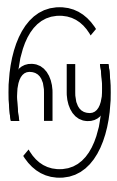
Macros étoilées (et plus) sans peine avec `xparse`

version 1

Le TEXnicien de surface

Lille 1 & GUTenberg

Dunkerque 2015



Copyright et licence

 Ce document et ses sources sont publiées sous la licence Creative Commons Attribution 4.0 International (CC BY 4.0)

https://creativecommons.org/licenses/by/4.0/

© Yvon Henel, 2015

Charger xparse

Dans un document (`toto.tex`), on charge `xparse` avec :

Code

```
\usepackage{xparse}
```

Charger xparse

Dans un document (`toto.tex`), on charge `xparse` avec :

Code

```
\usepackage{xparse}
```

Dans une extension (`toto.sty`), on charge `xparse` avec :

Code

```
\RequirePackage{xparse}
```

Charger xparse

Dans un document (`toto.tex`), on charge `xparse` avec :

Code

```
\usepackage{xparse}
```

Dans une extension (`toto.sty`), on charge `xparse` avec :

Code

```
\RequirePackage{xparse}
```

On aura intérêt alors à créer un *module* :

Code

```
\NeedsTeXFormat{LaTeX2e}[1999/12/01]
\RequirePackage{l3keys2e,xparse}
\ProvidesExplPackage
{toto} {2015/01/01} {0.1} {<description>}
```

Charger xparse

Dans un document (`toto.tex`), on charge `xparse` avec :

Code

```
\usepackage{xparse}
```

Dans une extension (`toto.sty`), on charge `xparse` avec :

Code

```
\RequirePackage{xparse}
```

On aura intérêt alors à créer un *module* :

Code

```
\NeedsTeXFormat{LaTeX2e}[1999/12/01]
\RequirePackage{l3keys2e,xparse}
\ProvidesExplPackage
{toto} {2015/01/01} {0.1} {<description>}
```

Ce que charge `xparse`

À savoir

`xparse` charge `expl3`

Ce que charge `xparse`

À savoir

`xparse` charge `expl3`

On pourra donc ensuite utiliser la syntaxe de LaTeX3

Ce que charge `xparse`

À savoir

`xparse` charge `expl3`

On pourra donc ensuite utiliser la syntaxe de LaTeX3

- directement si on écrit un module (`\ProvidesExplPackage`) ;

Ce que charge `xparse`

À savoir

`xparse` charge `expl3`

On pourra donc ensuite utiliser la syntaxe de LaTeX3

- directement si on écrit un module (`\ProvidesExplPackage`) ;
- entre `\ExplSyntaxOn` et `\ExplSyntaxOff` dans un document.

Créer de nouvelles macros

et de nouveaux environnements

Voyons d'abord les commandes disponibles :

Créer de nouvelles macros
et de nouveaux environnements

Voyons d'abord les commandes disponibles :

Macros

- `\DeclareDocumentCommand`;

Créer de nouvelles macros
et de nouveaux environnements

Voyons d'abord les commandes disponibles :

Macros

- `\DeclareDocumentCommand`;
- `\NewDocumentCommand`;

Créer de nouvelles macros
et de nouveaux environnements

Voyons d'abord les commandes disponibles :

Macros

- `\DeclareDocumentCommand`;
- `\NewDocumentCommand`;
- `\RenewDocumentCommand`;

Créer de nouvelles macros
et de nouveaux environnements

Voyons d'abord les commandes disponibles :

Macros

- `\DeclareDocumentCommand`;
- `\NewDocumentCommand`;
- `\RenewDocumentCommand`;
- `\ProvideDocumentCommand`.

Créer de nouvelles macros
et de nouveaux environnements

Voyons d'abord les commandes disponibles :

Macros

- `\DeclareDocumentCommand`;
- `\NewDocumentCommand`;
- `\RenewDocumentCommand`;
- `\ProvideDocumentCommand`.

