The lambdax package[*]

Erwann Rogard[†]

Released 2021-08-03

Abstract

This is a LaTeXpackage that provides 'lambda expressions', in other words an interface by which one consecutively, first, specifies the parameters and replacement code of a document-command[**xparse**], and, second, evaluates it with compatible arguments. Optionally, one can recurse. For example, `\LambdaX[mm]<t^>{$#1$#2}{x}{,~}^{y}{~and~}^{z}{.}`, where `<.>` specifies the recurse paramater, expands to "x, y and z."

Part I
Usage

Contents

[*]This file describes version v1.0, last revised 2021-08-03.

[†]first.lastname at gmail.com

1 Settings

The options hereafter are load-time-only.

xparse-command

Side effect Sets the xparse-document-command used by `\lambdax:nn`
Initial `\DeclareDocumentCommand`

2 Programming

`\lambdax:nn`

`\lambdax:nn{⟨argspec⟩}{⟨code⟩}⟨args⟩`

Expands to ⟨*code*⟩, ⟨*args*⟩ replacing the parameters implied by ⟨*argspec*⟩

`\lambdax:nnn`

`\lambdax:nn{⟨argspec⟩}{⟨code⟩}{⟨bool-arg-type⟩}⟨args⟩⟨bool-arg⟩`

Limitation That of lex[**erw-lex**]'s `argspec` collection.
Argspec Examples of ⟨*bool-arg-type*⟩[**xparse**] and ⟨*bool-arg*⟩ are `s` and `*`, respectively.
Semantics That of `\lambdax:nn` and recurse if applicable.

`\lambdax_xcmd_if:NTF` ⋆

`\lambdax_xcmd_if:NTF:Nn⟨xparse-command⟩{⟨code if true⟩}{⟨code if false⟩}`

3 Document

`\LambdaX`

`\LambdaX[⟨argspec⟩]<⟨bool-arg-type⟩>{⟨code⟩}`

Adapts `\lambda:nn` and `\lambda:nnn`

Part II
Other

1 Acknowledgment

The basis for `\lambdax:nn` originates with [**a-188053**]. Except for chaining, it was already provided by [**erw-ccool**].

2 Bibliograhy

Part III
Implementation

```
⟨*package⟩
⟨@@=lambdax⟩
\ExplSyntaxOn
```

1 Auxiliary

```
\cs_generate_variant:Nn\tl_count:n{e}
\cs_generate_variant:Nn\int_eval:n{e}
\cs_generate_variant:Nn\bool_if:nT{o, e}
```

__lambdax_str_case_empty:n

```
\cs_new:Nn
\__lambdax_str_case_empty:n
{{#1}
  {\c_empty_tl}}
```

(*End definition for* `\__lambdax_str_case_empty:n`.)

2 xcmdif

not-xparse

```
\msg_new:nnn{__lambdax}
{not-xparse}
{Expecting~an~xparse~command,~got~#2}
```

(*End definition for* `not-xparse`.)

\c__lambdax_xcmdname_tl

```
\tl_const:Nn
\c__lambdax_xcmdname_tl
{  {NewDocumentCommand}
  {RenewDocumentCommand}
  {ProvideDocumentCommand}
  {DeclareDocumentCommand}
  {NewExpandableDocumentCommand}
  {RenewExpandableDocumentCommand}
  {ProvideExpandableDocumentCommand}
  {DeclareExpandableDocumentCommand} }
```

(*End definition for* `\c__lambdax_xcmdname_tl`.)

__lambdax_xcmd_if:nTF
__lambdax_xcmd_if:eTF
\lambdax_xcmd_if:NTF
__lambdax_xcmd_else_error:Nn

```
\prg_new_conditional:Nnn
\__lambdax_xcmd_if:n{TF}
{\exp_args:Nnx
  \str_case:nnTF{#1}
  {  \tl_map_function:NN
    \c__lambdax_xcmdname_tl
    \__lambdax_str_case_empty:n}
  {\prg_return_true:}
  {\prg_return_false:}}
\cs_generate_variant:Nn\__lambdax_xcmd_if:nTF{e}
\cs_new:Nn
\lambdax_xcmd_if:NTF
{\__lambdax_xcmd_if:eTF
  {\cs_to_str:N#1}{#2}{#3}}
\cs_new:Nn
\__lambdax_xcmd_else_error:Nn
{  \lambdax_xcmd_if:NTF#1
  { #2 }
  { \msg_error:nne{__lambdax}
    {not-xparse}
    {\token_to_str:N#1} } }
```

