Le package tabularx*

David CARLISLE†

1999/01/07

Résumé

Ce package définit un nouvel environnement, `tabularx`, qui accepte les mêmes arguments que `tabular*`, mais modifie les largeurs de certaines colonnes, plutôt que l'espacement entre les colonnes, pour composer une table avec la largeur totale demandée. Les colonnes qui peuvent s'étirer sont marquées par le nouvel indicateur `X` dans l'argument-préambule.

Ce package utilise (automatiquement) le package `array`.

1 Introduction

Ce package définit une version de l'environnement `tabular` dans laquelle les largeurs de certaines colonnes sont calculées pour que la table ait une largeur spécifiée. Il semble qu'un tel environnement soit souvent demandé sur `comp.text.tex`.

`tabularx` `\begin{tabularx}{`⟨*width*⟩`}{`⟨*preamble*⟩`}`

Les arguments de `tabularx` sont essentiellement les mêmes que ceux de l'environnement `tabular*` classique. Cependant, plutôt que d'ajouter un espace entre les colonnes pour atteindre la largeur désirée, il ajuste la largeur de certaines colonnes. Les colonnes qui sont affectées par l'environnement `tabularx` doivent être marquées par la lettre `X` dans l'argument préambule. La spécification de colonne `X` sera convertie en `p{`⟨*dimension*⟩`}` une fois que la largeur correcte pour la colonne aura été calculée.

2 Exemples

La table suivante est composée avec
`\begin{tabularx}{250pt}{|c|X|c|X|} ....`

Entrée multicolonne !		TROIS	QUATRE
un	La largeur de cette colonne dépend de la largeur de la table.[1]	trois	La colonne quatre sera traitée comme la colonne deux, avec la même largeur.

Si nous changeons la première ligne en `\begin{tabularx}{300pt}{|c|X|c|X|}` nous obtenons :

*version v2.07, date 1999/01/07.

†traduction française du 3 Décembre 1999 par Jean-Pierre DRUCBERT <Jean-Pierre.Drucbert@onecert.fr>. Titre original « The tabularx package ».

[1] Vous pouvez maintenant utiliser `\footnote` dans l'environnement `tabularx` !

Entrée multicolonne !		TROIS	QUATRE
un	La largeur de cette colonne dépend de la largeur de la table.	trois	La colonne quatre sera traitée comme la colonne deux, avec la même largeur.

3 Différences entre `tabularx` et `tabular*`

Ces deux environnements acceptent les mêmes arguments, pour produire une table de la largeur spécifiée. Les principales différences entre elles sont :

- `tabularx` modifie les largeurs des *colonnes*, tandis que `tabular*` modifie les largeurs des *espaces* entre les colonnes ;
- les environnements `tabular` et `tabular*` peuvent être imbriqués sans restriction, mais si un environnement `tabularx` se trouve à l'intérieur d'un autre, alors celui à l'intérieur de l'autre *doit* être placé entre `{ }` ;
- le corps de l'environnement `tabularx` est utilisé en réalité comme argument d'une commande, et donc certaines constructions qui ne sont pas permises dans les arguments de commandes (comme `\verb`) ne peuvent pas être utilisées[2] ;
- `tabular*` utilise une possibilité primitive de TEX pour modifier l'espacement entre les colonnes d'un alignement. `tabularx` doit composer plusieurs fois la table dans sa recherche des meilleures largeurs de colonnes, et est donc bien plus lent. De plus, le fait que le corps soit expansé plusieurs fois peut mettre en défaut certaines constructions TEX.

4 Adaptation du comportement de `tabularx`

4.1 Sorties au terminal

`\tracingtabularx` Si la déclaration est faite, par exemple dans le préambule du document, alors tous les environnements `tabularx` ultérieurs afficheront des informations sur les largeurs de colonnes lors des essais pour trouver les largeurs correctes.

Comme alternative, on peut utiliser la déclaration de `\tracingtabularx`, soit pour les options `infoshow` ou `debugshow` qui peuvent être données, soit dans la commande `\usepackage` qui charge `tabularx`, ou comme option globale de la commande `\documentclass`.

