L'extension pour TEX/LATEX

simplekv

v 0.1

8 aout 2017

Christian Tellechea
unbonpetit@netc.fr

Cette petite extension est une implémentation d'un système dit à « clé/valeurs » pour TEX ou LATEX. Aucune fioriture inutile n'a été codée, elle comporte juste l'essentiel.

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 ⟨*clés*⟩ et ⟨*valeurs*⟩. Beaucoup d'extensions allant en ce sens existent déjà.
Le but de celle-ci n'est pas de grossir le nombre déjà trop grand de ces extensions, c'est simplement un morceau de l'extension `hlist` qui a été converti en extension car ce système ⟨*clé*⟩/⟨*valeurs*⟩ est également utilisé par l'extension `scratch`.

Voici brièvement les définitions et les limitations des structures mises à disposition :

- une ⟨*clé*⟩ 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 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` ;
- la syntaxe pour assigner une ⟨*valeur*⟩ à une ⟨*clé*⟩ est : ⟨*clé*⟩=⟨*valeur*⟩ ;
- les espaces qui précèdent et qui suivent la ⟨*clé*⟩ et la ⟨*valeur*⟩ sont ignorés, mais *pas ceux* qui se trouvent à l'intérieur de la ⟨*clé*⟩ ou de la ⟨*valeur*⟩ ;
- une ⟨*valeur*⟩ est un ⟨*code*⟩ arbitraire ;
- si une ⟨*valeur*⟩ est entourée d'accolades, ces dernières seront retirées : ⟨`clé`⟩=⟨`valeur`⟩ est donc équivalent à ⟨`clé`⟩=`{`⟨`valeur`⟩`}` ;
- lorsque plusieurs couples de ⟨*clés*⟩/⟨*valeurs*⟩ doivent être spécifiés, ils sont séparés les uns des autres par des virgules ;
- une virgule ne peut figurer dans une ⟨*valeur*⟩ que si la virgule est dans un niveau d'accolades ; par exemple, `foo=1,5` n'est pas valide car la ⟨*valeur*⟩ s'étend jusqu'au 1. Il faudrait écrire `foo={1,5}` pour spécifier une valeur de `1,5` ;
- lorsqu'une valeur est entourée de *plusieurs* d'accolades, seul le niveau externe est retiré ;
- les ⟨*valeurs*⟩ sont stockées *telles qu'elles sont lues* ; en particulier, aucun développement n'est effectué ;
- les définitions sont *locales* : par conséquent, toute ⟨*clé*⟩ définie ou modifiée dans un groupe est restaurée à son état antérieur à la sortie du groupe ;
- des ⟨*clé*⟩/⟨*valeurs*⟩ 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 ⟨*trousseau*⟩.

2 Commandes mises à disposition

La macro `\setKVdefault` Cette commande est un préalable à toute utilisation de ⟨*clés*⟩ puisqu'elle définit un ⟨*trousseau*⟩ contenant des ⟨*clés*⟩ et leurs ⟨*valeurs*⟩ par défaut.
On écrit

`\setKVdefault[`⟨`trousseau`⟩`]{`⟨`clé 1`⟩=⟨`valeur 1`⟩`,`⟨`clé 2`⟩=⟨`valeur 2`⟩`,...,`⟨`clé i`⟩=⟨`valeur i`⟩`}`

Il faut noter que

- l'argument entre accolades contenant les ⟨*clés*⟩ et les ⟨*valeurs*⟩ ne peut pas être vide ;
- le nom du ⟨*trousseau*⟩, bien qu'entre crochet, est *obligatoire*, mais il peut être vide bien que cela ne soit pas conseillé ;
- si plusieurs ⟨*clés*⟩ sont identiques, seule la dernière ⟨*valeur*⟩ sera prise en compte ;
- les ⟨*valeurs*⟩ peuvent être booléennes, auquel cas, elles *doivent* être « `true` » ou « `false` » en caractères de catcode 11 ;
- si une ⟨*valeur*⟩ est omise, elle est comprise comme étant « `true` ». Ainsi, écrire

  ```
  \setKVdefault[foo]{mon bool}
  ```

  est équivalent à

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

- il est *déconseillé* d'exécuter plusieurs fois `\setKVdefault` pour le même ⟨*trousseau*⟩. Si cela est indispensable, il convient de redéfinir *toutes* les ⟨*clés*⟩ qui figuraient dans les précédents appels de `\setKVdefault`.

