Title: HATPro source code as Training Material

|  |  |
| --- | --- |
| 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

1.1 Purpose 1

Appendix A - Definitions, Acronyms, Buzzwords and Abbreviations 1

Appendix B - R&D Tax Credit Considerations 1

# Introduction -

What I'm coming to the conclusion is to either a) define templates for enums, classes, etc. or b) have enum and class definitions identified as useful examples. I'm inclined to go with (b) and provide some additional puml metadata to allow a search for examples on.

Love that direction. Go with **(b)**: keep a few **living examples** in-tree and tag them with lightweight metadata so anyone can discover “good patterns” fast.

Here’s a tiny, practical setup you can adopt in minutes.

1. **Authoring pattern (PUML metadata)**

Add a few keys to your SCHEMAHINTS note when a file is meant as an example:

@startuml

class CurrencyCodeEnum {}

note right of CurrencyCodeEnum

SCHEMAHINTS

title: CurrencyCodeEnum

example: true

x-tags: [common, enum, currency, iso4217]

x-summary: Minimal ISO-4217 enum with valuePattern & metadata

enumDefine:

enumId: /core/commonLib/CurrencyCodeEnum

title: ISO 4217 Currency Codes

description: Three-letter ISO 4217 codes (e.g., USD, EUR, CAD)

x-standard: ISO 4217

x-standardRef: https://www.iso.org/iso-4217-currency-codes.html

valuePattern: ^[A-Z]{3}$

generate: true

enum: [USD, EUR, CAD, GBP, AUD]

end note

@enduml

And a class example:

@startuml

class TravelProfile {}

note right of TravelProfile

SCHEMAHINTS

title: TravelProfile

example: true

x-tags: [class, profile, refs]

x-summary: Top-level class referencing commonLib classes & enums

additionalProperties: false

required:[version, encoding]

field version:

type: string

pattern: ^\d+\.\d+\.\d+(-[A-Za-z0-9.-]+)?$

default: 0.1.0

field encoding:

enumDefine:

enumId: /core/commonLib/TextEncodingEnum

field techString:

ref$: /core/commonLib/TechString

end note

@enduml

**Conventions**

* example: true flags the file for discovery.
* x-tags: [...] provides search facets (e.g., enum, class, iso4217, profile, array, composition).
* x-summary: a one-liner for humans.

1. **Indexing script (ready to use)**

I made a tiny Node script that scans packages/\*\*/puml/\*\*.puml, finds SCHEMAHINTS with example: true, and prints a list (or writes JSON).

**Download:** index-examples.mjs

Place it at tools/index-examples.mjs, then run:

# list all examples (table)

node tools/index-examples.mjs ./packages

# filter by tag(s)

node tools/index-examples.mjs ./packages --filter enum,currency

# write a JSON index

node tools/index-examples.mjs ./packages --json examples-index.json

# list the distinct tags in the repo

node tools/index-examples.mjs ./packages --list-tags

Output rows look like:

enum CurrencyCodeEnum CurrencyCodeEnum common,enum,currency,iso4217 core/puml/commonLib/CurrencyCodeEnum.puml

class TravelProfile TravelProfile class,profile,refs profile/puml/TravelProfile.puml

1. **Linter compatibility**

You’re already using the improved lint-schemahints.mjs. To avoid warnings for the new metadata keys, keep (or add) these to the enumDefine allow-list:

* description
* valuePattern (validated as a regex by the linter)
* x-standard
* x-standardRef

For class-level metadata (example, x-tags, x-summary), the linter won’t care—those keys sit **outside** enumDefine. No rule change required.

1. **Repo layout & naming**

* Keep examples **in-place** near the real components (developers see context), not a detached examples/ folder.
* Use tags to slice by purpose: enum, class, array, composition, identity, currency, address, etc.
* Add a short **Examples** section in the Developer Guide explaining example: true, x-tags, and how to run index-examples.

1. **Optional next steps**

* Add an npm script:
* { "scripts": { "examples:index": "node tools/index-examples.mjs ./packages --json examples-index.json" } }
* Publish examples-index.json as a build artifact so newcomers can browse examples via GitHub UI.
* If you want, I can extend the linter later to flag missing x-summary when example: true (helps enforce quality).

If you’d like me to craft 3–4 high-value seed examples (enum; class with $ref; array of classes; composition + oneOf), say the word and I’ll hand you ready-to-paste PUMLs.

1. **You said:**

I'm going to capture this mini-thread and come back to it in preparation for bringing additional developers and users of the Travel Profile. This validates the ideal of using the deliverables as also a library for learning.

1. **ChatGPT said:**

Exactly — that’s a great takeaway.  
What you’re doing here (treating the deliverables as both **production artifacts** and **educational exemplars**) is precisely what mature standards groups do. By embedding “explainable patterns” directly in your PUML + JSON Schema ecosystem, you get:

1. **Living documentation** — every example doubles as working code.
2. **Traceable provenance** — examples evolve with the same governance as schemas.
3. **Self-training materials** — new contributors learn from curated, tagged examples, not scattered wiki pages.

Your plan to return to this thread for onboarding prep is perfect — when you’re ready, we can expand it into a mini “Contributor Onboarding Kit” containing:

* an **Examples Index** (using the index-examples.mjs tool),
* a **Style Guide** explaining tags and SCHEMAHINTS metadata,
* and a **Learning Path** that walks new developers through one enum, one class, and one composite schema.

That will give newcomers a clear progression from “what is a PUML hint block?” to “how do I add a new modular component and validate it?”

Would you like me to mark this thread for later consolidation into that onboarding kit outline when you revisit it?

1. **You said:**

I was just going to grab this particular mini-thread and save it. Would that be a sufficient starting point?

1. **ChatGPT said:**

Absolutely — that’s a **perfect** starting point.

This mini-thread already captures the essentials you’ll want later for the onboarding materials:

* ✅ **Philosophy:** turning the schema repo into a *living learning library*.
* ✅ **Mechanics:** how to tag examples (example: true, x-tags, x-summary).
* ✅ **Tooling:** the index-examples.mjs scanner, its CLI, and how to filter or export JSON.
* ✅ **Governance tie-in:** how it fits with the linter and developer guide conventions.

If you simply save this thread (or drop it into your tools/ or docs/dev-notes/ folder as something like ToolsNote\_IndexExamples.md), you’ll have a clean foundation for expanding later into a formal “Example Library / Developer Onboarding” section.

When you’re ready to extend it, we can add:

* a few curated seed examples (enum, class, composition, oneOf),
* a standard metadata vocabulary for x-tags,
* and a short onboarding script (first-day walkthrough).

Would you like me to format this mini-thread into a ready-to-save Markdown note (ToolsNote\_IndexExamples.md) so you can drop it directly into your repo?