Diverses choses que je sais…
sur la programmation avec LaTeX3

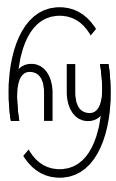
version 1.1

Le TeXnicien de surface

GUTenberg

Dunkerque 2013

révision 2013-06-13



Captatio benevolentia

Il s'agit d'une présentation **élémentaire** de LaTeX3.

Aujourd'hui

LaTeX3 disponible par l'intermédiaire de l'extension `expl3`.

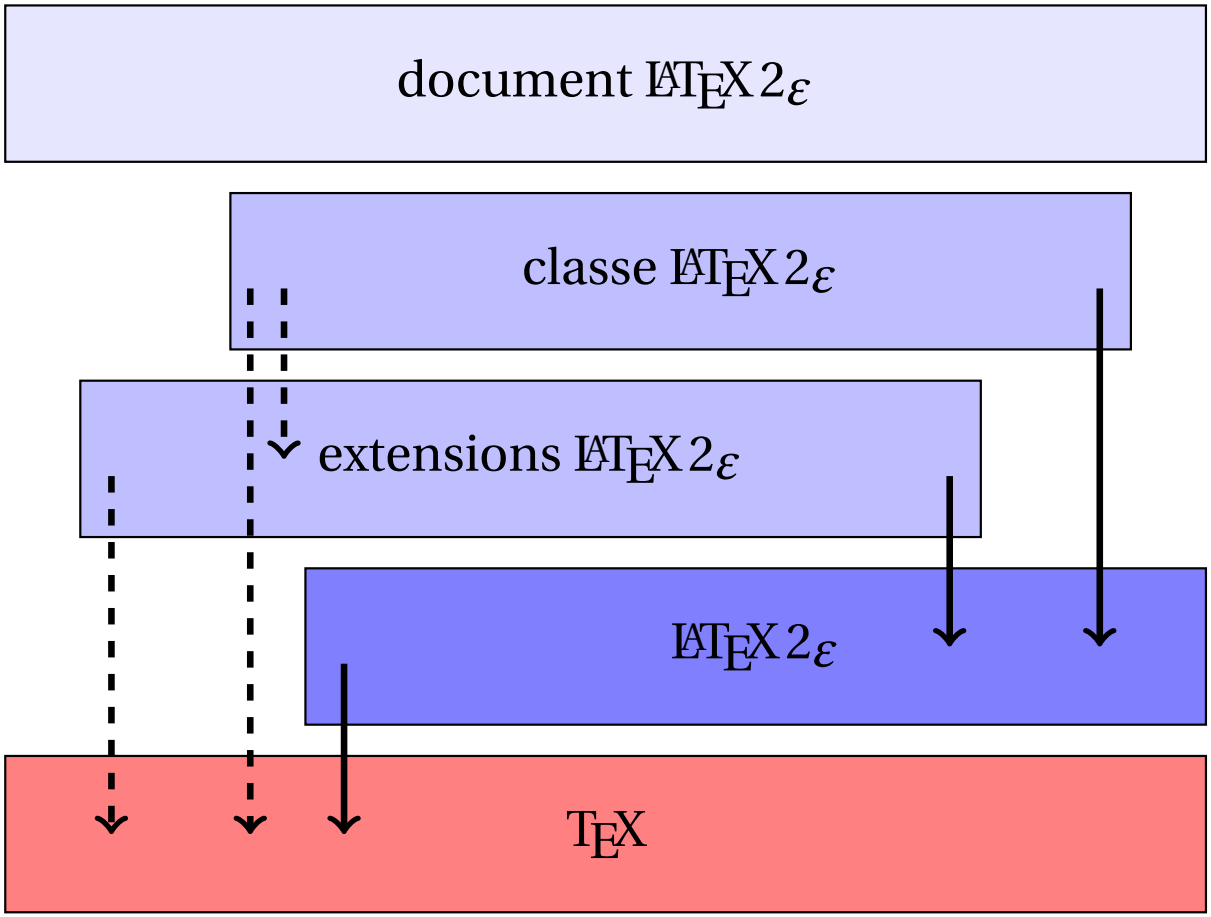
Aujourd'hui

LaTeX3 disponible par l'intermédiaire de l'extension `expl3`.
Dans un document, on utilise les commandes
`\ExplSyntaxOn` et `\ExplSyntaxOff`.