La macro `\setKV` Cette macro fonctionne selon les mêmes principes et limitations que `\setKVdefault`, sauf qu'elle redéfinit une ou plusieurs ⟨*clés*⟩ dont les ⟨*valeurs*⟩ ont précédemment été définies par `\setKVdefault`.
Si une ⟨*clé*⟩ n'a pas été prédéfinie par `\setKVdefault`, une erreur sera émise.

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 prédéfinie par \setKVdefault, 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*⟩.

```
\setKVdefault[foo]{nombre = 5 , lettres= AB \textit{CD} , mon bool}
a) \useKV[foo]{nombre}.\qquad  b) \useKV[foo]{lettres}.\qquad  c) \useKV[foo]{mon bool}.\par
\setKV[foo]{lettres = X Y Z \textbf{123} }
a) \useKV[foo]{nombre}.\qquad  b) \useKV[foo]{lettres}.\qquad  c) \useKV[foo]{mon bool}.
```

a) 5. b) AB *CD*. c) true.
a) 5. b) X Y Z **123**. c) true.

La macro \useKVdefault L'exécution de cette macro par \useKVdefault[⟨*trousseau*⟩] réinitialise toutes les ⟨*clés*⟩ du ⟨*trousseau*⟩ aux ⟨*valeurs*⟩ qui ont été définies lors de l'exécution \setKVdefault.

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 \showKV Cette commande écrit dans le fichier log la ⟨*valeur*⟩ assignée à une ⟨*clé*⟩ d'un ⟨*trousseau*⟩ :

\showKV[⟨*trousseau*⟩]{⟨*clé*⟩}

Si la ⟨*clé*⟩ n'est pas définie, « not defined » est affiché dans le fichier log.

3 Un exemple d'utilisation

Voici comment on pourrait programmer une macro qui affiche un cadre sur une ligne. 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 dédié s'il est faux ;
- left code et right code qui sont deux codes exécutés avant et après le cadre si le booléen est faux (par défaut : \hfill et \hfill pour un centrage sur la ligne) ;
- 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).

```
\setKVdefault[frame]{inline , left code = \hfill, right code = \hfill, sep = 3pt, width=0.5pt }
\newcommand\frametxt[2][]{%
  \setKV[frame]{#1}% lit les arguments optionnels
  \ifboolKV[frame]{inline}{}{\par\noindent\useKV[frame]{left code}}%
  \fboxsep=\useKV[frame]{sep}\relax
  \fboxrule=\useKV[frame]{width}\relax
  \fbox{#2}%
  \ifboolKV[frame]{inline}{}{\useKV[frame]{right code}\null\par}%
}
Un essai en ligne par défaut \frametxt{essai} puis un autre \frametxt[sep=5pt,width=2pt]{essai}
et un dernier \frametxt[sep=0.5pt]{essai}.

\useKVdefault[frame]% revenir au valeurs par défaut
Un essai centré \frametxt[inline = false]{essai centré}
un autre fer à gauche \frametxt[left code={}]{essai à gauche}
un dernier indenté \frametxt[left code=\hskip 5em, right code={}, sep=1pt, width=2pt]{essai indenté}
```

