Le package `tnscom` : outils communs à la suite `tns`

Code source disponible sur `https://github.com/typensee-latex/tnscom.git`.

Version `0.4.0-beta` développée et testée sur Mac OS X.

Christophe BAL

2020-08-05

Table des matières

I. Introduction

Ce package contient quelques macros utiles à différents packages de la série `tns` qui a pour modeste ambition de rendre un peu plus sémantique l'écriture de document avec LaTeX. Une partie des macros sont pour les coulisses et d'autres directement pour des mises en forme communes à différents packages.

II. Packages utilisés

La roue ayant déjà été inventée, le package `tnscom` utilise les packages suivants sans aucun scrupule.

- `amssymb`
- `nicematrix`
- `relsize`
- `siunitx`
- `tikz`
- `xstring`

III. A propos des macros standards redéfinies

Certaines macros comme `\frac` sont un peu revues par `tnscom`. Dans ce cas, les versions standard restent accessibles en utilisant le préfixe `std` ce qui donne ici la macro `\stdfrac`.

IV. Travailler avec des unités S.I.

L'excellent package `siunitx` étant chargé par `tnscom`, il devient facile d'obtenir les choses suivantes de façon sémantique. Indiquons que **les conventions d'écriture sont françaises dès lors que vous aurez chargé `babel` avec l'option `french`**[1] comme c'est le cas pour cette documentation. Ceci permet par exemple de taper `3.6` pour obtenir 3,6 ci-dessous.

```
$\ang{180} = \SI{\pi}{\radian}$

$\SI{1}{m.s^{-1}} = \SI{3.6}{km.h^{-1}}$
```

$180^\circ = \pi\,\mathrm{rad}$
$1\,\mathrm{m\,s^{-1}} = 3{,}6\,\mathrm{km\,h^{-1}}$

Dans l'exemple suivant, notez que les nombres à quatre chiffres comme 1234 sont écrits sans espace contrairement à ceux en ayant au moins cinq comme 12 345 car c'est l'usage typographique en France. Notez aussi les facilités données via la saisie de `*` et `e` dans l'argument de `\num` *(c'est comme si l'on tapait sur une calculatrice)*, ainsi que la possibilité d'utiliser des espaces pour améliorer la saisie des nombres dans le code LaTeX.

```
$\num{123*1000} = \num{123000}$

$\num{123e3} = \num{123 000}$
```

$123 \times 1000 = 123\,000$
$123 \times 10^3 = 123\,000$

Remarque. Pour les curieux, les réglages utilisés pour le moment sont les suivants.

1. Ceci est un petit réglage simple à ajouter mais que `siunitx` ne propose par défaut. C'est un peu dommage.

```
\sisetup{
    list-final-separator  = {et},
    list-pair-separator   = {et},
    range-phrase          = {à},
    input-product         = {*},
    output-decimal-marker = {,},
    group-minimum-digits  = 5
}
```

V. Deux séparateurs d'arguments par défaut

La macro `\tnsmathsep` définit le séparateur d'arguments de premier niveau et `\tnsmathsubsep` celui des arguments de deuxième niveau. Cette documentation utilisant l'option `french` de `babel`, la valeur de `\tnsmathsep` est ; et celle de `\tnsmathsubsep` est , . Sans ce choix, les valeurs de `\tnsmathsep` et `\tnsmathsubsep` seront , et ; respectivement à l'anglaise.

VI. Quelques modifications générales

1. Espace et point-virgule avec l'option `french` de `babel`

Seulement si vous utilisez `babel` avec l'option `french`, comme c'est le cas dans cette documentation alors vous verrez le même espacement autour du point-virgule en mode mathématique comme dans $A(x\,;y)$. Que c'est beau !

2. Espace et fractions

Quand on utilise `\frac` ou `\dfrac` de petits espaces sont automatiquement ajoutés pour éviter d'avoir des traits de fraction trop petits. Le comportement par défaut se retrouve en utilisant les macros `\stdfrac` et `\stddfrac`. Voici un exemple.

```
$\frac{2}{3}  = \stdfrac{2}{3}$  ,
$\dfrac{2}{3} = \stddfrac{2}{3}$
```

