is not None and v is None:
            raise ValueError("pin_mode ('INPUT' or 'INPUT_PULLUP') is required if 'pin' is specified for a SensorDataHardwarePoint.")
        if pin is None and v is not None:
            raise ValueError("pin_mode should not be set if 'pin' is not specified (e.g., for I2C sensors).")
        return v

class SystemInfoHardwarePoint_MicrocontrollerImpl(HardwarePointBase_MicrocontrollerImpl):
    point_kind: Literal["system_info"] = "system_info"
    data_point_uuid_ref: str # UUID of the master data point definition in system_definition.yaml.

# Timing Configuration
class TimingConstants(BaseModel):
    wifi_connect_timeout_ms: int = 30000
    mqtt_connect_timeout_ms: int = 20000
    ntp_sync_timeout_ms: int = 15000
    max_time_no_publish_ms: Optional[int] = Field(None, description="Max time in ms a monitored point can go without publishing before a restart is triggered.")
    publish_interval_ms: Optional[int] = Field(None, description="Default interval in ms for publishing data (e.g., sensor readings by the FSM).")

# Top-Level Microcontroller Configuration Model
class MicrocontrollerConfig(BaseModel):
    """Pydantic model for validating a controller-specific YAML configuration file (e.g., c1_config.yaml)."""
    device_id: str = Field(..., description="Unique device ID for this microcontroller. Must match the 'id' in system_definition.yaml.")
    description: str = Field(..., description="Human-readable description of the microcontroller's role.")
    wifi: WiFiConfig
    mqtt_broker: Optional[MQTTBrokerConfigOptional] = Field(None, description="Optional MQTT broker override. If None or fields are None, global settings from system_definition.yaml are used.")
    ntp_server: Optional[NTPServerConfigOptional] = Field(None, description="Optional NTP server override. If None or fields are None, global settings from system_definition.yaml are used.")
    hardware_points: List[Union[ActuatorHardwarePoint_MicrocontrollerImpl, SensorDataHardwarePoint_MicrocontrollerImpl, SystemInfoHardwarePoint_MicrocontrollerImpl]]
    timing_constants: TimingConstants
    # Other device-specific configs like i2c, onewire can be added here if they were in the original PDF and are still relevant.
    # Example (ensure these sub-models like I2CConfig, I2CDevice are also defined if used):
    # i2c_config: Optional[I2CConfig] = None
    # i2c_devices: Optional[List[I2CDeviceConfig]] = None
```

2. `autogen_config.h` Structure (Per Microcontroller)

This C++ header file will be generated by `build.py` for each microcontroller. `build.py` will iterate through the `hardware_points` list in the validated `MicrocontrollerConfig` Pydantic object. For each point, it will use the local `name` field (e.g., "C2_HeatingElement") to generate unique C++ define prefixes.

All `TOPIC_...` macros will contain full MQTT topic strings derived by `build.py` as per ADR-20 v2.5, Section II.C. The examples below for `TOPIC_...` macros are illustrative of the *kind* of topics, but their exact strings are determined by `build.py` and ADR-20 v2.5.

Example for Controller C2 (`microcontroller/controller2/autogen_config.h`):

```
#ifndef AUTOGEN_CONFIG_H_C2 // Unique header guard per controller
#define AUTOGEN_CONFIG_H_C2

// --- General Device Configuration ---
#define DEVICE_ID "c2" // From MicrocontrollerConfig.device_id
#define DEVICE_DESCRIPTION "Controller for Actuators (Heater, Fan, etc.)" // From MicrocontrollerConfig.description

// --- WiFi Configuration ---
#define WIFI_SSID "MyWiFiNetworkSSID" // From MicrocontrollerConfig.wifi.ssid
#define WIFI_PASSWORD "MyWiFiPassword" // From MicrocontrollerConfig.wifi.password

// --- MQTT Broker Configuration ---
// These values will be derived by build.py from system_definition.yaml global_settings.mqtt_broker
// or overridden by MicrocontrollerConfig.mqtt_broker if specified there.
#define MQTT_BROKER_ADDRESS "192.168.1.100"
#define MQTT_BROKER_PORT 1883
#define MQTT_USER "mqtt_username"      // or "" if not defined
#define MQTT_PASSWORD "mqtt_password"  // or "" if not defined

// --- NTP Server Configuration ---
// Similar to MQTT, derived by build.py from system_definition.yaml global_settings.ntp_server
// or overridden by MicrocontrollerConfig.ntp_server.
#define NTP_SERVER "pool.ntp.org"
#define NTP_UTC_OFFSET_SECONDS 0
#define NTP_UPDATE_INTERVAL_MS 3600000 // 1 hour

// --- Hardware Point Definitions (from c2_config.yaml hardware_points list) ---

// Example based on MicrocontrollerConfig entry:
// - name: "C2_HeatingElement"
//   point_kind: "actuator"
//   write_point_uuid_ref: "uuid-for-heatingpad-command"
//   readback_point_uuid_ref: "uuid-for-heatingpad-readback"
//   pin: 12
//   initial_state: "LOW"