4.2 Environnement utilisé pour composer les colonnes `X`

Par défaut, la spécification `X` est transformée en `p{`⟨*une valeur*⟩`}`. De telles colonnes étroites exigent souvent un format spécial, ce qui peut être obtenu en utilisant la syntaxe `>` du package `array.sty`. Ainsi par exemple vous pouvez spécifier `>{\small}X`. Un autre format utile dans les colonnes étroites est le « déchiqueté à droite » (ou « fer à gauche »), mais la commande `\raggedright` de LATEX redéfinit la commande `\\` d'une manière incompatible avec son utilisation

`\arraybackslash` dans les environnements `array` et `tabular`. Pour cette raison, ce package intro-

[2] en fait, depuis la version 1.02, `\verb` peut être utilisée, mais d'une part les espaces ne seront pas correctement représentés, d'autre part, il **faut** que les accolades soient appariées à l'intérieur de son argument, ce qui en limite vraiment l'intérêt

duit la commande `\arraybackslash`, qui peut être utilisée après une déclaration `\raggedright`, `\raggedleft` ou `\centering`. Donc un préambule de `tabularx` peut spécifier
`>{\raggedright\arraybackslash}X`.

`\newcolumntype` Ces spécifications de préambule peuvent évidemment être conservées en utilisant la commande `\newcolumntype` définie par le package `array.sty`. Donc nous pouvons dire
`\newcolumntype{Y}{>{\small\raggedright\arraybackslash}X}`
puis utiliser `Y` dans l'argument préambule de `tabularx`.

`\tabularxcolumn` Les colonnes `X` sont composées en utilisant le type `p` de colonne qui correspond à `\parbox[t]`. Vous pouvez vouloir qu'elles soient composées avec, par exemple, le type `m` de colonne, qui correspond à `\parbox[c]`. Il n'est pas possible de changer le type de colonne en utilisant la syntaxe `>`, donc une autre méthode est fournie. `\tabularxcolumn` doit être définie comme une commande à un seul argument, qui s'expanse en la spécification de préambule `tabular` que vous voulez faire correspondre à `X`. L'argument sera remplacé par la largeur calculée d'une colonne.

Par défaut, c'est `\newcommand{\tabularxcolumn}[1]{p{#1}}`. Nous pouvons donc changer ceci en une commande telle que :
`\renewcommand{\tabularxcolumn}[1]{>{\small}m{#1}}`

4.3 Largeurs des colonnes

Normalement toutes les colonnes `X` dans une même table auront finalement la même largeur ; cependant il est possible de forcer `tabularx` à leur donner des largeurs différentes. Un argument de préambule tel que
`{>{\hsize=.5\hsize}X>{\hsize=1.5\hsize}X}` spécifie deux colonnes, la seconde étant trois fois plus large que la première. Mais si vous voulez vous amuser (et vous risquer) à de telles choses il vous faudra suivre les deux règles suivantes :

- Assurez vous que la somme des largeurs de toutes les colonnes `X` reste inchangée. (Dans l'exemple précédent, les nouvelles largeurs faisaient un total de deux fois la largeur par défaut, comme deux colonnes `X` normales) ;
- N'utilisez pas d'entrées `\multicolumn` qui englobe une ou plusieurs colonnes `X`.

Comme la plupart des règles, il est possible de passer outre, si vous savez ce que vous faites.

4.4 Si l'algorithme échoue. . .

C'est peut-être par ce que les largeurs des colonnes « normales » de la table sont déjà plus grandes que la largeur totale demandée. `tabularx` refuse de composer des colonnes `X` de largeur négative, alors, dans ce cas, vous aurez un message d'alerte « X Columns too narrow (table too wide) »[3].

Dans ce cas, les colonnes `X` devront être composées avec une largeur de 1 em et donc la table elle-même sera plus large que la largeur totale demandée et donnée par l'argument d'environnement. Ce comportement du package peut être légèrement modifié, comme décrit dans le commentaire du code.

