The xmeaning package*

Valentin Dao†

Released 2025-12-18

Abstract

The *xmeaning* package's main objective is to enhance the \meaning primitive by enlarging the amount of given informationG. It requires *expl3* as well as *pkginfograb*. Please note that this package is considered to be highly experimental for the moment.

Limitations

The \xmeaning macro will fail under certain precise circumstances:

1. If the parameter text *or* the replacement text of a macro contains "->".
2. If you use \xmeaning for a constant defined using \chardef or \mathchardef.

Whether the compilation stops or succeeds, but with erroneous results, completely depends on what the argument is.

\xmeaning `\xmeaning* {⟨list of tokens⟩}`

On its simplest form, \xmeaning has a strictly similar usage to that of \meaning. The main argument can either correspond to a character or a control sequence. With *xmeaning*, it is possible to specify a list of tokens to display all their meanings, avoiding the need to repeat the macro. Unlike the TEX primitive, which doesn't always provide useful information, \xmeaning is capable of recognising 5 *token classes* among:

`Characters` A unique token.

`Character definitions` A control sequence declared with \chardef or \mathchardef.

`TEX registers` Control sequences representing boxes, counts…

`expl3 data types` All the variable types implemented by LATEX3.

`Macros` A macro, or any control sequence that doesn't fall into the previous classes.

By *class*, we mean that the macro will produce an output adapted to each token type. Its starred variant will display the meaning of all auxiliary macros as described later.

*This file describes vv0.1.0, published 2025-12-18.

†E-mail: vdao.texdev@gmail.com

1 Characters

Unlike the `meaning` primitive, which gives us only little information regarding characters, the package lists all there is to know about them.

- Character code, with its ASCII, Unicode, octal and hexadecimal representations.
- Category codes.
- Upper or lower case code (depending on the case of the queried character)†.
- Space factor code†.
- Delimiter code.
- Math code†.

† These codes will be displayed only if they are not null.

For math code, the package goes beyond simply displaying it, as it analyses it according to its functioning as described in the TEXbook. The same applies to the delimiter code if it is not 0. Furthermore, it is not necessary to escape special characters with a backslash. The macro may produce incorrect results if you pass it an argument that is not recognised by the encoding. If you are using an engine other than XeLaTeX or LuaLaTeX, keep this in mind.

LaTeX Code:

```
\xmeaning o
```

LaTeX Result:

```
            Glyph = the letter o
    Category code = 11
   Character code = 111
          Unicode = U+006F
            Octal = 157
      Hexadecimal = 6f
   Uppercase code = 79 (O)
Space factor code = 1000
        Math code = 1e0006f
                    Class: Variable family
                    Math family: 1
                    Font position: 111
```

2 Character definitions

The package calls any character defined by `\chardef` and `\mathchardef` a *character definition*. The advantage over the primitive `\meaning` is that here the glyph is produced, allowing you to see the concrete result of the control sequence. It is precisely for this reason that constants defined with these two macros do not work (for example, certain plainTeX constants such as `\z@`, `\@M`…). They correspond to numbers and not to real characters. Since the package cannot tell the difference, it will attempt to typeset a glyph with a character code that is certainly outside those defined by the font table. It also indicates whether the character should be used in text mode or math mode.

LaTeX Code:

```
\xmeaning{\%, \gamma}
```

LaTeX Result:

```
Character definition = \%
               Glyph = %
                Mode = text

Character definition = \gamma
               Glyph = γ
                Mode = math
```

3 Registers

For registers, the package simply displays its type as well as its content.

LaTeX Code:

```
\newtoks\temptoks
\temptoks={Token register}
\xmeaning\temptoks
```

LaTeX Result:

```
Register name = \temptoks
         Type = token (no. 35)
      Content = Token register
```

4 Data types

LaTeX3 data types are much more complex than registers. That is why a few more entries compared to registers are present. `\xmeaning` will fail if the naming conventions for data types are not followed.

LaTeX Code:

```
\ExplSyntaxOn
\tl_new:N \l_my_token_tl
\tl_set:Nn \l_my_token_tl { Token~data~type }
\xmeaning\l_my_token_tl
\ExplSyntaxOff
```

LaTeX Result:

```
 Variable = \l_my_token_tl
Data type = token
  Content = Token data type
    Scope = local
      Use = public
```

5 Macros

The package produces a very complete output for macros, where the following information is typeset:

- The name of the macro.
- The type of macro.
- The prefix of the macro (`\long`, `\protected`, and `\outer`)†.
- The parameter text†.
- The replacement text†.

