Table of Contents

Dictionnaire

- `Cargo.toml` définit les dépendances (les crates) et leur versions minimum à inclure dans le projet, équivalent du `package.json` de NPM
- `crate`: la plus petite unité de compilation avec cargo, concrètement chaque projet contient un ou plusieurs dossiers avec un `Cargo.toml`
- `crates.io`: le registre officiel des crates publiée pour l'écosystème Rust, l'équivalent de `npmjs.com` pour l'écosystème Javascript, ou `mvnrepository.com` pour Java

Etat de l'art

Format de données existant orienté humainement éditable

Ces recherches ignorent les formats de données largement supporté et répandu tel que le XML, JSON, YAML et TOML. Ils sont tout à fait adapter pour des configurations, de la sérialisation et de l'échange de donnée et sont pour la plupart facilement lisible. Cependant la quantité de séparateurs et délimiteurs en plus du contenu qu'ils n'ont pas été optimisé pour la rédaction par des humains. Le YAML et le TOML, bien que plus léger que le JSON, inclue de nombreux types de données autre que les strings, des tabulations et des guillemets, ce qui rend la rédaction plus fastidieuse qu'en Markdown.

On cherche quelque chose du niveau de simplicité du Markdown en terme de rédaction, mais avec une validation poussée customisable par le projet qui définit le schéma.

TODO: continuer markdown inspiration + besoin

Ces recherches se focalisent sur les syntaxes qui ne sont pas spécifique à un domaine ou qui seraient complètement déliée de l'informatique ou de l'éducation. Ainsi, l'auteur ne présente pas Cooklang [1], qui se veut une langage de balise pour les recettes de cuisines, même si l'implémentation du parseur en Rust [2] pourra servir pour d'autres recherches. On ignore également les projets qui créent une syntaxe très proche du Rust, comme la Rusty Object Notation (RON) [3], de par leur nécessité de connaître un peu la syntaxe du Rust et surtout parce qu'elle ne simplifie pas vraiment l'écriture comparé à du YAML. On ignore également les projets dont la spécification ou l'implémentation est en état de « brouillon » et n'est pas encore utilisable en production.

Contrairement aux languages de programmation qui existent par centaines, les syntaxes de ce genre ne sont pas monnaies courantes. Différentes manières de les nommer existent: language de balise (markup language), format de donnée, syntaxes, langage de donnée, language spécifique à un domaine (de l'anglais Domain Specific Language - DSL), … Les mots-clés utilisés suivants ont été utilisés sur Google, la barre de recherche de Github.com et de crates.io: `data format`, `human friendly`, `human writable`, `human readable`.

KHI - Le langage de données universel

D'abord nommée UDL (Universal Data Language) [4], cette syntaxe a été inventée pour mixer les possibilités du JSON, YAML, TOML, XML, CSV et Latex, afin de supporter toutes les structures de données modernes. Plus concrètement le markup, les structs, les listes, les tuples, les tables/ matrices, les enums, les arbres hiérarchiques sont supportés. Les objectifs sont la polyvalence, un format source (fait pour être rédigé à la main), l'esthétisme et la simplicité.

```
{article}:
uuid: 0c5aacfe-d828-43c7-a530-12a802af1df4
type: chemical-element
key: aluminium
title: Aluminium
description: The <@element>:{chemical element} aluminium.
tags: [metal; common]

{chemical-element}:
symbol: Al
number: 13
stp-phase: <Solid>
melting-point: 933.47
boiling-point: 2743
density: 2.7
electron-shells: [2; 8; 3]

{references}:
wikipedia: \https://en.wikipedia.org/wiki/Aluminium
snl: \https://snl.no/aluminium
```

Liste 1. – Un exemple simplifié tiré de leur README [5], décrivant un exemple d'article d'encyclopédie.

Une implémentation en Rust en proposée [6]. Son dernier commit sur ces 2 repositorys date du 11.11.2024, le projet a l'air de ne pas être fini au vu des nombreux `todo!()` présent dans le code.

Bitmark - le standard des formats éducatifs digitaux

Bitmark est un standard open-source, qui vise à uniformiser tous les formats de données utilisés pour décrire du contenu éducatif digital sur les nombreuses plateformes existantes [7]. Cette diversité de formats rend l'interropérabilité très difficile et freine l'accès à la connaissance et restreint les créateurs de contenus et les éditeurs dans les possibilités de migration entre plateformes. La stratégie est de définir un format basé sur le contenu (Content-first) plus que basé sur son rendu (layout-first) permettant un affichage sur tous type d'appareils incluant les appareils mobiles [7]. C'est la Bitmark Association en Suisse à Zurich qui développe ce standard, notamment à travers des Hackatons organisés en 2023 et 2024 [8].

