Title: Draft Merge Plan

|  |  |
| --- | --- |
| Document ID: | **DocID001** |
| Version: | **1.0** |
| Last Saved: | **0000-00-00 0:00:00** |
| Document Status: | **Draft** |
| Author: | **Neil Thomson** |
| Product/Release |  |

|  |  |
| --- | --- |
| Abstract:  (Comments) | <Enter the abstract in Properties/Summary/Comments> |

Detailed History of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Ver. | Date | Who | Detailed description of Changes |
| 1.0 | 08Dec2004 | NJT | Created |

Table of Contents

1. Introduction 1

2. Draft Merge Plan — TravelProfile PUML (v0.1) 1

2.1 1) Goals & Success Criteria 1

2.2 2) Target Directory Layout (steady state) 1

2.3 3) Phase 0 — Safety Net (15 min) 2

2.4 4) Phase 1 — Common Library First 2

2.4.1 4.1 Classes (copy/replace into core/puml/commonLib) 2

2.4.2 4.2 Enums (copy into core/puml/commonLib) 2

2.5 5) Phase 2 — Root + Segment Stubs 3

2.6 6) Phase 3 — Segment Deep Merge (iterative) 3

2.7 7) SCHEMAHINTS / ENUMHINTS Conventions to Enforce 4

2.8 8) $id / $ref Mapping Guardrails 4

2.9 9) Validation & Lint 4

2.10 10) Risk & Mitigation 5

2.11 11) Daily Working Loop 5

2.12 12) Manifest Template (optional but handy) 5

2.13 13) Hand-off Checklist (per batch) 6

# Introduction

ChatGPT proposed draft merge plan to bring on board the existing .PUML models and upgrade them with JSON-Schema generation hints

Here’s a tight, practical **Draft Merge Plan** to bring your existing PUML model set into the structure we outlined, without disturbing your working generators.

# Draft Merge Plan — TravelProfile PUML (v0.1)

## 1) Goals & Success Criteria

* **Preserve semantics** of your existing model while aligning folders/$id/$ref.
* **Zero generator surprises:** gen:enums, gen:schemas, validate:\*, and lint:hints:strict all pass.
* **Deterministic build:** clean baseId mapping + stable output paths.
* **Day-by-day continuity:** daily snapshot remains reproducible.

**Done =** all migrated classes emit valid JSON Schemas; all enum refs resolve; linter shows 0 errors (warnings optional).

## 2) Target Directory Layout (steady state)

packages/

core/

puml/commonLib/...

json/schemas/commonLib/...

json/enums/commonLib/...

travelProfile/

puml/

root/TravelerProfile.puml

identity/...

contact/...

preferences/...

support/...

json/schemas/...

json/enums/...

tools/

generate-json-schema-from-puml.mjs

generate\_json-enums-from-puml.mjs

lint-schemahints.mjs

**Mapping rule:**  
packages/<seg>/puml/<subs>/<Class>.puml → packages/<seg>/json/schemas/<subs>/<Class>.schema.json  
--baseId https://…/hatpro/ → $id/$ref = {baseId}<seg>/<subs>/<Name>.json|.schema.json

## 3) Phase 0 — Safety Net (15 min)

1. **Branch**: git checkout -b feat/tp-merge-structure
2. **Freeze working tools**: copy your current tools/\*.mjs to tools/\_locked/.
3. **Record baseline**:
   * npm run lint:hints:json → save lint-report.baseline.json
   * npm run validate:schemas & validate:enums (if any) — note outcome.

## 4) Phase 1 — Common Library First

Migrate the shared building blocks so downstream files can $ref them immediately.

### 4.1 Classes (copy/replace into core/puml/commonLib)

* InternationalString
* TechnicalString (a.k.a. TechString)
* NameComponents
* PreferenceValue (+ XOR score|flag)
* Any other foundational types you already use (Address, Email, Phone, etc.)

**SCHEMAHINTS checks**

* Use multi-line note ... end note
* 2-space indents; **no blank lines** inside field or enumDefine blocks
* If a field references an enum → **prefer enumDefine** (most robust)

### 4.2 Enums (copy into core/puml/commonLib)

* TextEncodingEnum, LanguageLocaleEnum, GenderEnum, PreferenceFlag, etc.
* Ensure ENUMHINTS includes:
* enumId: /core/commonLib/<EnumName>
* type: string
* generate: true
* enum:[...]
* Optional: x-enumNames, x-enumDescriptions, x-order, x-deprecated, x-aliases

**Build check**

npm run gen:enums

npm run lint:hints:strict

## 5) Phase 2 — Root + Segment Stubs

Bring in your **actual** segment classes (replacing the starter stubs), but keep the structure:

