

# Makeline Server

Deterministic makeline generation and description

# Directory Structure

```
profiles/  
  simulation.json          # Source profile  
  simulation/  
    makeline.json         # Expanded config (watched)  
    settings.json        # backup_limit config  
    backups/             # Timestamped backups  
  
generated/profiles/  
  simulation/  
    simulation.json       # Generated profile  
    spawner.json         # Process launcher config  
    config.json          # Machine config  
    watch.json           # File watch config
```

# Generate a Makeline

`just generate-makeline simulation` calls generator twice:

**First:** `makeline_generator profile generate --preset simulation`

- Outputs `generated/profiles/simulation/simulation.json`

**Second:** `makeline_generator profile expand --use-makeline-server`

- Reads `simulation.json`
- Outputs `makeline.json` + `spawner.json` + `config.json`

# Profile Structure

```
{  
  "layouts": { "default": { "cabinets": ["Initial", "Denest"] }},  
  "layer_groups": { "base": [layer1, layer2] },  
  "line_builds": { "production": {  
    "layout_name": "default",  
    "layer_groups": ["base"]  
  }}  
}
```

# Profile Expansion

Generator expands profile to module graph:

1. Read line\_build → get layout (CabinetKind list)
2. Expand CabinetKind → DeviceKind instances
3. Expand DeviceKind → ModuleKind instances
4. Expand ModuleKind → child ModuleKind instances (if any)
5. Result: Cabinet → Device → Module → Module graph

**Cabinet** = physical grouping, **Device** = logical grouping (corresponds to a DUC), **Module** = IPC process or DUC task

# ModuleKey Structure

```
ModuleKey {  
    cabinet: i32,           // Cabinet index  
    device: i32,            // Device index within cabinet  
    module: i32,            // Module index within device  
    kind: ModuleKind,       // Module type (Buffer, Conveyance, etc.)  
}
```

Network-addressable identifier used by subsystems to request config from makeline\_server

# locations() Method

Generator calls `makeline.locations()` after graph expansion:

- Topologically traverses graph (Root → Cabinet → Device → Module → Child)
- Assigns cabinet/device/module indices
- Tracks global `module_kind_index` per ModuleKind (1st Buffer, 2nd Buffer, etc.)
- Tracks per-device module counts for ModuleKey
- Returns `NodeLocations` with all locations

# Identity Assignment: Parents

Parent modules get identity from graph position:

```
module.identity = Identity {  
  owner: format!("{}", module.kind_index), module.kind),  
  subject: "self"  
}
```

Example: 1st Buffer → { owner: "buffer-1", subject: "self" }



# Identity Assignment: Children

Child modules inherit parent's owner:

```
child_module.identity = Identity {  
  owner: parent_module.identity.owner,  // Inherit parent  
  subject: format!("{}", child_index), child_module.kind)  
}
```

Example: Buffer's motor →

```
{ owner: "buffer-1", subject: "motor-1" }
```

Special cases: DripTray GPIO uses "lifecycler-1", Sequencer hardcoded

# Apply Layers

Generator applies layers sequentially:

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

# Generator Outputs

`profiles/simulation/makeline.json`: Full module graph

- Each module: Identity + Sections

`generated/profiles/simulation/spawner.json`: Process list

- Each entry: `-M` flag + Identity

# Launch Processes

`just simulate` launches spawner:

```
{  
  "name": "buffer-1",  
  "executable": "path/to/buffer",  
  "args": ["-M", "--identity", "buffer-1"]  
}
```

**-M flag:** Query makeline\_server for config

# makeline\_server: What it Does

- Loads `profiles/simulation/makeline.json` → builds ModuleKey → Module graph
- Watches `profiles/` directory for changes (500ms debounce)
- Subscribes to MakelineContract::command\_topic()
- Handles `RequestMakelineAvailable { key }` → publishes `MakelineAvailable` to module topic
- Handles `RequestAllSections { key }` → publishes all `ReportSection` events to module topic
- Handles `RequestSection { key, name }` → publishes specific `ReportSection` to module topic

# Subsystems Request Config

Subsystems with `-M` flag use makeline adapter:

- Send `RequestAllSections` with `ModuleKey` to `MakelineContract::command_topic()`
- Example: `{ owner: "buffer-1", subject: "self" }`

`makeline_server` responds:

- Looks up `ModuleKey` in graph
- Publishes `ReportSection` events to subsystem's topic

Subsystem receives sections and starts

# Hot-Reload: Edit makeline.json

**Edit** `profiles/simulation/makeline.json` → **Save**

**File watcher detects** → `handle_makeline_change` :

1. Reloads file, rebuilds graph
2. Diffs old vs new (per-module, per-section)
3. Queues `SectionChanged` events

**Update loop publishes** events to module topics

Modules receive events and reconfigure

# Hot-Reload: Edit Active Profile

**Edit** `profiles/simulation.json` (active profile) → **Save**

**File watcher detects** → `handle_profile_change`:

1. Reads profile, validates JSON
2. Calls `load_and_expand_profile`
3. Expands profile to makeline (with additional layers)
4. Saves expanded makeline to `profiles/simulation/makeline.json`
5. Loads new makeline, sends `ProfileReloaded` event
6. Updates spawner config if enabled



# Summary

1. `just generate-makeline` → generator outputs files
2. `just simulate` → spawner launches with `-M`
3. `makeline_server` connects to broker, loads graph, subscribes
4. Modules send `RequestAllSections` command by Identity
5. `makeline_server` responds with sections via events
6. Edit `makeline.json` → watcher detects → diff → queue → publish
7. Modules receive `SectionChanged` events → reconfigure