quickDoc

An ergonomic lightweight markup language with mnemonic inspirations for writing all kind of documents

Cyprien PIERRE

2025-10-14

Résumé

Rédiger le résumé

Mots-clés : Mot clé 1, Mot clé 2

1 Introduction

```
parler en revue de litérature de l'avantage des LML // LaTeX et // word

parler des problématiques d'incohérences et des specs non rigides

parler des limites actuelles
```

2 Etat de l'art

Il existe de multiples approches à la production de documents formatés. L'utilisation de logiciels de traitement de texte tels que MS Word, Google Docs ou encore Only Office est probablement la voies la plus largement répendue. Ces solutions proposent des environnements graphiques qui présentent aux utilisateurs les modifications qu'ils peuvent apporter à leurs document. Cette approche a des limitation notamment dans la réalisations de documents exécutable [a].

Une autre approche très répendue est d'écrire le formatage du texte en même temps que le contenu. Cette seconde approche implique généralement l'utilisation d'un éditeur de texte

a. **Document exécutable :** Désigne un document dont des éléments sont executé (p. ex. code) et remplacés par le produit de l'exécution (p. ex. schéma ou diagrammes).

comme Neovim, Emacs, iA Writter, etc. associé à un logiciel adapté à la convention de rédaction choisie par l'utilisateur. De manière générale, l'utilisateur utilisera un langage de balisage pour déclarer le formatage.

Il existe ainsi deux grandes catégories de langages de balisages appliqués à la rédaction de documents : les langages formels (TEX, HTML, TEI…) et les langages informels (Markdown, Org-Mode, AsciiDoc…)(??, ????). Cette seconde catégorie constitue la famille des LML.

2.1 Langages de définition de documents

Le standard actuel de rédaction de documents aux mises en formes rigoureuse est TEX. Il s'agit à la fois un langage de programmation et un logiciel d'impression de document. Pour l'utiliser lors de la rédaction de document nous utilisons un format de rédaction tels que LATEX, ConTeXt, Texinfo, ou encore OpTeX. Ces formats emploient des macros TEX pour faciliter la mise en page. Le format le plus répendu, nottament dans le monde académique, est le LATEX. La production d'un fichier TEX prêt à être exporté (en PDF, DVI, HTML, PostScript…) requiert une ou plusieurs étapes de compilation du texte source. Les compilateurs (ou moteurs) les plus fréquents sont pdfTeX, LuaTeX, et XeTeX et ils utilisent respectivement les variantes LATEX que sont pdfLaTeX, LuaLaTeX et XeLaTeX. Ces variantes sont similaires à LATEX avec des fonctions complémentaires adaptées au moteur choisis. LATEX bénéficie d'extensions communautaires compatibles ou spécialisées à un moteur et permettant de créer programmatiquement les documents. Nous pouvons citer ici les paquets XyMTeX pour dessiner des structures moléculaires, Biber pour générer les bibliographies, TikZ pour réaliser tout type de dessins ou encore ArabTeX qui permet la prise en charge des langues arabes. Pour rédiger des documents LATEX, il est nécessaire d'installer une distribution TEX. La plus connue est TEX Live dont il existe de multiples dérivés et pour les utilisateurs de Windows il semblerait que MikTeX soit apprécié. Enfin, l'emploi d'un éditeur (Emacs avec AUCTeX, le logiciel LyX, ou encore l'application web Overleaf) facilite la rédaction grace à des fonctionnalités d'assistance et de précomplétion. Des projets comme LTeX+(??, 2025) et Writefull(??, a) permettent de porter ces techniques d'assistances dans d'autres logiciels.

A la lecture de ce descriptif nous remarquons que la production de document avec LATEX implique en de multiples endroits de réaliser des choix et des arbitrages avant même de commencer à rédiger. Quel compileur utiliser ? Quels sont les extensions les plus à jours ou les plus stables ? Dans quel environnement vais-je travailler ? etc. Ces considérations persistent durant tout le processus de rédaction ce qui entraine des périodes de déconcentration et parasite le travail de l'utilisateur. Ce constat semble inciter à la création d'un système probablement plus restreint mais résiliant.

