

# Makeline System

Hot-reloadable configuration for food assembly hardware

# Quick Start

```
just generate-makeline simulation  
just simulate simulation
```

```
# Edit profiles/simulation.json  
# Ctrl+S → reloads in 2.5s
```

Aliases: `gm` / `s`

**Always use** `just generate-makeline` (not `just generate`)

# Where Files Live

```
generated/profiles/simulation/  
├─ simulation.json          # Don't edit (gets regenerated)  
├─ spawner.json            # Process launch config  
└─ custom_layers.json      # Your customizations  
  
profiles/                  # Edit here  
├─ simulation.json         # Active profile (watched)  
└─ simulation/  
    ├─ makeline.json       # Expanded result  
    └─ backups/            # Timestamped copies
```

**Edit** `profiles/` **for hot-reload** | `spawner.json` launches processes

# Profile Structure

Three sections define your makeline:

**layouts:** CabinetKind list (hardware topology)

- ```
{ "default": { "cabinets": ["Initial", "Denest", "Dispense",  
"Lift"] }}
```

**layer\_groups:** Named edit collections (modifications)

- ```
{ "base": [layer1, layer2], "prod": [layer3] }
```
- Layers applied sequentially (order matters!)

**line\_builds:** Combine layout + layer groups (final config)

# Editing Workflow

Edit `profiles/simulation.json` → Save → 2.5s → Changed modules  
reload

**What happens:** Diff → backup to `backups/{timestamp}/` → reload  
affected modules

**What keeps running:** Planner, unchanged modules, active orders

# Common Tasks

## Change dispenser ingredient:

```
{ "EditSectionField": {  
  "identity": { "owner": "dispenser-3", "subject": "self" },  
  "section_name": "inputs",  
  "field_key": "assigned_ingredient_id",  
  "field_value": "black_beans"  
}}
```

## Adjust buffer motion timeout:

```
{ "EditSectionField": {  
  "identity": { "owner": "buffer-1", "subject": "self" },  
  "section_name": "configuration",  
  "field_key": "motion_timeout",  
  "field_value": "10"
```

# Layer Edit Types - Part 1

## EditSectionField: Change single config field (most common)

```
{ "EditSectionField": {
  "identity": { "owner": "buffer-1", "subject": "self" },
  "section_name": "configuration",
  "field_key": "motion_timeout_ms",
  "field_value": 20000
}}
```

## AssignSections: Replace entire sections (multiple related fields)

```
{ "AssignSections": {  
  "identity": { "owner": "lifecycler-1", "subject": "self" },  
  "sections": {  
    "configuration": { "cooldown complete ms": 24000, "timeout fault ms": 600000 }.
```

# Layer Edit Types - Part 2

**AssignDispensers:** Populate all dispensers for cabinets

```
{ "AssignDispensers": {  
  "dispensers": [  
    { "cabinet_index": 2, "dispenser_index": 0, "ingredient_id": "black_beans",  
      "position": { "x": 0.0, "y": 0.0, "z": 0.0 }, "kind": "Auger" },  
    { "cabinet_index": 2, "dispenser_index": 1, "ingredient_id": "rice",  
      "position": { "x": 0.1, "y": 0.0, "z": 0.0 }, "kind": "Auger" }  
  ]  
}}
```

**OmitModules:** Remove modules from graph (testing/debugging)

```
{ "OmitModules": { "identities": [  
  { "owner": "hvac-1", "subject": "self" },  
  ...  
]}
```



# Module Sections Reference

Different modules have different section names. Common sections:

**Buffer modules:** `configuration`

- Fields: `motion_timeout_ms`, `homing_velocity`, `home_to_lower_mrad`, etc.

**Dispenser modules:** `inputs`, `outputs`

- `inputs`: `assigned_ingredient_id`, `dispenser_kind`
- `outputs`: Runtime state (read-only)

**Lifecycler module:** `configuration`, `light`

# Custom Layers

## Dev tweaks without modifying preset profiles

just generate-makeline-custom simulation → creates  
custom\_layers.json

```
[{
  "metadata": { "name": "Faster Buffer Motion" },
  "edits": [{
    "EditSectionField": {
      "identity": { "owner": "buffer-1", "subject": "self" },
      "section_name": "configuration",
      "field_key": "motion_timeout_ms",
      "field_value": 10000
    }
  ]
}]
```

# Line Builds & Switching

**One profile, multiple configs** via different layer combinations

```
"line_builds": {  
  "production": { "layer_groups": ["base", "prod_ingredients"] },  
  "testing": { "layer_groups": ["base", "test_ingredients"] }  
}
```

**At generation:** just generate-makeline simulation testing

**At runtime (Explorer):**

- AvailableLineBuilds → see options
- SelectLineBuild { line\_build\_name: "production" } → switch (10- 11

# How Targeting Works

**Identity structure:** `{ owner: "module-name", subject: "target" }`

**owner:** Module instance name (numbered instances from graph expansion)

- `"buffer-1"`, `"buffer-2"` (dispense cabinet buffers)
- `"lifecycler-1"` (system singleton)
- `"lift-1"` (lift cabinet)
- `"dispenser-1"` through `"dispenser-12"` (v5111) or `"dispenser-18"` (v5112)