[3] « Colonnes X trop étroites (table trop large) »

5 Les Macros

```
⟨*package⟩
```

```
\DeclareOption{infoshow}{\AtEndOfPackage\tracingtabularx}
\DeclareOption{debugshow}{\AtEndOfPackage\tracingtabularx}
\ProcessOptions
```

Cela nécessite le package `array.sty`.

```
\RequirePackage{array}[1994/02/03]
```

D'abord, quelques registres, etc., dont nous avons besoin.

```
\newdimen\TX@col@width
\newdimen\TX@old@table
\newdimen\TX@old@col
\newdimen\TX@target
\newdimen\TX@delta
\newcount\TX@cols
\newif\ifTX@
```

Maintenant une astuce pour construire le corps d'un environnement dans un identificateur de registre, sans faire aucune expansion. Cela ne vérifie pas vraiment les environnements imbriqués, donc, si vous aviez besoin d'imbriquer un environnement **`tabularx`** dans un autre, l'environnement imbriqué à l'intérieur de l'autre doit être entouré par **`{ }`**.

`\tabularx` Avant la v1.06, cette macro prenait deux arguments, qui étaient sauvegardés dans des registres séparés, avant que le corps de la table ne soit sauvegardé par **`\TX@get@body`**. Malheureusement, cela désactivait l'argument optionnel **`[t]`**. Maintenant, on sauvegarde seulement et de façon séparée la spécification de largeur, puis on efface l'identificateur de registre **`\toks@`**. Finalement, on appelle **`\TX@get@body`** pour commencer à sauvegarder le corps de la table. **`{\ifnum0=`}\fi`** a été rajouté dans la v1.03 pour permettre à **`tabularx`** d'apparaître dans un **`\halign`**. [4]

Ce mecanisme de saisie du corps d'un environnement a l'inconvénient (partagé avec les environnements d'alignement de l'AMS) d'empêcher de faire des environnement d'extension comme

```
\newenvironment{foo}{\begin{tabularx}{XX}}{\end{tabularx}}
```

puisque le code cherche une chaîne littérale **`\end{tabularx}`** pour s'arrêter de scruter. Depuis la version 2.02, on peut éviter ce problème en utilisant **`\tabularx`** et **`\endtabularx`** directement dans la définition :

```
\newenvironment{foo}{\tabularx{XX}}{\endtabularx}
```

Maintenant, le scruteur cherche la fin de l'environnement courant (**`foo`** dans cet exemple). Il y a quelques limitations à cette utilisation, la principale étant que **`\endtabularx`** est le *premier* identificateur de la « fin du code » de l'environnement.

```
\def\tabularx#1{%
```

[4] Cela ajoute un niveau supplémentaire de groupement qui n'est pas vraiment nécessaire. À la place, je (D. Carlisle, NdT) pourrais utiliser **`\iffalse{\fi\ifnum0=`}\fi`** ici, et **`\ifnum0=`{}\fi`** en-dessous, cependant ici le code devrait être déplacé après la première ligne, à cause de la note du bas de la page 386 du TeXBook, et je ne sais pas si je devrais écrire du code si obscur puisque commenté dans une note de bas de page d'un appendice appelé « Dirty Tricks » !

Permet d'utiliser `\tabularx` `\endtabularx` (mais pas `\begin{tabularx}` `\end{tabularx}`) dans les définitions `\newenvironment`.

```
\edef\TX@{\@currenvir}%
  {\ifnum0=`}\fi
```

`\relax` a été ajouté dans la v1.05 pour que les identificateurs de longueur non-expansibles, comme `\textwidth`, ne génèrent pas d'espace supplémentaire et un débordement de boîte. `\relax` a été de nouveau supprimé dans la v4.09[5] remplacé par `\setlength` pour que vous puissiez utiliser une largeur de `(\textwidth-12pt)/2` si vous employez le package calc.

```
  \setlength\TX@target{#1}%
  \TX@typeout{Target width: #1 = \the\TX@target.}%
  \toks@{}\TX@get@body}
```