$\frac{2}{3} = \stdfrac{2}{3}$, $\dfrac{2}{3} = \stddfrac{2}{3}$

3. Espace et racines n-ièmes d'un réel

`\sqrt` a été redéfinie pour ajouter un peu d'espaces sur la droite sous le radical avec aussi un meilleur placement de l'exposant pour l'éloigner du radical. Le comportement par défaut se retrouve en utilisant la macro `\stdsqrt`. Voici ce que cela donne.

```
$\sqrt{2}   = \stdsqrt{2}$   ,
$\sqrt{x_2} = \stdsqrt{x_2}$

$\sqrt[n]{45} = \stdsqrt[n]{45}$ ,
$\sqrt[p]{7}  = \stdsqrt[p]{7}$
```

$\sqrt{2} = \stdsqrt{2}$, $\sqrt{x_2} = \stdsqrt{x_2}$
$\sqrt[n]{45} = \stdsqrt[n]{45}$, $\sqrt[p]{7} = \stdsqrt[p]{7}$

4. Une variable « symbolique »

Dans certains contextes, comme celui des opérateurs fonctionnels, il peut être utile d'indiquer un argument symboliquement via • par exemple sans faire référence précisément à une ou des variables nommées. Voici un exemple d'utilisation où `symvar` est pour `sym`-bolic `var`-iable soit *« variable symbolique »* en anglais.

```
 $| \symvar |$ ou
 $\frac{d\symvar}{dx}
= \frac{d}{dx} \symvar$
```

$|\bullet|$ ou $\frac{d\bullet}{dx} = \frac{d}{dx}\bullet$

Si besoin, vous disposez d'autres symboles via l'argument optionnel de `\symvar` qui vaut `1` par défaut. Voici tous les symboles disponibles.

```
$\symvar[1]$
$\symvar[2]$
$\symvar[3]$
```

$\bullet \circ \blacksquare$

Remarque. Comme les symboles sont juste des caractères au sens LaTeX, il faudra si besoin gérer les espaces autour via par exemple des `\kern` pour obtenir des choses comme $|\bullet|$ et $d\circ$ qui ont été tapées `$|\kern.2ex\symvar\kern.2ex|$` et `$d\kern.25ex\symvar[2]$`.

VII. Pour les mathématiques

1. Intervalles « généralisés »

Il est possible définir facilement des sortes d'intervalles « généralisés ».

Exemple 1 – Mode extensible

Il est assez facile de définir une macro ayant le comportement suivant.

```
$\strangeset{a}{\dfrac{b}{c}}$
```

$\left\{a :: \dfrac{b}{c}\right)$

La macro `\strangeset` a été définie comme suit via `\tns@generic@interval@ext`.

```
\newcommand\strangeset[2]{%
    \tns@generic@interval@ext{\{} % 1e délimiteur
                             {#1} % 1e élément
                             {::} % Séparateur entre les deux éléments
                             {#2} % 2e élément
                             {)}  % 2e délimiteur
}
```

Exemple 2 – Mode semi-extensible

Le mode semi-extensible correspond à des délimiteurs un peu plus grand qu'en mode non extensible comme le montre l'exemple ci-après.

```
$\myinter{a}{\dfrac{b}{c}}$ ou
$\{ a :: \dfrac{b}{c} )$
```

$\left\{a :: \dfrac{b}{c}\right)$ ou $\{a :: \dfrac{b}{c})$

Il suffit d'utiliser `\tns@generic@interval@semi@ext` au lieu de `\tns@generic@interval@ext`. Voici le code utilisé.

```
\newcommand\myinter[2]{%
    \tns@generic@interval@semi@ext{\{}%
                                  {#1}{::}{#2}%
                                  {)}%
}
```

2. Décorer un opérateur

Il est facile d'obtenir l'effet suivant via la macro `\tns@over@math@symbol`.

```
$1 \eqtxt un$
```

$1 \overset{\text{texte}}{=} un$

Le code qui a été utilisé est le suivant où l'on fournit en 1er le texte décoratif et en 2e l'opérateur.

```
\newcommand\eqtxt{\tns@over@math@symbol{texte}{=}}
```

3. Produit en croix

Comme ce type de calculs apparait très souvent dans divers domaines, une macro privée est chargée de ce tout petit travail de mise en forme.

```
\makeatletter
$\tns@prop@prod{\cdot} % Opérateur
                {1}{2}  % Les éléments
                {3}{4}$ % du tableau
\makeatother
```

$1 \cdot 4 - 3 \cdot 2$

4. Déterminant 2D et critère de colinéarité

Des macros privées permettent, suivant le contexte, d'écrire le critère de colinéarité ou bien le calcul d'un déterminant 2D avec différentes mises en forme possibles.

