Air LiHa Implementation Plan

Architecture Decision

Option Considered: New LiHa Subclass

The LiHa class (`EVO_backend.py` lines 893-1203) is a thin firmware command wrapper — it sends commands like `PVL`, `SEP`, `PPR` etc. It doesn't contain conversion logic or volume calculations. The conversion factors (`* 3` for steps, `* 6` for speed) live in the EVOBackend methods (`_aspirate_action`, `_dispense_action`, `_aspirate_airgap`, `pick_up_tips`). A LiHa subclass would not help.

Option Considered: Parameterize Existing EVOBackend

Adding `steps_per_ul` and `speed_factor` parameters to EVOBackend would change the constructor signature and add complexity for all users, even those with syringe LiHa.

Recommended: EVOBackend Subclass (`AirEVOBackend`)

Rationale: - The Air LiHa differs from syringe LiHa in **init sequence**, **conversion factors**, and **per-operation commands** — these are all EVOBackend-level concerns - Subclassing keeps the existing EVOBackend completely untouched - Clean separation: syringe users use `EVOBackend`, air users use `AirEVOBackend` - The ZaapMotion configuration is specific to the Air LiHa hardware variant - `TipType.AIRDITI` already exists in the codebase — just needs liquid class data

Implementation Steps

Step 1: Create `AirEVOBackend` class

File: `pylabrobot/liquid_handling/backends/tecan/air_evo_backend.py`

```
class AirEVOBackend(EVOBackend):
```

Overrides: - `setup()` — adds ZaapMotion boot exit + motor config before calling `super().setup()` - `_aspirate_action()` — uses 106.4/213 conversion factors instead of 3/6 - `_dispense_action()` — same conversion factor change - `_aspirate_airgap()` — same conversion factor change - `pick_up_tips()` — same conversion factor change - `aspirate()` — wraps with SFR/SFP/SDP commands around plunger operations - `dispense()` — wraps with SFR/SFP/SDP commands around plunger operations - `drop_tips()` — adds SDT command before AST, wraps with SFR/SFP/SDP

Constants (class-level):

```
STEPS_PER_UL = 106.4       # vs 3 for syringe
SPEED_FACTOR = 213          # vs 6 for syringe (half-steps/sec per µL/s)
```

```
SFR_ACTIVE = 133120        # force ramp during plunger movement
SFR_IDLE = 3752            # force ramp at rest
SDP_DEFAULT = 1400         # default dispense parameter
```

ZaapMotion config data:

```
ZAAPMOTION_CONFIG = [        # 33 commands per tip, from USB capture
  "CFE 255,500",
  "CAD ADCA,0,12.5",
  ...
  "WRP",
]
```

Step 2: Add ZaapMotion liquid classes

File: `pylabrobot/liquid_handling/liquid_classes/tecan.py`

Add entries to the `mapping` dict with `TipType.AIRDITI` key, parsed from `DefaultLCs.XML`. Start with the most common liquid classes: - Water free dispense (DiTi 50) - Water wet contact (DiTi 50) - DMSO free dispense (DiTi 50)

Each liquid class entry maps a `(min_vol, max_vol, Liquid, TipType.AIRDITI)` tuple to a `TecanLiquidClass` with the calibration factor, speeds, air gap volumes, and LLD settings from the XML.

Step 3: Export the new backend

File: `pylabrobot/liquid_handling/backends/tecan/__init__.py`

Add `AirEVOBackend` to exports.

Step 4: Add helper method for ZaapMotion commands

In `AirEVOBackend`:

```
async def _zaapmotion_force_mode(self, enable: bool):
    """Send SFR/SFP/SDP to all tips for force mode control."""
    sfr_val = self.SFR_ACTIVE if enable else self.SFR_IDLE
    for tip in range(8):
        await self.send_command("C5", command=f"T2{tip}SFR{sfr_val}")
    if enable:
        for tip in range(8):
            await self.send_command("C5", command=f"T2{tip}SFP1")
    else:
        for tip in range(8):
            await self.send_command("C5", command=f"T2{tip}SDP{self.SDP_DEFAULT}")
```

Step 5: Tests

File: `pylabrobot/liquid_handling/backends/tecan/air_evo_tests.py`

- Test conversion factors: verify `_aspirate_action` produces correct steps for known volumes
- Test ZaapMotion config sequence: mock USB and verify all 33 × 8 config commands are sent
- Test SFR/SFP/SDP wrapping: verify force mode commands surround plunger operations
- Test setup sequence: verify boot exit + config + safety + PIA order

Files Changed

File	Change	Risk
`backends/tecan/air_evo_backend.py`	**NEW** — AirEVOBackend subclass	None (new file)
`backends/tecan/__init__.py`	Add export	Minimal
`liquid_classes/tecan.py`	Add ZaapDiTi mapping entries	None (additive)
`backends/tecan/air_evo_tests.py`	**NEW** — tests	None (new file)

Files NOT changed: - `EVO_backend.py` — untouched, syringe LiHa users unaffected - `tip_creators.py` — `TipType.AIRDITI` already exists - `machine.py`, `backend.py` — no framework changes

Usage

```python
from pylabrobot.liquid_handling import LiquidHandler
from pylabrobot.liquid_handling.backends.tecan import AirEVOBackend
from pylabrobot.resources.tecan.tecan_decks import EVO150Deck

backend = AirEVOBackend(diti_count=8)
deck = EVO150Deck()
lh = LiquidHandler(backend=backend, deck=deck)
await lh.setup()  # handles ZaapMotion boot exit, config, safety, PIA
```

Open Questions

1. **Should ZaapMotion config values be constructor parameters?** The PID gains, current limits, etc. may vary between instruments. For now, hardcode the values from our USB capture; parameterize later if needed.

2. **Should `_zaapmotion_force_mode` send to all 8 tips or only active channels?** EVOware always sends to all 8 regardless of which channels are in use. Start with all 8 for compatibility.

3. **Does `WRP` (write to flash) make the config persistent?** If so, the config commands only need to be sent on first boot after power cycle. We could check `RCS` (Report Configuration Status) first and skip config if already configured. EVOware does this — it checks `RCS` and skips the full config if the ZaapMotion is already in app mode with config present.

4. **Multi-dispense and other advanced operations** — the USB capture with multidispense shows the same SFR/SFP/SDP pattern. No additional ZaapMotion commands are needed beyond what's described here.