

# fmtcount.sty: Displaying the Values of L<sup>A</sup>T<sub>E</sub>X Counters

Nicola L.C. Talbot      Vincent Belaïche  
[dickimaw-books.com](http://dickimaw-books.com)

Erik Nijenhuis  
[github.com/Xerdi/fmtcount](https://github.com/Xerdi/fmtcount)

2025-12-02 (version 3.12)

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Available Commands</b>	<b>2</b>
<b>3</b>	<b>Package Options</b>	<b>8</b>
<b>4</b>	<b>Multilingual Support</b>	<b>8</b>
4.1	Options for setting ordinal ending position raise/level . . . . .	9
4.2	Options for French . . . . .	10
4.3	Prefixes . . . . .	14
<b>5</b>	<b>Configuration File <code>fmtcount.cfg</code></b>	<b>14</b>
<b>6</b>	<b>LaTeX2HTML style</b>	<b>15</b>
<b>7</b>	<b>Miscellaneous</b>	<b>15</b>
7.1	Handling of spaces with tailing optional argument . . . . .	15
7.2	Macro naming conventions . . . . .	16
<b>8</b>	<b>Acknowledgements</b>	<b>16</b>
<b>9</b>	<b>Troubleshooting</b>	<b>16</b>
<b>10</b>	<b>The Code</b>	<b>16</b>
10.1	Language definition files . . . . .	16
10.1.1	<code>fc-american.def</code> . . . . .	16
10.1.2	<code>fc-brazilian.def</code> . . . . .	17

10.1.3 fc-british.def . . . . .	19
10.1.4 fc-dutch.def . . . . .	19
10.1.5 fc-english.def . . . . .	27
10.1.6 fc-francais.def . . . . .	37
10.1.7 fc-french.def . . . . .	38
10.1.8 fc-frenchb.def . . . . .	69
10.1.9 fc-german.def . . . . .	70
10.1.10fc-germanb.def . . . . .	80
10.1.11fc-italian.def . . . . .	80
10.1.12fc-ngerman.def . . . . .	83
10.1.13fc-ngermanb.def . . . . .	83
10.1.14fc-portuges.def . . . . .	84
10.1.15fc-portuguese.def . . . . .	98
10.1.16fc-spanish.def . . . . .	99
10.1.17fc-UKenglish.def . . . . .	116
10.1.18fc-USenglish.def . . . . .	116
10.2 fcnumparser.sty . . . . .	117
10.3 fcprefix.sty . . . . .	127
10.4 fmtcount.sty . . . . .	137
10.4.1 Multilanguage Definitions . . . . .	162

## 1 Introduction

The `fmtcount` package provides commands to display the values of  $\LaTeX$  counters in a variety of formats. It also provides equivalent commands for actual numbers rather than counter names. Limited multilingual support is available. Currently, there is only support for English, French (including Belgian and Swiss variations), Spanish, Portuguese, German and Italian.

## 2 Available Commands

The commands can be divided into two categories: those that take the name of a counter as the argument, and those that take a number as the argument.

`\ordinal`     `\ordinal{<counter>}[<gender>]`

This will print the value of a  $\LaTeX$  counter `<counter>` as an ordinal, where the macro

`\fmtord`     `\fmtord{<text>}`

is used to format the st, nd, rd, th bit. By default the ordinal is formatted as a superscript, if the package option `level` is used, it is level with the text. For example, if the current section is 2, then `\ordinal{section}` will produce the output: 2<sup>nd</sup>. Note that the optional argument `<gender>` occurs *at the end*. This argument may only take one of the following values: m

(masculine), f (feminine) or n (neuter.) If  $\langle gender \rangle$  is omitted, or if the given gender has no meaning in the current language, m is assumed.

**Notes:**

1. the memoir class also defines a command called `\ordinal` which takes a number as an argument instead of a counter. In order to overcome this incompatibility, if you want to use the `fmtcount` package with the memoir class you should use

`\FCordinal`

```
\FCordinal
```

to access `fmtcount`'s version of `\ordinal`, and use `\ordinal` to use memoir's version of that command.

2. When the  $[\langle gender \rangle]$  optional argument is omitted, no ignoring of spaces following the final argument occurs. So both `\ordinal{section}_!` and `\ordinal{section}[m]_!` will produce: 2<sup>nd</sup> \_!, where \_ denotes a space. See § 7.1.

The commands below only work for numbers in the range 0 to 99999.

`\ordinalnum`

```
\ordinalnum{\langle n \rangle}[\langle gender \rangle]
```

This is like `\ordinal` but takes an actual number rather than a counter as the argument. For example: `\ordinalnum{2}` will produce: 2<sup>nd</sup>.

`\numberstring`

```
\numberstring{\langle counter \rangle}[\langle gender \rangle]
```

This will print the value of  $\langle counter \rangle$  as text. E.g. `\numberstring{section}` will produce: three. The optional argument is the same as that for `\ordinal`.

`\Numberstring`

```
\Numberstring{\langle counter \rangle}[\langle gender \rangle]
```

This does the same as `\numberstring`, but with initial letters in uppercase. For example, `\Numberstring{section}` will produce: Two.

`\NUMBERstring`

```
\NUMBERstring{\langle counter \rangle}[\langle gender \rangle]
```

This does the same as `\numberstring`, but converts the string to upper case. Note that `\MakeUppercase{\NUMBERstring{\langle counter \rangle}}` doesn't work, due to the way that `\MakeUppercase` expands its argument<sup>1</sup>.

`\numberstringnum`

```
\numberstringnum{\langle n \rangle}[\langle gender \rangle]
```

`\Numberstringnum`

```
\Numberstringnum{\langle n \rangle}[\langle gender \rangle]
```

<sup>1</sup>See all the various postings to `comp.text.tex` about `\MakeUppercase`

`\NUMBERstringnum`

```
\NUMBERstringnum{<n>}[<gender>]
```

These macros work like `\numberstring`, `\Numberstring` and `\NUMBERstring`, respectively, but take an actual number rather than a counter as the argument. For example: `\Numberstringnum{105}` will produce: One Hundred and Five.

`\ordinalstring`

```
\ordinalstring{<counter>}[<gender>]
```

This will print the value of `<counter>` as a textual ordinal. E.g. `\ordinalstring{section}` will produce: third. The optional argument is the same as that for `\ordinal`.

`\Ordinalstring`

```
\Ordinalstring{<counter>}[<gender>]
```

This does the same as `\ordinalstring`, but with initial letters in uppercase. For example, `\Ordinalstring{section}` will produce: Second.

`\ORDINALstring`

```
\ORDINALstring{<counter>}[<gender>]
```

This does the same as `\ordinalstring`, but with all words in upper case (see previous note about `\MakeUppercase`).

`\ordinalstringnum`

```
\ordinalstringnum{<n>}[<gender>]
```

`\Ordinalstringnum`

```
\Ordinalstringnum{<n>}[<gender>]
```

`\ORDINALstringnum`

```
\ORDINALstringnum{<n>}[<gender>]
```

These macros work like `\ordinalstring`, `\Ordinalstring` and `\ORDINALstring`, respectively, but take an actual number rather than a counter as the argument. For example, `\ordinalstringnum{2}` will produce: second.

As from version 1.09, textual representations can be stored for later use. This overcomes the problems encountered when you attempt to use one of the above commands in `\edef`.

Each of the following commands takes a label as the first argument, the other arguments are as the analogous commands above. These commands do not display anything, but store the textual representation. This can later be retrieved using

`\FMCuse`

```
\FMCuse{<label>}
```

Note: with `\storeordinal` and `\storeordinalnum`, the only bit that doesn't get expanded is `\fmtord`. So, for example, `\storeordinalnum{mylabel}{3}` will be stored as `3\relax \fmtord{rd}`.

\storeordinal	\storeordinal{<label>}{<counter>}[<gender>]
reordinalstring	\storeordinalstring{<label>}{<counter>}[<gender>]
reOrdinalstring	\storeOrdinalstring{<label>}{<counter>}[<gender>]
reORDINALstring	\storeORDINALstring{<label>}{<counter>}[<gender>]
renumberstring	\storenumberstring{<label>}{<counter>}[<gender>]
reNumberstring	\storeNumberstring{<label>}{<counter>}[<gender>]
reNUMBERstring	\storeNUMBERstring{<label>}{<counter>}[<gender>]
storeordinalnum	\storeordinalnum{<label>}{<number>}[<gender>]
reordinalstringnum	\storeordinalstring{<label>}{<number>}[<gender>]
reOrdinalstringnum	\storeOrdinalstringnum{<label>}{<number>}[<gender>]
reORDINALstringnum	\storeORDINALstringnum{<label>}{<number>}[<gender>]
renumberstringnum	\storenumberstring{<label>}{<number>}[<gender>]
reNumberstringnum	\storeNumberstring{<label>}{<number>}[<gender>]
reNUMBERstringnum	\storeNUMBERstring{<label>}{<number>}[<gender>]

`\binary` `\binary{<counter>}`

This will print the value of *<counter>* as a binary number. E.g. `\binary{section}` will produce: 10. The declaration

`\padzeroes` `\padzeroes[<n>]`

will ensure numbers are written to *<n>* digits, padding with zeroes if necessary. E.g. `\padzeroes[8]\binary{section}` will produce: 00000010. The default value for *<n>* is 17.

`\binarynum` `\binarynum{<n>}`

This is like `\binary` but takes an actual number rather than a counter as the argument. For example: `\binarynum{5}` will produce: 101.

The octal commands only work for values in the range 0 to 32768.

`\octal` `\octal{<counter>}`

This will print the value of *<counter>* as an octal number. For example, if you have a counter called, say `mycounter`, and you set the value to 125, then `\octal{mycounter}` will produce: 177. Again, the number will be padded with zeroes if necessary, depending on whether `\padzeroes` has been used.

`\octalnum` `\octalnum{<n>}`

This is like `\octal` but takes an actual number rather than a counter as the argument. For example: `\octalnum{125}` will produce: 177.

`\hexadecimal` `\hexadecimal{<counter>}`

This will print the value of *<counter>* as a hexadecimal number. Going back to the counter used in the previous example, `\hexadecimal{mycounter}` will produce: 7d. Again, the number will be padded with zeroes if necessary, depending on whether `\padzeroes` has been used.

`\HEXADecimal` `\HEXADecimal{<counter>}`

This does the same thing, but uses uppercase characters, e.g. `\HEXADecimal{mycounter}` will produce: 7D.

`\Hexadecimal` The macro `\Hexadecimal` is a deprecated alias of `\HEXADecimal`. Its name was confusing so it was changed. See [7.2](#).

`\hexadecimalnum` `\hexadecimalnum{<n>}`

`\HEXADecimalnum`

`\HEXADecimalnum{<n>}`

These are like `\hexadecimal` and `\Hexadecimal` but take an actual number rather than a counter as the argument. For example: `\hexadecimalnum{125}` will produce: 7d, and `\HEXADecimalnum{125}` will produce: 7D.

`\Hexadecimalnum`

The macro `\Hexadecimalnum` is a deprecated alias of `\HEXADecimalnum`. Its name was confusing so it was changed. See [7.2](#).

`\decimal`

`\decimal{<counter>}`

This is similar to `\arabic` but the number can be padded with zeroes depending on whether `\padzeroes` has been used. For example: `\padzeroes[8]\decimal{section}` will produce: 00000002 still assuming current section is section 2.

`\decimalnum`

`\decimalnum{<n>}`

This is like `\decimal` but takes an actual number rather than a counter as the argument. For example: `\padzeroes[8]\decimalnum{5}` will produce: 00000005.

`\aaalph`

`\aaalph{<counter>}`

This will print the value of `<counter>` as: a b ... z aa bb ... zz etc. For example, `\aaalpha{mycounter}` will produce: uuuuu if `mycounter` is set to 125.

`\AAAAlph`

`\AAAAlph{<counter>}`

This does the same thing, but uses uppercase characters, e.g. `\AAAAlph{mycounter}` will produce: UUUUU.

`\aaalphnum`

`\aaalphnum{<n>}`

`\AAAAlphnum`

`\AAAAlphnum{<n>}`

These macros are like `\aaalph` and `\AAAAlph` but take an actual number rather than a counter as the argument. For example: `\aaalphnum{125}` will produce: uuuuu, and `\AAAAlphnum{125}` will produce: UUUUU.

The `abalph` commands described below only work for values in the range 0 to 17576.

`\abalph`

`\abalph{<counter>}`

This will print the value of `<counter>` as: a b ... z aa ab ... az etc. For example, `\abalpha{mycounter}` will produce: du if `mycounter` is set to 125.

`\ABAlph` `\ABAlph{<counter>}`

This does the same thing, but uses uppercase characters, e.g. `\ABAlph{mycounter}` will produce: DU.

`\abalphnum` `\abalphnum{<n>}`

`\ABAlphnum` `\ABAlphnum{<n>}`

These macros are like `\abalph` and `\ABAlph` but take an actual number rather than a counter as the argument. For example: `\abalphnum{125}` will produce: du, and `\ABAlphnum{125}` will produce: DU.

### 3 Package Options

The following options can be passed to this package:

`<dialect>` load language `<dialect>`, supported `<dialect>` are the same as passed to `\FCloadlang`, see 4

`raise` make ordinal st,nd,rd,th appear as superscript

`level` make ordinal st,nd,rd,th appear level with rest of text

Options `raise` and `level` can also be set using the command:

`countsetoptions` `\fmtcountsetoptions{fmtord=<type>}`

where `<type>` is either `level` or `raise`. Since version 3.01 of `fmtcount`, it is also possible to set `<type>` on a language by language basis, see § 4.

### 4 Multilingual Support

Version 1.02 of the `fmtcount` package now has limited multilingual support. The following languages are implemented: English, Spanish, Portuguese, French, French (Swiss) and French (Belgian). German support was added in version 1.1.<sup>2</sup> Italian support was added in version 1.31.<sup>3</sup>

Actually, `fmtcount` has two modes:

- a multilingual mode, in which the commands `\numberstring`, `\ordinalstring`, `\ordinal`, and their variants will be formatted in the currently selected language, as per the `\language` macro set by `babel`, `polyglossia` or `suchlikes`, and

<sup>2</sup>Thanks to K. H. Fricke for supplying the information.

<sup>3</sup>Thanks to Edoardo Pasca for supplying the information.



- a default mode for backward compatibility in which these commands are formatted in English irrespective of `\languagename`, and to which `fmtcount` falls back when it cannot detect packages such as `babel` or `polyglossia` are loaded.