`\endtabularx` Ne fait vraiment pas grand chose...

```
\let\endtabularx\relax
```

`\TX@get@body` Place tous les identificateurs aussi loin que possible de `\end` dans un registre d'identificateur. Puis appelle `\TX@find@end` pour voir si nous sommes au `\end{tabularx}`.

```
\long\def\TX@get@body#1\end
  {\toks@\expandafter{\the\toks@#1}\TX@find@end}
```

`\TX@find@end` Si nous sommes au `\end{tabularx}`, appelle `\TX@endtabularx`, sinon ajoute `\end{...}` à l'identificateur et appelle une nouvelle fois `\TX@get@body`.

```
\def\TX@find@end#1{%
  \def\@tempa{#1}%
  \ifx\@tempa\TX@\expandafter\TX@endtabularx
  \else\toks@\expandafter
    {\the\toks@\end{#1}}\expandafter\TX@get@body\fi}
```

`\TX@` La chaîne `tabularx` est considérée comme une macro pour être testée avec `\ifx`.

```
\def\TX@{tabularx}
```

Maintenant, toutes les parties de spécification de table sont stockées dans les registres, nous pouvons commencer le travail de composition de la table.

L'algorithme de découverte des bonnes largeurs de colonne est le suivant. Tout d'abord, composition de la table avec chaque largeur de colonne `X` identique à la largeur de la table finale. En supposant qu'il y a au moins une colonne `X`, cela produira une table trop large. Puis, divise la largeur en excès par le nombre de colonnes `X` et réduit la largeur de colonne de cette valeur. Puis recompose la table. Si la table n'a pas encore la bonne largeur, une entrée `\multicolumn` doit « cacher » une des colonnes `X` et donc, il y a une colonne `X` de moins qui affecte la largeur de la table. Donc, nous réduisons de 1 le nombre de colonnes X et recommençons le processus.

`\TX@endtabularx` Bien que j'aie[6] essayé de construire un environnement `tabularx`, c'est en réalité une commande et tout le travail est fait par cette macro.

```
\def\TX@endtabularx{%
```

[5] Il s'agit d'une mise à jour majeure de la série tools et non de la version en cours de tabularx. NtD

[6] D. Carlisle, NdT

Définit la colonne `X` avec une version interne de la commande `\newcolumntype`. Les commandes `\expandafter` activent `\NC@newcol` pour considérer l'*expansion* de `\tabularxcolumn{\TX@col@width}` comme son argument. Ce sera la définition d'une colonne `X`, comme évoqué dans la section 4.

```
\expandafter\TX@newcol\expandafter{\tabularxcolumn{\TX@col@width}}%
```

Initialise la largeur de colonne et le nombre de colonnes `X`. Le nombre de colonnes `X` est fixé à un, ce qui signifie que le décompte initial sera trop haut de un, mais cette valeur est décrémentée avant son utilisation dans la boucle principale.

Depuis la v1.02, modifie la définition de `\verb`.

```
\let\verb\TX@verb
```

Depuis la v1.05, sauve les valeurs de tous les compteurs LaTeX, la liste `\cl@@ckpt` contenant le nom pour tous les compteurs LaTeX qui ont déjà été définis. Nous expansons `\setcounter` à ce moment-là, puisqu'il résulte de moins d'identificateurs stockés dans`\TX@ckpt`, mais la véritable ré-initialisation des compteurs a lieu quand `\TX@ckpt` est expansé après chaque passe d'essai. Réellement, depuis la v1.07, on utilise quelque chose d'équivalent à la définition originale de `\setcounter`, pour que `tabularx` soit compatible avec `calc.sty`.

```
\def\@elt##1{\global\value{##1}\the\value{##1}\relax}%
\edef\TX@ckpt{\cl@@ckpt}%
\let\@elt\relax
\TX@old@table\maxdimen
\TX@col@width\TX@target
\global\TX@cols\@ne
```

Sortie de certains en-têtes (à moins qu'ils ne soient désactivés).

```
\TX@typeout@
  {\@spaces Table Width\@spaces Column Width\@spaces X Columns}%
```