Le standard permet de décrire du contenu statique et interactif, comme des articles ou des quiz de divers formats. 2 formats équivalents sont définis: le bitmark markup language et le bitmark JSON data model [9]

La partie quizzes du standard inclut des textes à trous, des questions à choix multiple, du texte à surligner, des essais, des vrai/faux, des photos à prendre ou audios à enregister et de nombreux autres type d'exercices.

```
[.multiple-choice-1]
[!What color is milk?]
[?Cows produce milk.]
[+white]
[-red]
[-blue]
```

Liste 2. – Un exemple de question à choix multiple tiré de leur documentation [10]. L'option correcte `white` est préfixée par `+` et les 2 autres options incorrectes par `-`. Plus haut, `[!...]` décrit une consigne, `[?...]` décrit un indice.

```
{
    "markup": "[.multiple-choice-1]\n[!What color is milk?]\n[+white]\n[-red]\n[-
blue]",
    "bit": {
        "type": "multiple-choice-1",
        "format": "text",
        "item": [],
        "instruction": [ { "type": "text", "text": "What color is milk?" } ],
        "body": [],
        "choices": [
            { "choice": "white", "item": [], "isCorrect": true },
            { "choice": "red", "item": [], "isCorrect" : false },
            { "choice": "blue", "item": [], "isCorrect" : false }
        ],
        "hint": [ { "type": "text", "text": "Cows produce milk." } ],
        "isExample": false,
        "example": []
    }
}
```

Liste 3. – Extrait simplifié de la réponse JSON, respectant le standard Bitmark [11]. La phrase `There used to be a ___ here.` doit être complétée par le mot `school` en s'aidant du texte en allemand.

Open Taskpool, projet qui met à disposition des exercices d'apprentissage de langues [12], fournit une API JSON utilisant le Bitmark JSON data model.

Demander à Open Taskpool des exercices d'allemand vers anglais autour du mot `school` de format `cloze` (texte à trou), se fait avec cette simple requête:
`https://taskpool.taskbase.com/exercises?translationPair=de->en&word=school&exerciseType=bitmark.cloze`.

```
...
"cloze": {
    "type": "cloze",
    "format": "text",
    "instruction": "Gegeben: \"Früher war hier eine Schule.\", schreiben Sie das
fehlende Wort",
    "body": [
        { "type": "text", "text": "There used to be a " },
        {
            "type": "gap",
            "solutions": [ "school" ],
            "answer": { "text": "" }
        },
        { "type": "text", "text": " here." }
    ]
},
...
```

Liste 4. – Extrait simplifié de la réponse JSON, respectant le standard Bitmark [11]. La phrase `There used to be a ___ here.` doit être complétée par le mot `school` en s'aidant du texte en allemand.

Un autre exemple d'usage se trouve dans la documentation de Classtime [13], on voit que le système de création d'exercices est basé sur des formulaires. Ces 2 exemples donnent l'impression que la structure JSON est plus utilisée que le markup. Au vu de tous séparateurs et symboles de ponctuations à se rappeler, la syntaxe n'a peut-être pas été imaginée dans le but d'être rédigée à la main directement. Finalement, Bitmark ne spécifie pas de type d'exercices programmation nécessaire à PLX.

NestedText — Un meilleur JSON

NestedText se veut human-friendly, similaire au JSON mais pensé pour être facile à modifier et visualiser par les humains. Le seul type de donnée scalaire supporté est la chaîne de caractères, afin de simplifier la syntaxe et retirer le besoin de mettre des guillemets. La différence avec le YAML, en plus des types de données restreint est la facilité d'intégrer des morceaux de code sans échappements ni guillemets, les caractères de données ne peuvent pas être confondus avec NestedText [14].

```
Margaret Hodge:
    position: vice president
    address:
        > 2586 Marigold Lane
        > Topeka, Kansas 20682
    phone: 1-470-555-0398
    email: margaret.hodge@ku.edu
    additional roles:
        - new membership task force
        - accounting task force
```

Liste 5. – Exemple tiré de leur README [14]

Ce format a l'air assez léger visuellement et l'idée de faciliter l'intégration de blocs multi-lignes sans contraintes de charactères réservée serait utile à PLX. Cependant, tout comme le JSON la

validation du contenu n'est pas géré directement par le parseur mais par des librairies externes qui vérifient le schéma [15]. De plus, l'implémentation officielle est en Python et il n'y a pas d'implémentation Rust disponible; il existe une crate réservée mais vide [16].

