EY Prop Base — Architecture Overview

ESP32 Escape Room Prop Controller System

Version 1.0 — January 2026

1. System Overview

This firmware powers ESP32-based props in escape rooms. Each prop monitors sensors (RFID readers, reed switches, buttons, etc.), determines a 'solved' state, and communicates via MQTT with a central room controller (MiniPC).

The architecture follows a strict separation of concerns: the ESP32 handles physical I/O and local solve detection, while the MiniPC owns session logic, analytics, and persistence. The GM dashboard is purely a UI layer.

System Architecture

ESP32 Prop	← **MQTT** →	**MiniPC**	← **WebSocket** →	**GM Dashboard**
(this code)		(room controller)		(React UI)

2. File Structure

File	Purpose
`EY_Types.h`	Shared type definitions (enums, structs)
`EY_Config.h`	Per-prop configuration (identity, sensors, WiFi)
`EY_Sensors.h/cpp`	Generic sensor management system
`EY_Mqtt.h/cpp`	MQTT communication layer
`main.cpp`	Application entry point, LED feedback, reset logic

3. File Details

`EY_Types.h`

Contains shared type definitions used across multiple modules. Has no dependencies on other project headers.

PresentWhen — Enum defining sensor polarity
- HIGH_LEVEL: sensor is "present" when pin reads HIGH
- LOW_LEVEL: sensor is "present" when pin reads LOW

SolveMode — Enum defining solve condition
- ANY: solved when any single sensor triggers

- ALL: solved when all sensors are present simultaneously

SensorDef — Struct for compile-time sensor configuration
- id: unique identifier (e.g., "rfid1")
- pin: GPIO pin number
- presentWhen: polarity rule
- actionEvent: MQTT event string (e.g., "rfid_present")
- needsArming: require "not present" before "present" counts

SensorState — Struct for runtime sensor state
- armed, present, eventSent

`EY_Config.h`

The **only file you edit** when setting up a new prop. Contains all per-prop configuration.

Identity: SITE_ID, ROOM_ID, DEVICE_ID
Network: WIFI_SSID, WIFI_PASS, MQTT_HOST, MQTT_PORT
Hardware: LED_PIN, RESET_BTN_PIN
Sensors: SENSORS[] array, SENSOR_COUNT, SOLVE_MODE
Timing: blink rates, reset hold time, ignore window

`EY_Sensors.h / EY_Sensors.cpp`

Generic sensor management system. Handles any number of sensors defined in EY_Config.h.

EY_Sensors_Begin() — Initialize all sensor pins (INPUT_PULLUP)
EY_Sensors_Tick() — Poll sensors, publish events, return solve state
EY_Sensors_Reset() — Clear all sensor states (armed, present, eventSent)
EY_Sensors_IsSolved() — Check solve condition without side effects

Features:
- Automatic arming logic for sensors that need it
- One-shot event publishing (once per reset)
- Configurable solve mode (ANY / ALL)

`EY_Mqtt.h / EY_Mqtt.cpp`

MQTT communication layer implementing the v1 contract. Handles WiFi/MQTT reconnection non-blocking.

EY_Net_Begin(onReset, onSetSolved) — Start networking with callbacks
EY_Net_Tick() — Call every loop (handles reconnection)
EY_Mqtt_Connected() — Check connection status
EY_PublishEvent(action, source) — Publish transient event (not retained)
EY_PublishStatus(solved, source, override) — Publish retained status

MQTT Topics (contract v1):
- ey/<site>/<room>/prop/<propId>/status (retained)
- ey/<site>/<room>/prop/<propId>/event (not retained)
- ey/<site>/<room>/prop/<propId>/cmd (commands)

Features:

- LWT (Last Will Testament) for online/offline detection
- Legacy command format support for backward compatibility

`main.cpp`

Minimal application code. Glues together sensors, MQTT, and feedback. You should never need to edit this file.

setup()

- Initialize LED and reset button pins
- Call EY_Sensors_Begin()
- Call EY_Net_Begin() with reset/setSolved callbacks

loop()

- EY_Net_Tick() — maintain network connection
- Check BOOT button for long-press reset
- Fast-blink LED during reset feedback window
- EY_Sensors_Tick() — poll sensors (after ignore window)
- Latch solved state, publish status on transition
- Blink LED when solved

4. Setting Up a New Prop

To configure a new prop, you only need to edit **EY_Config.h**. No other files require changes.

Step-by-step:

1. Copy the entire project folder
2. Open EY_Config.h
3. Set DEVICE_ID to a unique name (e.g., "bookshelf_puzzle")
4. Set ROOM_ID if different from default
5. Define your SENSORS[] array with pin numbers and polarities
6. Set SOLVE_MODE to ANY or ALL
7. Flash to ESP32

Example — 3-Magnet Puzzle:

```
static const SensorDef SENSORS[] = {
  { "magnet_left",   25, PresentWhen::LOW_LEVEL, "magnet_left_placed",   false },
  { "magnet_center", 26, PresentWhen::LOW_LEVEL, "magnet_center_placed", false },
  { "magnet_right",  27, PresentWhen::LOW_LEVEL, "magnet_right_placed",  false },
};
static constexpr SolveMode SOLVE_MODE = SolveMode::ALL;
```

5. Quick Reference

Sensor Polarity

Sensor Type	PresentWhen	Wiring
Reed switch (to GND)	LOW_LEVEL	Switch between pin and GND, INPUT_PULLUP
RFID (high when tag)	HIGH_LEVEL	Module outputs HIGH when tag present
Button (to GND)	LOW_LEVEL	Button between pin and GND, INPUT_PULLUP
IR break-beam	HIGH_LEVEL	Output HIGH when beam broken (typical)

MQTT Message Types

Type	Topic Suffix	Retained	Purpose
Status	/status	Yes	Current state (solved, online, override)
Event	/event	No	Actions (sensor triggers, force_solved)
Command	/cmd	No	Instructions (reset, force_solved)

Source Values

Source	Meaning
player	Physical interaction by player
gm	Game master intervention
device	Automatic/system action