Pemier essai. Modifie la définition de `X` pour compter les colonnes `X`.

```
\TX@trial{\def\NC@rewrite@X{%
          \global\advance\TX@cols\@ne\NC@find p{\TX@col@width}}}%
```

Diminution répétitive de la largeur de colonne jusqu'à ce que la table ait la bonne largeur, ou stoppe la diminution sinon la colonne devient trop étroite. S'il n'y a pas d'entrée multicolonne, il n'y aura qu'un essai.

```
\loop
  \TX@arith
  \ifTX@
  \TX@trial{}%
\repeat
```

Une dernière fois, avec des messages d'alerte (voir Appendice D), utilise `tabular*` pour le mettre dans une boîte de dimension correcte, dans le cas ou l'algorithme ne réussit pas à trouver la bonne dimension.

Depuis la v1.04, sauvegarde localement l'argument de `\footnotetext` dans un identificateur de registre. Depuis la v1.06, `\toks@` contient la spécification de préambule et l'argument optionnel possible, ainsi que la corps de la table.

```
{\let\@footnotetext\TX@ftntext\let\@xfootnotenext\TX@xftntext
  \csname tabular*\expandafter\endcsname\expandafter\TX@target
    \the\toks@
  \csname endtabular*\endcsname}%
```

Maintenant, l'alignement est terminé et `}` a restauré la signification originale de `\@footnotetext` qui expanse le registre `\TX@ftn` qui exécutera une série de commandes
`\footnotetext[`⟨*num*⟩`]{`⟨*note*⟩`}`.
Nous avons besoin de faire attention à l'effacement du registre car nous pouvons être dans un `tabularx` imbriqué.

```
  \global\TX@ftn\expandafter{\expandafter}\the\TX@ftn
```

Maintenant, finit l'environnement `tabularx`. Il est à noter que nous avons besoin ici de `\end{tabularx}` puisque `\end{tabularx}` dans le fichier utilisateur n'est jamais expansé. Maintenant utilise `\TX@` plutôt que `tabularx`.

Nous avons également besoin de terminer le groupe commencé par `{\ifnum0=`}\fi` dans la macro `\tabularx`.

```
  \ifnum0=`{\fi}%
  \expandafter\end\expandafter{\TX@}}
```

`\TX@arith` Calcule la largeur de colonne pour l'essai suivant, fixant l'indicateur `\ifTX@` pour qu'il échoue si la boucle doit être interrompue.

```
\def\TX@arith{%
  \TX@false
  \ifdim\TX@old@table=\wd\@tempboxa
```

Si nous avons réduit la largeur de colonne, mais si la largeur de la table n'a pas été modifiée, nous arrêtons la boucle et composons la table (ce qui causera un alignement débordant) avec la valeur précédente de `\TX@col@width`.

```
    \TX@col@width\TX@old@col
    \TX@typeout@{Reached minimum width, backing up.}%
  \else
```

Autrement, calcule le dépassement de largeur de la table courante.

```
    \dimen@\wd\@tempboxa
    \advance\dimen@ -\TX@target
    \ifdim\dimen@<\TX@delta
```

Si le dépassement est inférieur à `\TX@delta`, stoppe. (`\TX@delta` doit être différent de zéro, sinon nous pouvons oublier la cible, à cause de l'erreur d'arrondi).

```
      \TX@typeout@{Reached target.}%
    \else
```

Diminue de 1 le nombre de colonnes `X` valides. (vérifiant que nous n'avons pas un 0, ce qui pourrait produire une future erreur). Puis divise la largeur en excès par le nombre de colonnes valides, et calcule la nouvelle largeur de colonne. Stocke temporairement cette valeur (multipliée par -1) dans `\dimen@`.

```
      \ifnum\TX@cols>\@ne
        \advance\TX@cols\m@ne
      \fi
      \divide\dimen@\TX@cols
      \advance\dimen@ -\TX@col@width
      \ifdim \dimen@ >\z@
```

Si la nouvelle largeur devait être trop étroite, la boucle est interrompue. Actuellement, trop étroite signifie moins de 0 pt !