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

Un essai centré

essai centré

un autre fer à gauche

essai à gauche

un dernier indenté

essai indenté

4 Le code

Le code ci-dessous est l'exact verbatim du fichier `simplekv.tex` :

```
% !TeX encoding = ISO-8859-1
% Ce fichier contient le code commenté de l'extension "simplekv"
%
% IMPORTANT : pour que les commentaires s'affichent correctement,
%             ouvrir ce fichier avec l'encodage ISO-8859-1
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
\def\skvname                    {simplekv}                                     %
\def\skvver                       {0.1}                                        %
%                                                                              %
\def\skvdate                    {2017/08/08}                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% ----------------------------------------------------------------------
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3
% of this license or (at your option) any later version.
% The latest version of this license is in
%
% %     http://www.latex-project.org/lppl.txt
%
% and version 1.3 or later is part of all distributions of LaTeX
% version 2005/12/01 or later.
% ----------------------------------------------------------------------
% This work has the LPPL maintenance status 'maintained'.
%
% The Current Maintainer of this work is Christian Tellechea
% Copyright : Christian Tellechea 2017
% email: unbonpetit@netc.fr
%        Commentaires, suggestions et signalement de bugs bienvenus !
%        Comments, bug reports and suggestions are welcome.
% Copyright: Christian Tellechea 2017
% ----------------------------------------------------------------------
% L'extension simplekv est composée des 5 fichiers suivants :
%   - code               : simplekv           (.tex et .sty)
%   - manuel en français : simplekv-fr        (.tex et .pdf)
%   - fichier lisezmoi   : README
% ----------------------------------------------------------------------

\expandafter\edef\csname skv_restorecatcode\endcsname{\catcode`\noexpand\_=\the\catcode`\_\relax}
\catcode`\_11

%##############################################
%##############################################
% Cette macro est équivalente à 0 et sert notamment à stopper le développement
% de \romannumeral
\chardef\skv_stop 0

% Définition du quark, notamment inséré à la fin d'une liste pour en reconnaitre
% la fin et stopper la récursivité
\def\skv_quark{\skv_quark}
```

```
54
55 % Voici les macros habituelles de sélection d'arguments
56 \long\def\skv_first#1#2{#1}
57 \long\def\skv_second#1#2{#2}
58
59 % Voici la macro pour 1-développer ou 2-développer le 2e argument (le 1er étant
60 % dépouillé des accolades)
61 %     \skv_exparg{<a>}{<b>} devient <a>{<b>}
62 %     \skv_eearg{<a>}{<b>} devient <a>{**<b>}
63 \long\def\skv_exparg#1#2{\expandafter\skv_exparg_i\expandafter{#2}{#1}}%
64 \long\def\skv_eearg#1#2{\expandafter\expandafter\expandafter\skv_exparg_i\expandafter\expandafter\↙
       expandafter{#2}{#1}}%
65 \long\def\skv_exparg_i#1#2{#2{#1}}
66
67 % Et la macro pour 1-développer le 2e argument (le 1er et le 2e argument sont
68 % dépouillés des accolades)%
69 %     \skv_expafter{<a>}{<b>} devient <a><*b>
70 \long\def\skv_expafter#1#2{\expandafter\skv_expafter_i\expandafter{#2}{#1}}
71 \long\def\skv_expafter_i#1#2{#2#1}
72
73 % Enfin, la macro pour former le nom du 2e argument (le 1er est dépouillé des
74 % accolades)
75 %     \skv_argcsname{<a>}{<b>} devient <a>\<b>
76 \def\skv_argcsname#1#{\skv_argcsname_i{#1}}
77 \def\skv_argcsname_i#1#2{\skv_expafter{#1}{\csname#2\endcsname}}
78
79 \def\skv_eaddtomacro#1#2{\skv_exparg{\skv_exparg{\def#1}}{\expandafter#1#2}}
80
81 %##############################################
82 %################ macros de test ################
83 %##############################################
84 % Voici quelques macros à sélection d'arguments pour les tests
85 \def\skv_ifcsname#1{\ifcsname#1\endcsname\expandafter\skv_first\else\expandafter\skv_second\fi}