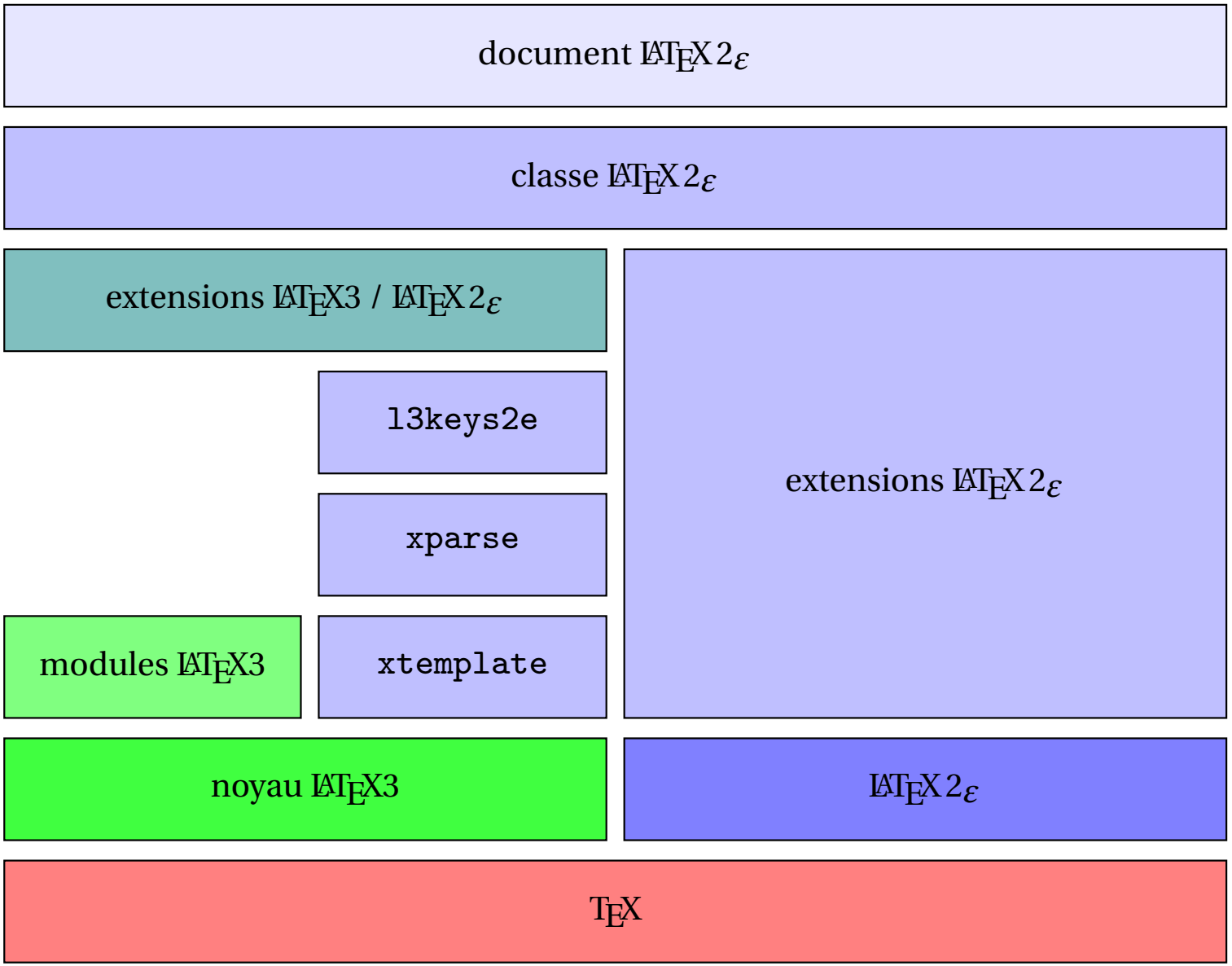
Aujourd'hui

LaTeX3 disponible par l'intermédiaire de l'extension `expl3`.
Dans un document, on utilise les commandes
`\ExplSyntaxOn` et `\ExplSyntaxOff`.
On peut créer des fichiers de styles avec
`\ProvidesExplPackage`, etc.

Modèle en couche pour LaTeX 2ε



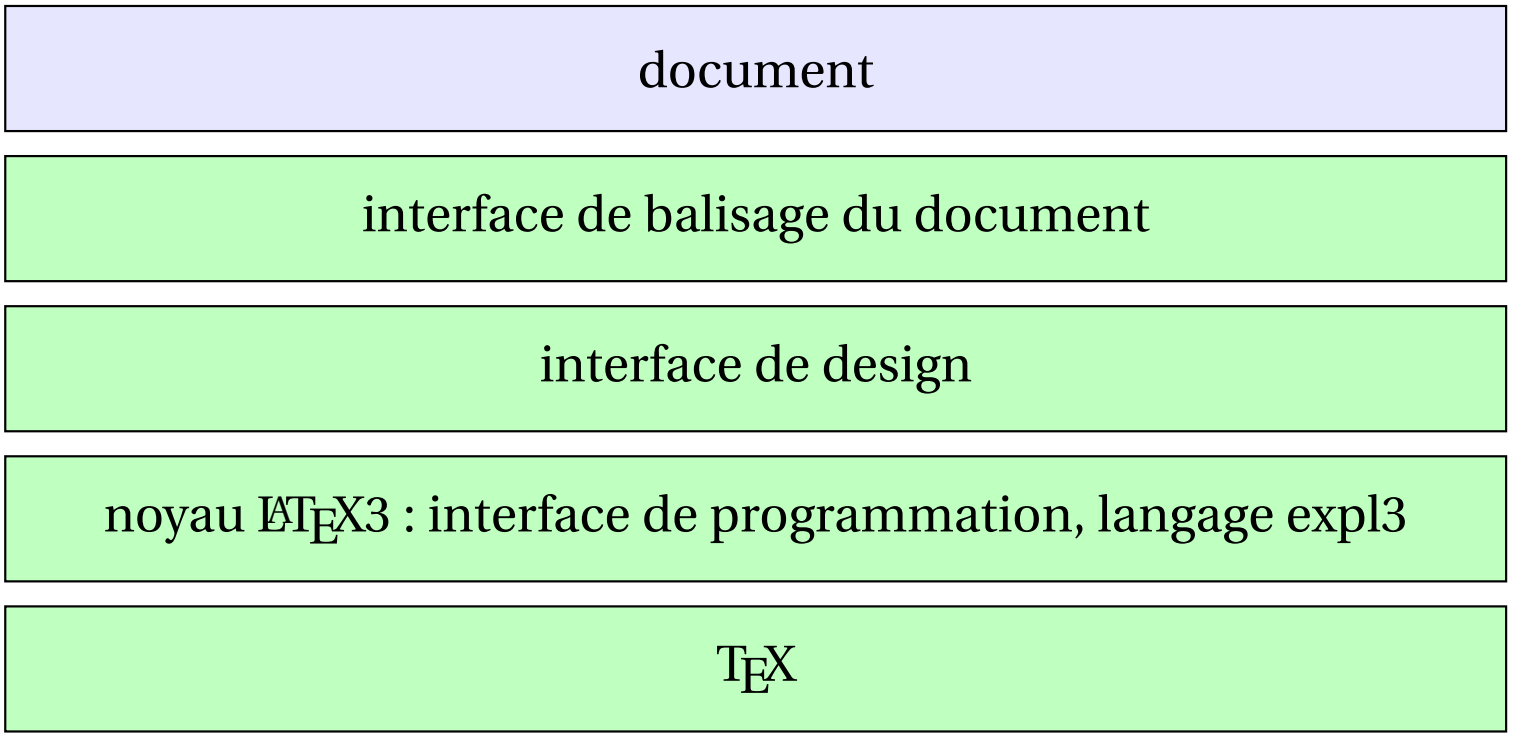
Modèle en couche pour LaTeX3 sur LaTeX 2ε