`Parler de Typst/Quatro et de Scribble (lisp)`

2.2 Langages de balisage léger

Markdown est le LML le plus populaire mais aussi le plus critiqué. Sa définition simple (MD Basics) en fait un langage très facile d'approche, facile à apprendre par tout individu. Ses faiblaisses résident néanmoins de la diversité de ses implémentations. MD Basic est supporté

de façon homogène mais n'inclus pas l'ensemble des usages textuels possibles. Ces ommissions ont tendu à la création de douzaines de dérivés non cohérents entres eux. Les utilisateurs sont donc contraint d'apprendre plusieurs Markdown selon l'environnement dans lequel ils rédigent leurs documents.

`Parler de RestructuredText, de Org-Mode et AsciiDoc au minimum`

Org-mode, dont le document Orgdown décris la synthaxe, est un LML très respecté par les utilisateurs d'Emacs. Ce langage est bien plus structuré que le Markdown et offre de nombreuses possibilités grâce à son étroite intégration avec LaTeX et les macros d'Emacs. Son intrication profonde avec Emacs le rend difficilement portable à d'autres environnements de rédaction.

AsciiDoc est une référence majeure des LML. Il bénéficie d'une gouvernance de projet communautaire éprouvée, administrée par l'Eclipse Foundation. Ce langage couvre probablement tous les besoins de rédaction existant. Sa documentation est riche mais peut être intimidante pour un nouvel utilisateur.

2.3 Emergence d'un besoin

=> Specify (github) => LLM => CBE

conclusion : création de **quickDoc**

3 Syntaxe

3.1 Définition

Un bloc de texte est constituée d'une ligne continue et de la première ligne vide immédiatement en dessous. Un retour chario n'entraine alors pas de création d'un nouveau bloc.

3.2 Balisage

Le texte d'un bloc peut être formaté au niveau d'une lettre, d'un mot, d'une ligne continue ou d'un ensemble de ligne continue espacées entres elles d'un unique retour chariot.

Le formattage se réalise en entourant *explicitement* le contenu à formater par les balises de formatage. Ces balises sont des symboles textuels accessibles facilement sur un clavier. Le tableau 1 liste les balises utilisées par **quickDoc** et présente 3 des modalitées d'emplois. Il existe une tollérance pour le formatage d'une ligne complète, comme l'ensemble de la ligne est formatée jusqu'au retour chariot alors il convient d'apposer *uniquement* une double marque en début de ligne.

TABLE 1 : Liste des balisages

Type	Balise	HTML5	Une lettre	Un mot	Une ligne
Superscript	+		+l+etter	+word+	++ this is a superscripted line
Subscript	-		-l-etter	-word-	-- this is a subscripted line
Strique	~		~l~etter	~word~	~~~this is a striked line
Highlight	=		=l=etter	=word=	== this is an highlighted line
Comment	;		;l;etter	;word;	;; this is a commented line
Bold	*		*l*etter	*word*	** this is a bolded line
Italique	/		/l/etter	/word/	// this is an italicised line
Underline	_		_l_etter	_word_	__ this is an underlined line
Verbatim	:		:l:etter	:word:	:: this is a line of verbatim
Math	$		$l$etter	$word$	$$ this is a line of math
Code	`		`l`etter	`word`	`` this is a line of code

La mise en forme d'un ensemble de ligne continues est réalisée par l'inscription de trois balises à la ligne précédent et celle suivant le texte à mettre en forme. Cette mise en forme est valide pour toutes les marques pour des raisons d'uniformité synthaxique bien que certains formatages n'ai que peu de raisons d'être employés de la sorte. La création d'un formatage multiligne se construit de la manière suivante :

```
```<parametres>
 <code
 sur plusieurs
 lignes>
```
```

```
$$$ <parametres>
 <formule mathématique
 sur plusieurs
 lignes>
$$$
```

```
;;; <parametres>
 <commentaires
 sur plusieurs
 lignes>
;;;
```