Créer de nouvelles macros
et de nouveaux environnements

Voyons d'abord les commandes disponibles :

Macros

- `\DeclareDocumentCommand`;
- `\NewDocumentCommand`;
- `\RenewDocumentCommand`;
- `\ProvideDocumentCommand`.

Environnements

- `\DeclareDocumentEnvironment`;
- `\NewDocumentEnvironment`;
- `\RenewDocumentEnvironment`;
- `\ProvideDocumentEnvironment`.

Syntaxe pour les macros

Code

```
\NewDocumentCommand {<macro>} {<sparg>} {<code>}
```

Syntaxe pour les macros

Code

```
\NewDocumentCommand {<macro>} {<sparg>} {<code>}
```

- *<macro>* est le nom de la macro, p. ex. `\majoliemacro` ;

Syntaxe pour les macros

Code

```
\NewDocumentCommand {<macro>} {<sparg>} {<code>}
```

- *<macro>* est le nom de la macro, p. ex. `\majoliemacro` ;
- *<sparg>* est une liste de *spécificateurs d'arguments* ;

Syntaxe pour les macros

Code

```
\NewDocumentCommand {<macro>} {<sparg>} {<code>}
```

- *<macro>* est le nom de la macro, p. ex. `\majoliemacro` ;
- *<sparg>* est une liste de *spécificateurs d'arguments* ;
- *<code>* est le code définissant la macro dans lequel on peut utiliser les habituels `#1`, …, `#9`.

Syntaxe pour les environnements

Code

```
\NewDocumentEnvironment {<environ>} {<sparg>}
{<ouverture>}
{<fermeture>}
```

Syntaxe pour les environnements

Code

```
\NewDocumentEnvironment {<environ>} {<sparg>}
{<ouverture>}
{<fermeture>}
```

- *<environ>* est le nom de l'environnemnent, p. ex. `boite` ;

Syntaxe pour les environnements

Code

```
\NewDocumentEnvironment {<environ>} {<sparg>}
{<ouverture>}
{<fermeture>}
```

- `<environ>` est le nom de l'environnemnent, p. ex. `boite` ;
- `<sparg>` est une liste de *spécificateurs d'arguments* ;

Syntaxe pour les environnements

Code

```
\NewDocumentEnvironment {<environ>} {<sparg>}
{<ouverture>}
{<fermeture>}
```

- `<environ>` est le nom de l'environnemnent, p. ex. `boite` ;
- `<sparg>` est une liste de *spécificateurs d'arguments* ;
- `<ouverture>` est le code exécuté en début d'environnement, code dans lequel on peut utiliser les habituels `#1`, …, `#9` ;

Syntaxe pour les environnements

Code

```
\NewDocumentEnvironment {<environ>} {<sparg>}
{<ouverture>}
{<fermeture>}
```

- `<environ>` est le nom de l'environnemnent, p. ex. `boite` ;
- `<sparg>` est une liste de *spécificateurs d'arguments* ;
- `<ouverture>` est le code exécuté en début d'environnement, code dans lequel on peut utiliser les habituels `#1`, …, `#9` ;
- `<fermeture>` est le code exécuté en fin d'environnement. On peut désormais utiliser les arguments passés à l'environnement avec les habituels `#1`, …, `#9`.

Spécificateurs d'arguments

Arguments obligatoires

- `m` : on passe l'argument entre accolades ;

Spécificateurs d'arguments

Arguments obligatoires

- `m` : on passe l'argument entre accolades ;
- `l` : l'argument est tout ce qui suit la macro jusqu'au premier lexème ouvrant un groupe (normalement {) ;

Spécificateurs d'arguments

Arguments obligatoires

- `m` : on passe l'argument entre accolades ;
- `l` : l'argument est tout ce qui suit la macro jusqu'au premier lexème ouvrant un groupe (normalement {) ;
- `r` : `r<lex1><lex2>`, argument délimité par *<lex1>* et *<lex2>* ;