†Here too, these elements are only displayed if they are present. For primitives, they will always be absent.

The macro *type* can be one of the following:

- TEXprimitive.
- TEXmacro.
- *expl3* primitive.
- *expl3* macro.
- argument specifier *expl3*, with distinction of the *unbraced* variants.
- *expl3* identity macro.
- *xparse* macro.

LaTeX Code:

```
\xmeaning{\kern, \newcommand}
```

LaTeX Result:

```
Control sequence = \kern
            Type = TeX primitive

Control sequence = \newcommand
            Type = TeX macro
Replacement text = \@star@or@long \new@command
```

6 Exhaustive meaning

With the starred variant of `\xmeaning`, it is possible to ask the package to display the meaning of all the auxiliary macros that make up the main argument. There is, however, no magical formula for knowing in advance what these auxiliary macros will be. The package therefore assumes that standard programming conventions are followed and, using a brute-force method, recursively applies `\xmeaning` to all of the following macros:

- `\foo`⟨␣⟩.
- `\foo`⟨␣`code`⟩ (for macros defined with *xparse*).
- `\foo`⟨`A-Z`⟩ (`\fooA`, `\fooB`…).
- ⟨`A-Z`⟩`\foo` (`\Afoo`, `\Bfoo`…).

- Nesting of @ (`\@foo`, `\f@oo…\foo@`).
- Replacement of *o* and *a* by @ (`\f@@`, `\b@r`).
- `\foo`⟨`@Roman numeral`⟩ (`\foo@ii`).
- ⟨`__`⟩`\foo` (for internal macros of *expl3*).

This method does not guarantee that if several macros are found, they are related to each other. Nor is it certain that auxiliary macros will be found even after applying all these rules, which are not absolute (after all, everyone is free to name their macros as they see fit).

LaTeX Code:

```
\xmeaning*{\newcommand, \integral}
```

LaTeX Result:

```
Control sequence = \newcommand
            Type = TeX macro
Replacement text = \@star@or@long \new@command

Control sequence = \@newcommand
            Type = TeX macro
      Parameters = #1[#2]
Replacement text = \kernel@ifnextchar [{\@xargdef #1[#2]}{\@argdef #1[#2]}

Control sequence = \new@command
            Type = TeX macro
      Parameters = #1
Replacement text = \@testopt {\@newcommand #1}0

Control sequence = \integral
            Type = TeX macro
          Prefix = \protected
Replacement text = \__cmd_start:nNNnnn {O{}m}\integral  \integral code {\__cmd_grab_D:
                   []\__cmd_grab_m_1:w }{{\prg_do_nothing: }\c_novalue_tl }{}

Control sequence = \integral␣code
            Type = xparse macro
          Prefix = \protected\long
      Parameters = #1#2
Replacement text = \group_begin: \tl_if_empty:nF
                   {#1}{\__intexgral_pkg_extract_first_key_name:n
                   {#1}\__intexgral_pkg_parse_macro_keys:n {#1}}\tl_set:Nn
                   \l__intexgral_pkg_integrand_tl {#2}\__intexgral_pkg_print_integral:
                   \group_end:
```