Bientôt (?)

LaTeX3 un format qui remplacera le format LaTeX 2_ε.

Modèle conceptuel de LaTeX3



Premier exemple

1re partie

```
\tl_const:Nn \Date_dif {iso}

\file_if_exist:nT {date-cal-\Date_icn.tex} {
  \file_input:n {date-cal-\Date_icn.tex} }

\cs_new:Nn \int_with_two_digits:n {
  \int_compare:nNnTF { #1 } < { \c_ten }
  { 0 \int_to_arabic:n { #1 } }
  { \int_to_arabic:n{#1} } }

\cs_new:cn {Date_write_to@iso:nnn} {
  \int_to_arabic:n { #1 } -
  \int_with_two_digits:n { #2 } -
  \int_with_two_digits:n { #3 } }
```

Premier exemple

2e partie

```
\DeclareDocumentCommand \testdate { m } {
  \group_begin:
  \int_set:Nn \l_tmpb_int
    { \julianday { #1 } }
  #1~:~JJ~=~
  \int_to_arabic:n { \l_tmpb_int }
  ~\quad~
  \calculatedate { \l_tmpb_int }
  \writedate
    { \YearNbr } { \MonthNbr } { \DayNbr }
  \group_end:
}
```

Premier exemple

3e partie

Code tiré de `siunitx` :

```
\NewDocumentCommand \numlist
  { o > { \SplitList { ; } } m } {
    \leavevmode
    \group_begin:
      \IfNoValueF {#1}
        { \keys_set:nn { siunitx } {#1} }
      \__siunitx_list_numbers:n {#2}
    \group_end:
  }
```

Premier exemple

3e partie

Code tiré de `siunitx` :

```
\NewDocumentCommand \numlist
  { o > { \SplitList { ; } } m } {
    \leavevmode
    \group_begin:
      \IfNoValueF {#1}
        { \keys_set:nn { siunitx } {#1} }
      \__siunitx_list_numbers:n {#2}
    \group_end:
  }
```

Utilisation :

```
\numlist{10; 20; 30}
```

Premier exemple

3e partie

Code tiré de `siunitx` :

```
\NewDocumentCommand \numlist
  { o > { \SplitList { ; } } m } {
    \leavevmode
    \group_begin:
      \IfNoValueF {#1}
        { \keys_set:nn { siunitx } {#1} }
      \__siunitx_list_numbers:n {#2}
    \group_end:
  }
```

Utilisation :

`\numlist{10; 20; 30}` ☞ « 10, 20 et 30 ».

Interface de programmation

Caractères disponibles

Expl3, langage du noyau de LaTeX3 n'utilise pas @ comme LaTeX 2ε.

Interface de programmation

Caractères disponibles

Expl3, langage du noyau de LaTeX3 n'utilise pas @ comme LaTeX 2_ε.
Il utilise les « lettres » _ et :.

Interface de programmation

Caractères disponibles

Expl3, langage du noyau de LaTeX3 n'utilise pas `@` comme LaTeX 2_ε.
Il utilise les « lettres » `_` et `:`.
Il suit un schéma de nommage très précis pour les *fonctions* et les *variables*.

Interface de programmation

Schéma de nommage, variables

Exemples de variables :

Interface de programmation

Schéma de nommage, variables

Exemples de variables :

- `l_tmpb_int` : variable locale — `l` —, temporaire — `tmp` —, contenant un entier relatif — `int` — ;

Interface de programmation

Schéma de nommage, variables

Exemples de variables :

- `l_tmpb_int` : variable locale — `l` —, temporaire — `tmp` —, contenant un entier relatif — `int` — ;
- `c_ten` : constante — `c` — ;

Interface de programmation

Schéma de nommage, variables

Exemples de variables :

- `l_tmpb_int` : variable locale — `l` —, temporaire — `tmp` —, contenant un entier relatif — `int` — ;
- `c_ten` : constante — `c` — ;
- `g_tmpa_bool` : variable globale — `g` —, temporaire, contenant un *booléen* — `bool`.

Interface de programmation

Schéma de nommage, variables

Exemples de variables :

- `l_tmpb_int` : variable locale — `l` —, temporaire — `tmp` —, contenant un entier relatif — `int` — ;
- `c_ten` : constante — `c` — ;
- `g_tmpa_bool` : variable globale — `g` —, temporaire, contenant un *booléen* — `bool`.

Les 1re et 2e variables sont définies dans le module `l3int`, la 3e dans le module `l3prg`.

Interface de programmation

Schéma de nommage, variables

Exemples de variables :

- `l_tmpb_int` : variable locale — `l` —, temporaire — `tmp` —, contenant un entier relatif — `int` — ;
- `c_ten` : constante — `c` — ;
- `g_tmpa_bool` : variable globale — `g` —, temporaire, contenant un *booléen* — `bool`.