SDLang - Simple Declarative Language

SDLang se définit comme « une manière simple et concise de représenter des données textuellement. Il a une structure similaire au XML: des tags, des valeurs et des attributs, ce qui en fait un choix polyvalent pour la sérialisation de données, des fichiers de configuration ou des langages déclaratifs. » (Traduction personnelle de leur site web [17]). SDLang définit également différents types de nombres (32bit, 64bit, entier, flottant, …), 4 valeurs de booléans (`true`, `false`, `on`, `off`) comme en YAML, différents formats de dates et un moyen d'intégrer des données binaires encodées en Base64.

```
// This is a node with a single string value
title "Hello, World"

// Multiple values are supported, too
bookmarks 12 15 188 1234

// Nodes can have attributes
author "Peter Parker" email="peter@example.org" active=true

// Nodes can be arbitrarily nested
contents {
section "First section" {
paragraph "This is the first paragraph"
paragraph "This is the second paragraph"
}
}

// Anonymous nodes are supported
"This text is the value of an anonymous node!"

// This makes things like matrix definitions very convenient
matrix {
1 0 0
0 1 0
0 0 1
}
```

Liste 6. – Exemple tiré de leur site web [17]

Ce format s'avère plus intéressant que les précédents de part le faible nombre de caractères réservés et la densité d'information: avec l'auteur décrit par son nom, email et un attribut booléan sur une seule ligne ou la matrice de 9 valeurs définie sur 5 lignes. Il est cependant regrettable de voir de les strings doivent être entourées de guillemets et les textes sur plusieurs lignes doivent être entourés de backticks `` ` ``. De même la définition de la hiéarchie d'objets définis nécessite d'utiliser une paire `{ }`, ce qui rend la rédaction un peu plus lente.

Librairies existantes de parsing en Rust
Après s'être intéressé aux syntaxes existantes, nous nous intéressons maintenant aux solutions existantes pour simplifier ce parsing de cette nouvelle syntaxe en Rust. Ecrire un parseur entièrement à la main est possible, mais s'il existe des librairies de parsing c'est probablement que l'option d'utiliser une librairie pour s'abstraire d'une partie de la complexité est une option à considérer.

De nombreuses librairies, ce travail ne considère que les 4 premières librairies liées à du parsing générale publiée sur `crates.io` avec le mot clé `parser` le 2025-05-02. avec la liste triée par `Recent Downloads` (c'est à dire dans l'ordre décroissant des compteurs de téléchargements des 3 derniers mois). les librairies non adaptées à des formats textes ou spécifique à un format, ou mis à jour il y a plus d'un an, sont également ignorés. [18]

J'ai retenu les librairies suivantes:
- Winnow
- Nom
- pest
- combine ?? - https://crates.io/crates/combine
- Chumsky far less download but much more github stars

Winnow -
54M téléchargements dans les 3 derniers mois

Chumsky

PEG

Nom

Systèmes de surglignage de code
Les IDEs modernes supportent possèdent des systèmes de surglignage de code (syntax highlighting en anglais) permettant de rendre le code plus lisible en colorisant les mots, charactères ou groupe de symboles de même type (séparateur, opérateur, mot clé du langage, variable, fonction, constante, …). Ces systèmes se distinguent par leur possibilités d'intégration. Les thèmes intégrés aux IDE peuvent définir directement les couleurs pour chaque type de token. Pour un rendu web, une version HTML contenant des classes CSS spécifiques à chaque type de token peut être générée, permettant à des thèmes écrits en CSS de venir appliquer les couleurs. Les possibilités de génération pour le HTML pour le web implique parfois une génération dans le navigateur ou sur le serveur directement.

Un système de surlignage est très différent d'un parseur. Même s'il traite du même langage, dans un cas, on cherche juste à découper le code en tokens et y définir un type de token. Ce qui s'apparente seulement à la premier étape du lexer/tokenizer généralement rencontré dans les parseurs.

Textmate
Textmate est un IDE pour MacOS qui a inventé un système de grammaire Textmate. Ces grammaires permettent de décrire comment tokeniser le code basée sur des expressions régulières. Ces expressions régulières viennent de la librairie Oniguruma [19]. VSCode utilise ces grammaires Textmate [20]. Intellij IDEA l'utilise également pour les langages non supportés par Intellij IDEA [21].

Tree-Sitter
Tree-Sitter est supporté dans Neovim [22], dans le nouvel éditeur Zed [23], ainsi que d'autres. Tree-Sitter a été inventé par l'équipe derrière Atom [24]

Semantic highlighting

Choix final
Si le temps le permet, une grammaire développée avec Tree-Sitter permettra de supporter du surglignage dans Neovim et VSCode dans le futur.

Les serveurs de language et librairies Rust existantes

Une part importante du support d'un langage dans un éditeur, consiste en l'intégration des erreurs, l'auto-complétion, les propositions de corrections et des informations au survol… et de nombreuses fonctionnalités qui améliorent l'interaction et la productivité autour du travail avec le code. Par ex. les erreurs de compilation étant intégrées à l'éditeur, il est possible de voir immédiatement les problèmes même avant d'avoir lancer une compilation manuelle dans une interface séparée.