7 Implementation

```
⟨*package⟩
⟨@@=xmeaning_pkg⟩

\NeedsTeXFormat{LaTeX2e}
\RequirePackage{expl3}[2025-05-14]
\RequirePackage{pkginfograb}
\pkginfograbProvidesExplPackage{xmeaning}
  {
    name          = {xmeaning}  ,
    author        = {Valentin Dao},
    date          = {2025-12-18},
    creation      = {2025-12-18},
    version       = {v0.1.0}    ,
    description   = {Enhanced implementation of the \string\meaning\ primitive.}
  }

\@ifpackagelater{expl3}{2025-05-14}{}
  {
    \PackageError { xmeaning } { Support~package~expl3~too~old }
      {
        expl3~version~too~old~
        'xmeaning'~requires~a~version~later~than:~2025-05-14.
        \MessageBreak
        Loading~xmeaning~will~abort.
      }
    \tex_endinput:D
  }

\cs_new:Npn \__xmeaning_pkg_char_value_delcode:n #1 {
  \tex_number:D \tex_delcode:D #1 \scan_stop:
}

\cs_new_protected:Npn \__xmeaning_pkg_tex_halign:n #1 {
  \tex_halign:D { #1 }
}

\cs_generate_variant:Nn \regex_if_match:nnTF { neTF }
\cs_generate_variant:Nn \regex_match_case:nn { ne }
\cs_generate_variant:Nn \regex_match_case:nnF { neF }
\cs_generate_variant:Nn \regex_extract_all:nnN { neN }
\cs_generate_variant:Nn \int_to_hex:n { e }
\cs_generate_variant:Nn \int_from_hex:n { e }
\cs_generate_variant:Nn \int_case:nn { Vn }
\cs_generate_variant:Nn \tl_range:nnn { nnV }
\cs_generate_variant:Nn \char_generate:nn { en }
\cs_generate_variant:Nn \token_if_cs:NTF { cTF }
\cs_generate_variant:Nn \clist_put_right:Nn { Nf }
\cs_generate_variant:Nn \int_compare_p:nNn { nNV }

\int_new:N \l__xmeaning_pkg_char_math_fam_int
\int_new:N \l__xmeaning_pkg_char_math_slot_int
\int_new:N \l__xmeaning_pkg_small_variant_family_int
```

```
\int_new:N \l__xmeaning_pkg_small_variant_position_int
\int_new:N \l__xmeaning_pkg_big_variant_family_int
\int_new:N \l__xmeaning_pkg_big_variant_position_int

\bool_new:N \l__xmeaning_pkg_undefined_cs_bool
\bool_set_false:N \l__xmeaning_pkg_undefined_cs_bool
\bool_new:N \l__xmeaning_pkg_exhaustive_meaning_bool
\bool_set_false:N \l__xmeaning_pkg_exhaustive_meaning_bool

\tl_new:N \l__xmeaning_pkg_register_type_tl

\skip_new:N \l__xmeaning_pkg_meaning_skip

\skip_set:Nn \l__xmeaning_pkg_meaning_skip { 6pt + 3pt - 2pt }

\cs_new_protected:Npn \__xmeaning_pkg_print_clist:c #1
  {
    \begin{enumerate}
    \clist_map_inline:cn { #1 }
      {
        \item ##1
      }
    \end{enumerate}
  }

\cs_new_protected:Npn \__xmeaning_pkg_print_seq:c #1
  {
    \begin{enumerate}
    \seq_map_inline:cn { #1 }
      {
        \item ##1
      }
    \end{enumerate}
  }

\cs_new_protected:Npn \__xmeaning_pkg_print_prop:c #1
  {
    \prop_map_inline:cn { #1 }
      { ##1~->~##2\tex_par:D }
  }

\cs_new_protected:Npn \__xmeaning_pkg_character_meaning:N #1 {
  \int_set:Nn \l_tmpa_int { \int_eval:n { `#1 } }
  \tl_set:Ne \l_tmpb_tl { \int_to_Hex:n { \l_tmpa_int } }
  \tl_set:Ne \l__xmeaning_pkg_utf_eight_representation_tl
    {
      U+
      \prg_replicate:nn { \int_eval:n { 4 - \tl_count:N \l_tmpb_tl } } { 0 }
      \tl_use:N \l_tmpb_tl
    }
  \__xmeaning_pkg_tex_halign:n { \tex_hfil:D##&\c_space_tl=\c_space_tl\vbox_top:n{\raggedrigh
    Glyph & \token_to_meaning:N #1\tex_cr:D
    Category~code & \token_to_catcode:N #1\tex_cr:D
    Character~code & \int_use:N \l_tmpa_int\tex_cr:D
```

```