Spécificateurs d'arguments

Arguments obligatoires

- `m` : on passe l'argument entre accolades ;
- `l` : l'argument est tout ce qui suit la macro jusqu'au premier lexème ouvrant un groupe (normalement {) ;
- `r` : `r<lex1><lex2>`, argument délimité par `<lex1>` et `<lex2>` ;
- `R` : `R<lex1><lex2>{<defaut>}` ;

Spécificateurs d'arguments

Arguments obligatoires

- `m` : on passe l'argument entre accolades ;
- `l` : l'argument est tout ce qui suit la macro jusqu'au premier lexème ouvrant un groupe (normalement {) ;
- `r` : `r<lex1><lex2>`, argument délimité par `<lex1>` et `<lex2>` ;
- `R` : `R<lex1><lex2>{<defaut>}` ;
- `u` : `u{<lexèmes>}`, la lecture de l'argument s'arrête quand LaTeX trouve `<lexèmes>` ;

Spécificateurs d'arguments

Arguments obligatoires

- `m` : on passe l'argument entre accolades ;
- `l` : l'argument est tout ce qui suit la macro jusqu'au premier lexème ouvrant un groupe (normalement {) ;
- `r` : `r<lex1><lex2>`, argument délimité par `<lex1>` et `<lex2>` ;
- `R` : `R<lex1><lex2>{<defaut>}` ;
- `u` : `u{<lexèmes>}`, la lecture de l'argument s'arrête quand LaTeX trouve `<lexèmes>` ;
- `v` : lecture *verbatim*, comme avec `\verb`.

Spécificateurs d'arguments

Arguments optionnels

- o : argument optionnel passé entre crochets ;

Spécificateurs d'arguments

Arguments optionnels

- `o` : argument optionnel passé entre crochets ;
- `O` : `O{<defaut>}`, si l'utilisateur ne donne pas d'argument, `<defaut>` est passé à la commande ;

Spécificateurs d'arguments

Arguments optionnels

- `o` : argument optionnel passé entre crochets ;
- `O` : `O{<defaut>}`, si l'utilisateur ne donne pas d'argument, *<defaut>* est passé à la commande ;
- `d` : `d<lex1><lex2>`, argument optionnel délimité par *<lex1>* et *<lex2>* ;

Spécificateurs d'arguments

Arguments optionnels

- `o` : argument optionnel passé entre crochets ;
- `O` : `O{<defaut>}`, si l'utilisateur ne donne pas d'argument, *<defaut>* est passé à la commande ;
- `d` : `d<lex1><lex2>`, argument optionnel délimité par *<lex1>* et *<lex2>* ;
- `D` : `D<lex1><lex2>{<defaut>}` ;

Spécificateurs d'arguments

Arguments optionnels

- `o` : argument optionnel passé entre crochets ;
- `O` : `O{<defaut>}`, si l'utilisateur ne donne pas d'argument, `<defaut>` est passé à la commande ;
- `d` : `d<lex1><lex2>`, argument optionnel délimité par `<lex1>` et `<lex2>` ;
- `D` : `D<lex1><lex2>{<defaut>}` ;
- `s` : une étoile * optionnelle ;

Spécificateurs d'arguments

Arguments optionnels

- `o` : argument optionnel passé entre crochets ;
- `O` : `O{<defaut>}`, si l'utilisateur ne donne pas d'argument, `<defaut>` est passé à la commande ;
- `d` : `d<lex1><lex2>`, argument optionnel délimité par `<lex1>` et `<lex2>` ;
- `D` : `D<lex1><lex2>{<defaut>}` ;
- `s` : une étoile * optionnelle ;
- `t` : `t<lexème>`, un `<lexème>` optionnel — `s` pourrait être écrit `t*` ;

