# Makeline Server

Deterministic makeline generation and description

# What is Makeline Server?

- Provides configuration for all modules on the makeline over IPC
- Loads a **profile** which contains **line builds**
- Expands profile into **makeline.json**, **config.json**, **watch.json**, and **spawner.json**
- Subsystems request config using **makeline adapter** with their **ModuleKey** (cabinet-device-kind-module)
- Active profile and makeline.json can be **hot-reloaded** → modules reconfigure without restart

# Makeline Server

- Takes profile via `-p` and line build via `-l`
- Uses makeline_generator to expand profile → saves to `profiles/{profile}/{profile}.json`
- Expansion produces `profiles/{profile}/makeline.json`, `config.json`, `watch.json`, and `spawner.json`
- Loads expanded Makeline into memory
- Watches `profiles/` directory for changes
- With `-s`, controls spawner when loading a profile

# Directory Structure: profiles/

- Created by makeline_server when run
- `profiles/` directory provided via `-w` switch to makeline_server

```
profiles/
  simulation.json          # Source profile
  simulation/
    makeline.json          # Expanded config (watched)
    settings.json          # backup_limit config
    backups/               # Timestamped backups
```

# Makeline Generator

- Generates profiles from presets (e.g., `simulation`, `v5111`)
- Expands profiles into makeline.json and config files
- Used by makeline_server to expand profiles at runtime
- Can be run offline as standalone tool

`just generate-makeline simulation`:

- `makeline_generator profile generate --preset simulation` → outputs profile
- `makeline_generator profile expand` → outputs makeline.json, spawner.json, config.json

# Directory Structure: generated/

- Created by `just generate-makeline`
- Offline, separate from makeline_server

```
generated/profiles/
  simulation/
    simulation.json          # Generated profile
    spawner.json             # Process launcher config
    config.json              # Machine config
    watch.json               # File watch config
```

# Makeline Profiles

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

# Makeline Layers

```
{
  "name": "custom_config",
  "edits": [
    { "EditSectionField": {
      "identity": { "owner": "buffer-1", "subject": "self" },
      "section_name": "configuration",
      "field_key": "timeout_ms",
      "field_value": 5000
    }},
    { "AssignSections": {
      "identity": { "owner": "conveyance-1", "subject": "self" },
      "sections": { "ingredients": { ... } }
    }}
  ]
}
```

# 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.)
}
```

String format: `cabinet-device-kind-module`

Example: `0-1-buffer-0`

Unique 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 based on cabinet and device:

```
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 string:

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

Example: Buffer's motor →

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

# Simulate

`just simulate` launches spawner:

```
{
  "name": "buffer-1",
  "executable": "buffer_server",
  "args": ["-M", "-c 1", "-d 1", "-m 0"]
}
```

`-M` **flag**: Use makeline adapter to request config from makeline_server and use queries instead of machine_config

# Subsystems Request Config

Subsystems with `-M` flag use makeline adapter:

- Send `RequestAllSections` with ModuleKey to MakelineContract::command_topic()
- Example ModuleKey: `1-1-buffer-0`

makeline_server:

- Receives section requests on `makeline/command`
- Looks up ModuleKey in graph (which contains Identity + Sections)
- Publishes responses to `makeline/event-{key}`

# 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. Subsystems send `RequestAllSections` command with ModuleKey
5. makeline_server responds with sections via events
6. Edit makeline.json → watcher detects → diff → publish
7. Subsystems receive `SectionChanged` events → reconfigure