86 \long\def\skv_ifx#1{\ifx#1\expandafter\skv_first\else\expandafter\skv_second\fi}
87 \long\def\skv_ifempty#1{\skv_exparg\skv_ifx{\expandafter\relax\detokenize{#1}\relax}}
88
89 % Ces macros sont utiles pour \skv_removeextremespaces, qui retire les
90 % espaces extrêmes de son argument
91 %     Voir codes 320-324 (http://progtex.fr/wp-content/uploads/2014/09/code.txt)
92 %     et pages 339-343 de "Apprendre à programmer en TeX"
93 \long\def\skv_ifspacefirst#1{\expandafter\skv_ifspacefirst_i\detokenize{#10} \_nil}
94 \long\def\skv_ifspacefirst_i#1 #2\_nil{\skv_ifempty{#1}}
95 \expandafter\def\expandafter\skv_gobspace\space{}
96 \def\skv_removefirstspaces{\romannumeral\skv_removefirstspaces_i}
97 \long\def\skv_removefirstspaces_i#1{\skv_ifspacefirst{#1}{\expandafter\skv_removefirstspaces_i\expandafter{\↙
       skv_gobspace#1}}{\skv_stop#1}}
98 \begingroup
99   \catcode0 12
100  \long\gdef\skv_removelastspaces#1{\romannumeral\skv_removelastspaces_i#1^^00 ^^00\_nil}
101  \long\gdef\skv_removelastspaces_i#1 ^^00{\skv_removelastspaces_ii#1^^00}
102  \long\gdef\skv_removelastspaces_ii#1^^00#2\_nil{\skv_ifspacefirst{#2}{\skv_removelastspaces_i#1^^00 ^^00\↙
       _nil}{\skv_stop#1}}
103 \endgroup
104 \long\def\skv_removeextremespaces#1{%
105   \romannumeral\expandafter\expandafter\expandafter\skv_removelastspaces\expandafter\expandafter\expandafter
106   {\expandafter\expandafter\expandafter\skv_stop\skv_removefirstspaces{#1}}%
107 }
108
109 %##############################################
110 %############### système clé/valeur ##############
111 %##############################################
112 % Ceci est le booléen indiquant si la lecture de <clés>=<valeurs> définit les
113 % <clés> _par défaut_ ou qu'il s'agit d'une _redéfinition_ des <clés> déjà
114 % existantes
115 \newif\ifskv_default
116
```

```
% macros chapeau appelant la même macro avec le booléen préalablement défini
\def\setKVdefault{\skv_defaulttrue\skv_readKV}
\def\setKV{\skv_defaultfalse\skv_readKV}

% L'argument obligatoire #1 est le nom du <trousseau> et #2 est l'ensemble
% des <clés>=<valeurs>
\def\skv_readKV[#1]#2{%
  \skv_ifempty{#2}
% Si aucune <clés>=<valeurs> alors qu'on définit les <valeurs> par défaut,
% message d'erreur et on s'arrête là
    {\ifskv_default\errmessage{No key/val found, no default key/val defined}\fi}
% Sinon, initialiser à <vide> la macro \skv_[<trousseau>] uniquement si on
% créé les <valeurs> par défaut
    {\ifskv_default\skv_argcsname\let{skv_[#1]}\empty\fi
% Puis on passe aux choses sérieuses, on va lire un par un tous les éléments
% <clé>=<valeur> (contenus dans #2) en mettant le quark comme dernier couple
% pour montrer la fin de la liste
    \skv_readKV_i[#1]#2,\skv_quark,%
    }%
}

\def\skv_readKV_i[#1]#2,{%
% #2 est le premier couple "<clé>=<valeur>" de la liste qui reste à traiter :
% tout d'abord, on se débarrasse des espaces extrêmes
  \skv_eearg{\def\__temp}{\skv_removeextremespaces{#2}}%
% Si ce qui en résulte est égal au <quark>,
  \skv_ifx{\__temp\skv_quark}
% alors, on a fini et on ne fait rien (fin du processus)
    {}
% Sinon, si ce qui en résulte est vide (le couple "<clé>=<valeur>" était donc
% vide ou composé d'un espace)
    {\skv_ifx{\__temp\empty}
% On a fini et on ne fait rien (fin du processus)
      {}
% dans le cas contraire, on va isoler la <clé> et la <valeur> du couple lu en
% prenant soin de mettre à la fin "=<quark>" pour se prémunir du cas où
% "<clé>=<valeur>" ne contient que la "<clé>" et pas de signe "=", ce qui
% ferait planter la macro à arguments délimités
      {\expandafter\skv_find_kv\__temp=\skv_quark\_nil[#1]%
      }%
% Lorsque la <clé> et la <valeur> est trouvée et stockée, recommencer et aller
% lire le prochain couple "<clé>=<valeur>"
    \skv_readKV_i[#1]%
    }%
}

% Voici la macro à arguments délimités à qui on a transmis
%     <clé>=<valeur>=<quark> (si <clé>=<valeur> est l'élément lu)
% ou
%     <clé>=<quark> (si <clé> est seule)
% et qui va isoler la <clé> de la <valeur>.