Avant la v2.03, si la boucle s'arrêtait ici, les colonnes X étaient laissées avec la largeur de la passe précédente, mais cela peut construire une table beaucoup

trop large puisque les estimations initiales sont toujours trop grossières. Maintenant, force `\TX@error@width` à avoir la valeur par défaut de 1 em. Si vous voulez conserver l'ancien comportement, mettez
`\renewcommand\TX@error@width{\TX@col@width}`
dans un package chargé après `tabularx`.

```
      \PackageWarning{tabularx}%
         {X Columns too narrow (table too wide)\MessageBreak}%
      \TX@col@width\TX@error@width\relax
    \else
```

Autrement, sauvegarde les anciens réglages et fixe la nouvelle largeur de colonne. Fixe l'indicateur à vrai pour que la table soit fixée et que la boucle puisse de nouveau s'exécuter.

```
      \TX@old@col\TX@col@width
      \TX@old@table\wd\@tempboxa
      \TX@col@width-\dimen@
      \TX@true
    \fi
  \fi
\fi}
```

`\TX@error@width` Si le calcul de largeur donne un résultat négatif, on utilise cela, à la place.

```
\def\TX@error@width{1em}
```

`\TX@delta` Accepte une table qui est dans un `\hfuzz` avec une largeur correcte.

```
\TX@delta\hfuzz
```

Initialise la colonne `X`. Ici, la définition peut être vide puisqu'elle est fixée par chaque environnement `tabularx`.

```
\newcolumntype{X}{}
```

`\tabularxcolumn` La définition par défaut de `X` est `p{#1}`.

```
\def\tabularxcolumn#1{p{#1}}
```

`\TX@newcol` Une petite macro juste utilisée pour limiter le nombre de commandes `\expandafter` nécessaires.

```
\def\TX@newcol{\newcol@{X}[0]}
```

`\TX@trial` Fait un test.

```
\def\TX@trial#1{%
  \setbox\@tempboxa\hbox{%
```

Toute commande supplémentaire. Cela est utilisé au premier passage pour compter le nombre de colonnes `X`.

```
    #1\relax
```

Depuis la v1.04, `\footnotetext` englobe ses arguments. Nettoie aussi localement `\TX@vwarn` pour que le message d'alerte soit généré par la passe `finale` et n'apparaîsse pas au milieu de la table si `\tracingtabularx` est utilisé.

```
  \let\@footnotetext\TX@trial@ftn
  \let\TX@vwarn\@empty
```

N'imbrique pas les environnements `tabularx` pendant les passes d'essai. Cela peut gaspiller du temps et la fixation globale de `\TX@cols` peut arrêter l'algorithme.

```
  \expandafter\let\expandafter\tabularx\csname tabular*\endcsname
  \expandafter\let\expandafter\endtabularx\csname endtabular*\endcsname
```

Ajouté dans la v1.05 : désactive `\write` pendant une passe d'essai. Cette astuce provient du TEXBook.[7]

```
\def\write{\begingroup
  \def\let{\afterassignment\endgroup\toks@}%
     \afterassignment\let\count@}%
```

Désactive les messages d'alerte (*cf* appendice D). Également, previent leur activation intempestive, en imposant l'enregistrement des noms de paramètres.

```
 \hbadness\@M
 \hfuzz\maxdimen
 \let\hbadness\@tempcnta
 \let\hfuzz\@tempdima
```

Réalise la table et termine la « hbox ». Depuis la v1.06, `\toks@` contient la spécification de préambule et son argument optionnel possible, ainsi que le corps de la table.

```
 \expandafter\tabular\the\toks@
 \endtabular}%
```

Depuis la v1.05, ré-initialise tous les compteurs LATEX en exécutant `\TX@ckpt`.

```
\TX@ckpt
```

Imprime certaines statistiques. Ajout de `\TX@align` dans la v1.05, pour aligner les colonnes.

```
\TX@typeout@{\@spaces
   \expandafter\TX@align
      \the\wd\@tempboxa\space\space\space\space\space\@@
   \expandafter\TX@align
      \the\TX@col@width\space\space\space\space\space\@@
   \@spaces\the\TX@cols}}
```