#define POINT_NAME_C2_HEATINGELEMENT "C2_HeatingElement" // From MicrocontrollerConfig.hardware_points[n].name
#define UUID_C2_HEATINGELEMENT_WRITE "uuid-for-heatingpad-command" // From .write_point_uuid_ref
#define UUID_C2_HEATINGELEMENT_READBACK "uuid-for-heatingpad-readback" // From .readback_point_uuid_ref
#define PIN_C2_HEATINGELEMENT 12 // From .pin
#define MODE_C2_HEATINGELEMENT OUTPUT // Implied by point_kind: "actuator" or explicit if model allows
#define INITIAL_STATE_C2_HEATINGELEMENT LOW // From .initial_state

// Full MQTT topics derived by build.py as per ADR-20 v2.5, Section II.C
// Example: Master command point for heating pad (UUID "uuid-for-heatingpad-comma
// Example: Master command point for heating pad (UUID "uuid-for-heatingpad-command")
// in system_definition.yaml has topic_directive_slug: "fruiting_chamber_heating_pad".
// build.py derives target "c2" and appends "/write" (or similar based on convention).
#define TOPIC_C2_HEATINGELEMENT_WRITE "mush/temperature_driver_fruiting/commands/c2/fruiting_chamber_heating_pad/write"

// Example: Master readback point for heating pad (UUID "uuid-for-heatingpad-readback")
// in system_definition.yaml has topic_device_slug: "fruiting_chamber_heating_pad".
#define TOPIC_C2_HEATINGELEMENT_READBACK "mush/c2/actuators/fruiting_chamber_heating_pad/readback"

// Example for a system info point:
// - name: "C2_SystemUptime"
//   point_kind: "system_info"
//   data_point_uuid_ref: "uuid-for-c2-uptime"

#define POINT_NAME_C2_SYSTEMUPTIME "C2_SystemUptime"
#define UUID_C2_SYSTEMUPTIME_DATA "uuid-for-c2-uptime"
// No PIN, MODE, INITIAL_STATE_ defines for system_info types
// Full MQTT topic derived by build.py as per ADR-20 v2.5, Section II.C
// Example: Master status point for uptime (UUID "uuid-for-c2-uptime")
// in system_definition.yaml has topic_status_slug: "uptime".
#define TOPIC_C2_SYSTEMUPTIME_DATA "mush/c2/statuses/uptime"

// --- Timing Constants ---
#define WIFI_CONNECT_TIMEOUT_MS 30000   // From MicrocontrollerConfig.timing_constants
#define MQTT_CONNECT_TIMEOUT_MS 20000
#define NTP_SYNC_TIMEOUT_MS 15000
// MAX_TIME_NO_PUBLISH_MS and PUBLISH_INTERVAL_MS would be #ifdef protected if optional
// #ifdef MAX_TIME_NO_PUBLISH_MS_C2_CONFIG
// #define MAX_TIME_NO_PUBLISH_MS MAX_TIME_NO_PUBLISH_MS_C2_CONFIG
// #endif
// #ifdef PUBLISH_INTERVAL_MS_C2_CONFIG
// #define PUBLISH_INTERVAL_MS PUBLISH_INTERVAL_MS_C2_CONFIG
// #endif

#endif // AUTOGEN_CONFIG_H_C2
```

3. Global Point Registry / UUID-Topic Mapping

(This section is largely superseded by ADR-20 v2.5, Section II.C, which details how `build.py` constructs topics and populates `global_point_registry.json`. The original PDF's Section 3 details on topic suffixes are obsolete.)

The `global_point_registry.json` is generated by `build.py`. It is keyed by UUID and contains master point attributes from `system_definition.yaml` and the fully derived MQTT topics as per **ADR-20 v2.5**.

4. Highlighted Areas for Discussion / Uncertainty (from original PDF v1.3, status updated for ADR-20 alignment)

1. **Pydantic `HardwarePoint.pin_mode` & C++ `MODE_...` Defines:**
    - Status: `build.py` will pass the string through (e.g., `#define MODE_MYPOINT OUTPUT`). This is consistent with current plans.
2. **Pydantic `HardwarePoint.initial_state` & C++ `INITIAL_STATE_...` Defines:**
    - Status: `build.py` translates appropriately. `build.py` should validate consistency with the logical `initial_value` from `system_definition.yaml`. This is consistent.
3. **Global vs. Local Timing Constants:**
    - Status: `TimingConstants` within `MicrocontrollerConfig` allow local overrides. `system_definition.yaml` can define global defaults for `build.py` to merge. This is consistent.
4. **Handling Optional Defines in `autogen_config.h`:**
    - Status: For optional values in `MicrocontrollerConfig` (e.g., `max_time_no_publish_ms`), if not provided, `build.py` can omit the `#define` (requiring `#ifdef` in C++) or define with a sentinel. MQTT Topic defines are always generated if the point aspect exists, as topics are now fully derived by `build.py` from `system_definition.yaml` point attributes. The original PDF's concern about optional topic suffixes is obsolete.
5. **Pin Mapping (`HardwarePoint.pin` to C++ integer):**
    - Status: `build.py` is responsible. Integer GPIOs preferred in YAML. String alias resolution is a potential future enhancement for `build.py`.
6. **`HardwarePointBase_MicrocontrollerImpl.attributes` field:**
    - Status: Remains a flexible `Dict[str, Any]` for extensibility. Consistent.