For multilingual mode, `fmtcount` needs to load correctly the language definition for document dialects. To do this use

`\FCloadlang`

```
\FCloadlang{<dialect>}
```

in the preamble — this will both switch on multilingual mode, and load the `<dialect>` definition. The `<dialect>` should match the options passed to `babel` or `polyglossia`. `fmtcount` currently supports the following `<dialect>`'s: `english`, `UKenglish`, `brazilian`, `british`, `USenglish`, `american`, `spanish`, `portuges`, `portuguese`, `french`, `frenchb`, `francais`, `german`, `germanb`, `ngerman`, `ngermanb`, `italian`, and `dutch`.

If you don't use `\FCloadlang`, `fmtcount` will attempt to detect the required dialects and call `\FCloadlang` for you, but this isn't guaranteed to work. Notably, when `\FCloadlang` is not used and `fmtcount` has switched on multilingual mode, but without detecting the needed dialects in the preamble, and `fmtcount` has to format a number for a dialect for which definition has not been loaded (via `\FCloadlang` above), then if `fmtcount` detects a definition file for this dialect it will attempt to load it, and cause an error otherwise. This loading in body has not been tested extensively, and may cause problems such as spurious spaces insertion before the first formatted number, so it's best to use `\FCloadlang` explicitly in the preamble.

If the French language is selected, the `french` option let you configure the dialect and other aspects. The `abbr` also has some influence with French. Please refer to § 4.2.

The male gender for all languages is used by default, however the feminine or neuter forms can be obtained by passing `f` or `n` as an optional argument to `\ordinal`, `\ordinalnum` etc. For example: `\numberstring{section}[f]`. Note that the optional argument comes *after* the compulsory argument. If a gender is not defined in a given language, the masculine version will be used instead.

Let me know if you find any spelling mistakes (has been known to happen in English, let alone other languages with which I'm not so familiar.) If you want to add support for another language, you will need to let me know how to form the numbers and ordinals from 0 to 99999 in that language for each gender.

#### 4.1 Options for setting ordinal ending position raise/level

`countsetoptions`

```
\fmtcountsetoptions{<language>={fmtord=<type>}}
```

where `<language>` is one of the supported language `<type>` is either `level` or `raise` or `undefine`. If the value is `level` or `raise`, then that will set the `fmtord` option accordingly<sup>4</sup> only for that language `<language>`. If the value is `undefine`, then the non-language specific behaviour is followed.

<sup>4</sup>see § 3

Some *<language>* are synonyms, here is a table:

language	alias(es)
english	british
french	frenchb
german	germanb ngerman ngermanb
USenglish	american

## 4.2 Options for French

This section is in French, as it is most useful to French speaking people.

Il est possible de configurer plusieurs aspects de la numérotation en français avec les options french et abbr. Ces options n'ont d'effet que si le langage french est chargé.

countsetoptions

```
\fmtcountsetoptions{french={<french options>}}
```

L'argument *<french options>* est une liste entre accolades et séparée par des virgules de réglages de la forme "*<clef>=<valeur>*", chacun de ces réglages est ci-après désigné par "option française" pour le distinguer des "options générales" telles que french.

Le dialecte peut être sélectionné avec l'option française dialect dont la valeur *<dialect>* peut être france, belgian ou swiss.

dialect

```
\fmtcountsetoptions{french={dialect={<dialect>}}}
```

french

```
\fmtcountsetoptions{french=<dialect>}
```

Pour alléger la notation et par souci de rétro-compatibilité france, belgian ou swiss sont également des *<clef>*s pour *<french options>* à utiliser sans *<valeur>*.

L'effet de l'option dialect est illustré ainsi :

france soixante-dix pour 70, quatre-vingts pour 80, et quatre-vingts-dix pour 90,

belgian septante pour 70, quatre-vingts pour 80, et nonante pour 90,

swiss septante pour 70, huitante<sup>5</sup> pour 80, et nonante pour 90

Il est à noter que la variante belgian est parfaitement correcte pour les francophones français<sup>6</sup>, et qu'elle est également utilisée en Suisse Romande hormis dans les cantons de Vaud, du Valais et de Fribourg. En ce qui concerne le mot "octante", il n'est actuellement pas pris en charge et n'est guère plus utilisé, ce qui est sans doute dommage car il est sans doute plus acceptable que le "huitante" de certains de nos amis suisses.

abbr

```
\fmtcountsetoptions{abbr=<boolean>}
```

<sup>5</sup>voir [Octante et huitante](#) sur le site d'Alain Lassine

<sup>6</sup>je précise que l'auteur de ces lignes est français

L'option générale `abbr` permet de changer l'effet de `\ordinal`. Selon  $\langle boolean \rangle$  on a :  
`true` pour produire des ordinaux de la forme 2<sup>e</sup> (par défaut), ou  
`false` pour produire des ordinaux de la forme 2<sup>ème</sup>

vingt plural `\fmtcountsetoptions{french={vingt plural=french plural control}}`

cent plural `\fmtcountsetoptions{french={cent plural=french plural control}}`

mil plural `\fmtcountsetoptions{french={mil plural=french plural control}}`

n-illion plural `\fmtcountsetoptions{french={n-illion plural=french plural control}}`

n-illiard plural `\fmtcountsetoptions{french={n-illiard plural=french plural control}}`

all plural `\fmtcountsetoptions{french={all plural=french plural control}}`

Les options `vingt plural`, `cent plural`, `mil plural`, `n-illion plural`, et `n-illiard plural`, permettent de contrôler très finement l'accord en nombre des mots respectivement vingt, cent, mil, et des mots de la forme  $\langle n \rangle$ illion et  $\langle n \rangle$ illiard, où  $\langle n \rangle$  désigne 'm' pour 1, 'b' pour 2, 'tr' pour 3, etc. L'option `all plural` est un raccourci permettant de contrôler de concert l'accord en nombre de tous ces mots. Tous ces paramètres valent reformé par défaut.

Attention, comme on va l'expliquer, seules quelques combinaisons de configurations de ces options donnent un orthographe correcte vis à vis des règles en vigueur. La raison d'être de ces options est la suivante :

- la règle de l'accord en nombre des noms de nombre dans un numéral cardinal dépend de savoir s'il a vraiment une valeur cardinale ou bien une valeur ordinale, ainsi on écrit « aller à la page deux-cent (sans s) d'un livre de deux-cents (avec s) pages », il faut donc pouvoir changer la configuration pour sélectionner le cas considéré,
- un autre cas demandant quelque configurabilité est celui de « mil » et « mille ». Pour rappel « mille » est le pluriel irrégulier de « mil », mais l'alternance mil/mille est rare, voire pédante, car aujourd'hui « mille » n'est utilisé que comme un mot invariable, en effet le sort des pluriels étrangers est systématiquement de finir par disparaître comme par exemple « scénarii » aujourd'hui supplanté par « scénarios ». Pour continuer à pouvoir écrire « mil », il aurait fallu former le pluriel comme « mils », ce qui n'est pas l'usage. Certaines personnes utilisent toutefois encore « mil » dans les dates, par exemple « mil neuf cent quatre-vingt quatre » au lieu de « mille neuf cent quatre-vingt quatre »,

- finalement les règles du français quoique bien définies ne sont pas très cohérentes et il est donc inévitable qu'un jour ou l'autre on les simplifie. Le paquetage `fmtcount` est déjà prêt à cette éventualité.

Le paramètre `<french plural control>` peut prendre les valeurs suivantes :

<code>traditional</code>	pour sélectionner la règle en usage chez les adultes à la date de parution de ce document, et dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur cardinale,
<code>reformed</code>	pour suivre toute nouvelle recommandation à la date de parution de ce document, , et dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur cardinale, l'idée des options <code>traditional</code> et <code>reformed</code> est donc de pouvoir contenter à la fois les anciens et les modernes, mais à dire vrai à la date où ce document est écrit elles ont exactement le même effet,
<code>traditional o</code>	pareil que <code>traditional</code> mais dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur ordinale,
<code>reformed o</code>	pareil que <code>reformed</code> mais dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur ordinale, de même que précédemment <code>reformed o</code> et <code>traditional o</code> ont exactement le même effet,
<code>always</code>	pour marquer toujours le pluriel, ceci n'est correct que pour « mil » vis à vis des règles en vigueur,
<code>never</code>	pour ne jamais marquer le pluriel, ceci est incorrect vis à vis des règles d'orthographe en vigueur,
<code>multiple</code>	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2, ceci est la règle en vigueur pour les nombres de la forme <code>&lt;n&gt;illion</code> et <code>&lt;n&gt;illiard</code> lorsque le nombre a une valeur cardinale,
<code>multiple g-last</code>	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 est est <i>globalement</i> en dernière position, où "globalement" signifie qu'on considère le nombre formaté en entier, ceci est incorrect vis à vis des règles d'orthographe en vigueur,
<code>multiple l-last</code>	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 et est <i>localement</i> en dernière position, où "localement" signifie qu'on considère seulement la portion du nombre qui multiplie soit l'unité, soit un <code>&lt;n&gt;illion</code> ou un <code>&lt;n&gt;illiard</code> ; ceci est la convention en vigueur pour le pluriel de "vingt" et de "cent" lorsque le nombre formaté a une valeur cardinale,
<code>multiple lng-last</code>	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 et est <i>localement</i> mais <i>non globalement</i> en dernière position, où "localement" et <i>globalement</i> on la même signification que pour les options <code>multiple g-last</code> et <code>multiple l-last</code> ; ceci est la convention en vigueur pour le pluriel de "vingt" et de "cent" lorsque le nombre formaté a une valeur ordinale,

multiple ng-last pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2, et *n* n'est pas *globalement* en dernière position, où "globalement" a la même signification que pour l'option multiple g-last; ceci est la règle que j'infère être en vigueur pour les nombres de la forme *<n>*illion et *<n>*illiard lorsque le nombre a une valeur ordinale, mais à dire vrai pour des nombres aussi grands, par exemple « deux millions », je pense qu'il n'est tout simplement pas d'usage de dire « l'exemplaire deux million(s?) » pour « le deux millionième exemplaire ».

L'effet des paramètres traditional, traditional o, reformed, et reformed o, est le suivant :

<i>&lt;x&gt;</i> dans " <i>&lt;x&gt;</i> plural"	traditional	reformed	traditional o	reformed o
vingt	multiple l-last		multiple lng-last	
cent				
mil	always			
n-illion	multiple		multiple ng-last	
n-illiard				

Les configurations qui respectent les règles d'orthographe sont les suivantes :

- `\fmtcountsetoptions{french={all plural=reformed o}}` pour formater les numéraux cardinaux à valeur ordinale,
- `\fmtcountsetoptions{french={mil plural=multiple}}` pour activer l'alternance mil/mille.
- `\fmtcountsetoptions{french={all plural=reformed}}` pour revenir dans la configuration par défaut.

dash or space

```
\fmtcountsetoptions{french={dash or space=<dash or space>}}
```

Avant la réforme de l'orthographe de 1990, on ne met des traits d'union qu'entre les dizaines et les unités, et encore sauf quand le nombre *n* considéré est tel que  $n \bmod 10 = 1$ , dans ce cas on écrit "et un" sans trait d'union. Après la réforme de 1990, on recommande de mettre des traits d'union de partout sauf autour de "mille", "million" et "milliard", et les mots analogues comme "billion", "billiard". Cette exception a toutefois été contestée par de nombreux auteurs, et on peut aussi mettre des traits d'union de partout. Mettre l'option (*dash or space*) à :

traditional pour sélectionner la règle d'avant la réforme de 1990,  
 1990 pour suivre la recommandation de la réforme de 1990,  
 reformed pour suivre la recommandation de la dernière réforme prise en charge, actuellement l'effet est le même que 1990, ou à  
 always pour mettre systématiquement des traits d'union de partout.

Par défaut, l'option vaut reformed.

scale

```
\fmtcountsetoptions{french={scale=<scale>}}
```

L'option `scale` permet de configurer l'écriture des grands nombres. Mettre  $\langle scale \rangle$  à :

- `recursive` dans ce cas  $10^{30}$  donne mille milliards de milliards de milliards, pour  $10^n$ , on écrit  $10^{n-9 \times \max\{(n \div 9) - 1, 0\}}$  suivi de la répétition  $\max\{(n \div 9) - 1, 0\}$  fois de “de milliards”
- `long`  $10^{6 \times n}$  donne un  $\langle n \rangle$ illion où  $\langle n \rangle$  est remplacé par “bi” pour 2, “tri” pour 3, etc. et  $10^{6 \times n + 3}$  donne un  $\langle n \rangle$ illiard avec la même convention pour  $\langle n \rangle$ . L'option `long` est correcte en Europe, par contre j'ignore l'usage au Québec.
- `short`  $10^{6 \times n}$  donne un  $\langle n \rangle$ illion où  $\langle n \rangle$  est remplacé par “bi” pour 2, “tri” pour 3, etc. L'option `short` est incorrecte en Europe.

Par défaut, l'option vaut `recursive`.

n-illiard upto

```
\fmtcountsetoptions{french={n-illiard upto=\langle n-illiard upto \rangle}}
```

Cette option n'a de sens que si `scale` vaut `long`. Certaines personnes préfèrent dire “mille  $\langle n \rangle$ illions” qu'un “ $\langle n \rangle$ illiard”. Mettre l'option `n-illiard upto` à :

- `infinity` pour que  $10^{6 \times n + 3}$  donne  $\langle n \rangle$ illards pour tout  $n > 0$ ,
- `infty` même effet que `infinity`,
- `k` où  $k$  est un entier quelconque strictement positif, dans ce cas  $10^{6 \times n + 3}$  donne “mille  $\langle n \rangle$ illions” lorsque  $n > k$ , et donne “ $\langle n \rangle$ illiard” sinon

mil plural mark

```
\fmtcountsetoptions{french={mil plural mark=\langle any text \rangle}}
```

La valeur par défaut de cette option est « le ». Il s'agit de la terminaison ajoutée à « mil » pour former le pluriel, c'est à dire « mille », cette option ne sert pas à grand chose sauf dans l'éventualité où ce pluriel serait francisé un jour — à dire vrai si cela se produisait une alternance mille/milles est plus vraisemblable, car « mille » est plus fréquent que « mil » et que les pluriels francisés sont formés en ajoutant « s » à la forme la plus fréquente, par exemple « blini/blinis », alors que « blini » veut dire « crêpes » (au pluriel).

### 4.3 Prefixes

latinnumeralstring

```
\latinnumeralstring{\langle counter \rangle}[\langle prefix options \rangle]
```

latinnumeralstringnum

```
\latinnumeralstringnum{\langle number \rangle}[\langle prefix options \rangle]
```

## 5 Configuration File `fmtcount.cfg`

You can save your preferred default settings to a file called `fmtcount.cfg`, and place it on the TeX path. These settings will then be loaded by the `fmtcount` package.