* travelProfile/puml/root/TravelerProfile.puml
  + fields: identity, contact, preferences, support?
  + $ref each to the segment roots below
* travelProfile/puml/identity/TravelerIdentity.puml
* travelProfile/puml/contact/TravelerContact.puml
* travelProfile/puml/preferences/TravelerPreferences.puml
* travelProfile/puml/support/TravelerSupport.puml

**Tip:** keep the **same path/segment names** to avoid later ref churn.

**Build check**

npm run gen:schemas

npm run validate:schemas

npm run lint:hints:strict

## 6) Phase 3 — Segment Deep Merge (iterative)

For each segment, migrate supporting classes and enums in small, testable batches:

1. **Identity**: name variants, transliteration, document-name models (if in scope), gender, DoB, etc.
2. **Contact**: email/phone channel classes, preferred language, address models.
3. **Preferences**: diet/food model (your declared “diet strategy” concept), communication preferences, lodging/bed preferences, etc.
4. **Support**: accessibility notes (use InternationalString), emergency contacts, assistive device info (if modeled), etc.

**Each mini-batch loop**

npm run gen:enums

npm run gen:schemas

npm run validate:enums

npm run validate:schemas

npm run lint:hints:strict

Commit when green.

## 7) SCHEMAHINTS / ENUMHINTS Conventions to Enforce

* **Enum reference (recommended):**
* field something:
* enumDefine:
* enumId: /<seg>/<path>/<EnumName>

(This yields a $ref to the enum JSON and avoids enumFrom parsing edge cases.)

* **XOR rule (PreferenceValue):** xor:(score|flag) with score -99..99 and flag → /core/commonLib/PreferenceFlag.
* **Array fields:**
* field emails:
* type: array
* items:
* type: string
* format: email
* **$ref to classes:**
* field legalName:
* $ref: /core/commonLib/NameComponents
* **No blank lines** inside a field or enumDefine block; 2-space indents only.

## 8) $id / $ref Mapping Guardrails

* Always run generators with a **trailing slash** in --baseId, e.g.:
* node tools/generate-json-schema-from-puml.mjs \
* --baseId https://schemas.example.org/hatpro/ \
* --packagesDir ./packages --debug
* $ref URIs resolve to:
* https://schemas.example.org/hatpro/<seg>/<subs>/<Name>.json (enums)
* https://schemas.example.org/hatpro/<seg>/<subs>/<Name>.schema.json (classes)

## 9) Validation & Lint

* **During merge**: npm run lint:hints:strict to catch structure/indent issues early.
* **After each batch**: npm run validate:schemas & validate:enums (ajv draft2020).
* **CI hook (later)**:
* "build:ci": "npm run lint:hints:strict && npm run gen:enums && npm run gen:schemas && npm run validate:enums && npm run validate:schemas"

## 10) Risk & Mitigation

| **Risk** | **Symptom** | **Mitigation** |
| --- | --- | --- |
| Hidden tabs / NBSP | property {} or missing $ref | Re-type lines; enforce .editorconfig with spaces + LF |
| Blank line inside block | field ignored | Remove blank lines within field/enumDefine |
| Wrong base path | files written under tools/ or missing | Pass --packagesDir ./packages (run from repo root) |
| Missing enums | linter “target not found” | npm run gen:enums before schemas |
| EnumFrom parsing quirks | {} instead of $ref | Prefer enumDefine with enumId |

## 11) Daily Working Loop

1. Migrate 5–10 PUMLs → run the 5-step build (gen:enums → lint).
2. Commit small, green batches.
3. Update **Daily-Session-Log.md** (files changed, open issues).
4. Export the day’s chat + artifacts.

## 12) Manifest Template (optional but handy)

Create packages/travelProfile/puml/\_MANIFEST.yaml to document intended order (for humans/tools later):

baseId: https://schemas.example.org/hatpro/

segments:

- core/commonLib:

enums: [TextEncodingEnum, LanguageLocaleEnum, GenderEnum, PreferenceFlag]

classes: [InternationalString, TechnicalString, NameComponents, PreferenceValue]

- travelProfile/root: [TravelerProfile]

- travelProfile/identity: [TravelerIdentity, ...]

- travelProfile/contact: [TravelerContact, ...]

- travelProfile/preferences: [TravelerPreferences, ...]

- travelProfile/support: [TravelerSupport, ...]

## 13) Hand-off Checklist (per batch)

* PUML files placed in correct folders
* enumDefine used for all enum fields
* $ref paths start with /
* Generators run clean
* Ajv validate clean
* Linter strict = 0 errors
* Commit created

If you want, tell me which **segment** you’re starting with (Identity, Contact, Preferences, Support), and I’ll tailor this into a segment-specific sub-plan with exact file moves and first-pass field/$ref stubs that match your existing class names.