Xparse and EkeysExt[1]

Eureka

October 7, 2024

[1]Read usrguide.pdf for more details.

Contents

Chapter 1

Xparse

1.1 argument sepcification

Three more characters have a special meaning when creating an argument specifier:

- First, `+` is used to make an argument **long** (to accept paragraph tokens-`\par`).
- Secondly, `>` is used to declare so-called "argument processors", see below.
- Thirdly, `!` is used to stress the **space(s)**, such that `\NewDocumentCommand\foobar{ m!o } {...}` and `\foobar{arg1} [arg2]` will raise error.
- Finally, `=` is used to declare that the following argument should be interpreted as a series of keyvals.

Recommendation: A very common syntax is to have one optional argument `o` treated as a key–value list (using for instance `l3keys`) followed by some mandatory arguments `m` (or `+m`).

Other argument type: `l, u` (mandatory), and the corresponding `g, G`(optional) is not reccomended to use.
An example of modify `=`:

```
\DeclareDocumentCommand\foo{=+{keyA}o m}
{%
  \texttt{ARGS:#1}|{\color{red}#2}\par%
}

\foo[long-text-A]{short-text-1}
\foo[keyA=long-text-A]{short-text-2}
\foo[keyB=New-text-B]{short-text-3}
\foo[{keyC=New-text-C}]{short-text-4}
\foo[=,New-text-D]{short-text-5}
```

ARGS:keyA=long-text-A|short-text-1
ARGS:keyA=long-text-A|short-text-2
ARGS:keyB=New-text-B|short-text-3
ARGS:keyA=keyC=New-text-C|short-text-4
ARGS:New-text-D|short-text-5

1.1.1 basic intro

The mandatory types are: `m, r, R, v, b`; The types which define optional arguments are: `o, d, D, s, t, e, E`, The first three types are **corresponding**.

```
\NewDocumentCommand{\foobar}{R(){DEFAULT-
ARG}}
{
  \#1~ is:~ #1
}
\foobar()\par
\foobar(a)\par
% \foobar
% > raise error and leave: "DEFAULT-ARG"
into the stream.
```

#1 is:
#1 is: a

To avoid the “argument missing error", you should use “The types which define optional arguments", the corresponding types for `R` is `D`.

```
\NewDocumentCommand{\foobar}{D(){DEFAULT-
ARG}}
{
  \#1~ is:~ #1
}
\foobar()\par
\foobar(a)\par
\foobar
```

#1 is:
#1 is: a
#1 is: DEFAULT-ARG

1.1.2 special case Example

- v-type:

```
\NewDocumentCommand\foo{v}
{
  BEFORE-(#1)-AFTER
}

\foo|\hello #1|\par
\foo{\hello #1}
```

BEFORE-(\hello #1)-AFTER
BEFORE-(\hello #1)-AFTER

- b-type:

```
% The prefix '+' is used to allow
multiple paragraphs in the environment's
body.
\NewDocumentEnvironment { twice }
  { O{\ttfamily} +b }
  {#2#1#2} {}

\begin{twice}[\itshape]
  Hello world!
\end{twice}
```

Hello world!*Hello world!*

- s-type:

```
% 's' must be the first argument type.
\NewDocumentCommand\foo{sm}
{
  \IfBooleanTF{#1}{TRUE--#2}{FALSE--#2}
}

\foo{ARG}\par
\foo*{ARG}
```

FALSE–ARG
TRUE–ARG

- t-type:; corresponding to optional type `s`:

```
% 't' must be the first argument type.
% \NewDocumentCommand\foo{t{|}m}{
% or
\NewDocumentCommand\foo{t|m}
{
  \IfBooleanTF{#1}{TRUE--#2}{FALSE--#2}
}

\foo{ARG}\par
\foo|{ARG}
```

FALSE–ARG
TRUE–ARG

- **!**-type:; Providing a simple example:

```
% amsmath package signature:
% \DeclareDocumentCommand \\ {!s !o}{...}
% When display breaks are enabled with
\allowdisplaybreaks, the \\* command can
be used to prohibit a pagebreak after a
given line, as usual.

\begin{align}
  a = 1 \\[1em]
  b = 2
\end{align}

\begin{align}
  a = 1 \\*[1em]
  b = 2
\end{align}

\begin{align}
  a = 1 \\* [1em]
  b = 2
\end{align}

\begin{align}
  a = 1 \\ *[1em]
  b = 2
\end{align}
```

$$a = 1 \tag{1.1}$$
$$b = 2 \tag{1.2}$$
$$a = 1 \tag{1.3}$$
$$b = 2 \tag{1.4}$$
$$a = 1 \tag{1.5}$$
$$[1em]b = 2 \tag{1.6}$$
$$a = 1 \tag{1.7}$$
$$*[1em]b = 2 \tag{1.8}$$

- e-type:

```
\NewDocumentCommand\foo{e:e|}
{
  \#1~ is:~ #1\par
  \#2~ is:~ #2\par
}
\NewDocumentCommand\foobar{e{:|}}
{
  \#1~ is:~ #1\par
  \#2~ is:~ #2\par
}

\foo:{ARG-A}|{ARG-B}\vskip2em
\foo|{ARG-B}:{ARG-A}

\dotfill\par
\foobar:{ARG-A}|{ARG-B}\vskip2em
\foobar|{ARG-B}:{ARG-A}
```