Note that if you are using the `datetime` package, the `datetime.cfg` configuration file will override the `fmtcount.cfg` configuration file. For example, if `datetime.cfg` has the line:

```
\renewcommand{\fmtord}[1]{\textsuperscript{\underline{#1}}}
```

and if `fmtcount.cfg` has the line:

```
\fmtcountsetoptions{fmtord=level}
```

then the former definition of `\fmtord` will take precedence.

## 6 LaTeX2HTML style

The LaTeX2HTML style file `fmtcount.perl` is provided. The following limitations apply:

- `\padzeroes` only has an effect in the preamble.
- The configuration file `fmtcount.cfg` is currently ignored. (This is because I can't work out the correct code to do this. If you know how to do this, please let me know.) You can however do:

```
\usepackage{fmtcount}  
\html{\input{fmtcount.cfg}}
```

This, I agree, is an unpleasant cludge.

## 7 Miscellaneous

### 7.1 Handling of spaces with tailing optional argument

Quite some of the commands in `fmtcount` have a tailing optional argument, notably a [*gender*] argument, which is due to historical reasons, and is a little unfortunate.

When the tailing optional argument is omitted, then any subsequent space will:

- not be gobbled if the command make some typset output, like `\ordinal` or `\numbestring`, and
- be gobbled if the command stores a number into a label like `\storeordinalnum` or `\storenumberstring`, or make some other border effect like `\padzeroes` without any typeset output.

So (where we use visible spaces “`□`” to demonstrate the point):

- “`x\ordinalnum{2}□x`” will be typeset to “`x2nd□x`”, while
- “`x\storeordinalnum{mylabel}{2}□x`” will be typeset to “`xx`”.

The reason for this design choice is that the commands like `\ordinal` or `\numbestring` are usually inserted in the flow of text, and one usually does not want subsequent spaces gobbled, while the commands like `\storeordinalnum` or `\storenumberstring` usually stands on their own line, and one usually does not want the tailing end-of-line to produce an extra-space.

## 7.2 Macro naming conventions

Macros that refer to upper-casing have upper case only in the main part of their name. That is to say the words “store”, “string” or “num” are not upper-cased for instance in `\storeORDINALstringnum`, `\storeOrdinalstringnum` or in `\NUMBERstringnum`.

Furthermore, when upper-casing all the number letters is considered, the main part of the name is:

- all in upper-case when it consist of a single word that is not composed of a prefix+radix, for instance “ORDINAL” or “NUMBER”, and
- with the prefix all in upper-case, and only the first letter of the radix in upper-case for words that consist of a prefix+radix, for instance “HEXADecimal” or “AAAlph” because they can be considered as a prefix+radix construct “hexa+decimal” or “aa+alph”.

Observance of this rule is the reason why macros `\Hexadecimal` and `\Hexadecimalnum` were respectively renamed as `\HEXADecimal` and `\HEXADecimalnum` from v3.06.

## 8 Acknowledgements

I would like to thank all the people who have provided translations and made bug reports.

## 9 Troubleshooting

Bug reporting should be done via the GitHub issue manager at:

[github.com/Xerdi/fmtcount/issues/](https://github.com/Xerdi/fmtcount/issues/).

## 10 The Code

### 10.1 Language definition files

#### 10.1.1 fc-american.def

American English definitions

```
1 \ProvidesFCLanguage{american}[2016/01/12]%
```

Loaded fc-USenglish.def if not already loaded

```
2 \FCLoadlang{USenglish}%
```

These are all just synonyms for the commands provided by fc-USenglish.def.

```
3 \global\let\@ordinalMamerican\@ordinalMUSenglish
```

```
4 \global\let\@ordinalFamerican\@ordinalMUSenglish
```

```
5 \global\let\@ordinalNamerican\@ordinalMUSenglish
```

```
6 \global\let\@numberstringMamerican\@numberstringMUSenglish
```

```
7 \global\let\@numberstringFamerican\@numberstringMUSenglish
```

```
8 \global\let\@numberstringNamerican\@numberstringMUSenglish
```

```
9 \global\let\@NumberstringMamerican\@NumberstringMUSenglish
```



```

10 \global\let\@NumberstringFamerican\@NumberstringMUSenglish
11 \global\let\@NumberstringNamerican\@NumberstringMUSenglish
12 \global\let\@OrdinalstringMamerican\@OrdinalstringMUSenglish
13 \global\let\@OrdinalstringFamerican\@OrdinalstringMUSenglish
14 \global\let\@OrdinalstringNamerican\@OrdinalstringMUSenglish
15 \global\let\@OrdinalstringMamerican\@OrdinalstringMUSenglish
16 \global\let\@OrdinalstringFamerican\@OrdinalstringMUSenglish
17 \global\let\@OrdinalstringNamerican\@OrdinalstringMUSenglish

```

### 10.1.2 fc-brazilian.def

Brazilian definitions.

```
1 \ProvidesFCLanguage{brazilian}[2017/12/26]%
```

Load fc-portuges.def if not already loaded.

```
2 \FCloadlang{portuges}%
```

Set brazilian to be equivalent to portuges for all the numeral ordinals, and string ordinals.

```

3 \global\let\@ordinalMbrazilian=\@ordinalMportuges
4 \global\let\@ordinalFbrazilian=\@ordinalFportuges
5 \global\let\@ordinalNbrazilian=\@ordinalNportuges
6 \global\let\@ordinalstringFbrazilian\@ordinalstringFportuges
7 \global\let\@ordinalstringMbrazilian\@ordinalstringMportuges
8 \global\let\@ordinalstringNbrazilian\@ordinalstringMportuges
9 \global\let\@OrdinalstringMbrazilian\@OrdinalstringMportuges
10 \global\let\@OrdinalstringFbrazilian\@OrdinalstringFportuges
11 \global\let\@OrdinalstringNbrazilian\@OrdinalstringMportuges

```

Convert a number to a textual representation. To make it easier, split it up into units, tens, teens and hundreds. Units, tens, and hundreds are the same as for portuges and are not redefined, only the teens are Brazilian specific.

Teens (argument must be a number from 0 to 9):

```

12 \newcommand*{\@teenstringbrazilian[1]}{
13   \ifcase#1\relax
14     dez%
15     \or onze%
16     \or doze%
17     \or treze%
18     \or quatorze%
19     \or quinze%
20     \or dezesseis%
21     \or dezessete%
22     \or dezoito%
23     \or dezenove%
24   \fi
25 }%
26 \global\let\@teenstringbrazilian\@teenstringbrazilian

```

Teens (with initial letter in upper case):

```
27 \newcommand*{\@Teenstringbrazilian[1]}{
```

```

28 \ifcase#1\relax
29   Dez%
30   \or Onze%
31   \or Doze%
32   \or Treze%
33   \or Quatorze%
34   \or Quinze%
35   \or Dezesseis%
36   \or Dezessete%
37   \or Dezoito%
38   \or Dezenove%
39 \fi
40 }%
41 \global\let\@@Teenstringbrazilian\@@Teenstringbrazilian

```

This has changed in version 1.08, so that it now stores the result in the second argument, but doesn't display anything. Since it only affects internal macros, it shouldn't affect documents created with older versions. (These internal macros are not meant for use in documents.)

```

42 \newcommand*\@numberstringMbrazilian}[2]{%
43   \let\@unitstring=\@unitstringportuges
44   \let\@teenstring=\@teenstringbrazilian
45   \let\@tenstring=\@tenstringportuges
46   \let\@hundredstring=\@hundredstringportuges
47   \def\@hundred{cem}\def\@thousand{mil}%
48   \def\@andname{e}%
49   \@numberstringportuges{#1}{#2}%
50 }%
51 \global\let\@numberstringMbrazilian\@numberstringMbrazilian

```

As above, but feminine form:

```

52 \newcommand*\@numberstringFbrazilian}[2]{%
53   \let\@unitstring=\@unitstringFportuges
54   \let\@teenstring=\@teenstringbrazilian
55   \let\@tenstring=\@tenstringportuges
56   \let\@hundredstring=\@hundredstringFportuges
57   \def\@hundred{cem}\def\@thousand{mil}%
58   \def\@andname{e}%
59   \@numberstringportuges{#1}{#2}%
60 }%
61 \global\let\@numberstringFbrazilian\@numberstringFbrazilian

```

Make neuter same as masculine:

```

62 \global\let\@numberstringNbrazilian\@numberstringMbrazilian

```

As above, but initial letters in upper case:

```

63 \newcommand*\@NumberstringMbrazilian}[2]{%
64   \let\@unitstring=\@unitstringportuges
65   \let\@teenstring=\@Teenstringbrazilian
66   \let\@tenstring=\@Tenstringportuges
67   \let\@hundredstring=\@Hundredstringportuges
68   \def\@hundred{Cem}\def\@thousand{Mil}%

```

```

69 \def\@andname{e}%
70 \@numberstringportuges{#1}{#2}%
71 }%
72 \global\let\@NumberstringMbrazilian\@NumberstringMbrazilian

```

As above, but feminine form:

```

73 \newcommand*\@NumberstringFbrazilian}[2]{%
74 \let\@unitstring=\@UnitstringFportuges
75 \let\@teenstring=\@Teenstringbrazilian
76 \let\@tenstring=\@Tenstringportuges
77 \let\@hundredstring=\@HundredstringFportuges
78 \def\@hundred{Cem}\def\@thousand{Mil}%
79 \def\@andname{e}%
80 \@numberstringportuges{#1}{#2}%
81 }%
82 \global\let\@NumberstringFbrazilian\@NumberstringFbrazilian

```

Make neuter same as masculine:

```

83 \global\let\@NumberstringNbrazilian\@NumberstringMbrazilian

```

### 10.1.3 fc-british.def

British definitions

```

1 \ProvidesFCLanguage{british}[2013/08/17]%

```

Load fc-english.def, if not already loaded

```

2 \FCloadlang{english}%

```

These are all just synonyms for the commands provided by fc-english.def.

```

3 \global\let\@ordinalMbritish\@ordinalMenglish
4 \global\let\@ordinalFbritish\@ordinalMenglish
5 \global\let\@ordinalNbritish\@ordinalMenglish
6 \global\let\@numberstringMbritish\@numberstringMenglish
7 \global\let\@numberstringFbritish\@numberstringMenglish
8 \global\let\@numberstringNbritish\@numberstringMenglish
9 \global\let\@NumberstringMbritish\@NumberstringMenglish
10 \global\let\@NumberstringFbritish\@NumberstringMenglish
11 \global\let\@NumberstringNbritish\@NumberstringMenglish
12 \global\let\@ordinalstringMbritish\@ordinalstringMenglish
13 \global\let\@ordinalstringFbritish\@ordinalstringMenglish
14 \global\let\@ordinalstringNbritish\@ordinalstringMenglish
15 \global\let\@OrdinalstringMbritish\@OrdinalstringMenglish
16 \global\let\@OrdinalstringFbritish\@OrdinalstringMenglish
17 \global\let\@OrdinalstringNbritish\@OrdinalstringMenglish

```

### 10.1.4 fc-dutch.def

Dutch definitions, initially added by Erik Nijenhuis.

```

1 \ProvidesFCLanguage{dutch}[2024/01/27]%

```

Define macro that converts a number or count register (first argument) to a numeric ordinal, and stores the result in the second argument, which should be a control sequence. Dutch numeric ordinals use “ste” for multiples of 10 from 20 upward (e.g., 20ste, 30ste, 100ste, 120ste), and “e” for all other numbers (e.g., 1e, 10e, 18e, 21e, 101e).

```

2 \newcommand{\@ordinalMdutch}[2]{%
3   \@tempcnta=#1\relax
4   \@FCmodulo{\@tempcnta}{10}%
5   \ifnum\@tempcnta=0\relax
6     \ifnum#1<20\relax
7       \edef#2{\number#1\relax e}%
8     \else
9       \edef#2{\number#1\relax ste}%
10    \fi
11  \else
12    \edef#2{\number#1\relax e}%
13  \fi
14}%
15 \global\let\@ordinalMdutch\@ordinalMdutch

```

Like English, there is no gender difference in Dutch, so make feminine and neuter the same as the masculine.

```

16 \global\let\@ordinalFdutch\@ordinalMdutch
17 \global\let\@ordinalNdutch\@ordinalMdutch

```

Define the macro that prints the value of a TeX count register as text. To make it easier, break it up into units, teens and tens. First, the units: the argument should be between 0 and 9 inclusive.

```

18 \newcommand*{\@unitstringdutch}[1]{%
19   \ifcase#1%
20     nul%
21     \or een% één and \'e\'en not working atm
22     \or twee%
23     \or drie%
24     \or vier%
25     \or vijf%
26     \or zes%
27     \or zeven%
28     \or acht%
29     \or negen%
30   \fi
31}%
32 \global\let\@unitstringdutch\@unitstringdutch

```

Next the tens, again the argument should be between 0 and 9 inclusive.

```

33 \global\let\@unitstringdutch\@unitstringdutch
34 \newcommand*{\@tenstringdutch}[1]{%
35   \ifcase#1%
36     \or tien%
37     \or twintig%
38     \or dertig%

```

```

39 \or veertig%
40 \or vijftig%
41 \or zestig%
42 \or zeventig%
43 \or tachtig%
44 \or negentig%
45 \or honderd%
46 \fi
47 }%
48 \global\let\@@tenstringdutch\@@tenstringdutch

```

Finally the teens, again the argument should be between 0 and 9 inclusive.

```

49 \newcommand*\@@teenstringdutch[1]{%
50 \ifcase#1%
51 tien%
52 \or elf%
53 \or twaalf%
54 \or dertien%
55 \or veertien%
56 \or vijftien%
57 \or zestien%
58 \or zeventien%
59 \or achttien%
60 \or negentien%
61 \fi
62 }%
63 \global\let\@@teenstringdutch\@@teenstringdutch

```

Hunderd and thousand:

```

64 \providecommand*\honderd{\honderd}%
65 \providecommand*\duizend{\duizend}%
66 \global\let\honderd\honderd
67 \global\let\duizend\duizend

```

The numberstring implementation. Converts numbers to Dutch words (e.g., 18 becomes “achttien”, 100 becomes “honderd”). Supports numbers from 0 to 99999.

```

68 \newcommand*\@@numberstringdutch[2]{%
69 \ifnum#1>99999\relax
70 \PackageError{fmtcount}{Out of range}%
71 {This macro only works for values less than 100000}%
72 \else
73 \ifnum#1<0\relax
74 \PackageError{fmtcount}{Negative numbers not permitted}%
75 {This macro does not work for negative numbers, however
76 you can try typing "minus" first, and then pass the modulus of
77 this number}%
78 \fi
79 \fi
80 \def#2{}%
81 \@strctr=#1\relax \divide\@strctr by 1000\relax
82 \ifnum\@strctr>1\relax