Exemple 1 – La totale en version courbée

La macro `\tns@det@plane@deco` est facile d'utilisation comme le montre l'exemple suivant.

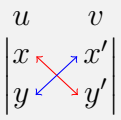
```
\makeatletter
$\tns@det@plane@deco@loop%
     {vec}     % Vecteurs visibles
     {u}       % 1e vecteur 2D
     {x}{y}    % Coord. du 1e vecteur
     {v}       % 2e vecteur 2D
     {x'}{y'}$ % Coord. du 2e vecteur
\makeatother
```

$$\begin{matrix} u & v \\ \begin{vmatrix} x & x' \\ y & y' \end{vmatrix} \end{matrix}$$

Exemple 2 – La totale en version produits en croix

Une autre mise en forme avec les explications est disponible.

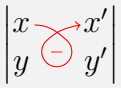
```
\makeatletter
$\tns@det@plane@deco@cross%
     {vec}      % Vecteurs visibles
     {u}        % 1e vecteur 2D
     {x}{y}     % Coord. du 1e vecteur
     {v}        % 2e vecteur 2D
     {x'}{y'}$ % Coord. du 2e vecteur
\makeatother
```



Exemple 3 – Décoration mais sans vecteur

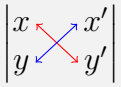
En choisissant `novec` au lieu de `vec`, les vecteurs ne seront pas affichés.

```
\makeatletter
$\tns@det@plane@deco@loop{novec}    %
                         {u}{x}{y}  %
                         {v}{x'}{y'}$
\makeatother
```



Ceci permet de produire un calcul de type produits en croix comme ci-dessous.

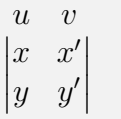
```
\makeatletter
$\tns@det@plane@deco@cross{novec}    %
                          {u}{x}{y}  %
                          {v}{x'}{y'}$
\makeatother
```



Exemple 4 – Sans décoration mais avec les vecteurs

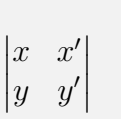
En utilisant `\tns@det@plane@no@deco`, la boucle fléchée ne sera pas imprimée. Voici une 1re utilisation possible.

```
\makeatletter
$\tns@det@plane@no@deco{vec}      %
                       {u}{x}{y}  %
                       {v}{x'}{y'}$
\makeatother
```



Exemple 5 – Sans décoration ni vecteur

```
\makeatletter
$\tns@det@plane@no@deco{novec}    %
                       {u}{x}{y}  %
                       {v}{x'}{y'}$
\makeatother
```



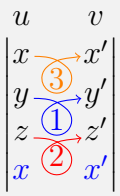
5. Produit vectoriel expliqué

Une macro privée permet d'écrire un calcul de produit vectoriel [2].

Exemple 1 – Avec des boucles explicatives

La macro `\tns@cross@prod@deco` est facile d'utilisation comme le montre l'exemple suivant.

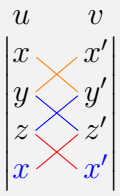
```
\makeatletter
$\tns@cross@prod@deco@loop%
     {vec}          % Vecteurs visibles
     {u}            % 1e vecteur 2D
     {x}{y}{z}      % Coord. du 1e vecteur
     {v}            % 2e vecteur 2D
     {x'}{y'}{z'}$ % Coord. du 2e vecteur
\makeatother
```



Exemple 2 – Avec des croix explicatives

Une autre mise en forme avec les explications est disponible.

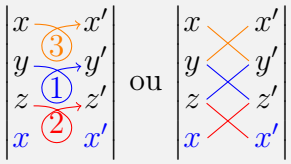
```
\makeatletter
$\tns@cross@prod@deco@cross%
     {vec}          %
     {u}{x}{y}{z}   %
     {v}{x'}{y'}{z'}$
\makeatother
```



Exemple 3 – Décoration mais sans vecteur

En choisissant `novec` au lieu de `vec`, les vecteurs ne seront pas affichés.

```
\makeatletter
$\tns@cross@prod@deco@loop%
     {novec}        %
     {u}{x}{y}{z}   %
     {v}{x'}{y'}{z'}$
ou
$\tns@cross@prod@deco@cross%
     {novec}        %
     {u}{x}{y}{z}   %
     {v}{x'}{y'}{z'}$
\makeatother
```



Exemple 4 – Sans décoration mais avec les vecteurs

En utilisant `\tns@cross@prod@no@deco`, la boucle fléchée ne sera pas imprimée. Voici une 1re utilisation possible.

