

L'extension pour $\text{T}_{\text{E}}\text{X}/\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$

simplekv

v 0.32

15 juin 2025

Christian TELLECHEA
unbonpetit@netc.fr

Cette petite extension est une implémentation d'un système dit à « *clés/valeurs* » pour $\text{T}_{\text{E}}\text{X}$ ou $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$. Elle comporte juste l'essentiel, aucune fioriture inutile n'a été codée et aucune extension tierce n'est nécessaire à son fonctionnement.

Cette petite extension se veut minimaliste. Trop sans doute puisqu'on ne la juge pas au niveau d'autres, jugées plus « sérieuses. »

Quoiqu'il en soit, simplekv a le mérite d'exister et se place à l'opposé des usines à gaz que l'on peut trouver dans cet exercice de style. Elle est écrite en T_EX, fonctionne donc sous tous les moteurs et ne requiert aucun package.

1 Clés, valeurs

Lorsqu'une macro doit recevoir des paramètres dont le nombre n'est pas fixe ou connu, il est commode de procéder par $\langle \text{clés} \rangle$ et $\langle \text{valeurs} \rangle$. Voici brièvement les définitions et les limitations des structures mises à disposition :

- une $\langle \text{clé} \rangle$ est un mot désignant un paramètre ; il est formé de préférence avec des caractères de code de catégorie 11 (lettres), 12 (autres caractères sauf la virgule et le signe =) et 10 (l'espace). On peut cependant y mettre des caractères ayant d'autres codes de catégorie, dans la limitation de ce qui est admis dans la primitive `\detokenize` ; une $\langle \text{clé} \rangle$, même si cela revêt peu de signification, peut être vide ;
- la syntaxe pour assigner une $\langle \text{valeur} \rangle$ à une $\langle \text{clé} \rangle$ est : $\langle \text{clé} \rangle = \langle \text{valeur} \rangle$;
- les espaces qui précèdent et qui suivent la $\langle \text{clé} \rangle$ et la $\langle \text{valeur} \rangle$ sont ignorés, mais *pas ceux* qui se trouvent à l'intérieur de la $\langle \text{clé} \rangle$ ou de la $\langle \text{valeur} \rangle$;
- une $\langle \text{valeur} \rangle$ est un $\langle \text{code} \rangle$ arbitraire ;
- si une $\langle \text{valeur} \rangle$ est entourée d'accolades, ces dernières seront retirées : $\langle \text{clé} \rangle = \langle \text{valeur} \rangle$ est donc équivalent à $\langle \text{clé} \rangle = \{ \langle \text{valeur} \rangle \}$;
- lorsqu'une valeur est entourée de *plusieurs* imbrications d'accolades, seul le niveau externe est retiré et donc $\langle \text{clé} \rangle = \{ \{ \langle \text{valeur} \rangle \} \}$ est compris comme $\langle \text{clé} \rangle = \{ \langle \text{valeur} \rangle \}$;
- lorsque plusieurs couples de $\langle \text{clés} \rangle / \langle \text{valeurs} \rangle$ doivent être spécifiés, ils sont séparés les uns des autres par des virgules ;
- une virgule ne peut figurer dans une $\langle \text{valeur} \rangle$ que si la virgule est entre accolades ; par exemple, `foo=1, 5` n'est pas valide car la $\langle \text{valeur} \rangle$ s'étend jusqu'au 1. Il faudrait écrire `foo=\{1, 5\}` pour spécifier une valeur de 1, 5 ;
- les $\langle \text{valeurs} \rangle$ sont stockées *telles qu'elles sont lues* ; en particulier, aucun développement n'est effectué ;
- les définitions sont *locales* : par conséquent, toute $\langle \text{clé} \rangle$ définie ou modifiée dans un groupe est restaurée à son état antérieur à la sortie du groupe ;
- des $\langle \text{clé} \rangle / \langle \text{valeurs} \rangle$ destinées à une même macro ou à un même usage doivent être regroupées dans un ensemble dont on choisit le nom. Un tel ensemble est appelé $\langle \text{trousseau} \rangle$.

2 Commandes mises à disposition

Les macros `\setKV` et `\setKVdefault` Ces commandes définissent des $\langle \text{clés} \rangle$ et leur assignent des $\langle \text{valeurs} \rangle$ dans un $\langle \text{trousseau} \rangle$. La seule différence entre les deux macros est que `\setKVdefault`, en plus d'assigner les $\langle \text{valeurs} \rangle$ aux $\langle \text{clés} \rangle$, les sauvegarde en vue d'une restauration ultérieure avec `\restoreKV`.