```

```

83   \@numberunderhundreddutch{\@strctr}{#2}%
84   \appto#2{duizend}%
85   \else
86   \ifnum\@strctr=1\relax
87   \eappto#2{\duizend}%
88   \fi
89   \fi
90   \@strctr=#1\relax
91   \@FCmodulo{\@strctr}{1000}%
92   \divide\@strctr by 100\relax
93   \ifnum\@strctr>1\relax
94   \eappto#2{\@unitstring{\@strctr}honderd}%
95   \else
96   \ifnum\@strctr=1\relax
97   \ifnum#1>1000\relax
98   \appto#2{honderd}%
99   \else
100  \eappto#2{\honderd}%
101  \fi
102  \fi
103  \fi
104  \@strctr=#1\relax
105  \@FCmodulo{\@strctr}{100}%
106  \ifnum#1=0\relax
107  \def#2{null}%
108  \else
109  \ifnum\@strctr=1\relax
110  \appto#2{een}% één and `e`en not working atm
111  \else
112  \@numberunderhundreddutch{\@strctr}{#2}%
113  \fi
114  \fi
115 }%
116 \global\let\@numberstringdutch\@numberstringdutch

```

All lower case version, the second argument must be a control sequence.

```

117 \newcommand*{\@numberstringMdutch}[2]{%
118   \let\@unitstring=\@unitstringdutch%
119   \let\@teenstring=\@teenstringdutch%
120   \let\@tenstring=\@tenstringdutch%
121   \def\@hundred{honderd}\def\@thousand{duizend}%
122   \@numberstringdutch{#1}{#2}%
123 }%
124 \global\let\@numberstringMdutch\@numberstringMdutch

```

There is no gender in Dutch, so make feminine and neuter the same as the masculine.

```

125 \global\let\@numberstringFdutch=\@numberstringMdutch
126 \global\let\@numberstringNdutch=\@numberstringMdutch

```

This version makes the first letter of each word an uppercase character (except “and”). The

second argument must be a control sequence.

```
127 \newcommand*{\@NumberstringMdutch}[2]{%
128   \@numberstringMdutch{#1}{\@@num@str}%
129   \edef#2{\noexpand\MakeUppercase\expandonce\@@num@str}%
130 }%
131 \global\let\@NumberstringMdutch\@NumberstringMdutch
```

There is no gender in Dutch, so make feminine and neuter the same as the masculine.

```
132 \global\let\@NumberstringFdutch=\@NumberstringMdutch
133 \global\let\@NumberstringNdutch=\@NumberstringMdutch
```

Define a macro that produces an ordinal as a string. Again, break it up into units, teens and tens. First the units:

```
134 \newcommand*\@@unitthstringdutch[1]{%
135   \ifcase#1%
136   nulde%
137   \or eerste% éérste and \’e\’erste not working atm
138   \or tweede%
139   \or derde%
140   \or vierde%
141   \or vijfde%
142   \or zesde%
143   \or zevende%
144   \or achtste%
145   \or negende%
146   \fi
147 }%
148 \global\let\@@unitthstringdutch\@@unitthstringdutch
```

Next the tens:

```
149 \newcommand*\@@tenthstringdutch[1]{%
150   \ifcase#1%
151   \or tiende%
152   \or twintigste%
153   \or dertigste%
154   \or veertigste%
155   \or vijftigste%
156   \or zestigste%
157   \or zeventigste%
158   \or tachtigste%
159   \or negentigste%
160   \fi
161 }%
162 \global\let\@@tenthstringdutch\@@tenthstringdutch
```

The teens:

```
163 \newcommand*\@@teenthstringdutch[1]{%
164   \ifcase#1%
165   tiende%
166   \or elfde%
167   \or twaalfde%
```

```

168 \or dertiende%
169 \or veertiende%
170 \or vijftiende%
171 \or zestiende%
172 \or zeventiende%
173 \or achttiende%
174 \or negentiende%
175 \fi
176 }%
177 \global\let\@@teenthstringdutch\@@teenthstringdutch

```

The ordinalstring implementation. Converts numbers to Dutch ordinal words (e.g., 18 becomes “achttiende”, 100 becomes “honderdste”). Supports numbers from 0 to 99999.

```

178 \newcommand*\@@ordinalstringdutch[2]{%
179 \orgargctr=#1\relax
180 \ifnum\orgargctr>99999\relax
181 \PackageError{fmtcount}{Out of range}%
182 {This macro only works for values less than 100000}%
183 \else
184 \ifnum\orgargctr<0\relax
185 \PackageError{fmtcount}{Negative numbers not permitted}%
186 {This macro does not work for negative numbers, however
187 you can try typing "minus" first, and then pass the modulus of
188 this number}%
189 \fi
190 \fi
191 \def#2{}%
192 \@strctr=\orgargctr\divide\@strctr by 1000\relax
193 \ifnum\@strctr>1\relax
194 \@numberunderhundreddutch{\@strctr}{#2}%
195 \@tmpstrctr=\orgargctr\@FCmodulo{\@tmpstrctr}{1000}%
196 \ifnum\@tmpstrctr=0\relax
197 \eappto#2{\@thousandth}%
198 \else
199 \appto#2{duizend}%
200 \fi
201 \else
202 \ifnum\@strctr=1\relax
203 \ifnum\orgargctr=1000\relax
204 \eappto#2{\@thousandth}%
205 \else
206 \eappto#2{\duizend}%
207 \fi
208 \fi
209 \fi
210 \@strctr=\orgargctr%
211 \@FCmodulo{\@strctr}{1000}%
212 \divide\@strctr by 100\relax
213 \ifnum\@strctr>1\relax
214 \@tmpstrctr=\orgargctr \@FCmodulo{\@tmpstrctr}{100}%

```



```

215 \ifnum\@tmpstrctr=0\relax
216 \ifnum\@strctr=1\relax
217 \eappto#2{\@hundredth}%
218 \else
219 \eappto#2{\@unitstring{\@strctr}\@hundredth}%
220 \fi
221 \else
222 \eappto#2{\@unitstring{\@strctr}honderd}%
223 \fi
224 \else
225 \ifnum\@strctr=1\relax
226 \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{100}%
227 \ifnum\@tmpstrctr=0\relax
228 \eappto#2{\@hundredth}%
229 \else
230 \ifnum\@orgargctr>1000\relax
231 \appto#2{honderd}%
232 \else
233 \eappto#2{\honderd}%
234 \fi
235 \fi
236 \fi
237 \fi
238 \@strctr=\@orgargctr%
239 \@FCmodulo{\@strctr}{100}%
240 \ifthenelse{\@strctr=0 \and \@orgargctr>0 }{-}{%
241 \@@numberunderhundredthdutch{\@strctr}{#2}%
242 }%
243 }%
244 \global\let\@@ordinalstringdutch\@@ordinalstringdutch

```

All lower case version. Again, the second argument must be a control sequence in which the resulting text is stored.

```

245 \newcommand*{\@ordinalstringMdutch}[2]{%
246 \let\@unitthstring=\@unitthstringdutch%
247 \let\@teenthstring=\@teenthstringdutch%
248 \let\@tenthstring=\@tenthstringdutch%
249 \let\@unitstring=\@unitstringdutch%
250 \let\@teenstring=\@teenstringdutch%
251 \let\@tenstring=\@tenstringdutch%
252 \def\@thousandth{duizendste}%
253 \def\@hundredth{honderdste}%
254 \@@ordinalstringdutch{#1}{#2}%
255 }%
256 \global\let\@@ordinalstringMdutch\@@ordinalstringMdutch

```

No gender in Dutch, so make feminine and neuter same as masculine:

```

257 \global\let\@@ordinalstringFdutch=\@@ordinalstringMdutch
258 \global\let\@@ordinalstringNdutch=\@@ordinalstringMdutch

```

First letter of each word in upper case:

```

259 \newcommand*{\@OrdinalstringMdutch}[2]{%
260   \@ordinalstringMdutch{#1}{\@num@str}%
261   \def\@hundred{Honderd}\def\@thousand{Duizend}%
262   \def\@hundredth{Honderdste}\def\@thousandth{Duizendste}%
263   \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
264 }%
265 \global\let\@OrdinalstringMdutch\@OrdinalstringMdutch

```

No gender in Dutch, so make feminine and neuter same as masculine:

```

266 \global\let\@OrdinalstringFdutch=\@OrdinalstringMdutch
267 \global\let\@OrdinalstringNdutch=\@OrdinalstringMdutch

```

Helper macro for formatting numbers under 100 as words. Handles units (1–9), teens (10–19), and compound numbers (20–99) with proper diaeresis where needed.

```

268 \newcommand*{\@numberunderhundreddutch}[2]{%
269   \ifnum#1<10\relax
270   \ifnum#1>0\relax
271     \eappto#2{\@unitstring{#1}}%
272     \fi
273   \else
274     \@tmpstrctr=#1\relax
275     \@FCmodulo{\@tmpstrctr}{10}%
276     \ifnum#1<20\relax
277       \eappto#2{\@teenstring{\@tmpstrctr}}%
278       \else
279         \ifnum\@tmpstrctr=0\relax
280           \else

```

For digits ending with an ‘e’, a diaeresis (trema) gets added to \@andname to prevent ambiguity in pronunciation. Examples: drieëntwintig (23), tweeënveertig (42).

```

281   \ifnum\@tmpstrctr=2\relax\def\@andname{ën}%
282   \else\ifnum\@tmpstrctr=3\relax\def\@andname{ëñ}%
283   \else\def\@andname{en}%
284   \fi\fi%
285   \eappto#2{\@unitstring{\@tmpstrctr}\@andname}%
286   \fi
287   \@tmpstrctr=#1\relax
288   \divide\@tmpstrctr by 10\relax
289   \eappto#2{\@tenstring{\@tmpstrctr}}%
290   \fi
291   \fi
292 }%
293 \global\let\@numberunderhundreddutch\@numberunderhundreddutch
294 \newcommand*{\@numberunderhundredthdutch}[2]{%
295   \ifnum#1<10\relax
296     \eappto#2{\@unitthstring{#1}}%
297     \else
298       \@tmpstrctr=#1\relax
299       \@FCmodulo{\@tmpstrctr}{10}%
300       \ifnum#1<20\relax

```

```

301 \eappto#2{\@teenthstring{\@tmpstrctr}}%
302 \else
303 \ifnum\@tmpstrctr=0\relax
304 \else

```

Again, for digits ending with an 'e', a trema gets added for \@andname (drieëntwintig or tweeënveertig).

```

305 \ifnum\@tmpstrctr=2\relax\def\@andname{ën}%
306 \else\ifnum\@tmpstrctr=3\relax\def\@andname{ën}%
307 \else\def\@andname{en}%
308 \fi\fi%
309 \eappto#2{\@unitstring{\@tmpstrctr}\@andname}%
310 \fi
311 \@tmpstrctr=#1\relax
312 \divide\@tmpstrctr by 10\relax
313 \eappto#2{\@tenthstring{\@tmpstrctr}}%
314 \fi
315 \fi
316 }%
317 \global\let\@@numberunderhundredthdutch\@@numberunderhundredthdutch

```

### 10.1.5 fc-english.def

English definitions

```
1 \ProvidesFCLanguage{english}[2016/01/12]%
```

Define macro that converts a number or count register (first argument) to an ordinal, and stores the result in the second argument, which should be a control sequence.

```

2 \newcommand*\@ordinalMenglish[2]{%
3 \def\@fc@ord{}%
4 \@orgargctr=#1\relax
5 \@ordinalctr=#1%
6 \@FCmodulo{\@ordinalctr}{100}%
7 \ifnum\@ordinalctr=11\relax
8 \def\@fc@ord{th}%
9 \else
10 \ifnum\@ordinalctr=12\relax
11 \def\@fc@ord{th}%
12 \else
13 \ifnum\@ordinalctr=13\relax
14 \def\@fc@ord{th}%
15 \else
16 \@FCmodulo{\@ordinalctr}{10}%
17 \ifcase\@ordinalctr
18 \def\@fc@ord{th}% case 0
19 \or \def\@fc@ord{st}% case 1
20 \or \def\@fc@ord{nd}% case 2
21 \or \def\@fc@ord{rd}% case 3
22 \else
23 \def\@fc@ord{th}% default case

```

```

24     \fi
25     \fi
26     \fi
27 \fi
28 \edef#2{\number#1\relax\noexpand\fmtord{\@fc@ord}}%
29 }%
30 \global\let\@ordinalMenglish\@ordinalMenglish

```

There is no gender difference in English, so make feminine and neuter the same as the masculine.

```

31 \global\let\@ordinalFenglish=\@ordinalMenglish
32 \global\let\@ordinalNenglish=\@ordinalMenglish

```

Define the macro that prints the value of a T<sub>E</sub>X count register as text. To make it easier, break it up into units, teens and tens. First, the units: the argument should be between 0 and 9 inclusive.

```

33 \newcommand*{\@unitstringenglish[1]}{%
34     \ifcase#1\relax
35         zero%
36         \or one%
37         \or two%
38         \or three%
39         \or four%
40         \or five%
41         \or six%
42         \or seven%
43         \or eight%
44         \or nine%
45 \fi
46 }%
47 \global\let\@unitstringenglish\@unitstringenglish

```

Next the tens, again the argument should be between 0 and 9 inclusive.

```

48 \newcommand*{\@tenstringenglish[1]}{%
49     \ifcase#1\relax
50         \or ten%
51         \or twenty%
52         \or thirty%
53         \or forty%
54         \or fifty%
55         \or sixty%
56         \or seventy%
57         \or eighty%
58         \or ninety%
59 \fi
60 }%
61 \global\let\@tenstringenglish\@tenstringenglish

```

Finally the teens, again the argument should be between 0 and 9 inclusive.

```

62 \newcommand*{\@teenstringenglish[1]}{%
63     \ifcase#1\relax

```

```

64     ten%
65     \or eleven%
66     \or twelve%
67     \or thirteen%
68     \or fourteen%
69     \or fifteen%
70     \or sixteen%
71     \or seventeen%
72     \or eighteen%
73     \or nineteen%
74     \fi
75 }%
76 \global\let\@@teenstringenglish\@@teenstringenglish

```

As above, but with the initial letter in uppercase. The units:

```

77 \newcommand*\@@Unitstringenglish[1]{%
78   \ifcase#1\relax
79     Zero%
80     \or One%
81     \or Two%
82     \or Three%
83     \or Four%
84     \or Five%
85     \or Six%
86     \or Seven%
87     \or Eight%
88     \or Nine%
89     \fi
90 }%
91 \global\let\@@Unitstringenglish\@@Unitstringenglish

```

The tens:

```

92 \newcommand*\@@Tenstringenglish[1]{%
93   \ifcase#1\relax
94     \or Ten%
95     \or Twenty%
96     \or Thirty%
97     \or Forty%
98     \or Fifty%
99     \or Sixty%
100    \or Seventy%
101    \or Eighty%
102    \or Ninety%
103    \fi
104 }%
105 \global\let\@@Tenstringenglish\@@Tenstringenglish