2. Bien qu'utilisée juste par `tnsgeo`, cette fonctionnalité a été placée dans `tnscom` car son codage a beaucoup de points communs avec celui des déterminants 2D et du critère de colinéarité. Ceci facilite donc la maintenance.

```
\makeatletter
$\tns@cross@prod@no@deco%
     {vec}          %
     {u}{x}{y}{z}   %
     {v}{x'}{y'}{z'}$
\makeatother
```

$$\begin{vmatrix} u & v \\ x & x' \\ y & y' \\ z & z' \\ x & x' \end{vmatrix}$$

Exemple 5 – Sans décoration ni vecteur

```
\makeatletter
$\tns@cross@prod@no@deco%
     {novec}        %
     {u}{x}{y}{z}   %
     {v}{x'}{y'}{z'}$
\makeatother
```

$$\begin{vmatrix} x & x' \\ y & y' \\ z & z' \\ x & x' \end{vmatrix}$$

VIII. En coulisse...

Cette section présente les macros dites *« privées »* qui sont proposées par `tnscom`. Toutes ces macros ont des noms commençant par `\tns@` et aucune version étoilée n'est proposée [3].

1. Nouvelle macro modifiant les arguments en amont de l'application d'une ancienne macro

Comme cette fonctionnalité est utilisée par plusieurs packages de la suite `tns`, une mini macro permet de faciliter la définition de ce type de nouvelle macro. Par exemple, ci-dessous la 2e macro ajoute juste une mise en forme particulière aux deux arguments juste avant d'appliquer la 1re macro.

```
\twoargs{A}{B} : \newtwoargs{A}{B}
```

A, B : **([A])**, **([B])**

Ceci est géré facilement via la macro `\tns@apply@macro@two@args` comme suit en fournissant comme 1er argument la macro cible et pour 2e celle qui va modifier en amont les arguments.

```
\newcommand\twoargs[2]{#1, #2}
\newcommand\modify[1]{\textbf{([#1])}}

\def\newtwoargs{\tns@apply@macro@two@args\twoargs\modify}
```

Remarque. En interne la macro `\tns@apply@macro@two@args` est définie avec quatre arguments. En fait ci-dessus nous utilisons la machinerie LaTeX qui va manger les deux arguments manquants lors de l'utilisation de `\newtwoargs`.

2. Macro avec un « multi-argument » à « envelopper »

La suite `tns` propose la possibilité d'avoir des macros avec un nombre variables d'arguments. Pour ce type de macros, le choix a été fait de passer via unique argument au sens LaTeX mais contenant

3. « Explicite » est mieux que « implicite ».

des *« sous-arguemnts »* séparés par des barres verticales *(on dit* « pipe » *en anglais)*. Nous parlerons de « multi-argument ».

Voici un exemple d'utilisation possible *(la fonctionnalité ci-dessous est en fait disponible via la macro* `\coord` *du package* `tnsgeo` *disponible sur* `https://github.com/typensee-latex/tnsgeo.git` *).*

```
$\verticalcoord{1}$                ,
$\verticalcoord{1 | 2}$            ,
$\verticalcoord{1 | 2 | 3}$        ,
$\verticalcoord{1 | 2 | 3 | 4}$    ,
$\verticalcoord{1 | 2 | 3 | 4 | 5}$ ...
```

$$\begin{bmatrix}1\end{bmatrix}\ ,\ \begin{bmatrix}1\\2\end{bmatrix}\ ,\ \begin{bmatrix}1\\2\\3\end{bmatrix}\ ,\ \begin{bmatrix}1\\2\\3\\4\end{bmatrix}\ ,\ \begin{bmatrix}1\\2\\3\\4\\5\end{bmatrix}\ \ ...$$

La macro `\verticalcoord` a été définie comme suit via la macro privée `\tns@multi@args` en utilisant l'environnement `bmatrix` vient du très pratique package `nicematrix` qu'utilise déjà `tnscom`.

```
\newcommand\verticalcoord[1]{%
    \tns@multi@args{|}{#1}           % Séparateur et argument au sens LaTeX
                   {\begin{bmatrix}} % Matériel avant
                   {\\}              % Ce qui remplace le séparateur
                   {\end{bmatrix}}   % Matériel après
}
```

3. Appliquer une macro sur chaque partie d'un « multi-argument »

Exemple 1 – Appliquer partout

Il est possible d'appliquer une macro chaque partie d'un mutli-argument comme ci-dessous.

```
\multiapply{1 | 2 | 3}
```