(*End definition for* `\__lambdax_xcmd_if:nTF`*,* `\lambdax_xcmd_if:NTF`*, and* `\__lambdax_xcmd_else_error:Nn`*. This function is documented on page 1.*)

`\c__lambdax_xenv_tl`

```
\tl_const:Nn
\c__lambdax_xenv_tl
{ {NewDocumentEnvironment}
  {RenewDocumentEnvironment}
  {ProvideDocumentEnvironment}
  {DeclareDocumentEnvironment} }
```

(*End definition for* `\c__lambdax_xenv_tl`*.*)

`\__lambdax_msg_name:n`

```
\cs_new:Nn
\__lambdax_msg_name:n{msg_\g__lambdax_opt_msg_tl{}:#1}
```

(*End definition for* `\__lambdax_msg_name:n`*.*)

3 lambda

`\__lambdax_placeholder:n`
`\__lambdax_placeholder:e`
`\__lambdax_argspec:n`
`\__lambdax_argspec_count:n`
`\__lambdax_chain_position:n`
`\__lambdax_chain_placeholder:n`

```
\cs_new:Nn\__lambdax_placeholder:n{#### #1}
\cs_generate_variant:Nn\__lambdax_placeholder:n{o,e}
\cs_new:Nn\__lambdax_argspec:n{\lex_eval:nn{argspec}{#1}}
\cs_new:Nn\__lambdax_argspec_count:n{\tl_count:e{\__lambdax_argspec:n{#1}}}
\cs_new:Nn\__lambdax_chain_position:n{\int_eval:e{\__lambdax_argspec_count:n{#1}+1}}
\cs_new:Nn\__lambdax_chain_placeholder:n
{\__lambdax_placeholder:e
  {\__lambdax_chain_position:n{#1}}}
```

(*End definition for* `\__lambdax_placeholder:n` *and others.*)

`\__lambdax_lambda:Nnn`
`\__lambdax_lambda_dev:N`
`\__lambdax_lambda_doc:NN`

```
\cs_new_protected:Nn \__lambdax_lambda:Nnn
{\exp_args:NNx
  #1 \__lambdax_lambda
  {#2}
  {#3}
  \__lambdax_lambda}
\cs_generate_variant:Nn\__lambdax_lambda:N{c}
\cs_new_protected:Nn
\__lambdax_lambda_chain:Nnnn
{ \tl_set:Nn
  \l__lambdax_head_tl
  {\exp_args:NNx#1 \__lambdax_lambda_chain
    {#2#3} }
  \exp_args:Nx
  \l__lambdax_head_tl
  {\exp_not:n{#4} \exp_not:N
    \bool_if:oT
    {\__lambdax_chain_placeholder:n{#2}}
    {\exp_not:N\__lambdax_lambda_chain}}
  \__lambdax_lambda_chain}
\cs_set_protected:Nn
\__lambdax_lambda_dev:N
{ \cs_new_protected:Nn
  \lambdax:nn
  {  \__lambdax_lambda:Nnn #1
    {##1}{##2} }
  \cs_new_protected:Nn
  \lambdax:nnn
  {  \__lambdax_lambda_chain:Nnnn #1
    {##1}{##2}{##3} } }
\cs_set_protected:Nn
\__lambdax_lambda_doc:N
{ \NewDocumentCommand
  #1 { O{m} d<>  m }
  {\IfValueTF{##2}
    { \lambdax:nnn { ##1 } { ##2 } { ##3 } }
    { \lambdax:nn { ##1 } { ##3 } } } }
\cs_generate_variant:Nn\__lambdax_lambda_doc:N{c}
```

(*End definition for* `\__lambdax_lambda:Nnn`*,* `\__lambdax_lambda_dev:N`*, and* `\__lambdax_lambda_doc:NN`*.*)

4 Settings

```
\keys_define:nn{ __lambdax }
{ dev.code:n = {
    \__lambdax_xcmd_else_error:Nn#1
    {\__lambdax_lambda_dev:N#1 }
  },
  internal / document-command-name.code:n = { \__lambdax_lambda_doc:c{#1} },
  internal / document-command-name.initial:n = { LambdaX },
  xparse-command.code:n =
  { \__lambdax_xcmd_else_error:Nn #1
    { \keys_set:nn{ __lambdax }{ dev = #1 } } },
  xparse-command .initial:n = { \DeclareDocumentCommand }
}
```

```
\ProcessKeysOptions{__lambdax}
\ExplSyntaxOff
```

```
⟨/package⟩
```