```

The teens:

```

106 \newcommand*\@@Teenstringenglish[1]{%
107   \ifcase#1\relax
108     Ten%

```

```

109 \or Eleven%
110 \or Twelve%
111 \or Thirteen%
112 \or Fourteen%
113 \or Fifteen%
114 \or Sixteen%
115 \or Seventeen%
116 \or Eighteen%
117 \or Nineteen%
118 \fi
119 }%
120 \global\let\@@Teenstringenglish\@@Teenstringenglish
This has changed in version 1.09, so that it now stores the result in the second argument, but
doesn't display anything. Since it only affects internal macros, it shouldn't affect documents
created with older versions. (These internal macros are not meant for use in documents.)
121 \newcommand*\@@numberstringenglish[2]{%
122 \ifnum#1>99999
123 \PackageError{fmtcount}{Out of range}%
124 {This macro only works for values less than 100000}%
125 \else
126 \ifnum#1<0
127 \PackageError{fmtcount}{Negative numbers not permitted}%
128 {This macro does not work for negative numbers, however
129 you can try typing "minus" first, and then pass the modulus of
130 this number}%
131 \fi
132 \fi
133 \def#2{}%
134 \@strctr=#1\relax \divide\@strctr by 1000\relax
135 \ifnum\@strctr>9
136 \divide\@strctr by 10
137 \ifnum\@strctr>1\relax
138 \let\@@fc@numstr#2\relax
139 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
140 \@strctr=#1 \divide\@strctr by 1000\relax
141 \@FCmodulo{\@strctr}{10}%
142 \ifnum\@strctr>0\relax
143 \let\@@fc@numstr#2\relax
144 \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
145 \fi
146 \else
147 \@strctr=#1\relax
148 \divide\@strctr by 1000\relax
149 \@FCmodulo{\@strctr}{10}%
150 \let\@@fc@numstr#2\relax
151 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
152 \fi
153 \let\@@fc@numstr#2\relax
154 \edef#2{\@@fc@numstr \@thousand}%

```

```

155 \else
156 \ifnum \@strctr>0\relax
157 \let\@@fc@numstr#2\relax
158 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@thousand}%
159 \fi
160 \fi
161 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
162 \divide\@strctr by 100
163 \ifnum \@strctr>0\relax
164 \ifnum#1>1000\relax
165 \let\@@fc@numstr#2\relax
166 \edef#2{\@@fc@numstr\ }%
167 \fi
168 \let\@@fc@numstr#2\relax
169 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@hundred}%
170 \fi
171 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
172 \ifnum#1>100\relax
173 \ifnum \@strctr>0\relax
174 \let\@@fc@numstr#2\relax
175 \edef#2{\@@fc@numstr\ \@andname\ }%
176 \fi
177 \fi
178 \ifnum \@strctr>19\relax
179 \divide\@strctr by 10\relax
180 \let\@@fc@numstr#2\relax
181 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
182 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
183 \ifnum \@strctr>0\relax
184 \let\@@fc@numstr#2\relax
185 \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
186 \fi
187 \else
188 \ifnum \@strctr<10\relax
189 \ifnum \@strctr=0\relax
190 \ifnum#1<100\relax
191 \let\@@fc@numstr#2\relax
192 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
193 \fi
194 \else
195 \let\@@fc@numstr#2\relax
196 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
197 \fi
198 \else
199 \@FCmodulo{\@strctr}{10}%
200 \let\@@fc@numstr#2\relax
201 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
202 \fi
203 \fi

```

```

204 }%
205 \global\let\@@numberstringenglish\@numberstringenglish

```

All lower case version, the second argument must be a control sequence.

```

206 \newcommand*\@numberstringMenglish}[2]{%
207   \let\@unitstring=\@unitstringenglish
208   \let\@teenstring=\@teenstringenglish
209   \let\@tenstring=\@tenstringenglish
210   \def\@hundred{hundred}\def\@thousand{thousand}%
211   \def\@andname{and}%
212   \@numberstringenglish{#1}{#2}%
213 }%
214 \global\let\@numberstringMenglish\@numberstringMenglish

```

There is no gender in English, so make feminine and neuter the same as the masculine.

```

215 \global\let\@numberstringFenglish=\@numberstringMenglish
216 \global\let\@numberstringNenglish=\@numberstringMenglish

```

This version makes the first letter of each word an uppercase character (except “and”). The second argument must be a control sequence.

```

217 \newcommand*\@NumberstringMenglish}[2]{%
218   \let\@unitstring=\@Unitstringenglish
219   \let\@teenstring=\@Teenstringenglish
220   \let\@tenstring=\@Tenstringenglish
221   \def\@hundred{Hundred}\def\@thousand{Thousand}%
222   \def\@andname{and}%
223   \@numberstringenglish{#1}{#2}%
224 }%
225 \global\let\@NumberstringMenglish\@NumberstringMenglish

```

There is no gender in English, so make feminine and neuter the same as the masculine.

```

226 \global\let\@NumberstringFenglish=\@NumberstringMenglish
227 \global\let\@NumberstringNenglish=\@NumberstringMenglish

```

Define a macro that produces an ordinal as a string. Again, break it up into units, teens and tens. First the units:

```

228 \newcommand*\@unitthstringenglish[1]{%
229   \ifcase#1\relax
230     zeroth%
231     \or first%
232     \or second%
233     \or third%
234     \or fourth%
235     \or fifth%
236     \or sixth%
237     \or seventh%
238     \or eighth%
239     \or ninth%
240   \fi
241 }%
242 \global\let\@unitthstringenglish\@unitthstringenglish

```



Next the tens:

```
243 \newcommand*{\@@tenthstringenglish[1]}{%
244   \ifcase#1\relax
245     \or tenth%
246     \or twentieth%
247     \or thirtieth%
248     \or fortieth%
249     \or fiftieth%
250     \or sixtieth%
251     \or seventieth%
252     \or eightieth%
253     \or ninetieth%
254   \fi
255 }%
256 \global\let\@@tenthstringenglish\@@tenthstringenglish
```

The teens:

```
257 \newcommand*{\@@teenthstringenglish[1]}{%
258   \ifcase#1\relax
259     tenth%
260     \or eleventh%
261     \or twelfth%
262     \or thirteenth%
263     \or fourteenth%
264     \or fifteenth%
265     \or sixteenth%
266     \or seventeenth%
267     \or eighteenth%
268     \or nineteenth%
269   \fi
270 }%
271 \global\let\@@teenthstringenglish\@@teenthstringenglish
```

As before, but with the first letter in upper case. The units:

```
272 \newcommand*{\@@Unitthstringenglish[1]}{%
273   \ifcase#1\relax
274     Zeroth%
275     \or First%
276     \or Second%
277     \or Third%
278     \or Fourth%
279     \or Fifth%
280     \or Sixth%
281     \or Seventh%
282     \or Eighth%
283     \or Ninth%
284   \fi
285 }%
286 \global\let\@@Unitthstringenglish\@@Unitthstringenglish
```

The tens:

```

287 \newcommand*\@@Tenthstringenglish[1]{%
288   \ifcase#1\relax
289     \or Tenth%
290     \or Twentieth%
291     \or Thirtieth%
292     \or Fortieth%
293     \or Fiftieth%
294     \or Sixtieth%
295     \or Seventieth%
296     \or Eightieth%
297     \or Ninetieth%
298   \fi
299 }%
300 \global\let\@@Tenthstringenglish\@@Tenthstringenglish

```

The teens:

```

301 \newcommand*\@@Teenthstringenglish[1]{%
302   \ifcase#1\relax
303     Tenth%
304     \or Eleventh%
305     \or Twelfth%
306     \or Thirteenth%
307     \or Fourteenth%
308     \or Fifteenth%
309     \or Sixteenth%
310     \or Seventeenth%
311     \or Eighteenth%
312     \or Nineteenth%
313   \fi
314 }%
315 \global\let\@@Teenthstringenglish\@@Teenthstringenglish

```

Again, as from version 1.09, this has been changed to take two arguments, where the second argument is a control sequence. The resulting text is stored in the control sequence, and nothing is displayed.

```

316 \newcommand*\@@ordinalstringenglish[2]{%
317   \@strctr=#1\relax
318   \ifnum#1>99999
319     \PackageError{fmtcount}{Out of range}%
320     {This macro only works for values less than 100000 (value given: \number\@strctr)}%
321   \else
322     \ifnum#1<0
323       \PackageError{fmtcount}{Negative numbers not permitted}%
324       {This macro does not work for negative numbers, however
325       you can try typing "minus" first, and then pass the modulus of
326       this number}%
327     \fi
328     \def#2{}%
329     \fi
330     \@strctr=#1\relax \divide\@strctr by 1000\relax

```

```

331 \ifnum \@strctr>9\relax
    #1 is greater or equal to 10000
332 \divide \@strctr by 10
333 \ifnum \@strctr>1\relax
334 \let \@fc@ordstr#2\relax
335 \edef#2{\@fc@ordstr\@tenstring{\@strctr}}%
336 \@strctr=#1\relax
337 \divide \@strctr by 1000\relax
338 \@FCmodulo{\@strctr}{10}%
339 \ifnum \@strctr>0\relax
340 \let \@fc@ordstr#2\relax
341 \edef#2{\@fc@ordstr-\@unitstring{\@strctr}}%
342 \fi
343 \else
344 \@strctr=#1\relax \divide \@strctr by 1000\relax
345 \@FCmodulo{\@strctr}{10}%
346 \let \@fc@ordstr#2\relax
347 \edef#2{\@fc@ordstr\@teenstring{\@strctr}}%
348 \fi
349 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
350 \ifnum \@strctr=0\relax
351 \let \@fc@ordstr#2\relax
352 \edef#2{\@fc@ordstr \@thousandth}%
353 \else
354 \let \@fc@ordstr#2\relax
355 \edef#2{\@fc@ordstr \@thousand}%
356 \fi
357 \else
358 \ifnum \@strctr>0\relax
359 \let \@fc@ordstr#2\relax
360 \edef#2{\@fc@ordstr\@unitstring{\@strctr}}%
361 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
362 \let \@fc@ordstr#2\relax
363 \ifnum \@strctr=0\relax
364 \edef#2{\@fc@ordstr \@thousandth}%
365 \else
366 \edef#2{\@fc@ordstr \@thousand}%
367 \fi
368 \fi
369 \fi
370 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
371 \divide \@strctr by 100
372 \ifnum \@strctr>0\relax
373 \ifnum#1>1000\relax
374 \let \@fc@ordstr#2\relax
375 \edef#2{\@fc@ordstr }%
376 \fi
377 \let \@fc@ordstr#2\relax
378 \edef#2{\@fc@ordstr\@unitstring{\@strctr}}%

```

```

379 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
380 \let\@@fc@ordstr#2\relax
381 \ifnum\@strctr=0\relax
382   \edef#2{\@@fc@ordstr\ \@hundredth}%
383 \else
384   \edef#2{\@@fc@ordstr\ \@hundred}%
385 \fi
386 \fi
387 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
388 \ifnum#1>100\relax
389   \ifnum\@strctr>0\relax
390     \let\@@fc@ordstr#2\relax
391     \edef#2{\@@fc@ordstr\ \@andname\ }%
392   \fi
393 \fi
394 \ifnum\@strctr>19\relax
395   \@tmpstrctr=\@strctr
396   \divide\@strctr by 10\relax
397   \@FCmodulo{\@tmpstrctr}{10}%
398   \let\@@fc@ordstr#2\relax
399   \ifnum\@tmpstrctr=0\relax
400     \edef#2{\@@fc@ordstr\@tenthstring{\@strctr}}%
401   \else
402     \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
403   \fi
404   \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
405   \ifnum\@strctr>0\relax
406     \let\@@fc@ordstr#2\relax
407     \edef#2{\@@fc@ordstr-\@unitthstring{\@strctr}}%
408   \fi
409 \else
410   \ifnum\@strctr<10\relax
411     \ifnum\@strctr=0\relax
412       \ifnum#1<100\relax
413         \let\@@fc@ordstr#2\relax
414         \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
415       \fi
416     \else
417       \let\@@fc@ordstr#2\relax
418       \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
419     \fi
420   \else
421     \@FCmodulo{\@strctr}{10}%
422     \let\@@fc@ordstr#2\relax
423     \edef#2{\@@fc@ordstr\@teenthstring{\@strctr}}%
424   \fi
425 \fi
426 }%
427 \global\let\@@ordinalstringenglish\@@ordinalstringenglish

```

All lower case version. Again, the second argument must be a control sequence in which the resulting text is stored.

```

428 \newcommand*{\@ordinalstringMenglish}[2]{%
429   \let\@unitthstring=\@unitthstringenglish
430   \let\@teenthstring=\@teenthstringenglish
431   \let\@tenthstring=\@tenthstringenglish
432   \let\@unitstring=\@unitstringenglish
433   \let\@teenstring=\@teenstringenglish
434   \let\@tenstring=\@tenstringenglish
435   \def\@andname{and}%
436   \def\@hundred{hundred}\def\@thousand{thousand}%
437   \def\@hundredth{hundredth}\def\@thousandth{thousandth}%
438   \@ordinalstringenglish{#1}{#2}%
439 }%
440 \global\let\@ordinalstringMenglish\@ordinalstringMenglish

```

No gender in English, so make feminine and neuter same as masculine:

```

441 \global\let\@ordinalstringFenglish=\@ordinalstringMenglish
442 \global\let\@ordinalstringNenglish=\@ordinalstringMenglish

```

First letter of each word in upper case:

```

443 \newcommand*{\@OrdinalstringMenglish}[2]{%
444   \let\@unitthstring=\@Unitthstringenglish
445   \let\@teenthstring=\@Teenthstringenglish
446   \let\@tenthstring=\@Tenthstringenglish
447   \let\@unitstring=\@Unitstringenglish
448   \let\@teenstring=\@Teenstringenglish
449   \let\@tenstring=\@Tenstringenglish
450   \def\@andname{and}%
451   \def\@hundred{Hundred}\def\@thousand{Thousand}%
452   \def\@hundredth{Hundredth}\def\@thousandth{Thousandth}%
453   \@ordinalstringenglish{#1}{#2}%
454 }%
455 \global\let\@OrdinalstringMenglish\@OrdinalstringMenglish

```

No gender in English, so make feminine and neuter same as masculine:

```

456 \global\let\@OrdinalstringFenglish=\@OrdinalstringMenglish
457 \global\let\@OrdinalstringNenglish=\@OrdinalstringMenglish

```

### 10.1.6 fc-francais.def

```

1 \ProvidesFCLanguage{francais}[2013/08/17]%
2 \FCloadlang{french}%

```

Set francais to be equivalent to french.

```

3 \global\let\@ordinalMfrancais=\@ordinalMfrench
4 \global\let\@ordinalFfrancais=\@ordinalFfrench
5 \global\let\@ordinalNfrancais=\@ordinalNfrench
6 \global\let\@numberstringMfrancais=\@numberstringMfrench
7 \global\let\@numberstringFfrancais=\@numberstringFfrench
8 \global\let\@numberstringNfrancais=\@numberstringNfrench

```

```

9 \global\let\@NumberstringMfrançais=\@NumberstringMfrench
10 \global\let\@NumberstringFfrançais=\@NumberstringFfrench
11 \global\let\@NumberstringNfrançais=\@NumberstringNfrench
12 \global\let\@OrdinalstringMfrançais=\@OrdinalstringMfrench
13 \global\let\@OrdinalstringFfrançais=\@OrdinalstringFfrench
14 \global\let\@OrdinalstringNfrançais=\@OrdinalstringNfrench
15 \global\let\@OrdinalstringMfrançais=\@OrdinalstringMfrench
16 \global\let\@OrdinalstringFfrançais=\@OrdinalstringFfrench
17 \global\let\@OrdinalstringNfrançais=\@OrdinalstringNfrench

```

### 10.1.7 fc-french.def

Definitions for French.

```
1 \ProvidesFCLanguage{french}[2017/06/15]%
```

Package fcprefix is needed to format the prefix  $\langle n \rangle$  in  $\langle n \rangle$ illion or  $\langle n \rangle$ illiard. Big numbers were developed based on reference: [http://www.alain.be/boece/noms\\_de\\_nombre.html](http://www.alain.be/boece/noms_de_nombre.html). Package fcprefix is now loaded by fmtcount.