Les 1re et 2e variables sont définies dans le module `l3int`,
la 3e dans le module `l3prg`.
La finale indique le type d'objet.
(à suivre…)

Interface de programmation

Types d'objets

expl3 fournit des objets de « haut niveau » :

Interface de programmation

Types d'objets

expl3 fournit des objets de « haut niveau » :

```
int  entiers ;
```

Interface de programmation

Types d'objets

expl3 fournit des objets de « haut niveau » :

int	entiers ;
fp	réels (nombres à virgules flottantes) ;

Interface de programmation

Types d'objets

expl3 fournit des objets de « haut niveau » :

int	entiers ;
fp	réels (nombres à virgules flottantes) ;
bool	valeurs logiques ;

Interface de programmation

Types d'objets

expl3 fournit des objets de « haut niveau » :

int	entiers ;
fp	réels (nombres à virgules flottantes) ;
bool	valeurs logiques ;
tl	*token list* : listes de lexèmes ;

Interface de programmation

Types d'objets

expl3 fournit des objets de « haut niveau » :

int	entiers ;
fp	réels (nombres à virgules flottantes) ;
bool	valeurs logiques ;
tl	*token list* : listes de lexèmes ;
seq	suites et piles ;

Interface de programmation

Types d'objets

expl3 fournit des objets de « haut niveau » :

int	entiers ;
fp	réels (nombres à virgules flottantes) ;
bool	valeurs logiques ;
tl	*token list* : listes de lexèmes ;
seq	suites et piles ;
clist	listes à virgules ;

Interface de programmation

Types d'objets

expl3 fournit des objets de « haut niveau » :

int	entiers ;
fp	réels (nombres à virgules flottantes) ;
bool	valeurs logiques ;
tl	*token list* : listes de lexèmes ;
seq	suites et piles ;
clist	listes à virgules ;
prop(erty list)	listes de propriétés ;

Interface de programmation

Types d'objets

expl3 fournit des objets de « haut niveau » :

int	entiers ;
fp	réels (nombres à virgules flottantes) ;
bool	valeurs logiques ;
tl	*token list* : listes de lexèmes ;
seq	suites et piles ;
clist	listes à virgules ;
prop(erty list)	listes de propriétés ;
ior / iow	flux d'entrée (*read*) et sortie (*write*) ;

Interface de programmation

Types d'objets

expl3 fournit des objets de « haut niveau » :

int	entiers ;
fp	réels (nombres à virgules flottantes) ;
bool	valeurs logiques ;
tl	*token list* : listes de lexèmes ;
seq	suites et piles ;
clist	listes à virgules ;
prop(erty list)	listes de propriétés ;
ior / iow	flux d'entrée (*read*) et sortie (*write*) ;
dim, skip, muskip	longueurs et ressorts ;

Interface de programmation

Types d'objets

expl3 fournit des objets de « haut niveau » :

int	entiers ;
fp	réels (nombres à virgules flottantes) ;
bool	valeurs logiques ;
tl	*token list* : listes de lexèmes ;
seq	suites et piles ;
clist	listes à virgules ;
prop(erty list)	listes de propriétés ;
ior / iow	flux d'entrée (*read*) et sortie (*write*) ;
dim, skip, muskip	longueurs et ressorts ;
box, coffin	boites et boites avec poignées.

Interface de programmation

Schéma de nommage, fonctions

On distingue, en général, dans les noms des fonctions le nom d'un module

```
\int
```

Interface de programmation

Schéma de nommage, fonctions

On distingue, en général, dans les noms des fonctions
le nom d'un module, une partie descriptive

```
\int_div_truncate
```

Interface de programmation

Schéma de nommage, fonctions

On distingue, en général, dans les noms des fonctions
le nom d'un module, une partie descriptive, deux points

```
\int_div_truncate:
```

Interface de programmation

Schéma de nommage, fonctions

On distingue, en général, dans les noms des fonctions le nom d'un module, une partie descriptive, deux points, la signature de la fonction.

```
\int_div_truncate:nn
```

Interface de programmation

Schéma de nommage, fonctions

On distingue, en général, dans les noms des fonctions le nom d'un module, une partie descriptive, deux points, la signature de la fonction.

`\int_div_truncate:nn`

La signature indique combien et quels types d'arguments sont utilisés par la fonction.

Interface de programmation

Schéma de nommage, fonctions

On distingue, en général, dans les noms des fonctions
le nom d'un module, une partie descriptive, deux points, la signature de la fonction.

`\int_div_truncate:nn`

La signature indique combien et quels types d'arguments sont utilisés par la fonction.
Il y a une lettre par argument.
(à suivre…)

Interface de programmation

Schéma de nommage, fonctions, signature

`n` / `N` lire *no manipulation* ; une liste de lexèmes placée entre accolades ou un seul lexème ;

Interface de programmation

Schéma de nommage, fonctions, signature

`n` / `N` lire *no manipulation* ; une liste de lexèmes placée entre accolades ou un seul lexème ;

`c` des caractères qui forment le nom d'une commande : `\toto:c {UnArg}` fait la même chose que `\toto:N \UnArg` ;

Interface de programmation

Schéma de nommage, fonctions, signature

`n` / `N` lire *no manipulation* ; une liste de lexèmes placée entre accolades ou un seul lexème ;

`c` des caractères qui forment le nom d'une commande : `\toto:c {UnArg}` fait la même chose que `\toto:N \UnArg` ;

`v` / `V` récupère le contenu d'une variable : `\toto:v {UneVar}` fait la même chose que `\toto:V \UneVar` ;