On écrit

$$\backslash \text{setKV}[\langle \text{trousseau} \rangle] \{ \langle \text{clé } 1 \rangle = \langle \text{valeur } 1 \rangle, \langle \text{clé } 2 \rangle = \langle \text{valeur } 2 \rangle, \dots, \langle \text{clé } n \rangle = \langle \text{valeur } n \rangle \}$$

Il faut noter que

- pour parvenir à leurs fins, ces macros peuvent être utilisées une ou plusieurs fois, avec un nombre de couples $\langle \text{clé} \rangle / \langle \text{valeur} \rangle$ quelconque ;
- un ensemble $\langle \text{clé} \rangle = \langle \text{valeur} \rangle$ réduit à 0 caractère après suppression des espaces est ignoré ;
- lors de la lecture des $\langle \text{clés} \rangle / \langle \text{valeurs} \rangle$, la virgule et le signe égal doivent avoir un catcode de 12 sans quoi ils ne seront pas compris comme frontières entre $\langle \text{clés} \rangle$ et $\langle \text{valeurs} \rangle$ et ne joueront pas leur rôle ;
- le nom du $\langle \text{trousseau} \rangle$, bien qu'entre crochets, est *obligatoire* , il peut éventuellement être vide bien que cela ne soit pas conseillé ;
- les espaces autour du nom du $\langle \text{trousseau} \rangle$ ne sont *pas* retirés et donc le trousseau « foo » n'est pas le même que le trousseau « foo_ » ;
- si une même $\langle \text{clé} \rangle$ est définie plusieurs fois, la $\langle \text{valeur} \rangle$ retenue sera celle de la dernière assignation ;
- les $\langle \text{valeurs} \rangle$ peuvent être booléennes auquel cas, elles *doivent* être « true » ou « false » en caractères de catcode 11 ;
- si une $\langle \text{valeur} \rangle$ est omise, elle est comprise comme étant « true ». Ainsi, écrire

$$\backslash \text{setKV}[\text{foo}]\{ \text{mon bool} \}$$

est équivalent à

```
\setKV[foo]{mon bool = true}
```

La macro `\useKV` Cette macro purement développable renvoie la *⟨valeur⟩* préalablement associée à une *⟨clé⟩* dans un *⟨trousseau⟩* :

```
\useKV[⟨trousseau⟩]{⟨clé⟩}
```

Il faut noter que

- si la *⟨clé⟩* n’a pas été définie, une erreur sera émise;
- si la *⟨clé⟩* est booléenne, le texte « true » ou « false » sera renvoyé;
- il faut 2 développements à `\useKV[⟨trousseau⟩]{⟨clé⟩}` pour donner la *⟨valeur⟩* associée à la *⟨clé⟩*.

<pre>\setKV[foo]{nombre = 5 , lettres= AB \textit{CD} , mon bool}</pre>		
a) <code>\useKV[foo]{nombre}.</code>	b) <code>\useKV[foo]{lettres}.</code>	c) <code>\useKV[foo]{mon bool}.</code>
<pre>\setKV[foo]{lettres = X Y Z \textbf{123} }</pre>		
a) <code>\useKV[foo]{nombre}.</code>	b) <code>\useKV[foo]{lettres}.</code>	c) <code>\useKV[foo]{mon bool}.</code>
<hr/>		
a) 5.	b) AB CD.	c) true.
a) 5.	b) X Y Z 123.	c) true.

La macro `\restoreKV` La macro `\restoreKV[⟨trousseau⟩]` réinitialise toutes les *⟨clés⟩* du *⟨trousseau⟩* aux *⟨valeurs⟩* qui ont été définies lors des précédentes exécutions de `\setKVdefault`. La macro `\useKVdefault[⟨trousseau⟩]` lui est équivalente.

La macro `\ifboolKV` Cette macro permet, selon la valeur d’une *⟨clé booléenne⟩*, d’exécuter un des deux *⟨codes⟩* donnés. La syntaxe est

```
\ifboolKV[⟨trousseau⟩]{⟨clé⟩}{⟨code si "true"⟩}{⟨code si "false"⟩}
```

La macro est purement développable, elle nécessite 2 développements pour donner l’un des deux codes, et exige que la *⟨clé⟩* soit booléenne sans quoi un message d’erreur est émis.

La macro `\ifkeyKV` Cette macro teste si une *⟨clé⟩* est définie selon la syntaxe

```
\ifkeyKV[⟨trousseau⟩]{⟨clé⟩}{⟨code si clé définie⟩}{⟨code si clé non définie⟩}
```

La macro est purement développable et nécessite 2 développements pour donner l’un des deux codes.

La macro `\ifemptyKV` Cette macro teste si la *⟨valeur⟩* d’une *⟨clé⟩* est vide selon la syntaxe

```
\ifemptyKV[⟨trousseau⟩]{⟨clé⟩}{⟨code si valeur vide⟩}{⟨code si valeur non vide⟩}
```

La macro est purement développable et nécessite 2 développements pour donner l’un des deux codes.

La macro `\showKV` Cette commande écrit dans le fichier log la *⟨valeur⟩* et le *⟨code⟩* assignés à une *⟨clé⟩* d’un *⟨trousseau⟩* :

```
\showKV[⟨trousseau⟩]{⟨clé⟩}
```

Si la *⟨clé⟩* n’est pas définie, « non défini » est affiché dans le fichier log. Il en est de même pour le *⟨code⟩*.