First of all we define two macros `\fc@gl@let` and `\fc@gl@def` used in place of `\let` and `\def` within options setting macros. This way we can control from outside these macros whether the respective `\let` or `\def` is group-local or global. By default they are defined to be group-local.

```

2 \ifcsundef{fc@gl@let}{\global\let\fc@gl@let\let}{\PackageError{fmtcount}{Command already defined.}}
3 \protect\fc@gl@let\space already defined.}}
4 \ifcsundef{fc@gl@def}{\global\let\fc@gl@def\def}{\PackageError{fmtcount}{Command already defined.}}
5 \protect\fc@gl@def\space already defined.}}

```

Options for controlling plural mark. First of all we define some temporary macro `\fc@french@set@plural` in order to factorize code that defines an plural mark option:

```

#1 key name,
#2 key value,
#3 configuration index for 'reformed',
#4 configuration index for 'traditional',
#5 configuration index for 'reformed o', and
#6 configuration index for 'traditional o'.
6 \gdef\fc@french@set@plural#1#2#3#4#5#6{%
7 \ifthenelse{\equal{#2}{reformed}}{%
8 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#3}%
9 }{%
10 \ifthenelse{\equal{#2}{traditional}}{%
11 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#4}%
12 }{%
13 \ifthenelse{\equal{#2}{reformed o}}{%
14 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#5}%
15 }{%
16 \ifthenelse{\equal{#2}{traditional o}}{%
17 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#6}%
18 }{%
19 \ifthenelse{\equal{#2}{always}}{%

```

```

20     \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{0}%
21 }{%
22     \ifthenelse{\equal{#2}{never}}{%
23         \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{1}%
24     }{%
25         \ifthenelse{\equal{#2}{multiple}}{%
26             \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{2}%
27         }{%
28             \ifthenelse{\equal{#2}{multiple g-last}}{%
29                 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{3}%
30             }{%
31                 \ifthenelse{\equal{#2}{multiple l-last}}{%
32                     \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{4}%
33                 }{%
34                     \ifthenelse{\equal{#2}{multiple lng-last}}{%
35                         \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{5}%
36                     }{%
37                         \ifthenelse{\equal{#2}{multiple ng-last}}{%
38                             \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{6}%
39                         }{%
40                             \PackageError{fmtcount}{Unexpected argument}{%
41                                 '#2' was unexpected: french option '#1 plural' expects 'reformed', 't
42                                 'reformed o', 'traditional o', 'always', 'never', 'multiple', 'multip
43                                 'multiple l-last', 'multiple lng-last', or 'multiple ng-last'.%
44                                 }}}}}}}}}}}}}

```

Now a shorthand `\@tempa` is defined just to define all the options controlling plural mark. This shorthand takes into account that ‘reformed’ and ‘traditional’ have the same effect, and so do ‘reformed o’ and ‘traditional o’.

```

45 \def\@tempa#1#2#3{%
46     \define@key{fcfrench}{#1 plural}[reformed]{%
47         \fc@french@set@plural{#1}{##1}{#2}{#3}{#3}%
48     }%

```

Macro `\@tempb` takes a macro as argument, and makes its current definition global. Like here it is useful when the macro name contains non-letters, and we have to resort to the `\csname... \endcsname` construct.

```

49     \expandafter\@tempb\csname KV@fcfrench@#1 plural\endcsname
50 }%
51 \def\@tempb#1{%
52     \global\let#1#1
53 }%
54 \@tempa{vingt}{4}{5}
55 \@tempa{cent}{4}{5}
56 \@tempa{mil}{0}{0}
57 \@tempa{n-illion}{2}{6}
58 \@tempa{n-illiard}{2}{6}

```

For option ‘all plural’ we cannot use the `\@tempa` shorthand, because ‘all plural’ is just a multiplexer.

```

59 \define@key{fcfrench}{all plural}[reformed]{%
60 \csname KV@fcfrench@vingt plural\endcsname{#1}%
61 \csname KV@fcfrench@cent plural\endcsname{#1}%
62 \csname KV@fcfrench@mil plural\endcsname{#1}%
63 \csname KV@fcfrench@n-illion plural\endcsname{#1}%
64 \csname KV@fcfrench@n-illiard plural\endcsname{#1}%
65 }%
66 \expandafter\@tempb\csname KV@fcfrench@all plural\endcsname

```

Now options ‘dash or space’, we have three possible key values:

traditional use dash for numbers below 100, except when ‘et’ is used, and space otherwise

reformed reform of 1990, use dash except with million & milliard, and suchlikes, i.e.  $\langle n \rangle$ illion and  $\langle n \rangle$ illiard,

always always use dashes to separate all words

```

67 \define@key{fcfrench}{dash or space}[reformed]{%
68 \ifthenelse{\equal{#1}{traditional}}{%
69 \let\fc@frenchoptions@supermillion@dos\space%
70 \let\fc@frenchoptions@submillion@dos\space
71 }{%
72 \ifthenelse{\equal{#1}{reformed}\or\equal{#1}{1990}}{%
73 \let\fc@frenchoptions@supermillion@dos\space
74 \def\fc@frenchoptions@submillion@dos{-}%
75 }{%
76 \ifthenelse{\equal{#1}{always}}{%
77 \def\fc@frenchoptions@supermillion@dos{-}%
78 \def\fc@frenchoptions@submillion@dos{-}%
79 }{%
80 \PackageError{fmtcount}{Unexpected argument}{%
81 French option ‘dash or space’ expects ‘always’, ‘reformed’ or ‘traditional’
82 }
83 }%
84 }%
85 }%
86 }%

```

Option ‘scale’, can take 3 possible values:

long for which  $\langle n \rangle$ illions &  $\langle n \rangle$ illiards are used with  $10^{6 \times n} = 1 \langle n \rangle$ illion, and  $10^{6 \times n + 3} = 1 \langle n \rangle$ illiard

short for which  $\langle n \rangle$ illions only are used with  $10^{3 \times n + 3} = 1 \langle n \rangle$ illion

recursive for which  $10^{18} =$  un milliard de milliards

```

87 \define@key{fcfrench}{scale}[recursive]{%
88 \ifthenelse{\equal{#1}{long}}{%
89 \let\fc@poweroften\fc@@pot@longscalefrench
90 }{%
91 \ifthenelse{\equal{#1}{recursive}}{%
92 \let\fc@poweroften\fc@@pot@recursivefrench
93 }{%
94 \ifthenelse{\equal{#1}{short}}{%

```



```

95     \let\fc@poweroften\fc@pot@shortscalefrench
96   }{%
97     \PackageError{fmtcount}{Unexpected argument}{%
98       French option ‘scale’ expects ‘long’, ‘recursive’ or ‘short’
99     }
100  }%
101 }%
102 }%
103 }%

```

Option ‘n-illiard upto’ is ignored if ‘scale’ is different from ‘long’. It can take the following values:

- infinity in that case  $\langle n \rangle$ illard are never disabled,
- infty this is just a shorthand for ‘infinity’, and
- $n$  any integer that is such that  $n > 0$ , and that  $\forall k \in \mathbb{N}, k \geq n$ , number  $10^{6 \times k + 3}$  will be formatted as “mille  $\langle n \rangle$ illions”

```

104 \define@key{fcfrench}{n-illiard upto}[infinity]{%
105   \ifthenelse{\equal{#1}{infinity}}{%
106     \def\fc@longscale@nilliard@upto{0}%
107   }{%
108     \ifthenelse{\equal{#1}{infty}}{%
109       \def\fc@longscale@nilliard@upto{0}%
110     }{%
111       \if Q\ifnum9<1#1Q\fi\else
112         \PackageError{fmtcount}{Unexpected argument}{%
113           French option ‘milliard threshold’ expects ‘infinity’, or equivalently ‘infty’, or a no
114           integer.}%
115         \fi
116         \def\fc@longscale@nilliard@upto{#1}%
117       }%
118 }%

```

Now, the options ‘france’, ‘swiss’ and ‘belgian’ are defined to select the dialect to use. Macro \@tempa is just a local shorthand to define each one of this option.

```

119 \def\@tempa#1{%
120   \define@key{fcfrench}{#1}[]{%
121     \PackageError{fmtcount}{Unexpected argument}{French option with key ‘#1’ does not take
122       any value}}%
123   \csgdef{KV{fcfrench@#1@default}}{%
124     \fc@gl@def\fmtcount@french{#1}}%
125 }%
126 \@tempa{france}\@tempa{swiss}\@tempa{belgian}%

```

Make ‘france’ the default dialect for ‘french’ language

```

127 \gdef\fmtcount@french{france}%

```

Now, option ‘dialect’ is now defined so that ‘france’, ‘swiss’ and ‘belgian’ can also be used as key values, which is more conventional although less concise.

```

128 \define@key{fcfrench}{dialect}[france]{%
129   \ifthenelse{\equal{#1}{france}}

```

```

130 \or\equal{#1}{swiss}
131 \or\equal{#1}{belgian}}{%
132 \def\fmtcount@french{#1}}{%
133 \PackageError{fmtcount}{Invalid value ‘#1’ to french option dialect key}
134 {Option ‘french’ can only take the values ‘france’,
135 ‘belgian’ or ‘swiss’}}}%
136 \expandafter\@tempb\csname KV@fcfrench@dialect\endcsname

```

The option `mil plural mark` allows to make the plural of `mil` to be regular, i.e. `mils`, instead of `mille`. By default it is `le`.

```

137 \define@key{fcfrench}{mil plural mark}[le]{%
138 \def\fc@frenchoptions@mil@plural@mark{#1}}
139 \expandafter\@tempb\csname KV@fcfrench@mil plural mark\endcsname

```

Definition of case handling macros. This should be moved somewhere else to be commonalized between all languages.

**UpperCaseFirstLetter** The macro `\fc@UpperCaseFirstLetter` is such that `\fc@UpperCaseFirstLetter<word>\@nil` expands to `\word` with first letter capitalized and remainder unchanged.

```

140 \gdef\fc@UpperCaseFirstLetter#1#2\@nil{%
141 \uppercase{#1}#2}

```

**\fc@CaseIden** The macro `\fc@CaseIden` is such that `\fc@CaseIden<word>\@nil` expands to `\word` unchanged.

```

142 \gdef\fc@CaseIden#1\@nil{%
143 #1%
144 }%

```

**\fc@UpperCaseAll** The macro `\fc@UpperCaseAll` is such that `\fc@UpperCaseAll<word>\@nil` expands to `\word` all capitalized.

```

145 \gdef\fc@UpperCaseAll#1\@nil{%
146 \uppercase{#1}%
147 }%

```

**\fc@wcase** The macro `\fc@wcase` is the capitalizing macro for word-by-word capitalization. By default we set it to identity, ie. no capitalization.

```

148 \global\let\fc@wcase\fc@CaseIden

```

**\fc@gcase** The macro `\fc@gcase` is the capitalizing macro for global (the completed number) capitalization. By default we set it to identity, ie. no capitalization.

```

149 \global\let\fc@gcase\fc@CaseIden

```

**\fc@apply@gcase** The macro `\fc@apply@gcase` simply applies `\fc@gcase` to `\@tempa`, knowing that `\@tempa` is the macro containing the result of formatting.

```

150 \gdef\fc@apply@gcase{%
    First of all we expand whatever \fc@wcase... \@nil found within \@tempa.
151 \protected@edef\@tempa{\@tempa}%
152 \protected@edef\@tempa{\expandafter\fc@gcase\@tempa\@nil}%
153 }

```

**OrdinalMfrench**

```

154 \newcommand*{\@ordinalMfrench}[2]{%

```

```

155 \iffmtord@abbrv
156 \ifnum#1=1 %
157 \edef#2{\number#1\relax\noexpand\fmtord{er}}%
158 \else
159 \edef#2{\number#1\relax\noexpand\fmtord{e}}%
160 \fi
161 \else
162 \PackageWarning{fmtcount}{Non abbreviated ordinal finals ('eme) are
163 considered incorrect in French.}%
164 \ifnum#1=1 %
165 \edef#2{\number#1\relax\noexpand\fmtord{er}}%
166 \else
167 \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'eme}}%
168 \fi
169 \fi}
170 \global\let\@ordinalMfrench\@ordinalMfrench

```

@ordinalFfrench

```

171 \newcommand*{\@ordinalFfrench}[2]{%
172 \iffmtord@abbrv
173 \ifnum#1=1 %
174 \edef#2{\number#1\relax\noexpand\fmtord{re}}%
175 \else
176 \edef#2{\number#1\relax\noexpand\fmtord{e}}%
177 \fi
178 \else
179 \PackageWarning{fmtcount}{Non abbreviated ordinal finals ('eme) are
180 considered incorrect in French.}%
181 \ifnum#1=1 %
182 \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'ere}}%
183 \else
184 \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'eme}}%
185 \fi
186 \fi}
187 \global\let\@ordinalFfrench\@ordinalFfrench

```

In French neutral gender and masculine gender are formally identical.

```

188 \global\let\@ordinalNfrench\@ordinalMfrench