Contrairement au surlignage de code, ces fonctionnalités demandent une compréhension beaucoup plus fine, ils sont implémentés des processus séparés de l'éditeur (étant donc agnostique du langage de programmation utilisé). Ces processus séparés sont appelés des serveurs de langage (language server en anglais).

La communication entre l'éditeur et un serveur de langage démarré pour le fichier en cours, se fait via le `Language Server Protocol (LSP)` inventé par Microsoft pour VSCode.

Fonctionnement général du protocole

- **JSON-RPC** est utilisé pour faire des appels

JSON-RPC is a bit like HTTP

```
Content-Length: ...\r\n
\r\n
{
"jsonrpc": "2.0",
"id": 1,
"method": "textDocument/completion",
"params": {
...
}
}
```

lsp-server

https://github.com/rust-lang/rust-analyzer/blob/master/lib/lsp-server/examples/goto_def.rs

async-lsp

tower-lsp-server

Choix final

Si le temps le permet…

Protocoles de synchronisation existants

gRPC

Websockets

Choix final

Bibliographie

[1] A. Dubovskoy, « Cooklang – Recipe Markup Language ». [En ligne]. Disponible sur: https://cooklang.org/

[2] A. Dubovskoy, « Canonical Cooklang parser in Rust ». [En ligne]. Disponible sur: https://github.com/cooklang/cooklang-rs

[3] C. de Ron, « Rusty Object Notation ». [En ligne]. Disponible sur: https://github.com/ron-rs/ron

[4] Torm, « udl v0.3.1 - Parser for UDL (Universal Data Language) ». [En ligne]. Disponible sur: https://crates.io/crates/udl

[5] Torm, « The Khi data language ». [En ligne]. Disponible sur: https://github.com/khilang/khi

[6] Torm, « Rust Khi parser & library ». [En ligne]. Disponible sur: https://github.com/khilang/khi.rs

[7] bitmark Association, « bitmark Association website ». [En ligne]. Disponible sur: https://www.bitmark-association.org/

[8] bitmark Association, « bitmark Hackathon ». [En ligne]. Disponible sur: https://www.bitmark-association.org/bitmarkhackathon

[9] bitmark Association, « bitmark Documentation ». [En ligne]. Disponible sur: https://docs.bitmark.cloud/

[10] bitmark Association, « Quizzes - .multiple-choice, .multiple-choice-1 ». [En ligne]. Disponible sur: https://docs.bitmark.cloud/quizzes/#multiple-choice-multiple-choice-1

[11] bitmark Association, « Quizzes - .cloze (gap text) ». [En ligne]. Disponible sur: https://docs.bitmark.cloud/quizzes/#cloze-gap-text

[12] Taskbase, « open-taskpool - 12,000 UK 🇺🇦 → DE 🇩🇪 & DE 🇩🇪 → EN🇬🇧 learning tasks ready for you to use. ». [En ligne]. Disponible sur: https://github.com/taskbase/open-taskpool

[13] Classtime, « Créer la première question / le premier jeu de questions ». [En ligne]. Disponible sur: https://help.classtime.com/fr/comment-commencer-a-utiliser-classtime/creer-la-premiere-question-le-premier-jeu-de-questions

[14] K. Kundert, « NestedText — A Human Friendly Data Format ». [En ligne]. Disponible sur: https://github.com/KenKundert/nestedtext

[15] Ken et K. Kundert, « NestedText documentation - Schemas ». [En ligne]. Disponible sur: https://nestedtext.org/en/latest/schemas.html

[16] bob22z, « docs.rs - Crate nestedtext ». [En ligne]. Disponible sur: https://docs.rs/nestedtext/latest/nestedtext/

[17] S. Ludwig, « SDLang, Simple Declarative Language ». [En ligne]. Disponible sur: https://sdlang.org/

[18] [En ligne]. Disponible sur: https://crates.io/keywords/parser

[19] [En ligne]. Disponible sur: https://macromates.com/manual/en/regular_expressions

[20] [En ligne]. Disponible sur: https://code.visualstudio.com/api/language-extensions/syntax-highlight-guide

[21] [En ligne]. Disponible sur: https://www.jetbrains.com/help/idea/textmate.html

[22] [En ligne]. Disponible sur: https://neovim.io/doc/user/treesitter.html

[23] [En ligne]. Disponible sur: https://zed.dev/docs/extensions/languages?#grammar

[24] « Creating a Grammar ». [En ligne]. Disponible sur: https://flight-manual.atom-editor.cc/hacking-atom/sections/creating-a-grammar/