    Unicode & \tl_use:N \l__xmeaning_pkg_utf_eight_representation_tl\tex_cr:D
    Octal & \int_to_oct:n { \l_tmpa_int }\tex_cr:D
    Hexadecimal & \int_to_hex:n { \l_tmpa_int }\tex_cr:D
    \bool_if:nF
      {
        \int_compare_p:nNn { \char_value_uccode:n { `#1 } } = { 0 }
        ||
        \int_compare_p:nNV { \char_value_uccode:n { `#1 } } = \l_tmpa_int
      }
      {
        Uppercase~code & \char_value_uccode:n { `#1 }~(\char_generate:nn { \char_value_uccode
      }
    \bool_if:nF
      {
        \int_compare_p:nNn { \char_value_lccode:n { `#1 } } = { 0 }
        ||
        \int_compare_p:nNV { \char_value_lccode:n { `#1 } } = \l_tmpa_int
      }
      {
        Lowercase~code & \char_value_lccode:n { `#1 }~(\char_generate:nn { \char_value_lccode
      }
    Space~factor~code & \char_value_sfcode:n { `#1 } \tex_cr:D
    Math~code & \int_to_hex:e { \char_value_mathcode:n { `#1 } }
    \sys_if_engine_luatex:TF
      {
        \int_set:Nn \l_tmpb_int { \Umathcharclass`#1 }
        \int_set:Nn \l__xmeaning_pkg_char_math_fam_int { \Umathcharfam`#1 }
        \int_set:Nn \l__xmeaning_pkg_char_math_slot_int { \Umathcharslot`#1 }
      }
      {
        \exp_args:NNe \int_set:Nn \l_tmpb_int
          { \tl_head:e { \int_to_hex:e { \char_value_mathcode:n { `#1 } } } }
        \exp_args:NNe \int_set:Nn \l__xmeaning_pkg_char_math_fam_int
          { \exp_args:Ne \tl_item:nn { \int_to_hex:e { \char_value_mathcode:n { `#1 } } } { 2
        \exp_args:NNe \int_set:Nn \l__xmeaning_pkg_char_math_slot_int
          {
            \int_from_hex:e {
            \exp_args:Ne
            \tl_range:nnn
              { \int_to_hex:e { \char_value_mathcode:n { `#1 } } }
              { 3 }
              { 4 }
            }
          }
      }\tex_par:D
      Class:~
        \int_case:Vn \l_tmpb_int
          {
            { 0 } { Ordinary }
            { 1 } { Large~operator }
            { 2 } { Binary~operation }
            { 3 } { Relation }
            { 4 } { Opening }
            { 5 } { Closing }
```

```
            { 6 } { Punctuation }
            { 7 } { Variable~family }
          }\tex_par:D
      Math~family:~\int_use:N \l__xmeaning_pkg_char_math_fam_int\tex_par:D
      Font~position:~\int_use:N \l__xmeaning_pkg_char_math_slot_int\tex_cr:D
    \int_compare:nNnF { \number\delcode`#1 } < { 0 }
      {
        Delimiter~code &
        \int_set:Nn \l__xmeaning_pkg_small_variant_family_int
          {
            \tl_head:e {
              \__xmeaning_pkg_char_value_delcode:n { `#1 }
            }
          }
        \exp_args:NNe
        \int_set:Nn \l__xmeaning_pkg_small_variant_position_int
          {
            \exp_args:Ne
            \tl_range:nnn
              { \__xmeaning_pkg_char_value_delcode:n { `#1 } }
              { 2 }
              { 3 }
          }
        \exp_args:NNe
        \int_set:Nn \l__xmeaning_pkg_big_variant_family_int
          {
            \exp_args:Ne
            \tl_item:nn
              { \__xmeaning_pkg_char_value_delcode:n { `#1 } }
              { 4 }
          }
        \exp_args:NNe
        \int_set:Nn \l__xmeaning_pkg_big_variant_position_int
          {
            \exp_args:Ne
            \tl_range:nnn
              { \__xmeaning_pkg_char_value_delcode:n { `#1 } }
              { 5 }
              { 6 }
          }