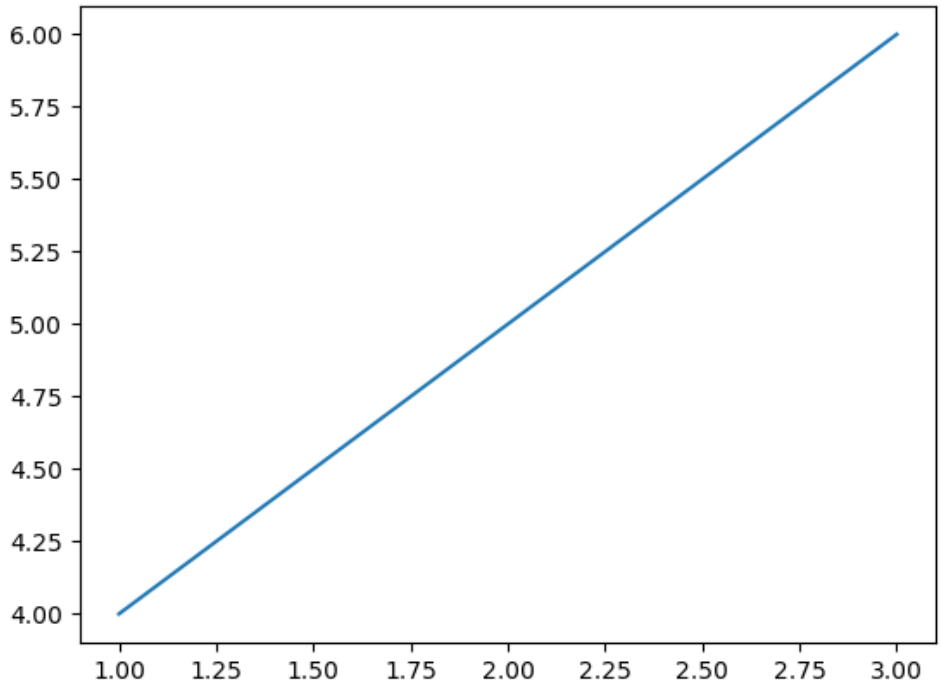
```
::: <parametres>
 <formule mathématique
 sur plusieurs
 lignes>
:::
```

Les blocs de codes peuvent être configurer de manière à obtenir plusieurs comportements. Les paramètres sont à inscrire sous la forme `[[[set <parameter-name>:<value>]]]` avec les parametres suivants :

— `lang` : défini le langage utilisé pour le formatage du texte,
— `runtime` : défini le moteur d'éxecution le cas échéant et si pertinent (e.g. deno, node, bun, etc.).
— `play` : autorise l'execution du code avec `t` et l'empêche par défaut (`nil`).
— `export` : précise ce qui doit être imprimé lors de l'export. Ce paramètre peut prendre spécifiquement les valeurs suivantes :
  — `verbatim` : imprime le code en police monospace avec son formatage et coloration synthaxique,
  — `result` : remplace le bloc de code avec son résultat (e.g. un graphique, un tableau, etc.),

— `both` : imprime successivement le code en verbatim puis son résultat,
— `neither` : n'imprime rien.

Un exemple complet de bloc de code : `` [[[set lang:python export:result]]]
import matplotlib.pyplot as plt; plt.plot([1, 2, 3], [4, 5, 6]); plt.show(



Les blocs de commentaires sont de 4 types :

— sans précision, ils ne sont pas exportées
— avec `type:todo-inline` ils impriment un bloc de type "TODO" en lieu et place ten que celui ci :

bloc "TODO" en ligne

— avec `type:todo` ils impriment un bloc de type "TODO" dans la marge du document tel que celui ci :

e.g.

— avec `type:properties` il permet de créer une liste de propriétées pour le bloc associée tel que :

```
Ceci est un bloc de texte avec des propriétées attachées.
;; [[[set type:properties key1:value1 key2:value2]]]
```

Les blocs de mathématiques peuvent être associées à un résolveur d'équation tel que : $$ `[[[set solver:<nom du résolveur>]]]` `<equation>` Quelques exemples de résolveurs : Math.js, SymPy, Maxima, SageMath, Wolfram Alpha, etc.
Les blocs verbatims peuvent être mis en forme en renseignant un type en lieu et place du

<parametres> sous la forme `[[[set type:<value>]]]`. Ces types de blocs s'appellent des *admonitions*(Donath, Martin, ????) et peuvent prendre les valeurs suivantes : note, abstract, info, tip, success, question, warning, failure, danger, bug, exemple, quote, keywords, answer, hypothesis, theorem, proof.

Les verbatims peuvent également être utilisés pour collecter des entrées utilisateurs et créer des formulaires en écrivant `:[[[get <property> <key>:<value>]]]:` où `<key>` correspond au type attendu et `<value>` correspond à la contrainte associée. Par exemple, voici des appels d'entrées valides :

— `:[[[get user-name string:20]]]:` donnera `____________________` et sera affecté à la propriété "user-name",
— `:[[[get user-lang string:2]]]:` donnera `__` et sera affecté à la propriété "user-lang".