Interface de programmation

Schéma de nommage, fonctions, signature

`n` / `N` lire *no manipulation* ; une liste de lexèmes placée entre accolades ou un seul lexème ;

`c` des caractères qui forment le nom d'une commande : `\toto:c {UnArg}` fait la même chose que `\toto:N \UnArg` ;

`v` / `V` récupère le contenu d'une variable : `\toto:v {UneVar}` fait la même chose que `\toto:V \UneVar` ;

`o` / `f` / `x` l'argument est développé une fois — `o` —, entièrement — `f` —, exhaustivement — `x` — ;

Interface de programmation

Schéma de nommage, fonctions, signature

`n` / `N` lire *no manipulation* ; une liste de lexèmes placée entre accolades ou un seul lexème ;

`c` des caractères qui forment le nom d'une commande : `\toto:c {UnArg}` fait la même chose que `\toto:N \UnArg` ;

`v` / `V` récupère le contenu d'une variable : `\toto:v {UneVar}` fait la même chose que `\toto:V \UneVar` ;

`o` / `f` / `x` l'argument est développé une fois — `o` —, entièrement — `f` —, exhaustivement — `x` — ;

`T` / `F` comme `n` mais indique les branches si-vrai et si-faux d'un test ;

Interface de programmation

Schéma de nommage, fonctions, signature

`n` / `N` lire *no manipulation* ; une liste de lexèmes placée entre accolades ou un seul lexème ;

`c` des caractères qui forment le nom d'une commande : `\toto:c {UnArg}` fait la même chose que `\toto:N \UnArg` ;

`v` / `V` récupère le contenu d'une variable : `\toto:v {UneVar}` fait la même chose que `\toto:V \UneVar` ;

`o` / `f` / `x` l'argument est développé une fois — `o` —, entièrement — `f` —, exhaustivement — `x` — ;

`T` / `F` comme `n` mais indique les branches si-vrai et si-faux d'un test ;

`p` / `w` des paramètres à la TEX — `p` —, ou étranges — `w`.

Interface de programmation

Schéma de nommage, fonctions de création

- `new` : création avec controle ;

Interface de programmation

Schéma de nommage, fonctions de création

- `new` : création avec controle ;
- `set` : création sans controle, local au groupe courant ;

Interface de programmation

Schéma de nommage, fonctions de création

- `new` : création avec controle ;
- `set` : création sans controle, local au groupe courant ;
- `gset` : création sans controle, global c.-à-d. transcende le groupe courant.

Interface de programmation

Schéma de nommage, fonctions de création

- `new` : création avec controle ;
- `set` : création sans controle, local au groupe courant ;
- `gset` : création sans controle, global c.-à-d. transcende le groupe courant.

Exemples, code tiré de `siunitx` :

```
\cs_new:Npn \__siunitx_font_shape: { }

\int_gset:cn { c__siunitx_math #1 _int } { \fam }

\tl_set:Nn \c__siunitx_minus_tl
       { \tex_char:D \c__siunitx_minus_int }
```

Interface de programmation

Variantes d'une fonction

On dispose de la fonction `\prop_get:NnNT` dont le 2e argument doit être donné entre accolades.

Interface de programmation

Variantes d'une fonction

On dispose de la fonction `\prop_get:NnNT` dont le 2e argument doit être donné entre accolades.
On voudrait la *même* fonction mais qui récupérerait le contenu d'une variable donnée par son nom :

Interface de programmation

Variantes d'une fonction

On dispose de la fonction `\prop_get:NnNT` dont le 2e argument doit être donné entre accolades.
On voudrait la *même* fonction mais qui récupérerait le contenu d'une variable donnée par son nom :

```
\cs_generate_variant:Nn \prop_get:NnNT { NV }
```

Interface de programmation

Variantes d'une fonction

On dispose de la fonction `\prop_get:NnNT` dont le 2e argument doit être donné entre accolades.
On voudrait la *même* fonction mais qui récupérerait le contenu d'une variable donnée par son nom :

```
\cs_generate_variant:Nn \prop_get:NnNT { NV }
```

On dispose maintenant de la fonction `\prop_get:NVNT` qui fait ce que l'on voulait.

XPARSE

Un petit air de famille

XPARSE

Un petit air de famille

- `\Declare` / `\New` / `\Renew` / `\ProvideDocumentCommand`;

XPARSE

Un petit air de famille

- `\Declare` / `\New` / `\Renew` / `\ProvideDocumentCommand` ;
- `\Declare` / `\New` / `\Renew` / `\ProvideDocumentEnvironment` ;

XPARSE

Un petit air de famille

- `\Declare` / `\New` / `\Renew` / `\ProvideDocumentCommand` ;
- `\Declare` / `\New` / `\Renew` / `\ProvideDocumentEnvironment` ;
- `\IfNoValue(TF)`, `\IfBoolean(TF)` ;

XPARSE

Un petit air de famille

- `\Declare` / `\New` / `\Renew` / `\ProvideDocumentCommand`;
- `\Declare` / `\New` / `\Renew` / `\ProvideDocumentEnvironment`;
- `\IfNoValue(TF)`, `\IfBoolean(TF)`;
- `\DeclareExpandableDocumentCommand`...

XPARSE

Un petit air de famille

- `\Declare` / `\New` / `\Renew` / `\ProvideDocumentCommand` ;
- `\Declare` / `\New` / `\Renew` / `\ProvideDocumentEnvironment` ;
- `\IfNoValue(TF)`, `\IfBoolean(TF)` ;
- `\DeclareExpandableDocumentCommand`...

mais...

