

# Technical Details

胡雨軒

November 14, 2014

Items in this files are displayed in something close to lexicographic order. The first hasn't to be the most important.

## Contents

<b>1</b>	<b>Gephi</b>	<b>1</b>
1.1	Troubleshoot whilst loading . . . . .	1
1.2	Can't display the whole character set . . . . .	2
<b>2</b>	<b>j00Q code generation</b>	<b>2</b>
<b>3</b>	<b>L<sup>A</sup>T<sub>E</sub>X</b>	<b>2</b>
3.1	Why using L <sup>A</sup> T <sub>E</sub> X instead of markdown? . . . . .	2
3.2	Preamble . . . . .	2
3.3	Babel and cjk . . . . .	3
3.4	Font and IDC . . . . .	3
3.5	Compilation . . . . .	3
<b>4</b>	<b>MariaDB</b>	<b>4</b>
4.1	Why not MySQL? . . . . .	4
<b>5</b>	<b>La carte de données PairMap</b>	<b>4</b>

## 1 Gephi

### 1.1 Troubleshoot whilst loading

Can't be loaded if Java 8 is default JVM version so keep lower.

### 1.2 Can't display the whole character set

Use GNU Unifont. That font contains almost a glyph for each Unicode code point.

## 2 jOOQ code generation

Refer to the jOOQ official website tutorial for further informations.

Given the modified `pom.xml` and the additionnal file `huawen.xml` (aimed at setting bindings with Java project), you may execute the following lines to update Java entities.

```
1 mvn compile #
2 mvn exec:java -Dexec.mainClass="org.jooq.util.GenerationTool" -
  Dexec.args="/huawen.xml"
```

## 3 L<sup>A</sup>T<sub>E</sub>X

### 3.1 Why using L<sup>A</sup>T<sub>E</sub>X instead of markdown?

You're right, markdown can be read directly on github with no downloads and is lighter than L<sup>A</sup>T<sub>E</sub>X but the latter is pretty easier than markdown in purpose of type-setting hypertext including links, bibliographies and so on. Documents written with L<sup>A</sup>T<sub>E</sub>X are far more beautiful and so much easier to compose instead of being a few bit slower to get typed.

Nonetheless I've been trying to automatically generate markdown files with pandoc then to not to upload 'pdf' here but just \*.tex and \*.md. I faced some issues (handling bibliography-related and \input commands) then I gave up. I know you more than me about it, please tell me :-)

### 3.2 Preamble

L<sup>A</sup>T<sub>E</sub>X files here use preambles as modules, so rather prefer to load preambles with commands from the package `import`. On the contrary, it's easier to enclose text with `\input` or even `\inputAllFiles` which import `*.tex` files from a directory and sort them lexicographically.

### 3.3 Babel and cjk<sup>1</sup>

An sinogram in standard text is rendered correctly but when typed in `\foreignlanguage` just produces a 華 character. Hum..., it seems it was a bug and it's been fixed.

### 3.4 Font and idc<sup>2</sup>

Using `xeCJK` is bad and short-minded :

```
1 \usepackage{fontspec,xeCJK}
2
3 \setmainfont{Linux Libertine}
4 \setCJKmainfont{AR PL New Sung}
```

It's better to use `ucharclasses` which is buggy but do the job when nicely asked to.

```
1 \usepackage{fontspec}
2
3 \defaultfontfeatures{Mapping=tex-text}
4 \setmainfont{Linux Libertine}
5 \newfontfamily{\chinesefont}{AR PL New Sung}
6 \newfontfamily{\idcfont}{Unifont}
7
8 \usepackage[CJK, IdeographicDescriptionCharacters]{ucharclasses}
9 \setTransitionsForCJK{\chinesefont}{\normalfont}
10 \setTransitionTo{IdeographicDescriptionCharacters}{\idcfont}
11 \setTransitionFrom{IdeographicDescriptionCharacters}{\normalfont}
```

Maybe the compiler automatically embeds needed font glyphs so no need to add arguments like `-dSubsetFonts=true -dEmbedAllFonts=true`.

---

<sup>1</sup>Stands for *Chinese, Japanese and Korean*.

<sup>2</sup>Stands for *ideographic description character*.

### 3.5 Compilation

I use the XeTeX compiler.

The option `recorder` is required by the package `currfile`. More specifically, the `abspath` option loads the sub-package `currfile-abspath` and requires compiler option `recorder` to be used. Thus, as `currfile` *could be used*, your compilation line should look something like :

```
xelatex -recorder
```

## 4 MariaDB

### 4.1 Why not MySQL ?

Because I prefer truly free and maintained software.

## 5 La carte de données PairMap

La structure de données dans laquelle on stocke les caractères est un dictionnaire qui a pour clef l'ids d'un caractère (ou son hash). Il faut une structure qui évite complètement les redondances, c'est-à-dire que la forêt des sinogrammes n'ait pas de doublon quel que soit l'ordre dans lequel les caractères sont entrés et quel que soit le détail des ids. PairMap n'a pas vocation à être générique : des structures existent déjà pour cela mais à offrir la solution la plus adaptée au stockage des sinogrammes.

Trois stratégies d'optimisation des accès mémoires :

- Un tas ;
- Un accès aléatoire ;
- Un arbre couvrant.

### Stratégie du tas

- Chaque caractère possède ses propres composants ;
- La taille d'un tas est 3 ;

- Solution tributaire des IDS ;
- Explosion combinatoire pour mettre à jour ;
- Analyse de caractère rapide.

### Stratégie de l'accès aléatoire

- Lecture et écriture rapide si vraiment aléatoire ;
- Chaque sinogramme contient un tableau de référence de composants.

### Stratégie de l'arbre couvrant

- Hypothèse : faiblement connecté ;
- Euh... mais ce n'est vraiment pratique à parcourir !

Un sinogramme comporte trois références vers ses composants et pour l'immense majeure partie deux références. Si le sinogramme est une entrée de la table on a un caractère, un point de code et une IDS. Si c'est un sinogramme induit on n'a qu'une IDS. Certains sinogrammes d'un pan Unicode (*unicode block*) sont décomposés en sinogrammes qui n'ont pas de décomposition dans ce pan mais se décomposent dans un autre pan. L'ordre de décomposition des sinogrammes n'est pas respecté globalement mais seulement à l'intérieur d'un pan. Bien qu'un point de code et une IDS renvoient à une même réalité, il faut les voir comme deux sous-clefs, facette d'une même clef.

Tous les caractères ont une IDS mais seulement les caractères explicites ont un point de code. Les caractères implicites n'ont pas de point de code, seulement une IDS. L'IDS peut donc être vue comme une clef principale à côté de laquelle on place lorsque c'est possible un point de code. L'organisation interne de PairMap devra donc être un dictionnaire  $(K1, V)$  avec  $K1$  une IDS et  $V$  une référence vers un objet Node. Il y a également un dictionnaire  $(K2, K1)$  avec  $K2$  un point de code. Le dictionnaire réciproque  $(K1, K2)$  n'est nécessaire que pour économiser la résolution d'une référence : on peut en effet accéder à  $K2$  très facilement à partir de  $K1$  :  $K1 \rightarrow V.K2$ .

L'interface de PairMap est finalement un dictionnaire  $(K1, K2, V)$ .

La classe [TreeMap](#) peut-elle apporter quelque chose de plus que HashMap ?