**subject:** Config target within the module hierarchy

# Generator Expansion Process

**Step 1:** Read `profiles/{preset}.json` → get layouts, layer\_groups, line\_builds

**Step 2:** Select line\_build (from CLI or "default") → determines layout + which layer\_groups to apply

**Step 3:** Expand layout into graph:

```
For each CabinetKind:  
  cabinet_kind.devices() → Vec<DeviceKind>  
  For each DeviceKind:  
    device_kind.modules() → Vec<ModuleKind>  
    Create numbered instances (buffer-1, buffer-2, ...)
```

# Generated Files Explained

**spawner.json:** Process launch configuration

- Executable paths, args, environment variables for each module
- `generate-makeline`: Includes makeline\_server references (enables hot-reload)
- `generate`: Includes machine\_config references only (legacy, no hot-reload)

**makeline.json:** Expanded module graph

- Lives in `profiles/{preset}/makeline.json` (not generated!)
- Contains all module instances with their sections

# System Rules & Presets

**Always use:** `just generate-makeline` (not `just generate`)

- Adds hot-reload support via `makeline_server` in `spawner.json`

**Watch mechanism:** Only `profiles/{preset}.json` triggers 2.5s auto-reload

**Spawner behavior:** Smart restart - only kills/restarts processes with config changes

**Never edit:** `generated/` directory - gets overwritten on every generation

# Architecture Rationale

## Why this design?

**Hardware abstraction:** Physical machines have cabinets → devices → hardware modules. Software mirrors this hierarchy so config matches physical reality.

**Hot-reload requirement:** Must change config without stopping production. Graph-based design allows:

- Diff detection (compare old vs new graph)
- Selective module restart (only changed modules)
- Process isolation (spawner manages independent processes)



# System Architecture

**Three-tier enum hierarchy:** CabinetKind → DeviceKind → ModuleKind

Generator expands each tier: Types define requirements → instances get created

**CabinetKind** enum (4 variants):

- Initial, Denest, Dispense, Lift

**DeviceKind** enum (10 variants):

- Core, CabinetCore, CabinetScreen, Denester, Dispenser

# Cabinet → Device Mappings

**Initial** cabinet (system-wide services):

- `Core` device → 20 modules

**Denest** cabinet (bowl handling):

- `CabinetCore`, `Conveyance`, `Denester`, `CabinetScreen`

**Dispense** cabinet (ingredient dispensing):

- `CabinetCore`, `Hvac`, `DispenseFillPositioner`
- Plus  $N \times$  `Dispenser` devices (12 for v5111, 18 for v5112)

# Device → Module Details

**Core** device (Initial cabinet only) - 20 system modules:

- Api-1, BowlRecovery-1, CabinetMonitor-1, Datalog-1
- Discovery-1, Echo-1, Fault-1, Follower-1
- Interlock-1, LifeCycler-1, MachineConfig-1, Makeline-1
- PartnerApi-1, PartnerWebhook-1, Planner-1, Preprocessor-1
- RfidClient-1, Sequencer-1, State-1, Telemetry-1, Tracker-1

**DispenseFillPositioner** device (one per dispense cabinet):

- Buffer, Conveyance, Shutter, Duc

# Physical Hardware Context

**Real-world machine structure drives software design:**

**Cabinet** = Physical enclosure unit

- Initial: System computer, no food hardware
- Denest: Unstacks bowls from dispenser
- Dispense: Contains ingredient hoppers (12-18 per cabinet)
- Lift: Presents finished bowls to customer

**Device** = Functional hardware subsystem within cabinet

- Core: Software-only (system services)

# Module Communication

**IPC via ZeroMQ:** Each module = separate process on pub/sub network

**Identity for routing:** Messages addressed by `{ owner, subject }`:

- `{ owner: "buffer-1", subject: "self" }` → message to buffer-1 process
- `{ owner: "buffer-1", subject: "motor-1" }` → message to buffer-1's motor child
- Makeline server routes based on Identity

**Message types:**

# Hot-Reload Mechanics

**What triggers reload?** Edit `profiles/{preset}.json` → `watch_server` detects → `makeline_server` reloads

## Diff algorithm:

1. Load new profile, expand to graph
2. Compare new graph vs old graph (structure + sections)
3. Identify changed modules (section values differ)
4. Send `SectionChanged` events to affected modules
5. Modules reconfigure without restarting process

**What causes module restart?**

# Core Modules Overview

## System orchestration:

- **Planner**: Assembly planning (which dispenser, what order)
- **Sequencer**: Executes plans as bowl moves through system
- **Follower**: Tracks individual bowls (position, state)
- **Preprocessor**: Pre-processing incoming orders

## Hardware control:

- **MakeLine**: Central coordinator, config hot-reload
- **MachineConfig**: Provides config to other modules

- Cabinet → Devices
- Device → Modules
- Module → Child modules

## **Why parent-child?**

- Config inheritance: Children can reference parent config
- Logical grouping: All modules in a device share context
- Traversal: Walk graph to find modules by type/location
- Validation: Ensure required modules exist in correct hierarchy

## **Graph traversal uses:**

- Planner: Find all dispensers in cabinet 2



# That's It

```
just generate-makeline simulation [line_build]
just simulate simulation
# Edit profiles/simulation.json
```

## Key features:

- Hot-reload (2.5s, no restarts)
- Custom layers (dev tweaks)
- Line builds (test/prod switching)
- Auto backups (rollback ready)

**Remember:** `profiles/` not `generated/`