Une syntaxe nouvelle :
Exemples tirés de `siunitx`

```
\NewDocumentCommand \DeclareBinaryPrefix { m m m } {
  \__siunitx_declare_prefix:Nnnn #1 {#2} { 2 } {#3}
}

\NewDocumentCommand \DeclareSIUnit { O { } m m } {
  \__siunitx_declare_unit:Nnn #2 {#3} {#1}
}

\NewDocumentCommand \SI { o m o m } {
  \leavevmode
  \group_begin:
    \IfNoValueTF {#1}
      { \__siunitx_combined:nnnn { } {#2} {#3} {#4} }
      {
        \keys_set:nn { siunitx } {#1}
        \__siunitx_combined:nnnn {#1} {#2} {#3} {#4}
      }
  \group_end:
}
```

XPARSE

Exemples tirés de enotez

```
\NewDocumentCommand \printendnotes { so }
  {
    \bool_if:NTF \l__enotez_split_bool
      { \enotez_print_endnotes:nn { \BooleanFalse } { #2 } }
      { \enotez_print_endnotes:nn { #1 } { #2 } }
  }

\NewDocumentCommand \setenotez { +m }
  { \keys_set:nn { enotez } { #1 } \ignorespaces }
```

Exemples tirés de exsheets

```
\NewDocumentCommand \currentpointssum { s }
  {
    \IfBooleanTF { #1 }
      { \exsheets_parse_points:n { \g__exsheets_points_sum_fp } }
      { \exsheets_print_points:n { \g__exsheets_points_sum_fp } }
  }
```

Syntaxe

```
\NewDocumentCommand \mamacro {<paramètres>} {<du code>}
```

Syntaxe

`\NewDocumentCommand \mamacro {<paramètres>} {<du code>}`

Les *<paramètres>* sont au plus 9 et on donne leur type à l'aide des notations suivantes :

Syntaxe

`\NewDocumentCommand \mamacro {<paramètres>} {<du code>}`

Les *<paramètres>* sont au plus 9 et on donne leur type à l'aide des notations suivantes :

- `m` : argument obligatoire *ordinaire*, non-`\long` ;

Syntaxe

`\NewDocumentCommand \mamacro {<paramètres>} {<du code>}`

Les *<paramètres>* sont au plus 9 et on donne leur type à l'aide des notations suivantes :

- `m` : argument obligatoire *ordinaire*, non-`\long` ;
- `l`, `u`, `r`, `R` : arguments obligatoires délimités ;

Syntaxe

`\NewDocumentCommand \mamacro {<paramètres>} {<du code>}`

Les *<paramètres>* sont au plus 9 et on donne leur type à l'aide des notations suivantes :

- `m` : argument obligatoire *ordinaire*, non-`\long` ;
- `l`, `u`, `r`, `R` : arguments obligatoires délimités ;
- `O`, `o` : argument optionnel *ordinaire* avec ou sans valeur par défaut ;

Syntaxe

`\NewDocumentCommand \mamacro {<paramètres>} {<du code>}`

Les *<paramètres>* sont au plus 9 et on donne leur type à l'aide des notations suivantes :

- `m` : argument obligatoire *ordinaire*, non-`\long` ;
- `l`, `u`, `r`, `R` : arguments obligatoires délimités ;
- `O`, `o` : argument optionnel *ordinaire* avec ou sans valeur par défaut ;
- `D`, `d` : argument optionnel délimité avec ou sans valeur par défaut ;

Syntaxe

`\NewDocumentCommand \mamacro {<paramètres>} {<du code>}`

Les *<paramètres>* sont au plus 9 et on donne leur type à l'aide des notations suivantes :

- `m` : argument obligatoire *ordinaire*, non-`\long` ;
- `l`, `u`, `r`, `R` : arguments obligatoires délimités ;
- `O`, `o` : argument optionnel *ordinaire* avec ou sans valeur par défaut ;
- `D`, `d` : argument optionnel délimité avec ou sans valeur par défaut ;
- `G`, `g` : argument optionnel délimité par des accolades avec ou sans valeur par défaut ;

Syntaxe

`\NewDocumentCommand \mamacro {<paramètres>} {<du code>}`

Les *<paramètres>* sont au plus 9 et on donne leur type à l'aide des notations suivantes :

- `m` : argument obligatoire *ordinaire*, non-`\long` ;
- `l`, `u`, `r`, `R` : arguments obligatoires délimités ;
- `O`, `o` : argument optionnel *ordinaire* avec ou sans valeur par défaut ;
- `D`, `d` : argument optionnel délimité avec ou sans valeur par défaut ;
- `G`, `g` : argument optionnel délimité par des accolades avec ou sans valeur par défaut ;
- `s`, `t` : une étoile ou un lexème optionnel, si présent entraine `\BooleanTrue` ;

Syntaxe

`\NewDocumentCommand \mamacro {<paramètres>} {<du code>}`

Les *<paramètres>* sont au plus 9 et on donne leur type à l'aide des notations suivantes :

- `m` : argument obligatoire *ordinaire*, non-`\long` ;
- `l`, `u`, `r`, `R` : arguments obligatoires délimités ;
- `O`, `o` : argument optionnel *ordinaire* avec ou sans valeur par défaut ;
- `D`, `d` : argument optionnel délimité avec ou sans valeur par défaut ;
- `G`, `g` : argument optionnel délimité par des accolades avec ou sans valeur par défaut ;
- `s`, `t` : une étoile ou un lexème optionnel, si présent entraine `\BooleanTrue` ;
- `v` : lit l'argument `verbatim`.

Syntaxe

Paramètres, suite...

Argument délimité :

Syntaxe

Paramètres, suite...