Les balises peuvent être combinées entre elles dans les limites de la matrice de compatibilité présentée par le tableau 2. La combinaison du superscript et du subscript produit un middlescript.

TABLE 2 : Compatibility matrix between formatings

Beacon Type	N°	1	2	3	4	5	6	7	8	9	10	11
Highlight	1	x	1	1	1	1	1	1	1	1	0	0
Strike	2	1	x	1	1	1	1	1	1	1	0	0
Subscript	3	1	1	x	1	1	1	1	1	1	0	0
Superscript	4	1	1	1	x	1	1	1	1	1	0	0
Bold	5	1	1	1	1	x	1	1	1	0	0	0
Comment	6	1	1	1	1	1	x	1	1	0	0	0
Italique	7	1	1	1	1	1	1	x	1	0	0	0
Underline	8	1	1	1	1	1	1	1	x	0	0	0
Verbatim	9	1	1	1	1	0	0	0	0	x	0	0
Inline Math	10	0	0	0	0	0	0	0	0	0	x	0
Inline Code	11	0	0	0	0	0	0	0	0	0	0	x

Lists begins with a bullets that represent their meaning followed by a space. The following is supported :

— Ordered lists starts with `1.`,
— Unordered lists starts with `-`,
— Headers are lists too and starts with a `#`, the number of `#` set the level of the header.

Unordered lists and headers bullets can be followed by a multivalue key in the form of `[<state>:<priorit` with all optionnal values :

— a formal `<state>` : TODO, PLANED, DOING, PAUSE, DONE, ABORTED,
— an estimated `<priority>` value : A, B, C, D, E,
— an `<impact>` factor : 5, 4, 3, 2 ,1.

```
# [TODO:B:3] Title with a full key
```

```
- [DOING:5] a task with a high impact factor
    - [ ] a check box action
    - [x] a checked check box action
```

Sublevels (nested lists) are supported for headers, ordered and unordered lists. There must be 4 spaces before the sublevel bullet.

```
# First level header  - First unordered list 1. First ordered list
    ## Second              - Second               1.1. Second
        ### Third              - Third                1.1.1. Third
```

The export backend shall provide settings to customise desired formatting outputs of ordered lists (alpha-numeric numbering, dots, parentheses…)

3.3 Snippets

Tous les blocs de textes peuvent être associées à des tags. Un tag s'écrit `#<tag>` peut être insérée à n'importe quel endroit du bloc de texte en respectant les rêgles de balisage décrites au paragraphe 3.2.

TABLE 3 : liste des snippets

Type	Lemma	Behavior
Ref Link	`@element`	Insère un `[[lien]]` unidirectionnel
Tag	`#tag`	Insère un `[[lien]]` bidirectionnel

Les tags sont utilisés pour faire de l'analyse sémantique. Ecrire un tag entraine la création ou la mise à jour d'un fichier "tag_<nom-du-tag>.qdo" constitué comme ceci :

```
# Nom-du-tag
[[[get count:?]]] ;; nombre d'occurence dans le projet
[[[get blocs:?nom-du-tag]]] ;; tous les blocs utilisants le tag
```

L'instruction `[[[get blocs:?nom-du-tag]]]` peut être complété par un système de tris. Par exemple, pour lister les blocs par ordre de date décroissante on écrira `[[[get blocs:?nom-du-tag order-by:date-desc]]]`.

Un système de trâme permettant aux utilisateurs de préconfigurer ces documents et de modifier en lot leurs configuration serait un atoût en matière d'expérience utilisateur.

3.4 Callouts

Callouts are bidirectional links inside the current document. They are used to quickly jump to content, a reference, or a definition.