        \__xmeaning_pkg_char_value_delcode:n { `#1 } \tex_par:D
        Small~variant~family:~\int_use:N \l__xmeaning_pkg_small_variant_family_int \tex_par:D
        Small~variant~position:~\int_use:N \l__xmeaning_pkg_small_variant_position_int \tex_p
        Big~variant~family:~\int_use:N \l__xmeaning_pkg_big_variant_family_int \tex_par:D
        Big~variant~position:~\int_use:N \l__xmeaning_pkg_big_variant_position_int
      }
    \tex_crcr:D
  }
}

\cs_new_protected:Npn \__xmeaning_pkg_register_meaning:n #1 {
  \regex_extract_all:neN { \\([a-z]+)(\d+) } { \cs_meaning:c { #1 } } \l_tmpa_seq
  \str_case_e:nnF { \seq_item:Nn \l_tmpa_seq { 2 } }
    {
```

```
      { toks   } { \tl_set:Nn \l__xmeaning_pkg_register_type_tl { token     } }
      { dimen  } { \tl_set:Nn \l__xmeaning_pkg_register_type_tl { dimension } }
    }
    { \tl_set:Ne \l__xmeaning_pkg_register_type_tl { \seq_item:Nn \l_tmpa_seq { 2 } } }
  \__xmeaning_pkg_tex_halign:n { \tex_hfil:D###&\c_space_tl=\c_space_tl\vbox_top:n{\raggedrigh
    Register~name & \c_backslash_str #1 \tex_cr:D
    Type & \tl_use:N \l__xmeaning_pkg_register_type_tl\ (no.~\seq_item:Nn \l_tmpa_seq { 3 })\
    Content & \exp_last_unbraced:Ne \tex_the:D { \use:c { #1 } }
    \tex_crcr:D
  }
}

\cs_new_protected:Npn \__xmeaning_pkg_data_type_meaning:n #1 {
  \__xmeaning_pkg_tex_halign:n { \tex_hfil:D###&\c_space_tl=\c_space_tl\vbox_top:n{\raggedrigh
    Variable & \slshape \str_use:N \c_backslash_str \tl_to_str:n { #1 } \tex_cr:D
    Data~type & \str_case_e:nn { \seq_item:Nn \l_tmpb_seq { 4 } }
      {
        { tl       } { token                    }
        { str      } { string                   }
        { seq      } { sequence                 }
        { int      } { integer                  }
        { fp       } { floating~point           }
        { flag     } { flag                     }
        { clist    } { comma~list               }
        { prop     } { property~list            }
        { dim      } { dimension                }
        { intarray } { integer~array            }
        { fparray  } { floating~point~array     }
        { bitset   } { bitset                   }
        { cctab    } { category~code~table      }
        { bool     } { boolean                  }
        { regex    } { regular~expression       }
        { ior      } { input/output~read~stream }
        { iow      } { input/output~write~stream }
        { box      } { box                      }
        { coffin   } { coffin                   }
      } \tex_cr:D
    Content &
    \exp_last_unbraced:Ne
    \scan_stop:
      {
        \str_case_e:nn { \seq_item:Nn \l_tmpb_seq { 4 } }
          {
            { tl       } { \exp_not:N \tl_use:c }
            { str      } { \exp_not:N \str_use:c }
            { seq      } { \exp_not:N \__xmeaning_pkg_print_seq:c }
            { int      } { \exp_not:N \int_use:c }
            { fp       } { \exp_not:N \fp_use:c }
            { flag     } { \exp_not:N \flag_height:c }
            { clist    } { \exp_not:N \__xmeaning_pkg_print_clist:c }
            { prop     } { \exp_not:N \__xmeaning_pkg_print_prop:c }
            { dim      } { \exp_not:N \dim_use:c }
            { bitset   } { \exp_not:N \bitset_use:c }
            { bool     } { \exp_not:N \bool_to_str:c }