\def\skv_find_kv#1=#2\_nil[#3]{%
% #1 est ce qui se trouve avant le _premier_ signe "=" et
% #2 est ce qui se trouve entre le premier signe "=" et le \_nil
% Pour la <clé>, pas de problème, c'est _obligatoirement_ ce qui est avant le
% signe "=", que ce signe soit présent dans le couple lu ou pas puisqu'on y a
% rajouté "=<quark>".
% On élimine les espaces extrêmes pour obtenir la <clé> définitive stockée dans
% \__key (il faut 2-développer \skv_removeextremespaces pour qu'elle donne son
% argument sans espace extrême)
  \edef\__key{\detokenize\expandafter\expandafter\expandafter{\skv_removeextremespaces{#1}}}%
% Pour la <valeur>, on lui ôte d'abord les espaces extrêmes
  \skv_eearg{\def\__val}{\skv_removeextremespaces{#2}}%
  \skv_ifx{\__val\skv_quark}
% Si elle est égale au <quark>, alors la <valeur> vaut "true"
    {\def\__val{true}}%
```

```
% Sinon, ôter "=<quark>" de la fin de l'argument #2, éliminer les espaces
% extrêmes de ce qui en résulte et stocker le tout dans \__val (tout ceci est
% effectué par la macro \skv_find_val <valeur>=<quark>)
    {\skv_find_val#2}%
% Si on lit les <clés>=<valeurs> par défaut,
  \ifskv_default
% assigner à la macro "\skv_[<trousseau>]_<clé>" la <valeur> trouvée
    \skv_argcsname\let{skv_[#3]_\detokenize\expandafter{\__key}}\__val
% Puis ajouter à la macro "\skv_[<trousseau>]", qui a été préalablement
% initialisée à <vide> :
%     \def\skv_[<trousseau>]_<clé>{<valeur>}
    \skv_argcsname\skv_eaddtomacro{skv_[#3]}%
      {\expandafter\def\csname skv_[#3]_\detokenize\expandafter{\__key}\expandafter\endcsname\expandafter{\↙
          __val}}%
% C'est selon ce hashage que sont enregistrés les couples <clé>/<valeur> : les
% <clés> sont contenues dans les noms des macros tandis que les <valeurs> sont
% les textes de remplacement de ces macros.
% C'est rapide et simple :
%    a)  pour trouver une <valeur> d'après sa <clé>, il suffit de développer la
%        macro \skv_[<trousseau>]_<clé>
%    b) pour redéfinir une <clé>, il suffit de redéfinir cette macro avec la
%       nouvelle <valeur>
%    c) il est facile de vérifier qu'une <clé> existe en vérifiant que la macro
%       associée est définie, la primitive \ifcsname le fait très bien
%    d) en revanche, on ne peut pas faire de recherche _inverse_ de façon
%       pratique : il est en effet plus difficile de trouver la (ou les) <clé>
%       contenant une <valeur> donnée, mais cette limitation n'a pas grande
%       importance ici (je ne sais pas si les autres systèmes de <clé>/<valeur>
%       sont programmés de telle sorte que cela soit simple...)
% Bref, la macro "\skv_[<trousseau>]" contient donc _toutes_ les définitions
% des macros définissant les <clés>/<valeurs> _par défaut_ et exécuter
% "\skv_[<trousseau>]" remet donc toutes les <clés> à leur <valeur> par défaut.
  \else
% Dans le cas où on _lit_ des nouvelles <valeurs> pour des <clés>
    \skv_ifcsname{skv_[#3]_\__key}
% Si la <clé> existe (ssi la macro "\skv_[<trousseau>]_<clé>" est définie),
% alors assigner la <valeur> à cette macro
      {\skv_argcsname\let{skv_[#3]_\__key}\__val}%
% Sinon, émettre un message d'erreur et ne rien faire de plus
      {\errmessage{Key "\__key" is not defined: nothing is modified}}%
  \fi
}

% Cette macro à qui on a transmis "<valeur>=<quark>" ne garde que <valeur>, en
% ôte les espaces extrêmes et stocke le résultat dans \__val
\def\skv_find_val#1=\skv_quark{\skv_eearg{\def\__val}{\skv_removeextremespaces{#1}}}

% Cette macro remet toutes les <clés> à leur <valeurs> par défaut en exécutant
% la macro "\skv_[<trousseau>]"
\def\useKVdefault[#1]{%
  \skv_ifcsname{skv_[#1]}
    {\csname skv_[#1]\endcsname}
% Si la macro "\skv_[<trousseau>]" n'existe pas, message d'erreur
    {\errmessage{Undefined set of keys "#1"}}%
}

% Cette macro donne la <valeur> correspondant à la <clé> contenue dans #2
\def\useKV[#1]#2{%
% Avec \romannumeral, la <valeur> sera obtenue après _2_ développements de
% \useKV[<trousseau>]{<clé>}
  \romannumeral\skv_ifempty{#2}