3 Code

En plus d’une *⟨valeur⟩*, un *⟨code⟩* arbitraire peut être assigné à n’importe quelle *⟨clé⟩*. Pour ce faire, on écrit

```
\defKV[⟨trousseau⟩]{⟨clé 1⟩=⟨code 1⟩,⟨clé 2⟩=⟨code 2⟩,...,⟨clé n⟩=⟨code n⟩}
```

Chaque *⟨code⟩* peut contenir « #1 » qui représente la *⟨valeur⟩* de la *⟨clé⟩*. Ce *⟨code⟩* est exécuté lorsque une *⟨valeur⟩* est assignée à la *⟨clé⟩* avec `\setKV`, `\setKVdefault` ou `\restoreKV`.

Ainsi déclarer

```
\defKV[x]{ mykey = \def\foo{\textbf{#1}}
```

va définir une macro `\foo` dès que la *clé* « mykey » va être définie (ou redéfinie) et donc, si l'on écrit

```
\setKV[x]{ mykey = bonjour }
```

le code qui est exécuté en coulisses est

```
\def\foo{\textbf{bonjour}}
```

```
\defKV[x]{ mykey = \def\foo{\textbf{#1}} }
\setKV[x]{ mykey = bonjour }% définition
1) \meaning\foo\par
2) \useKV[x]{ mykey }

\setKV[x]{ mykey = hello }% redéfinition
3) \meaning\foo\par
4) \useKV[x]{ mykey }

1) macro:->\textbf{bonjour}
2) bonjour
3) macro:->\textbf{hello}
4) hello
```

La macro `\testboolKV` permet de tester, par exemple dans un *code*, si son argument est « true » ou « false »

```
\testboolKV{<argument>}{<code si true>}{<code si false>}
```

La macro est purement développable, elle nécessite 2 développements pour donner l'un des deux codes, et exige que l'*argument* soit booléen sans quoi un message d'erreur est émis.

```
\defKV[x]{ x = \def\test{\testboolKV{#1}{test positif}{test négatif}}
\setKV[x]{ x = true}
1) \test

\setKV[x]{ x= false}
2) \test

1) test positif
2) test négatif
```

Toute autre valeur que « true » ou « false » générera un message d'erreur.

4 Un exemple d'utilisation

Voici comment on pourrait programmer une macro qui affiche un cadre sur une ligne, grâce à la macro `\fbox` et l'environnement `center` de \TeX . Pour cela les *clés* suivantes seront utilisées :

- le booléen `inline` qui affichera le cadre dans le texte s'il est vrai et sur une ligne dédiée s'il est faux ;
- `sep` qui est une dimension mesurant la distance entre le texte et le cadre (par défaut 3pt) ;
- `width` qui est la largeur des traits du cadre (par défaut 0.5pt) ;
- `style` qui contient le code exécuté avant le texte.

Une première façon de faire, sans recours à `\defKV` ;

```
\setKVdefault[frame]{
  sep      = 3pt,
  line width = 0.5pt,
  style    = \bfseries,
  inline
}
\newcommand\frametxt[2][{}]{%
  \restoreKV[frame]% revenir au valeurs par défaut
  \setKV[frame]{#1}% lit les arguments optionnels
  \fboxsep = \useKV[frame]{sep}
  \fboxrule= \useKV[frame]{line width}
  \ifboolKV[frame]{inline}
```

```

{}
{\begin{center}}}%
\fbbox{\useKV[frame]{style}#2}%
\ifboolKV[frame]{inline}
{}
{\end{center}}}%
}
Un essai en ligne par défaut \frametxt{essai} puis un autre \frametxt[sep=5pt,line width=2pt]{essai}
et un dernier \frametxt[sep=1pt,style=\itshape]{essai}.

```

Un essai hors ligne : \frametxt[inline = false, style=\bfseries\color{red}]{essai centré}

Un essai en ligne par défaut essai puis un autre essai et un dernier *essai*.

Un essai hors ligne :

essai centré

Dans l'exemple repris ci-dessous et grâce à \defKV, on stocke tous les paramètres lors de leur assignation. Il y a bien moins de verbosité dans le code de frametxt ce qui le rend plus léger et plus lisible.

```

\defKV[frame]{%
  sep      = {\fbboxsep = #1 },
  line width = {\fbboxrule= #1 },
  inline    = \testboolKV{#1}
              {\def\hookpre{}\def\hookpost{}}
              {\def\hookpre{\begin{center}}\def\hookpost{\end{center}}},
  style     = \def\fstyle{#1}
}
\setKVdefault[frame]{
  sep      = 3pt,
  line width = 0.5pt,
  style     = \bfseries,
  inline
}
\newcommand\frametxt[2][{}]{%
  \restoreKV[frame]% revenir au valeurs par défaut
  \setKV[frame]{#1}% lit les arguments optionnels
  \hookpre
  \fbbox{\fstyle #2}%
  \hookpost
}
Un essai en ligne par défaut \frametxt{essai} puis un autre \frametxt[sep=5pt,line width=2pt]{essai}
et un dernier \frametxt[sep=1pt,style=\itshape]{essai}.

```

Un essai hors ligne : \frametxt[inline = false, style=\bfseries\color{red}]{essai centré}

Un essai en ligne par défaut essai puis un autre essai et un dernier *essai*.

Un essai hors ligne :

essai centré