`\TX@align` Macro ajoutée dans la v1.05 pour améliorer l'impression des informations de traçage.

```
\def\TX@align#1.#2#3#4#5#6#7#8#9\@@{%
  \ifnum#1<10 \space\fi
  \ifnum#1<100 \space\fi
  \ifnum#1<\@m\space\fi
  \ifnum#1<\@M\space\fi
  #1.#2#3#4#5#6#7#8\space\space}
```

`\arraybackslash` Rupture \\.

```
\def\arraybackslash{\let\\\@arraycr}
```

`\tracingtabularx` Imprime des statistiques pour les largeurs de colonne et de table.

```
\def\tracingtabularx{%
  \def\TX@typeout{\PackageWarningNoLine{tabularx}}%
  \def\TX@typeout@##1{\typeout{(tabularx) ##1}}}
```

`\TX@typeout` Par défaut, est silencieux.

```
\let\TX@typeout\@gobble
\let\TX@typeout@\@gobble
```

`\TX@ftn` Un identificateur de registre pour le texte des notes de bas de page.

```
\newtoks\TX@ftn
```

[7] L'astuce du TEXBook ne fonctionne pas correctement, donc cela a été modifié dans la v2.05

\TX@ftntext \TX@xftntext À l'intérieur, sauvegarde juste le texte de note de bas de page dans un identificateur de registre.

```
\long\def\TX@ftntext#1{%
  \edef\@tempa{\the\TX@ftn\noexpand\footnotetext
                    [\the\csname c@\@mpfn\endcsname]}%
  \global\TX@ftn\expandafter{\@tempa{#1}}}%
\long\def\TX@xftntext[#1]#2{%
  \global\TX@ftn\expandafter{\the\TX@ftn\footnotetext[#1]{#2}}}
```

\TX@trial@ftn Dans les passes d'essai, englobe les textes des notes de bas de page.

```
\long\def\TX@trial@ftn#1{}
```

Cette dernière section a été ajouté dans la Version 1.02. Les versions précédentes indiquaient que `\verb` ne fonctionnait pas à l'intérieur de `tabularx`, mais ça n'empêchait pas les gens de l'utiliser ! Cela entraîne LaTeX dans une erreur irrécupérable, avec un message d'erreur qui ne mentionne pas la cause de l'erreur. Le `\verb` « du pauvre » (et `\verb*`) défini ici, est basé sur les indications de la page 382 du TeXBook. Comme il est expliqué, faire un verbatim de cette façon signifie que les espaces ne sont pas traités correctement et donc que `\verb*` serait bien inutile ; cependant, je considère que cette section du code est une correction d'erreur, plutôt qu'une véritable implémentation de verbatim.

Le mécanisme est totalement global et toute macro qui veut autoriser l'utilisation d'une forme de `\verb` à l'intérieur de son argument peut être
`\let\verb=\TX@verb` (tout en s'assurant de restaurer la vraie définition ensuite !).

`\verb` et `\verb*` ont les restrictions suivantes :

1. Les espaces dans l'argument ne sont pas lus tels que, mais peuvent être ignorés, en accord avec les règles habituelles de TeX ;
2. les espaces doivent être ajoutés en sortie après le contrôle des mots, même s'ils sont absents en entrée ;
3. sauf si l'argument est un espace isolé, tout espace traînant, s'il est dans l'argument original, ou ajouté comme en (2), sera omit ;
4. l'argument ne doit pas finir par un `\`, puisque `\verb|\|` n'est pas permis, et, de toute façon, à cause de (3), `\verb|\ |` donne `\` ;
5. l'argument doit être équilibré par `{` et `}`. Donc, `\verb|{|` n'est pas permis ;
6. un caractère de commentaire comme `%` n'apparaîtra pas tel quel. Il agira comme d'habitude, commentant le reste de la ligne en entrée !
7. les combinaisons `?‘` et `!‘` appaîtront comme ¿ et ¡ si la fonte `cmtt` est utilisée.