Argument délimité :
`r/d<lexème1><lexème2>`

Syntaxe

Paramètres, suite...

Argument délimité :

`r/d<lexème1><lexème2>`

`R/D<lexème1><lexème2>{<valeur par défaut>}`

Syntaxe

Paramètres, suite...

Argument délimité :

`r/d<lexème1><lexème2>`

`R/D<lexème1><lexème2>{<valeur par défaut>}`

`o/O` se comporte comme `d/D[]`

Syntaxe

Paramètres, suite...

Argument délimité :

`r/d<lexème1><lexème2>`

`R/D<lexème1><lexème2>{<valeur par défaut>}`

`o/O` se comporte comme `d/D[]`

`g/G` se comporte comme `d/D{}`

Syntaxe

Paramètres, suite...

Argument délimité :

`r/d<lexème1><lexème2>`

`R/D<lexème1><lexème2>{<valeur par défaut>}`

`o/O` se comporte comme `d/D[]`

`g/G` se comporte comme `d/D{}`

`u{<lexèmes>}` lit jusqu'à rencontrer `<lexèmes>`

Syntaxe

Paramètres, suite...

Argument délimité :

`r/d<lexème1><lexème2>`

`R/D<lexème1><lexème2>{<valeur par défaut>}`

`o/O` se comporte comme `d/D[]`

`g/G` se comporte comme `d/D{}`

`u{<lexèmes>}` lit jusqu'à rencontrer `<lexèmes>`

`l` se comporte comme un impossible `u{{}`

Syntaxe

Paramètres, suite...

Argument délimité :
`r/d<lexème1><lexème2>`
`R/D<lexème1><lexème2>{<valeur par défaut>}`
`o/O` se comporte comme `d/D[]`
`g/G` se comporte comme `d/D{}`
`u{<lexèmes>}` lit jusqu'à rencontrer `<lexèmes>`
`l` se comporte comme un impossible `u{{}`
Quand on ne donne pas de valeur à un argument optionnel, cet argument est passé avec la valeur spéciale `-NoValue-` que l'on peut tester avec `\If(No)Value(TF)`.

Syntaxe

Paramètres, suite...

Argument délimité :
`r/d<lexème1><lexème2>`
`R/D<lexème1><lexème2>{<valeur par défaut>}`
`o/O` se comporte comme `d/D[]`
`g/G` se comporte comme `d/D{}`
`u{<lexèmes>}` lit jusqu'à rencontrer `<lexèmes>`
`l` se comporte comme un impossible `u{{}`
Quand on ne donne pas de valeur à un argument optionnel, cet argument est passé avec la valeur spéciale `-NoValue-` que l'on peut tester avec `\If(No)Value(TF)`.
Étoile et autre lexème optionnel :

Syntaxe

Paramètres, suite...

Argument délimité :
`r/d<lexème1><lexème2>`
`R/D<lexème1><lexème2>{<valeur par défaut>}`
`o/O` se comporte comme `d/D[]`
`g/G` se comporte comme `d/D{}`
`u{<lexèmes>}` lit jusqu'à rencontrer `<lexèmes>`
`l` se comporte comme un impossible `u{{}`
Quand on ne donne pas de valeur à un argument optionnel, cet argument est passé avec la valeur spéciale `-NoValue-` que l'on peut tester avec `\If(No)Value(TF)`.
Étoile et autre lexème optionnel :
`s` se comporte comme `t*`

Syntaxe

paramètres, suite...

Argument long :

Syntaxe

Paramètres, suite...

Argument long :
Si on veut qu'un argument soit long, on fait précéder la lettre d'un +.

Syntaxe

paramètres, suite...

Argument long :
Si on veut qu'un argument soit long, on fait précéder la lettre d'un +.
Différences avec LaTeX 2ε :

Syntaxe

Paramètres, suite...

Argument long :
Si on veut qu'un argument soit long, on fait précéder la lettre d'un +.
Différences avec LaTeX 2_ε :

- logique inverse : argument court par défaut ;

Syntaxe

Paramètres, suite...

Argument long :
Si on veut qu'un argument soit long, on fait précéder la lettre d'un +.
Différences avec LaTeX 2ε :

- logique inverse : argument court par défaut ;
- qualité par argument et non pas globalement.

Syntaxe

Paramètres, suite... et fin

Préparateur — *processor* — d'argument :

Syntaxe

Paramètres, suite... et fin

Préparateur — *processor* — d'argument :

```
>{\ProcessorB} >{\ProcessorA} m
```

Syntaxe

Paramètres, suite... et fin

Préparateur — *processor* — d'argument :
`>{\ProcessorB} >{\ProcessorA} m`
un préparateur retourne la variable préparée comme valeur de
`\ProcessedArgument`

Syntaxe

Paramètres, suite... et fin

Préparateur — *processor* — d'argument :
`>{\ProcessorB} >{\ProcessorA} m`
un préparateur retourne la variable préparée comme valeur de `\ProcessedArgument`

Préparateurs prédéfinis dans `xparse` :

Syntaxe

Paramètres, suite... et fin

Préparateur — *processor* — d'argument :
`>{\ProcessorB} >{\ProcessorA} m`
un préparateur retourne la variable préparée comme valeur de `\ProcessedArgument`

Préparateurs prédéfinis dans `xparse` :

- `\ReverseBoolean` ;

Syntaxe

Paramètres, suite... et fin

Préparateur — *processor* — d'argument :
`>{\ProcessorB} >{\ProcessorA} m`
un préparateur retourne la variable préparée comme valeur de `\ProcessedArgument`

Préparateurs prédéfinis dans `xparse` :

- `\ReverseBoolean` ;
- `\SplitArgument{<nombre>}{<lexème>}` : découpe au niveau du `<lexème>` en, au plus, $n+1$ éléments ;

Syntaxe

Paramètres, suite... et fin

Préparateur — *processor* — d'argument :
`>{\ProcessorB} >{\ProcessorA} m`
un préparateur retourne la variable préparée comme valeur de `\ProcessedArgument`

Préparateurs prédéfinis dans `xparse` :

- `\ReverseBoolean` ;
- `\SplitArgument{<nombre>}{<lexème>}` : découpe au niveau du `<lexème>` en, au plus, $n+1$ éléments ;
- `\SplitList{<lexème>}` : découpe au niveau du `<lexème>` et renvoie les éléments entre accolades.