Spécificateurs d'arguments

Arguments optionnels

- `o` : argument optionnel passé entre crochets ;
- `O` : `O{<defaut>}`, si l'utilisateur ne donne pas d'argument, `<defaut>` est passé à la commande ;
- `d` : `d<lex1><lex2>`, argument optionnel délimité par `<lex1>` et `<lex2>` ;
- `D` : `D<lex1><lex2>{<defaut>}` ;
- `s` : une étoile `*` optionnelle ;
- `t` : `t<lexème>`, un `<lexème>` optionnel — `s` pourrait être écrit `t*` ;
- `g` : argument optionnel délimité par des accolades ;

Spécificateurs d'arguments

Arguments optionnels

- `o` : argument optionnel passé entre crochets ;
- `O` : `O{<defaut>}`, si l'utilisateur ne donne pas d'argument, `<defaut>` est passé à la commande ;
- `d` : `d<lex1><lex2>`, argument optionnel délimité par `<lex1>` et `<lex2>` ;
- `D` : `D<lex1><lex2>{<defaut>}` ;
- `s` : une étoile * optionnelle ;
- `t` : `t<lexème>`, un `<lexème>` optionnel — `s` pourrait être écrit `t*` ;
- `g` : argument optionnel délimité par des accolades ;
- `G` : `G{<defaut>}`.

Argument optionnel manquant

À savoir

Quand on ne précise pas de valeur par défaut (`o`, `d`, `g`), si l'argument optionnel n'est pas donné par l'utilisateur, la commande reçoit la valeur spéciale `-NoValue-`.

Argument optionnel manquant

À savoir

Quand on ne précise pas de valeur par défaut (`o`, `d`, `g`), si l'argument optionnel n'est pas donné par l'utilisateur, la commande reçoit la valeur spéciale `-NoValue-`.

Quand le <*lexème*> manque (`s`, `t`), l'argument correspondant contient la valeur `\BooleanFalse`.

Argument optionnel manquant

À savoir

Quand on ne précise pas de valeur par défaut (`o`, `d`, `g`), si l'argument optionnel n'est pas donné par l'utilisateur, la commande reçoit la valeur spéciale `-NoValue-`.

Quand le <*lexème*> manque (`s`, `t`), l'argument correspondant contient la valeur `\BooleanFalse`. Il contient `\BooleanTrue` dans le cas contraire.

Premier exemple

Code

```
\NewDocumentCommand {\truc} { s O{1} m} {<déf.>}
```

Premier exemple

Code

```
\NewDocumentCommand {\truc} { s O{1} m} {<déf.>}
```

Premier exemple

Code

```
\NewDocumentCommand {\truc} { s O{1} m} {<déf.>}
```

Avec

Code

```
\truc*{AA}
```

Premier exemple

Code

```
\NewDocumentCommand {\truc} { s O{1} m} {<déf.>}
```

Avec

Code

```
\truc*{AA}
```

on a

- #1 = \BooleanTrue ;

Premier exemple

Code

```
\NewDocumentCommand {\truc} { s O{1} m} {<déf.>}
```

Avec

Code

```
\truc*{AA}
```

on a

- #1 = `\BooleanTrue`;
- #2 = 1;

Premier exemple

Code

```
\NewDocumentCommand {\truc} { s O{1} m} {<déf.>}
```

Avec

Code

```
\truc*{AA}
```

on a

- #1 = `\BooleanTrue`;
- #2 = 1;
- #3 = AA.

Contrôler la présence d'un argument

Les trois macros `\IfNoValue`*TF* permettent de contrôler la présence d'un argument optionnel.

Contrôler la présence d'un argument

Les trois macros `\IfNoValue`*TF* permettent de contrôler la présence d'un argument optionnel.

Code

```
\IfNoValueTF{<arg>}{<si vrai>}{<si faux>}
```

Contrôler la présence d'un argument

Les trois macros `\IfNoValue`*TF* permettent de contrôler la présence d'un argument optionnel.

Code

```
\IfNoValueTF{<arg>}{<si vrai>}{<si faux>}
\IfNoValueT{<arg>}{<si vrai>}
```

Contrôler la présence d'un argument

Les trois macros `\IfNoValue`*TF* permettent de contrôler la présence d'un argument optionnel.