```

```
            { box      } { \exp_not:N \box_use:c }
          }
      }
      { #1 }\tex_cr:D
    Scope & \str_case_e:nn { \seq_item:Nn \l_tmpb_seq { 2 } }
      {
        { g } { global   }
        { l } { local    }
        { c } { constant }
      } \tex_cr:D
    Use & \int_case:nn { \int_eval:n { \tl_count:e { \seq_item:Nn \l_tmpb_seq { 3 } } } }
      {
        { 1 } { public }
        { 2 } { private }
      }
    \tex_crcr:D
  }
}

\cs_new_protected:Npn \__xmeaning_pkg_chardef_meaning:n #1 {
  \regex_extract_all:neN { \\(math)?char"([0-9A-Z]+) } { \cs_meaning:c { #1 } } \l_tmpa_seq
  \__xmeaning_pkg_tex_halign:n { \tex_hfil:D##&\c_space_tl=\c_space_tl\vbox_top:n{\raggedrigh
  Character~definition & \c_backslash_str #1 \tex_cr:D
  Glyph &
  \tl_if_empty:eTF { \seq_item:Nn \l_tmpa_seq { 2 } }
    { \char\exp_last_unbraced:Ne"{ \seq_item:Nn \l_tmpa_seq { 3 } } }
    { $ \mathchar\exp_last_unbraced:Ne"{ \seq_item:Nn \l_tmpa_seq { 3 } } $ }
    \tex_cr:D
  Mode & \tl_if_empty:eTF { \seq_item:Nn \l_tmpa_seq { 2 } } { text } { math }
  \tex_crcr:D
  }
}

\cs_new_protected:Npn \__xmeaning_pkg_macro_meaning:n #1 {
  \__xmeaning_pkg_tex_halign:n { \tex_hfil:D##&\c_space_tl=\c_space_tl\vbox_top:n{\raggedrigh
    Control~sequence &
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_replace_all:Nnn \l_tmpa_tl { ~ } { \textvisiblespace }
    \slshape \str_use:N \c_backslash_str \tl_use:N \l_tmpa_tl \tex_cr:D
    Type &
      \exp_args:Nc \token_if_macro:NT { #1 }
        {
          \regex_match_case:nnF
            {
              { \w+:[NnVvcofexTFwD]*          } { expl3~macro }
              { :{2}[NnVvcofexTFwD]_unbraced  } { expl3~argument~specifier~(unbraced~type) }
              { :{2}[NnVvcofexTFwD]\Z         } { expl3~argument~specifier }
              { :{3}                          } { expl3~identity~macro }
              { [a-zA-Z]\s code               } { xparse~macro }
            }
            { #1 }
            { \TeX\ macro }
        }
      \exp_args:Nc \token_if_primitive:NT { #1 }
```

```
        {
          \regex_if_match:nnTF { tex\_\w+:D } { #1 }
            { expl3~primitive }
            { \TeX\ primitive }
        }
      \tex_cr:D
    \exp_args:Nc \token_if_primitive:NF { #1 }
      {
        \tl_if_empty:eF { \exp_args:Nc \cs_prefix_spec:N { #1 } }
          { Prefix & \exp_args:Nc \cs_prefix_spec:N { #1 } \tex_cr:D }
      }
    \exp_args:Nc \token_if_primitive:NF { #1 }
      {
        \tl_if_empty:eF { \exp_args:Nc \cs_parameter_spec:N { #1 } }
          { Parameters & \exp_args:Nc \cs_parameter_spec:N { #1 } \tex_cr:D }
      }
    \exp_args:Nc \token_if_primitive:NF { #1 }
      { Replacement~text & \cs_replacement_spec:c { #1 } }
    \tex_crcr:D
  }
}

\cs_new_protected:Npn \__xmeaning_pkg_control_sequence_meaning:n #1 {
  \cs_if_exist:cTF { #1 }
    {
      \regex_extract_all:nnNTF { \A(g|l|c)(_{1,2})\w*_([a-z]*) } { #1 } \l_tmpb_seq
        { \__xmeaning_pkg_data_type_meaning:n { #1 } }
        {
          \regex_match_case:neF
            {
              { \\(math)?char"([0-9A-Z]+) } { \__xmeaning_pkg_chardef_meaning:n { #1 } }
              { \\[a-z]+\d+ } { \__xmeaning_pkg_register_meaning:n { #1 } }
            }
            { \cs_meaning:c { #1 } }
            { \__xmeaning_pkg_macro_meaning:n { #1 } }