Syntaxe

Paramètres, suite... et fin

Préparateur — *processor* — d'argument :
`>{\ProcessorB} >{\ProcessorA} m`
un préparateur retourne la variable préparée comme valeur de `\ProcessedArgument`

Préparateurs prédéfinis dans `xparse` :

- `\ReverseBoolean` ;
- `\SplitArgument{<nombre>}{<lexème>}` : découpe au niveau du `<lexème>` en, au plus, $n+1$ éléments ;
- `\SplitList{<lexème>}` : découpe au niveau du `<lexème>` et renvoie les éléments entre accolades.
  `\ProcessList{<liste>}{<fonction>}` : applique la `<fonction>` unaire à chaque élément de la liste ;

Syntaxe

Paramètres, suite... et fin

Préparateur — *processor* — d'argument :
`>{\ProcessorB} >{\ProcessorA} m`
un préparateur retourne la variable préparée comme valeur de `\ProcessedArgument`

Préparateurs prédéfinis dans `xparse` :

- `\ReverseBoolean` ;
- `\SplitArgument{<nombre>}{<lexème>}` : découpe au niveau du `<lexème>` en, au plus, $n+1$ éléments ;
- `\SplitList{<lexème>}` : découpe au niveau du `<lexème>` et renvoie les éléments entre accolades.
  `\ProcessList{<liste>}{<fonction>}` : applique la `<fonction>` unaire à chaque élément de la liste ;
- `\TrimSpaces` : retire les espaces de tête et de queue.

Exemples

Extraits de la documentation de xparse

```
\NewDocumentCommand \chapter { s o m } {
  \IfBooleanTF {#1}
  { \typesetstarchapter {#3} }
  { \typesetnormalchapter {#2} {#3} } }
```

Exemples

Extraits de la documentation de xparse

```
\NewDocumentCommand \chapter { s o m } {
  \IfBooleanTF {#1}
  { \typesetstarchapter {#3} }
  { \typesetnormalchapter {#2} {#3} } }
```

```
\NewDocumentCommand \chapter
  { > {\ReverseBoolean} s o m } {
    \IfBooleanTF {#1}
    { \typesetnormalchapter {#2} {#3} }
    { \typesetstarchapter {#3} } }
```

Bibliographie

- Sur TEX

Bibliographie

- Sur TEX
  - *The TEXbook*, D. E. Knuth ; traduction française *Le TEXlivre* ;

Bibliographie

- Sur TEX
  - *The TEXbook*, D. E. Knuth ; traduction française *Le TEXlivre* ;
  - *TEX by topic*, V. Eijkhout.

Bibliographie

- Sur TEX
  - *The TEXbook*, D. E. Knuth ; traduction française *Le TEXlivre* ;
  - *TEX by topic*, V. Eijkhout.
- Sur LATEX 2ε

Bibliographie

- Sur TEX
  - *The TEXbook*, D. E. Knuth ; traduction française *Le TEXlivre* ;
  - *TEX by topic*, V. Eijkhout.
- Sur LATEX 2ε
  - Les sources commentées du noyau de LATEX 2ε sont disponibles sous le nom `source2e.pdf`.

Bibliographie

- Sur TEX
  - *The TEXbook*, D. E. Knuth ; traduction française *Le TEXlivre* ;
  - *TEX by topic*, V. Eijkhout.
- Sur LATEX 2ε
  - Les sources commentées du noyau de LATEX 2ε sont disponibles sous le nom `source2e.pdf`.
- Sur LATEX3

Bibliographie

- Sur TEX
  - *The TEXbook*, D. E. Knuth ; traduction française *Le TEXlivre* ;
  - *TEX by topic*, V. Eijkhout.
- Sur LATEX 2ε
  - Les sources commentées du noyau de LATEX 2ε sont disponibles sous le nom `source2e.pdf`.
- Sur LATEX3
  - *The expl3 package and LATEX3 programming*, The LATEX3 Project ;

Bibliographie

- Sur TEX
  - *The TEXbook*, D. E. Knuth ; traduction française *Le TEXlivre* ;
  - *TEX by topic*, V. Eijkhout.
- Sur LATEX 2ε
  - Les sources commentées du noyau de LATEX 2ε sont disponibles sous le nom `source2e.pdf`.
- Sur LATEX3
  - *The expl3 package and LATEX3 programming*, The LATEX3 Project ;
  - *The LATEX3 Interfaces*, The LATEX3 Project ;

Bibliographie

- Sur TEX
  - *The TEXbook*, D. E. Knuth ; traduction française *Le TEXlivre* ;
  - *TEX by topic*, V. Eijkhout.
- Sur LATEX 2ε
  - Les sources commentées du noyau de LATEX 2ε sont disponibles sous le nom `source2e.pdf`.
- Sur LATEX3
  - *The expl3 package and LATEX3 programming*, The LATEX3 Project ;
  - *The LATEX3 Interfaces*, The LATEX3 Project ;
  - *The xparse package, Document command parser*, The LATEX3 Project.