#1 is: ARG-A
#2 is: ARG-B

#1 is: -NoValue-
#2 is: ARG-B
:ARG-A
.....
#1 is: ARG-A
#2 is: ARG-B

#1 is: ARG-A
#2 is: ARG-B

- E-type:

```
\NewDocumentCommand\foo{E{:|/}{{ARG-1}{AR
G-2}{ARG-3}}}
{
  \#1~ is:~ #1\par
  \#2~ is:~ #2\par
  \#3~ is:~ #3\par
}

\foo:{ARG-A}/{ARG-C}
```

#1 is: ARG-A
#2 is: ARG-2
#3 is: ARG-C

1.2 Argument Processor

Syntax: `>{<processor>}` in the specification, `>{\ProcessorB} >{\ProcessorA} m` would apply `\ProcessorA` followed by `\ProcessorB` to the tokens grabbed by the `m` argument.

1.2.1 SplitArgument Processor

Split the argument by the separator:

```
\ExplSyntaxOn
\NewDocumentCommand\foo{>{\SplitArgument{
2}{,}}m}
{
  \tl_set:Nn \l_tmpa_tl {#1}
  % \tl_show:N \l_tmpa_tl
  % > \l_tmpa_tl=\{a\}\{b\}\{c\}.
  \argsRead #1
}
\NewDocumentCommand\argsRead{mmm}
{
  \#1-(1)~ is:~ { #1 }\par
  \#1-(2)~ is:~ { #2 }\par
  \#1-(3)~ is:~ { #3 }\par
}
\ExplSyntaxOff

\foo{a, b, c}
```

#1-(1) is: a
#1-(2) is: b
#1-(3) is: c

Before `#1` passed to command `\argsRead`, it has been splited into 3 parts, and passed to `\argsRead` as 3 arguments.(`something like {#1.1}{#1.2}{#1.3} inside of #1`). Thus we can use LaTeX3 based functions `\use_i:nn {}{}, \use_ii:nn {}{}` to get each part.

```
\ExplSyntaxOn
\NewDocumentCommand\foo{>{\SplitArgument{
1}{,}}m}
{
  \#1-(1)~ is:~ \use_i:nn #1\par
  \#1-(2)~ is:~ \use_ii:nn #1
}
\ExplSyntaxOff

\foo{a, b}
```

#1-(1) is: a
#1-(2) is: b

1.2.2 SplitList Processor

```
\ExplSyntaxOn
\NewDocumentCommand\foo{>{\SplitList{|}}m
}
{
  \tl_set:Nn \l_tmpa_tl {#1}
  % \tl_show:N \l_tmpa_tl
  % > \l_tmpa_tl=\{a\}\{b\}\{c\}\{d\}.
  \use_iii:nnnn #1
}
\ExplSyntaxOff

\foo{a | b | c | d}
```

c

1.2.3 ProcessList Processor

```
\ExplSyntaxOn
\NewDocumentCommand\foo{
>{\SplitList{;}}m }
{
  \ProcessList {#1} { \addBrace }
}
\NewDocumentCommand\addBrace{m}{
  (#1)\par
}
\ExplSyntaxOff

\foo{a ; b ; c ; d}
```

(a)
(b)
(c)
(d)

1.2.4 TrimSpaces Processor

Use command: `>{\TrimSpaces}m`

1.2.5 ProcessedArgument Processor

This processor provide a way to process the argument using our own function.

```
\ExplSyntaxOn
\NewDocumentCommand\foo{
>{\_eq_and_braket:n}m }
{
  \#1~ is:~ #1
}
\cs_set:Npn \_eq_and_braket:n #1
{
  \tl_set:Nn \ProcessedArgument
  {[=~#1~=]}
  % or in this way
  % \def\ProcessedArgument{[=~#1~=]}
}
\ExplSyntaxOff

\foo{a}
```

#1 is: [= a =]

1.3 Expandable command and environment

Parsing arguments expandably imposes a number of restrictions on both the type of arguments that can be read and the error checking available:

- The last argument (if any are present) must be one of the mandatory types `m, r, R, l, u`.

- All short arguments appear before long arguments.
- The mandatory argument types `l` and `u` may not be used after optional arguments.
- The optional argument types `g` and `G` are not available.
- The "verbatim" argument type `v` is not available.
- Argument processors (using `>`) are not available.
- ..., checking for optional arguments is less robust than in the standard version.

1.4 Access to the argument specification

```
\ExplSyntaxOn
\NewDocumentCommand\foo{oO{DEFAULT}R()me{
:|}m}{}
\GetDocumentCommandArgSpec \foo
\tl_to_str:N \ArgumentSpecification
% \tl_show:N \ArgumentSpecification
% > \ArgumentSpecification=oO{DEFAULT}R()
me{:|}m.
\ExplSyntaxOff
```

oO{DEFAULT}R()me{:|}m

Chapter 2

EkeysExt

This package is too difficult for me. Wait a moment.