```

unitstringfrench

```

189 \newcommand*{\@unitstringfrench}[1]{%
190 \noexpand\fc@wcase
191 \ifcase#1 %
192 z\'ero%
193 \or un%
194 \or deux%
195 \or trois%
196 \or quatre%
197 \or cinq%
198 \or six%
199 \or sept%

```

```

200 \or huit%
201 \or neuf%
202 \fi
203 \noexpand\@nil
204 }%
205 \global\let\@@unitstringfrench\@@unitstringfrench

```

tenstringfrench

```

206 \newcommand*\@@tenstringfrench}[1]{%
207 \noexpand\fc@wcase
208 \ifcase#1 %
209 \or dix%
210 \or vingt%
211 \or trente%
212 \or quarante%
213 \or cinquante%
214 \or soixante%
215 \or septante%
216 \or huitante%
217 \or nonante%
218 \or cent%
219 \fi
220 \noexpand\@nil
221 }%
222 \global\let\@@tenstringfrench\@@tenstringfrench

```

teenstringfrench

```

223 \newcommand*\@@teenstringfrench}[1]{%
224 \noexpand\fc@wcase
225 \ifcase#1 %
226     dix%
227 \or onze%
228 \or douze%
229 \or treize%
230 \or quatorze%
231 \or quinze%
232 \or seize%
233 \or dix\noexpand\@nil-\noexpand\fc@wcase sept%
234 \or dix\noexpand\@nil-\noexpand\fc@wcase huit%
235 \or dix\noexpand\@nil-\noexpand\fc@wcase neuf%
236 \fi
237 \noexpand\@nil
238 }%
239 \global\let\@@teenstringfrench\@@teenstringfrench

```

seventiesfrench

```

240 \newcommand*\@@seventiesfrench}[1]{%
241 \@tenstring{6}%
242 \ifnum#1=1 %
243 \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
244 \else

```

```

245 -%
246 \fi
247 \@teenstring{#1}%
248 }%
249 \global\let\@@seventiesfrench\@@seventiesfrench
\@eightiesfrench Macro \@@eightiesfrench is used to format numbers in the interval [80..89]. Argument as
follows:
#1 digit  $d_w$  such that the number to be formatted is  $80 + d_w$ 
Implicit arguments as:
\count0 weight  $w$  of the number  $d_{w+1}d_w$  to be formatted
\count1 same as \#1
\count6 input, counter giving the least weight of non zero digits in top level formatted
number integral part, with rounding down to a multiple of 3,
\count9 input, counter giving the power type of the power of ten following the eighties to
be formatted; that is '1' for "mil" and '2' for "<n>illion|<n>illiard".
250 \newcommand*\@@eightiesfrench[1]{%
251 \fc@wcase quatre\@nil-\noexpand\fc@wcase vingt%
252 \ifnum#1>0 %
253 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
254 s%
255 \fi
256 \noexpand\@nil
257 -\@unitstring{#1}%
258 \else
259 \ifcase\fc@frenchoptions@vingt@plural\space
260 s% 0: always
261 \or
262 % 1: never
263 \or
264 s% 2: multiple
265 \or
266 % 3: multiple g-last
267 \ifnum\count0=\count6\ifnum\count9=0 s\fi\fi
268 \or
269 % 4: multiple l-last
270 \ifnum\count9=1 %
271 \else
272 s%
273 \fi
274 \or
275 % 5: multiple lng-last
276 \ifnum\count9=1 %
277 \else
278 \ifnum\count0>0 %
279 s%
280 \fi
281 \fi
282 \or

```

```

283 % or 6: multiple ng-last
284 \ifnum\count0>0 %
285     s%
286 \fi
287 \fi
288 \noexpand\@nil
289 \fi
290 }%
291 \global\let\@@eightiesfrench\@@eightiesfrench
292 \newcommand*\@@ninetiesfrench}[1]{%
293 \fc@wcase quatre\@nil-\noexpand\fc@wcase vingt%
294 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
295     s%
296 \fi
297 \noexpand\@nil
298 -\@teenstring{#1}%
299 }%
300 \global\let\@@ninetiesfrench\@@ninetiesfrench
301 \newcommand*\@@seventiesfrenchswiss}[1]{%
302 \@tenstring{7}%
303 \ifnum#1=1\ \@andname\ \fi
304 \ifnum#1>1-\fi
305 \ifnum#1>0 \@unitstring{#1}\fi
306 }%
307 \global\let\@@seventiesfrenchswiss\@@seventiesfrenchswiss
308 \newcommand*\@@eightiesfrenchswiss}[1]{%
309 \@tenstring{8}%
310 \ifnum#1=1\ \@andname\ \fi
311 \ifnum#1>1-\fi
312 \ifnum#1>0 \@unitstring{#1}\fi
313 }%
314 \global\let\@@eightiesfrenchswiss\@@eightiesfrenchswiss
315 \newcommand*\@@ninetiesfrenchswiss}[1]{%
316 \@tenstring{9}%
317 \ifnum#1=1\ \@andname\ \fi
318 \ifnum#1>1-\fi
319 \ifnum#1>0 \@unitstring{#1}\fi
320 }%
321 \global\let\@@ninetiesfrenchswiss\@@ninetiesfrenchswiss

```

c@french@common Macro \fc@french@common does all the preliminary settings common to all French dialects & formatting options.

```

322 \newcommand*\fc@french@common{%
323 \let\fc@wcase\fc@CaseIden
324 \let\@unitstring=\@unitstringfrench
325 \let\@teenstring=\@teenstringfrench
326 \let\@tenstring=\@tenstringfrench
327 \def\@hundred{cent}%
328 \def\@andname{et}%
329 }%

```

```

330 \global\let\fc@french@common\fc@french@common
331 \newcommand*{\@numberstringMfrenchswiss}[2]{%
332 \fc@french@common
333 \let\fc@gcase\fc@CaseIden
334 \let\@seventies=\@seventiesfrenchswiss
335 \let\@eighties=\@eightiesfrenchswiss
336 \let\@nineties=\@ninetiesfrenchswiss
337 \let\fc@nbrstr@preamble\@empty
338 \let\fc@nbrstr@postamble\@empty
339 \@@numberstringfrench{#1}{#2}}
340 \global\let\@numberstringMfrenchswiss\@numberstringMfrenchswiss
341 \newcommand*{\@numberstringMfrenchfrance}[2]{%
342 \fc@french@common
343 \let\fc@gcase\fc@CaseIden
344 \let\@seventies=\@seventiesfrench
345 \let\@eighties=\@eightiesfrench
346 \let\@nineties=\@ninetiesfrench
347 \let\fc@nbrstr@preamble\@empty
348 \let\fc@nbrstr@postamble\@empty
349 \@@numberstringfrench{#1}{#2}}
350 \global\let\@numberstringMfrenchfrance\@numberstringMfrenchfrance
351 \newcommand*{\@numberstringMfrenchbelgian}[2]{%
352 \fc@french@common
353 \let\fc@gcase\fc@CaseIden
354 \let\@seventies=\@seventiesfrenchswiss
355 \let\@eighties=\@eightiesfrench
356 \let\@nineties=\@ninetiesfrench
357 \let\fc@nbrstr@preamble\@empty
358 \let\fc@nbrstr@postamble\@empty
359 \@@numberstringfrench{#1}{#2}}
360 \global\let\@numberstringMfrenchbelgian\@numberstringMfrenchbelgian
361 \let\@numberstringMfrench=\@numberstringMfrenchfrance
362 \newcommand*{\@numberstringFfrenchswiss}[2]{%
363 \fc@french@common
364 \let\fc@gcase\fc@CaseIden
365 \let\@seventies=\@seventiesfrenchswiss
366 \let\@eighties=\@eightiesfrenchswiss
367 \let\@nineties=\@ninetiesfrenchswiss
368 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
369 \let\fc@nbrstr@postamble\@empty
370 \@@numberstringfrench{#1}{#2}}
371 \global\let\@numberstringFfrenchswiss\@numberstringFfrenchswiss
372 \newcommand*{\@numberstringFfrenchfrance}[2]{%
373 \fc@french@common
374 \let\fc@gcase\fc@CaseIden
375 \let\@seventies=\@seventiesfrench
376 \let\@eighties=\@eightiesfrench
377 \let\@nineties=\@ninetiesfrench
378 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble

```

```

379 \let\fc@nbrstr@postamble\@empty
380 \@@numberstringfrench{#1}{#2}}
381 \global\let\@numberstringFfrenchfrance\@numberstringFfrenchfrance
382 \newcommand*{\@numberstringFfrenchbelgian}[2]{%
383 \fc@french@common
384 \let\fc@gcase\fc@CaseIden
385 \let\@seventies=\@@seventiesfrenchswiss
386 \let\@eighties=\@@eightiesfrench
387 \let\@nineties=\@@ninetiesfrench
388 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
389 \let\fc@nbrstr@postamble\@empty
390 \@@numberstringfrench{#1}{#2}}
391 \global\let\@numberstringFfrenchbelgian\@numberstringFfrenchbelgian
392 \global\let\@numberstringFfrench=\@numberstringFfrenchfrance
393 \global\let\@ordinalstringNfrench\@ordinalstringMfrench
394 \newcommand*{\@NumberstringMfrenchswiss}[2]{%
395 \fc@french@common
396 \let\fc@gcase\fc@UpperCaseFirstLetter
397 \let\@seventies=\@@seventiesfrenchswiss
398 \let\@eighties=\@@eightiesfrenchswiss
399 \let\@nineties=\@@ninetiesfrenchswiss
400 \let\fc@nbrstr@preamble\@empty
401 \let\fc@nbrstr@postamble\fc@apply@gcase
402 \@@numberstringfrench{#1}{#2}}
403 \global\let\@NumberstringMfrenchswiss\@NumberstringMfrenchswiss
404 \newcommand*{\@NumberstringMfrenchfrance}[2]{%
405 \fc@french@common
406 \let\fc@gcase\fc@UpperCaseFirstLetter
407 \let\@seventies=\@@seventiesfrench
408 \let\@eighties=\@@eightiesfrench
409 \let\@nineties=\@@ninetiesfrench
410 \let\fc@nbrstr@preamble\@empty
411 \let\fc@nbrstr@postamble\fc@apply@gcase
412 \@@numberstringfrench{#1}{#2}}
413 \global\let\@NumberstringMfrenchfrance\@NumberstringMfrenchfrance
414 \newcommand*{\@NumberstringMfrenchbelgian}[2]{%
415 \fc@french@common
416 \let\fc@gcase\fc@UpperCaseFirstLetter
417 \let\@seventies=\@@seventiesfrenchswiss
418 \let\@eighties=\@@eightiesfrench
419 \let\@nineties=\@@ninetiesfrench
420 \let\fc@nbrstr@preamble\@empty
421 \let\fc@nbrstr@postamble\fc@apply@gcase
422 \@@numberstringfrench{#1}{#2}}
423 \global\let\@NumberstringMfrenchbelgian\@NumberstringMfrenchbelgian
424 \global\let\@NumberstringMfrench=\@NumberstringMfrenchfrance
425 \newcommand*{\@NumberstringFfrenchswiss}[2]{%
426 \fc@french@common
427 \let\fc@gcase\fc@UpperCaseFirstLetter

```



```

428 \let\@seventies=\@seventiesfrenchswiss
429 \let\@eighties=\@eightiesfrenchswiss
430 \let\@nineties=\@ninetiesfrenchswiss
431 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
432 \let\fc@nbrstr@postamble\fc@apply@gcase
433 \@@numberstringfrench{#1}{#2}}
434 \global\let\@NumberstringFfrenchswiss\@NumberstringFfrenchswiss
435 \newcommand*\@NumberstringFfrenchfrance}[2]{%
436 \fc@french@common
437 \let\fc@gcase\fc@UpperCaseFirstLetter
438 \let\@seventies=\@seventiesfrench
439 \let\@eighties=\@eightiesfrench
440 \let\@nineties=\@ninetiesfrench
441 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
442 \let\fc@nbrstr@postamble\fc@apply@gcase
443 \@@numberstringfrench{#1}{#2}}
444 \global\let\@NumberstringFfrenchfrance\@NumberstringFfrenchfrance
445 \newcommand*\@NumberstringFfrenchbelgian}[2]{%
446 \fc@french@common
447 \let\fc@gcase\fc@UpperCaseFirstLetter
448 \let\@seventies=\@seventiesfrenchswiss
449 \let\@eighties=\@eightiesfrench
450 \let\@nineties=\@ninetiesfrench
451 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
452 \let\fc@nbrstr@postamble\fc@apply@gcase
453 \@@numberstringfrench{#1}{#2}}
454 \global\let\@NumberstringFfrenchbelgian\@NumberstringFfrenchbelgian
455 \global\let\@NumberstringFfrench=\@NumberstringFfrenchfrance
456 \global\let\@NumberstringNfrench\@NumberstringMfrench
457 \newcommand*\@ordinalstringMfrenchswiss}[2]{%
458 \fc@french@common
459 \let\fc@gcase\fc@CaseIden
460 \let\fc@first\fc@@firstfrench
461 \let\@seventies=\@seventiesfrenchswiss
462 \let\@eighties=\@eightiesfrenchswiss
463 \let\@nineties=\@ninetiesfrenchswiss
464 \@@ordinalstringfrench{#1}{#2}%
465 }%
466 \global\let\@ordinalstringMfrenchswiss\@ordinalstringMfrenchswiss
467 \newcommand*\fc@@firstfrench{premier}
468 \global\let\fc@@firstfrench\fc@@firstfrench

469 \newcommand*\fc@@firstFfrench{premi\protect\`ere}
470 \global\let\fc@@firstFfrench\fc@@firstFfrench
471 \newcommand*\@ordinalstringMfrenchfrance}[2]{%
472 \fc@french@common
473 \let\fc@gcase\fc@CaseIden
474 \let\fc@first=\fc@@firstfrench
475 \let\@seventies=\@seventiesfrench
476 \let\@eighties=\@eightiesfrench