Code

```
\IfNoValueTF{<arg>}{<si vrai>}{<si faux>}
\IfNoValueT{<arg>}{<si vrai>}
\IfNoValueF{<arg>}{<si faux>}
```

Contrôler la présence d'un argument

Les trois macros `\IfNoValue`*TF* permettent de contrôler la présence d'un argument optionnel.

Code

```
\IfNoValueTF{<arg>}{<si vrai>}{<si faux>}
\IfNoValueT{<arg>}{<si vrai>}
\IfNoValueF{<arg>}{<si faux>}
```

Contrôler la présence d'un argument

Les trois macros `\IfNoValue`*TF* permettent de contrôler la présence d'un argument optionnel.

Code

```
\IfNoValueTF{<arg>}{<si vrai>}{<si faux>}
\IfNoValueT{<arg>}{<si vrai>}
\IfNoValueF{<arg>}{<si faux>}
```

Les macros `\IfValue`*TF* sont les inverses des précédentes.

Tenir compte d'un argument optionnel

```
\DeclareDocumentCommand \truc { o m }
{
  \IfNoValueTF {#1}
  { \FaireUnTrucAvecUnArgumentObligatoire {#2} }
  { \FaireUnTrucAvecLesDeuxArguments {#1} {#2} }
}
```

Contrôler la présence d'un lexème optionnel

Les trois macros `\IfBoolean`*TF* permettent de contrôler la présence d'un lexème optionnel déclaré avec `t` ou `s`.

Contrôler la présence d'un lexème optionnel

Les trois macros `\IfBoolean`*TF* permettent de contrôler la présence d'un lexème optionnel déclaré avec `t` ou `s`.

Code

```
\IfBooleanTF<arg>{<si vrai>}{<si faux>}
```

Contrôler la présence d'un lexème optionnel

Les trois macros `\IfBoolean`*TF* permettent de contrôler la présence d'un lexème optionnel déclaré avec `t` ou `s`.

Code

```
\IfBooleanTF<arg>{<si vrai>}{<si faux>}
\IfBooleanT<arg>{<si vrai>}
```

Contrôler la présence d'un lexème optionnel

Les trois macros `\IfBoolean`*TF* permettent de contrôler la présence d'un lexème optionnel déclaré avec `t` ou `s`.

Code

```
\IfBooleanTF<arg>{<si vrai>}{<si faux>}
\IfBooleanT<arg>{<si vrai>}
\IfBooleanF<arg>{<si faux>}
```

Contrôler la présence d'un lexème optionnel

Les trois macros `\IfBoolean`*TF* permettent de contrôler la présence d'un lexème optionnel déclaré avec `t` ou `s`.

Code

```
\IfBooleanTF<arg>{<si vrai>}{<si faux>}
\IfBooleanT<arg>{<si vrai>}
\IfBooleanF<arg>{<si faux>}
```

Tenir compte d'une étoile

```
\DeclareDocumentCommand \truc { s m }
{
  \IfBooleanTF #1
  { \CommandeAvecEtoile {#2} }
  { \CommandeSansEtoile {#2} }
}
```

Prétraitement des arguments

À savoir

`xparse` permet le prétraitement des arguments.

Prétraitement des arguments

À savoir

`xparse` permet le prétraitement des arguments.

Code

```
>{\ProcesseurB} >{\ProcesseurA} m
```

Quelques processeurs prédéfinis

À savoir

`xparse` ne définit que quelques processeurs.

Quelques processeurs prédéfinis

À savoir

`xparse` ne définit que quelques processeurs.

Code

```
\ReverseBoolean
```

Quelques processeurs prédéfinis

À savoir

`xparse` ne définit que quelques processeurs.

Code

```
\ReverseBoolean
\SplitArgument
```

Quelques processeurs prédéfinis

À savoir

`xparse` ne définit que quelques processeurs.

Code

```
\ReverseBoolean
\SplitArgument
\SplitList
```

Quelques processeurs prédéfinis

À savoir

`xparse` ne définit que quelques processeurs.