((1)) ((2)) ((3))

La macro `\multiapply` a été définie comme suit. L'exemple 2 un peu plus bas explique à quoi sert le 1er argument de `\decoone` non utilisé ici.

```
\newcommand\decoone[2]{ ((#2)) }

\newcommand\multiapply[1]{%
    \tns@multi@apply@each{\decoone}{#1}
}
```

Exemple 2 – Choisir où appliquer

La macro `\tns@multi@apply@each` donne accès au numéro de l'argument à traiter via le 1er argument non utilisé dans l'exemple précédent. Ceci rend possible de traiter à part le tout premier argument comme ci-dessous.

```
\multiapplyafterone{1 | 2 | 3}
```

1 «2» «3»

La macro `\multiapplyafterone` a été définie comme suit.

```
\newcommand\decofirstandothers[2]{%
    \ifnum#1=1
        \fbox{#2} %
    \else%
        <<#2>> %
    \fi%
}

\newcommand\multiapplyafterone[1]{%
    \tns@multi@apply@each{\decofirstandothers}{#1}
}
```

4. Appliquer une macro sur chaque couple de parties successives d'un « multi-argument »

Exemple 1 – Appliquer partout

Il est possible d'appliquer une macro sur chaque couple de parties successives partie d'un mutli-argument comme ci-dessous.

`\multiapplycouple{1 | 2 | 3}`	(1-2)(2-3)

La macro `\multiapplycouple` a été définie comme suit.

```
\newcommand\decocouple[3]{%
    (#2-#3)%
}

\newcommand\multiapplycouple[1]{%
    \tns@multi@apply@couple{\decocouple}{#1}
}
```

Exemple 2 – Choisir où appliquer

Le compteur `tns@multi@apply@couple@position` permet de connaître le numéro du couple à traiter. Ceci permet de différencier par exemple le traitement du tout premier couple comme dans l'exemple suivant *(c'est pour cela que le compteur a été créé)*.

`\multiapplycoupleafterone{1 | 2 | 3}`	1-2(2-3)

La macro `\multiapplycoupleafterone` a été définie comme suit.

```
\newcommand\decocoupleafterfirst[3]{%
    \ifnum#1=1
        \fbox{#2-#3}%
    \else%
        (#2-#3)%
    \fi
}

\newcommand\multiapplycoupleafterone[1]{%
    \tns@multi@apply@couple{\decocoupleafterfirst}{#1}
}
```

IX. Historique

Nous ne donnons ici qu'un très bref historique récent [4] de `tnscom` à destination de l'utilisateur principalement. Tous les changements sont disponibles uniquement en anglais dans le dossier `change-log` : voir le code source de `tnscom` sur `github`.

2020-08-05 Nouvelle version mineure `0.4.0-beta`.

- **Multi-argument.**
  - Ajout de `\tns@multi@apply@each`.
  - Ajout de `\tns@multi@apply@couple`.

2020-07-30 Nouvelle version mineure `0.3.0-beta`.

- **Produit vectoriel :** intégration de cette fonctionnalité propre à `tnsgeo` pour faciliter la maintenance.
- **Déterminant 2D :** l'API est devenue similaire à celle pour le produit vectoriel.

2020-07-29 Nouvelle version mineure `0.2.0-beta`.

- **Déterminant 2D :** ajout de l'explication du calcul à la sauce produit en croix.
- **Symboles :** la macro `\symvar*` a été supprimée. À la place on fera appel à l'argument optionnel de `\symvar`.
- **Nouvelle dépendance :** le package `siunitx` est chargé par défaut.

2020-07-15 Nouvelle version mineure `0.1.0-beta`.

- **Symboles :** les nouvelles macros `\symvar` et `\symvar*` produisent un disque plein et un carré plein permettant par exemple d'indiquer symboliquement une ou des variables.

2020-07-10 Première version `0.0.0-beta`.

4. On ne va pas au-delà de un an depuis la dernière version.

X. Toutes les fiches techniques

1. Deux séparateurs d'arguments par défaut

```
\tnsmathsep{}
\tnsmathsubsep{}
```

2. Quelques modifications générales

i. Espace et fractions

```
\frac    {#1..#2}
\dfrac   {#1..#2}
\stdfrac {#1..#2}
\stddfrac{#1..#2}
```

— `Argument 1:` le numérateur.