        }
    }
    {
      \bool_set_true:N \l__xmeaning_pkg_undefined_cs_bool
          \bool_if:NF \l__xmeaning_pkg_exhaustive_meaning_bool
            { Undefined }
    }
}

\cs_new_protected:Npn \__xmeaning_pkg_xmeaning_ii:n #1 {
  \group_begin:
  \ttfamily
  \frenchspacing
  \clist_set:Nn \l_tmpa_clist { #1 }
  \seq_set_from_clist:NN \l_tmpa_seq \l_tmpa_clist
  \seq_map_inline:Nn \l_tmpa_seq
    {
      \token_if_cs:NTF ##1
        { \exp_args:Ne \__xmeaning_pkg_control_sequence_meaning:n { \cs_to_str:N ##1 } }
```

```
        { \__xmeaning_pkg_character_meaning:N ##1 }
      \bool_if:NF \l__xmeaning_pkg_undefined_cs_bool
        { \skip_vertical:N \l__xmeaning_pkg_meaning_skip }
    }
  \group_end:
}

\cs_new:Npn \__xmeaning_pkg_xmeaning_tl_map:nw #1#2#3 \q_stop
  {
    \str_if_eq:nnTF { #2 } { #1 }
      { @ }
      { #2 }
    \tl_if_empty:nF { #3 }
      { \__xmeaning_pkg_xmeaning_tl_map:nw { #1 } #3 \q_stop }
  }

\cs_new:Npn \__xmeaning_pkg_xmeaning_replace_a_tokens:n #1
  { \__xmeaning_tl_map:nw { a } #1 \q_stop }

\cs_new:Npn \__xmeaning_pkg_xmeaning_replace_o_tokens:n #1
  { \__xmeaning_tl_map:nw { o } #1 \q_stop }

\cs_new_protected:Npn \__xmeaning_pkg_xmeaning_i:n #1 {
  \group_begin:
  \bool_set_true:N \l__xmeaning_pkg_exhaustive_meaning_bool
  \ttfamily
  \frenchspacing
  \clist_set:Nn \l_tmpa_clist { #1 }
  \clist_map_inline:Nn \l_tmpa_clist
    {
      \clist_clear:N \l_tmpb_clist
      \tl_set:Ne \l_tmpa_tl
        {
          \token_if_cs:NTF ##1
            { \cs_to_str:N ##1 }
            { ##1 }
        }
      \clist_put_right:Nf \l_tmpb_clist { \l_tmpa_tl\c_space_tl }
      \clist_put_right:Nf \l_tmpb_clist { \l_tmpa_tl\c_space_tl code}
      \clist_put_right:Ne \l_tmpb_clist { end\l_tmpa_tl}
      \exp_args:NNe \int_set:Nn \l_tmpa_int { \tl_count:V \l_tmpa_tl }
      \int_step_inline:nn { 26 }
        {
          \clist_put_right:Ne \l_tmpb_clist { \int_to_Alph:n { ####1 } \l_tmpa_tl, }
          \clist_put_right:Ne \l_tmpb_clist { \l_tmpa_tl \int_to_Alph:n { ####1 }, }
        }
      \int_step_inline:nn { 10 }
        { \clist_put_right:Ne \l_tmpb_clist { \l_tmpa_tl @\int_to_roman:n { ####1 } } }
      \int_step_inline:nnn { 0 } { \l_tmpa_int }
        {
          \clist_put_right:Ne \l_tmpb_clist
            {
              \exp_args:NV \tl_range:nnn { \l_tmpa_tl } { 1 } { ####1 }
              @
```

```
              \exp_args:NV \tl_range:nnV { \l_tmpa_tl } { \int_eval:n{ ####1+1 } } \l_tmpa_ir
            }
        }
      \tl_if_in:NnT \l_tmpa_tl { a }
        {
          \clist_put_right:Ne \l_tmpb_clist
            { \exp_args:NV \__xmeaning_pkg_xmeaning_replace_a_tokens:n \l_tmpa_tl }
        }
      \tl_if_in:NnT \l_tmpa_tl { o }
        {
          \clist_put_right:Ne \l_tmpb_clist
            { \exp_args:NV \__xmeaning_pkg_xmeaning_replace_o_tokens:n \l_tmpa_tl }
        }
      \clist_put_right:Ne \l_tmpb_clist
        { __\l_tmpa_tl }
      \__xmeaning_pkg_xmeaning_ii:n { ##1 }
      \clist_map_inline:Nn \l_tmpb_clist
        { \exp_args:Nc \__xmeaning_pkg_xmeaning_ii:n { ####1 } }
    }
  \group_end:
}

\NewDocumentCommand{\xmeaning}{ s m }{
  \IfBooleanTF{#1}
    { \__xmeaning_pkg_xmeaning_i:n { #2 } }
    { \__xmeaning_pkg_xmeaning_ii:n { #2 } }
}

\ExplSyntaxOff
```