Code

```
\ReverseBoolean
\SplitArgument
\SplitList (\ProcessList)
```

Quelques processeurs prédéfinis

À savoir

`xparse` ne définit que quelques processeurs.

Code

```
\ReverseBoolean
\SplitArgument
\SplitList (\ProcessList)
\TrimSpaces
```

Quelques processeurs prédéfinis

À savoir

`xparse` ne définit que quelques processeurs.

Code

```
\ReverseBoolean
\SplitArgument
\SplitList (\ProcessList)
\TrimSpaces
```

Remarque

Attention !

Toutes les définitions qui vont suivre sont placées entre `\ExplSyntaxOn` et `\ExplSyntaxOff`.

Processeur \SplitArgument
Définitions

```
\NewDocumentCommand {\ladate}
  { > { \SplitArgument { 2 } { / } } m }
  { \EcrireDate #1 }

\NewDocumentCommand {\EcrireDate} { m m m } {
  \IfNoValueTF{#3}{\the\year}{#3}-
  \IfNoValueTF{#2}{\the\month}{#2}-
  #1
}
```

Processeur \SplitArgument
Utilisations

- `\ladate{28/5/2014}` ☞ 2014-5-28 ;

Processeur \SplitArgument
Utilisations

- `\ladate{28/5/2014}` ☞ 2014-5-28 ;
- `\ladate{28/5}` ☞ 2015-5-28 ;

Processeur \SplitArgument
Utilisations

- `\ladate{28/5/2014}` ☞ 2014-5-28 ;
- `\ladate{28/5}` ☞ 2015-5-28 ;
- `\ladate{28}` ☞ 2015-7-28.

Processeur \SplitList
Définitions

```
\NewDocumentCommand {\Traitera} { }
  { \quad \textcolor{red} }

\NewDocumentCommand {\fooa}
  { > { \SplitList { ; } } m }
  { \ProcessList {#1} { \Traitera } }
%%-------------------------------------
\NewDocumentCommand {\Traiterb} { m }
  { \quad |\textcolor{red}{#1}| }

\NewDocumentCommand {\foob}
  { > { \SplitList { ; } } m }
  { \ProcessList {#1} { \Traiterb } }
```

Processeur \SplitList
Utilisations

- `\fooa{aa; bb; cc; dd; ee}` ☞
  aa bb cc dd ee;

Processeur \SplitList
Utilisations

- `\fooa{aa; bb; cc; dd; ee}` ☞
  aa bb cc dd ee;
- `\foob{aa; bb; cc; dd; ee}` ☞
  |aa| |bb| |cc| |dd| |ee|.

Processeur \TrimSpaces

```
\NewDocumentCommand {\truc}
  { >{\TrimSpaces} m}
  { |#1| }
```

Processeur \TrimSpaces

```
\NewDocumentCommand {\truc}
  { >{\TrimSpaces} m}
  { |#1| }
```

\truc{␣aaa␣␣␣␣} ☞ |aaa|

Le signe plus dans les spécificateurs d'arguments

À savoir

Le signe `+` est utilisé devant un spécificateur d'argument (`m`, `o`, &c) pour faire de cet argument un argument *long* c.-à-d. un argument qui peut contenir un saut de paragraphe.

Le signe plus dans les spécificateurs d'arguments

À savoir

Le signe + est utilisé devant un spécificateur d'argument (m, o, &c) pour faire de cet argument un argument *long* c.-à-d. un argument qui peut contenir un saut de paragraphe.

Code

```
\NewDocumentCommand { \chose } { o +m }
{<définition>}
```

Le signe plus dans les spécificateurs d'arguments

À savoir

Le signe + est utilisé devant un spécificateur d'argument (m, o, &c) pour faire de cet argument un argument *long* c.-à-d. un argument qui peut contenir un saut de paragraphe.

Code

```
\NewDocumentCommand { \chose } { o +m }
{<définition>}
```