\TX@verb La définition interne de `\verb`. Les espaces seront remplacés par `~`, pour que dans la forme « étoile », `\let ~` soit ␣, que nous obtenons par `\uppercase{*}`. L'utilisation de `{\ifnum0=‘}\fi` plutôt que `\bgroup` permet que `&` apparaîsse dans l'argument.

```
{\uccode‘\*=‘\ %
\uppercase{\gdef\TX@verb{%
  \leavevmode\null\TX@vwarn
  {\ifnum0=‘}\fi\ttfamily\let\\\ignorespaces
  \@ifstar{\let~*\TX@vb}{\TX@vb}}}}
```

\TX@vb Donne le « presque verbatim » par `\meaning`. « `!` » est ajouté au début du texte fourni, pour s'assurer que l'argument entier ne consiste pas en un groupe `{ }` isolé. TeX enlèvera les accolades extérieures d'un tel groupe. « `!` » sera enlevé plus tard.

À l'origine, je suivais KNUTH et j'avais `\def\@tempa{##1}`, cependant cela ne permettait pas à `#` d'apparaître dans l'argument. Donc, dans la v1.04, j'ai changé ça et utilisé un identificateur de registre et `\edef`. Cela permet à `#` d'apparaître mais chacun apparaît deux fois !, donc plus tard je fais passer une boucle remplaçant `##` par `#`.

```
\def\TX@vb#1{\def\@tempa##1#1{\toks@{##1}\edef\@tempa{\the\toks@}%
    \expandafter\TX@v\meaning\@tempa\\ \\\ifnum0=`{\fi}}\@tempa!}
```

`\TX@v` Enlève le segment initial de `\meaning`, incluant « `!` » ajouté plus tôt.

```
\def\TX@v#1!{\afterassignment\TX@vfirst\let\@tempa= }
```

Comme expliqué ci-dessus, nous allons remplacer les paires `##` par `#`. Pour faire cela, nous avons besoin d'un identificateur `#` non spécifique. Fait un `*` dans un identificateur de paramètre pour que nous puissions définir des macro avec des arguments. La signification normale sera restaurée par le dernier `\endgroup`.

```
\begingroup
\catcode`\*=\catcode`\#
\catcode`\#=12
```

`\TX@vfirst` Comme un cas spécial, empêche que le premier caractère soit éliminé. Alors `\verb*| |` produit ␣. Puis, appelle `\TX@v@`. Cela est légèrement astucieux depuis la v1.04, puisque je me suis assuré qu'un vrai `#` plutôt qu'une commande `\let` to `#` est pris en compte si le premier caractère est `#`.

```
\gdef\TX@vfirst{%
  \if\@tempa#%
    \def\@tempb{\TX@v@#}%
  \else
    \let\@tempb\TX@v@
    \if\@tempa\space~\else\@tempa\fi
  \fi
  \@tempb}
```

`\TX@v@` Boucle à travers `\meaning`, remplaçant tous les espaces par `~`. Si le dernier caractère est un espace, il sera éliminé, puisque `\verb*|\LaTeX|` produit `\LaTeX` et non `\LaTeX␣`. Les identificateurs ré-écrits sont ensuite traités pour remplacer les paires `##`.

```
\gdef\TX@v@*1 *2{%
  \TX@v@hash*1##\relax\if*2\\\else~\expandafter\TX@v@\fi*2}
```

`\TX@v@hash` La boucle interne, remplaçant `##` par `#`.

```
\gdef\TX@v@hash*1##*2{*1\ifx*2\relax\else#\expandafter\TX@v@hash\fi*2}
```

Comme promis, nous restaurons la signification normale de `#` et `*`.

```
\endgroup
```

`\TX@vwarn` Avertit l'utilisateur la première fois que `\verb` est utilisé.

```
\def\TX@vwarn{%
  \@warning{\noexpand\verb may be unreliable inside tabularx}%
  \global\let\TX@vwarn\@empty}
```

```
</package>
```