TABLE 4 : Text callouts

Type	Lemma	Behavior
Reference	`[ref:@entry]`	
Foot Note	`[fn:note]`	
Quote	`[cite:@entry]`	
Figure ref	`[fig:name]`	
Table ref	`[tbl:name]`	
Code ref	`[src:name]`	
Header jump	`[header]`	

3.5 Links

Tous les liens suivent l'écriture `[[<CONTEXT>:<LINK>:<QUERY>][<TEXT>]]` pù seul le `<LINK>` **doit** être renseigné. Les autres éléments sont :

— `<CONTEXT>` fournis des informations complémentaires pour l'affichage du lien. Cela permet de mettre en oeuvre des affichages adaptés aux images, vidéos, player de musique, flux RSS, etc.
— `<LINK>` is the path of the resource on the World Wide Web, in a file system, or on any supported network. We can call a 'Header ID' from the current document or from another one.
— `<QUERY>` est utilisé pour renvoyer vers une section particulière du lien (un titre, un callout, etc.). Une requete s'écris `?[<key>:<value>]`, où la clé est n'importe quelle propriété et la valeur est n'importe quelle chaine de charactère.
— `<TEXT>` est le texte de remplacement à afficher à la place du lien.

TABLE 5 : <LINK> types

Type	Lemma	Can be used to
URI		
Web URL	`[[https:link]]`	Display a bookmark
Local File	`[[file:<path>]]`	
Musique	`[[:<path or url>]]`	Display a music player
Image	`[[img:<path or url>]]`	
Document	`[[doc:<path or url>]]`	Display a document viewer
Video	`[[vid:<path or url>]]`	Display a video player
Header ID	`[[id:headerId]]`	
RSS Flow	`[[rss:<url>]]`	Display a list of last entries
IRC Flow	`[[irc:<url>]]`	Display a list of last messages
Email	`[[mailto:<email>]]`	Display a contact form

3.6 Macroprogramme

Des utilisations avancées de documentation programmatique peuvent faire appel à des macros définies par l'utilisateur ou par la plateforme de service.

Dans quickDoc, ces macros sont appellées avec trois crochets tels que : `[[[<macro-name>]]]`

Elles sont définies par l'utilisateur comme ceci :

```
[[[defmacro <macro-name>
       <macro-code>
]]]
```

3.7 Configuration

One might consider using a separate file to manage, preserve, and share all its configurations. That might be a file for LML styles and tweaks, one for special export settings (e.g., TEX, ODT, or HTML). A good way to go is by declaring at the first line of the document

```
;;[[[set config file:file1 file:file... file:fileN]]]
```

3.7.1 Text layout

First, define a style like

```
[[[set text_style name:paraghaphe scope:raw-text al:justif size:12 long:80
sans-pro columns:nil]]]
```

Then apply it, it will be set from the statement until the new one.

```
[[[use text-style name:paragraphe]]]
```

4 Consistency analysis

A consistent lightweight markup language shall have only one way to format text.

Markdown variants on the 6 are limited to those listed by IANA's "Markdown Variant" (??, 2023). We exclude SSW and Quarto, the first one is too contextual and the second is based on Pandoc markdown.

- Bol : Bold
- Ita : Italique
- OrL : Ordered List
- UnL : Unordered List
- Und : Underline
- Hig : Highlight
- Str : Strike
- Ver : Verbatim
- Cod : Inline code
- Sup : Superscript
- Sub : Subscript
- Com : Comment

TABLE 6 : Text formatting consistency versus Markdown variants

Format	Bol	Ita	OrL	UnL	Und	Hig	Str	Ver	Cod	Sup	Su
quickDoc	1	1	1	1	1	1	1	1	1	1	
Common Markdown(??, a)											
MultiMarkdown(??, a)											
Github Flavored Markdown(??, a)											
Pandoc(??, a)											
Fountain(??, a)											
Markdown for RFCs(Thomas Leitner, ????)											
Pandoc2rfc											
Markdown Extra(??, a)											
Markedly Structured Text(??, 2025a)											
AsciiDoc											
reStructuredText											
Orgdown (Org-Mode)											
Textile											
Djot											
Wikitext											
Creole											
txt2tags											
Setext											

5 Capacity analysis

6 Typesystem compatibility

7 Conclusion

8 Références du document

8.1 Liste des figures

8.2 Liste des tableaux

8.3 Liste des codes sources

Listings

8.4 Liste des glosses

8.5 Liste des acronymes

9 Bibliographie

Donath, Martin (). *Admonitions - Material for MkDocs*.

(2025). *Ltex-plus/Ltex-Ls-Plus*.

(a). *Writefull*.

(2023). *Markdown Variants*.

(a). *Syntax – Fountain*.

(a). *PHP Markdown Extra*.

(2025a). *Jupyter-Book/Mystmd*.

Thomas Leitner (). *Kramdown Syntax*.