— `Argument 2:` le dénominateur.

ii. Espace et racines n-ièmes d'un réel

```
\sqrt    [#opt]{#1}
\stdsqrt [#opt]{#1}
```

— `Option:` l'exposant à indiquer pour une racine n-ième.

— `Argument:` le radicande, c'est à dire ce qui sera écrit sous le radical.

iii. Une variable « symbolique »

```
\symvar [#opt]
```

— `Option:` le numéro du symbole qui vaut `1` par défaut.

- `1` donne • .
- `2` donne ◦ .
- `3` donne ▪ .

3. Pour les mathématiques

i. Intervalles « généralisés »

```
\tns@generic@interval@ext     {#1..#5}
\tns@generic@interval@semi@ext{#1..#5}
```

— `Argument 1:` le 1er délimiteur qui est à gauche.

— `Argument 2:` le 1er élément de l'intervalle « généralisé ».

— `Argument 3:` le séparateur entre les deux éléments.

— `Argument 4:` le 2e élément de l'intervalle « généralisé ».

— `Argument 5:` le 2er délimiteur qui est à droite.

ii. Décorer un opérateur

`\tns@over@math@symbol{#1..#2}`

— `Argument 1:` le texte à ajouter au-dessus.

— `Argument 2:` le symbole à décorer.

iii. Produit en croix

`\tns@prop@prod{#1..#5}`

— `Argument 1:` l'opérateur de multiplication à imprimer.

— `Arguments 2..5:` les 4 éléments du tableau de proportionnalité lus ligne par ligne.

iv. Déterminant 2D et critère de colinéarité

```
\tns@det@plane@deco@loop{#1..#7}
\tns@det@plane@deco@cross{#1..#7}
\tns@det@plane@no@deco{#1..#7}
```

— `Argument 1:` `vec` ou `novec` suivant que l'on veut afficher ou non les vecteurs.

— `Argument 2:` le 1er vecteur.

— `Arguments 3..4:` les coordonnées du 1er vecteur.

— `Argument 5:` le 2e vecteur.

— `Arguments 6..7:` les coordonnées du 2e vecteur.

v. Produit vectoriel expliqué

```
\tns@cross@prod@deco@loop {#1..#9}
\tns@cross@prod@deco@cross{#1..#9}
\tns@cross@prod@no@deco   {#1..#9}
```

— `Argument 1:` `vec` ou `novec` suivant que l'on veut afficher ou non les vecteurs.

— `Argument 2:` le 1er vecteur.

— `Argument 3..5:` les coordonnées du 1er vecteur.

— `Argument 6:` le 2e vecteur.

— `Argument 7..9:` les coordonnées du 2e vecteur.

4. En coulisse...

i. Nouvelle macro modifiant les arguments en amont de l'application d'une ancienne macro

`\tns@apply@macro@two@args{#1..#4}`

— `Argument 1:` macro finale avec deux arguments et c'est tout.

— `Argument 2:` macro qui sera appliqué sur chacun des deux arguments avant appel la macro donné en argument 1.

— `Arguments 3..4:` ils sont là pour la définition abstraite mais en pratique ils ne seront pas utilisés Ils correspondent aux arguments de la nouvelle macro fournie par `\tns@apply@macro@two@args`.

ii. Macro avec un « multi-argument » à « envelopper »

`\tns@multi@args{#1..#5}`

— `Argument 1:` le séparateur de *« sous-arguemnts »*.

— `Argument 2:` le « multi-argument ».

— `Argument 2:` le matériel a ajouté avant.

— `Argument 4:` le matériel que l'on met à la place du séparateur donné en 1er argument.

— `Argument 5:` le matériel a ajouté après.

iii. Appliquer une macro sur chaque partie d'un « multi-argument »

`\tns@multi@apply@each{#1..#2}`

— `Argument 1:` la macro à appliquer à chaque partie du « multi-argument ». Cette macro recevra deux arguments *(le 1er est le numéro de la partie et le 2e la partie elle-même)*.

— `Argument 2:` le « multi-argument ».

iv. Appliquer une macro sur chaque couple de parties successives d'un « multi-argument »

`\tns@multi@apply@couple{#1..#2}`

— `Argument 1:` la macro à appliquer à chaque couple de parties successives du « multi-argument ». Cette macro recevra trois arguments *(le 1er est le numéro du couple tandis que le 2e et le 3e sont les parties du couple)*.

— `Argument 2:` le « multi-argument ».