% Si la <clé> est vide, message d'erreur (il ne peut y avoir de <valeur>
% associée à une <clé> vide)
    {\skv_stop\errmessage{Key name missing}}
    {\skv_ifcsname{skv_[#1]_\skv_removeextremespaces{#2}}
% Si la macro "\skv_[<trousseau>]_<clé>" existe, 2-développer le \csname pour
```

```
% avoir la <valeur>
      {\expandafter\expandafter\expandafter\skv_stop\csname skv_[#1]_\skv_removeextremespaces{#2}\endcsname}
% Sinon, message d'erreur
      {\skv_stop\errmessage{Key "\skv_removeextremespaces{#2}" not defined}}%
    }%
}

% Voici une macro purement développable qui teste si #2 (la <clé> du <trousseau>
% #1) est égale à "true" ou à "false" et selon l'issue, exécute le 1er ou 2e
% argument qui suit (arguments appelés <vrai> et <faux>)
\def\ifboolKV[#1]#2{%
% Cette macro donnera un des 2 arguments <vrai> ou <faux> en _2_ développements
% grâce au \romannumeral
  \romannumeral\skv_ifempty{#2}
% Si la <clé> est vide, message d'erreur
    {\skv_stop\errmessage{Key name missing}\skv_second}
    {\skv_ifcsname{skv_[#1]_\skv_removeextremespaces{#2}}
% Si la <clé> débarrassée de ses espaces extrêmes existe, tester son contenu
      {\skv_eearg\ifboolKV_i{\csname skv_[#1]_\skv_removeextremespaces{#2}\endcsname}}
% Sinon, message d'erreur
      {\skv_stop\errmessage{Key "\skv_removeextremespaces{#2}" not defined}\skv_second}%
    }%
}

% Cette macro teste si #1, qui est une <valeur>, vaut "true" ou "false"
\def\ifboolKV_i#1{%
% Tester d'abord si elle vaut "true"
  \skv_ifargtrue{#1}
    {\expandafter\skv_stop\skv_first}
    {\skv_ifargfalse{#1}
% Puis si elle vaut "false"
      {\expandafter\skv_stop\skv_second}
% Si ni l'un ni l'autre, la <valeur> n'est pas un booléen acceptable
      {\skv_stop\errmessage{Value "#1" is not a valid boolean}\skv_second}%
    }%
}

% La macro \skv_ifargtrue{<argument>} teste de façon purement développable si
% <argument> vaut "true" ou "false".
% Pour cela, on transmet à \skv_ifargtrue_i l'argument "<argument>true" qui est
% délimité par \_nil
\def\skv_ifargtrue#1{\skv_ifargtrue_i#1true\_nil}
% Dans la macro \skv_ifargtrue_i, l'argument #1 est ce qui se trouve avant
% "true" dans "<argument>true" :
%     - s'il n'est pas vide, sélectionner l'argument <faux>
%     - s'il est vide, cela signifie que <argument> commence par "true" ; il est
%       donc de la forme "true<autre>"
%       L'argument #2 est ce qui se trouve après "true" dans "true<autre>true",
%       c'est donc "<autre>true".
%       Pour être sûr que <autre> est <vide>, on transmet "<autre>true" à
%       \skv_ifargtrue_ii qui teste si la réunion de ce qui est avant le
%       premier "true" et ce qui est après est <vide>
\def\skv_ifargtrue_i#1true#2\_nil{\skv_ifempty{#1}{\skv_ifargtrue_ii#2\_nil}\skv_second}
\def\skv_ifargtrue_ii#1true#2\_nil{\skv_ifempty{#1#2}}

% On procède de même pour tester "false"
\def\skv_ifargfalse#1{\skv_ifargfalse_i#1false\_nil}
\def\skv_ifargfalse_i#1false#2\_nil{\skv_ifempty{#1}{\skv_ifargfalse_ii#2\_nil}\skv_second}
\def\skv_ifargfalse_ii#1false#2\_nil{\skv_ifempty{#1#2}}

\def\showKV[#1]#2{%
% Ecrire dans le fichier log "Key [<trousseau>]<clé>="
  \immediate\write-1 {Key [#1]\skv_removeextremespaces{#2}=%
    \skv_ifcsname{skv_[#1]_\skv_removeextremespaces{#2}}
% si la <clé> est définie, prendre le \meaning de la macro correspondante
    {\expandafter\expandafter\expandafter\skv_show\expandafter
```

```
    \meaning\csname skv_[#1]_\skv_removeextremespaces{#2}\endcsname}
% Sinon, afficher
    {not defined}%
  }%
}
% Mange ce qui se trouve jusqu'à "->" dans le développement de \meaning
\def\skv_show#1->{}
\skv_restorecatcode
\endinput

Versions :
 ____________________________________________________________________________
| Version |    Date    | Changements                                         |
|----------------------------------------------------------------------------|
|   0.1   | 08/08/2017 | Première version                                    |
|----------------------------------------------------------------------------|
```