```

```

477 \let\@nineties=\@ninetiesfrench
478 \@ordinalstringfrench{#1}{#2}}
479 \global\let\@ordinalstringMfrenchfrance\@ordinalstringMfrenchfrance
480 \newcommand*{\@ordinalstringMfrenchbelgian}[2]{%
481 \fc@french@common
482 \let\fc@gcase\fc@CaseIden
483 \let\fc@first=\fc@@firstfrench
484 \let\@seventies=\@seventiesfrench
485 \let\@eighties=\@eightiesfrench
486 \let\@nineties=\@ninetiesfrench
487 \@ordinalstringfrench{#1}{#2}}%
488 }%
489 \global\let\@ordinalstringMfrenchbelgian\@ordinalstringMfrenchbelgian
490 \global\let\@ordinalstringMfrench=\@ordinalstringMfrenchfrance
491 \newcommand*{\@ordinalstringFfrenchswiss}[2]{%
492 \fc@french@common
493 \let\fc@gcase\fc@CaseIden
494 \let\fc@first\fc@@firstFfrench
495 \let\@seventies=\@seventiesfrenchswiss
496 \let\@eighties=\@eightiesfrenchswiss
497 \let\@nineties=\@ninetiesfrenchswiss
498 \@ordinalstringfrench{#1}{#2}}%
499 }%
500 \global\let\@ordinalstringFfrenchswiss\@ordinalstringFfrenchswiss
501 \newcommand*{\@ordinalstringFfrenchfrance}[2]{%
502 \fc@french@common
503 \let\fc@gcase\fc@CaseIden
504 \let\fc@first=\fc@@firstFfrench
505 \let\@seventies=\@seventiesfrench
506 \let\@eighties=\@eightiesfrench
507 \let\@nineties=\@ninetiesfrench
508 \@ordinalstringfrench{#1}{#2}}%
509 }%
510 \global\let\@ordinalstringFfrenchfrance\@ordinalstringFfrenchfrance
511 \newcommand*{\@ordinalstringFfrenchbelgian}[2]{%
512 \fc@french@common
513 \let\fc@gcase\fc@CaseIden
514 \let\fc@first=\fc@@firstFfrench
515 \let\@seventies=\@seventiesfrench
516 \let\@eighties=\@eightiesfrench
517 \let\@nineties=\@ninetiesfrench
518 \@ordinalstringfrench{#1}{#2}}%
519 }%
520 \global\let\@ordinalstringFfrenchbelgian\@ordinalstringFfrenchbelgian
521 \global\let\@ordinalstringFfrench=\@ordinalstringFfrenchfrance
522 \global\let\@ordinalstringNfrench\@ordinalstringMfrench
523 \newcommand*{\@OrdinalstringMfrenchswiss}[2]{%
524 \fc@french@common
525 \let\fc@gcase\fc@UpperCaseFirstLetter

```

```

526 \let\fc@first=\fc@@firstfrench
527 \let\@seventies=\@@seventiesfrenchswiss
528 \let\@eighties=\@@eightiesfrenchswiss
529 \let\@nineties=\@@ninetiesfrenchswiss
530 \@@ordinalstringfrench{#1}{#2}%
531 }%
532 \global\let\@OrdinalstringMfrenchswiss\@OrdinalstringMfrenchswiss
533 \newcommand*{\@OrdinalstringMfrenchfrance}[2]{%
534 \fc@french@common
535 \let\fc@gcase\fc@UpperCaseFirstLetter
536 \let\fc@first\fc@@firstfrench
537 \let\@seventies=\@@seventiesfrench
538 \let\@eighties=\@@eightiesfrench
539 \let\@nineties=\@@ninetiesfrench
540 \@@ordinalstringfrench{#1}{#2}%
541 }%
542 \global\let\@OrdinalstringMfrenchfrance\@OrdinalstringMfrenchfrance
543 \newcommand*{\@OrdinalstringMfrenchbelgian}[2]{%
544 \fc@french@common
545 \let\fc@gcase\fc@UpperCaseFirstLetter
546 \let\fc@first\fc@@firstfrench
547 \let\@seventies=\@@seventiesfrench
548 \let\@eighties=\@@eightiesfrench
549 \let\@nineties=\@@ninetiesfrench
550 \@@ordinalstringfrench{#1}{#2}%
551 }%
552 \global\let\@OrdinalstringMfrenchbelgian\@OrdinalstringMfrenchbelgian
553 \global\let\@OrdinalstringMfrench=\@OrdinalstringMfrenchfrance
554 \newcommand*{\@OrdinalstringFfrenchswiss}[2]{%
555 \fc@french@common
556 \let\fc@gcase\fc@UpperCaseFirstLetter
557 \let\fc@first\fc@@firstfrench
558 \let\@seventies=\@@seventiesfrenchswiss
559 \let\@eighties=\@@eightiesfrenchswiss
560 \let\@nineties=\@@ninetiesfrenchswiss
561 \@@ordinalstringfrench{#1}{#2}%
562 }%
563 \global\let\@OrdinalstringFfrenchswiss\@OrdinalstringFfrenchswiss
564 \newcommand*{\@OrdinalstringFfrenchfrance}[2]{%
565 \fc@french@common
566 \let\fc@gcase\fc@UpperCaseFirstLetter
567 \let\fc@first\fc@@firstFfrench
568 \let\@seventies=\@@seventiesfrench
569 \let\@eighties=\@@eightiesfrench
570 \let\@nineties=\@@ninetiesfrench
571 \@@ordinalstringfrench{#1}{#2}%
572 }%
573 \global\let\@OrdinalstringFfrenchfrance\@OrdinalstringFfrenchfrance
574 \newcommand*{\@OrdinalstringFfrenchbelgian}[2]{%

```

```

575 \fc@french@common
576 \let\fc@gcase\fc@UpperCaseFirstLetter
577 \let\fc@first\fc@@firstFfrench
578 \let\@seventies=\@@seventiesfrench
579 \let\@eighties=\@@eightiesfrench
580 \let\@nineties=\@@ninetiesfrench
581 \@@ordinalstringfrench{#1}{#2}%
582 }%
583 \global\let\@OrdinalstringFfrenchbelgian\@OrdinalstringFfrenchbelgian
584 \global\let\@OrdinalstringFfrench=\@OrdinalstringFfrenchfrance
585 \global\let\@OrdinalstringNfrench\@OrdinalstringMfrench

```

`\do@plural@mark` Macro `\fc@@do@plural@mark` will expand to the plural mark of  $\langle n \rangle$ illiard,  $\langle n \rangle$ illion, mil, cent or vingt, whichever is applicable. First check that the macro is not yet defined.

```

586 \ifcsundef{fc@@do@plural@mark}{}%
587 {\PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
588   'fc@@do@plural@mark'}}

```

Arguments as follows:

**#1** plural mark, 's' in general, but for mil it is `\fc@frenchoptions@mil@plural@mark`

Implicit arguments as follows:

- `\count0` input, counter giving the weight  $w$ , this is expected to be multiple of 3,
- `\count1` input, counter giving the plural value of multiplied object  $\langle n \rangle$ illiard,  $\langle n \rangle$ illion, mil, cent or vingt, whichever is applicable, that is to say it is 1 when the considered objet is not multiplied, and 2 or more when it is multiplied,
- `\count6` input, counter giving the least weight of non zero digits in top level formatted number integral part, with rounding down to a multiple of 3,
- `\count10` input, counter giving the plural mark control option.

```

589 \def\fc@@do@plural@mark#1{%
590   \ifcase\count10 %
591     #1% 0=always
592     \or% 1=never
593     \or% 2=multiple
594     \ifnum\count1>1 %
595       #1%
596       \fi
597     \or% 3= multiple g-last
598     \ifnum\count1>1 %
599       \ifnum\count0=\count6 %
600         #1%
601         \fi
602       \fi
603     \or% 4= multiple l-last
604     \ifnum\count1>1 %
605       \ifnum\count9=1 %
606         \else
607         #1%
608         \fi
609     \fi

```

```

610 \or% 5= multiple lng-last
611   \ifnum\count1>1 %
612     \ifnum\count9=1 %
613     \else
614       \if\count0>\count6 %
615         #1%
616       \fi
617     \fi
618   \fi
619 \or% 6= multiple ng-last
620   \ifnum\count1>1 %
621     \ifnum\count0>\count6 %
622       #1%
623     \fi
624   \fi
625 \fi
626 }%
627 \global\let\fc@@do@plural@mark\fc@@do@plural@mark

```

`\@nbrstr@Fpreamble` Macro `\fc@@nbrstr@Fpreamble` do the necessary preliminaries before formatting a cardinal with feminine gender.

```

628 \ifcsundef\fc@@nbrstr@Fpreamble}{%
629   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
630     'fc@@nbrstr@Fpreamble'}}

```

`\@nbrstr@Fpreamble`

```

631 \def\fc@@nbrstr@Fpreamble{%
632   \fc@read@unit{\count1}{0}%
633   \ifnum\count1=1 %
634     \let\fc@wcase@save\fc@wcase
635     \def\fc@wcase{\noexpand\fc@wcase}%
636     \def\@nil{\noexpand\@nil}%
637     \let\fc@nbrstr@postamble\fc@@nbrstr@Fpostamble
638   \fi
639 }%
640 \global\let\fc@@nbrstr@Fpreamble\fc@@nbrstr@Fpreamble

```

`\@nbrstr@Fpostamble`

```

641 \def\fc@@nbrstr@Fpostamble{%
642   \let\fc@wcase\fc@wcase@save
643   \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
644   \def\@tempd{un}%
645   \ifx\@tempc\@tempd
646     \let\@tempc\@tempa
647     \edef\@tempa{\@tempb\fc@wcase une\@nil}%
648   \fi
649 }%
650 \global\let\fc@@nbrstr@Fpostamble\fc@@nbrstr@Fpostamble

```

`\@pot@longscalefrench` Macro `\fc@@pot@longscalefrench` is used to produce powers of ten with long scale con-

vention. The long scale convention is correct for French and elsewhere in Europe. First we check that the macro is not yet defined.

```
651 \ifcsundef{fc@pot@longscalefrench}{}%
652 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
653   'fc@pot@longscalefrench'}}
```

Argument are as follows:

- #1 input, plural value of  $d$ , that is to say: let  $d$  be the number multiplying the considered power of ten, then the plural value #2 is expected to be 0 if  $d = 0$ , 1 if  $d = 1$ , or  $> 1$  if  $d > 1$
- #2 output, counter, maybe 0 when power of ten is 1, 1 when power of ten starts with “mil(le)”, or 2 when power of ten is a “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)”
- #3 output, macro into which to place the formatted power of ten

Implicit arguments as follows:

`\count0` input, counter giving the weight  $w$ , this is expected to be multiple of 3

```
654 \def\fc@pot@longscalefrench#1#2#3{%
655   {%
```

First the input arguments are saved into local objects: #1 and #1 are respectively saved into `\@tempa` and `\@tempb`.

```
656   \edef\@tempb{\number#1}%
```

Let `\count1` be the plural value.

```
657   \count1=\@tempb
```

Let  $n$  and  $r$  the the quotient and remainder of division of weight  $w$  by 6, that is to say  $w = n \times 6 + r$  and  $0 \leq r < 6$ , then `\count2` is set to  $n$  and `\count3` is set to  $r$ .

```
658   \count2\count0 %
659   \divide\count2 by 6 %
660   \count3\count2 %
661   \multiply\count3 by 6 %
662   \count3-\count3 %
663   \advance\count3 by \count0 %
664   \ifnum\count0>0 %
```

If weight  $w$  (a.k.a. `\count0`) is such that  $w > 0$ , then  $w \geq 3$  because  $w$  is a multiple of 3. So we *may* have to append “mil(le)” or “ $\langle n \rangle$ illion(s)” or “ $\langle n \rangle$ illiard(s)”.

```
665   \ifnum\count1>0 %
```

Plural value is  $> 0$  so have at least one “mil(le)” or “ $\langle n \rangle$ illion(s)” or “ $\langle n \rangle$ illiard(s)”. We need to distinguish between the case of “mil(le)” and that of “ $\langle n \rangle$ illion(s)” or “ $\langle n \rangle$ illiard(s)”, so we `\define \@temp` to ‘1’ for “mil(le)”, and to ‘2’ otherwise.

```
666   \edef\@temp{%
667   \ifnum\count2=0 % weight=3
```

Here  $n = 0$ , with  $n = w \div 6$ , but we also know that  $w \geq 3$ , so we have  $w = 3$  which means we are in the “mil(le)” case.

```
668       1%
669   \else
670       \ifnum\count3>2 %
```

Here we are in the case of  $3 \leq r < 6$ , with  $r$  the remainder of division of weight  $w$  by 6, we should have “ $\langle n \rangle$ illiard(s)”, but that may also be “mil(le)” instead depending on option ‘n-illiard upto’, known as `\fc@longscale@nilliard@upto`.

```
671         \ifnum\fc@longscale@nilliard@upto=0 %
```

Here option ‘n-illiard upto’ is ‘infinity’, so we always use “ $\langle n \rangle$ illiard(s)”.

```
672             2%
```

```
673         \else
```

Here option ‘n-illiard upto’ indicate some threshold to which to compare  $n$  (a.k.a. `\count2`).

```
674             \ifnum\count2>\fc@longscale@nilliard@upto
```

```
675                 1%
```

```
676             \else
```

```
677                 2%
```

```
678             \fi
```

```
679         \fi
```

```
680     \else
```

```
681         2%
```

```
682     \fi
```

```
683 \fi
```

```
684 }%
```

```
685 \ifnum\@temph=1 %
```

Here  $10^w$  is formatted as “mil(le)”.

```
686     \count10=\fc@frenchoptions@mil@plural\space
```

```
687     \edef\@tempe{%
```

```
688         \noexpand\fc@wcase
```

```
689         mil%
```

```
690         \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
```

```
691         \noexpand\@nil
```

```
692     }%
```

```
693 \else
```

```
694     % weight >= 6
```

```
695     \expandafter\fc@@latin@cardinal@pefix\expandafter{\the\count2}\@tempg
```

```
696     % now form the xxx-illion(s) or xxx-illiard(s) word
```

```
697     \ifnum\count3>2 %
```

```
698         \toks10{illiard}%
```

```
699         \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
```

```
700     \else
```

```
701         \toks10{illion}%
```

```
702         \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
```

```
703     \fi
```

```
704     \edef\@tempe{%
```

```
705         \noexpand\fc@wcase
```

```
706         \@tempg
```

```
707         \the\toks10 %
```

```
708         \fc@@do@plural@mark s%
```

```
709         \noexpand\@nil
```

```
710     }%
```

```
711     \fi
712     \else
```

Here plural indicator of  $d$  indicates that  $d = 0$ , so we have  $0 \times 10^w$ , and it is not worth to format  $10^w$ , because there are none of them.

```
713     \let\@tempe\@empty
714     \def\@temph{0}%
715     \fi
716     \else
```

Case of  $w = 0$ .

```
717     \let\@tempe\@empty
718     \def\@temph{0}%
719     \fi
```

Now place into `cs@tempa` the assignment of results `\@temph` and `\@tempe` to #2 and #3 for further propagation after closing brace.

```
720     \expandafter\toks\expandafter1\expandafter{\@tempe}%
721     \toks0{#2}%
722     \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
723     \expandafter
724   }\@tempa
725 }%
726 \global\let\fc@pot@longscalefrench\fc@pot@longscalefrench
```

`\fc@pot@shortscalefrench` Macro `\fc@pot@shortscalefrench` is used to produce powers of ten with short scale convention. This convention is the US convention and is not correct for French and elsewhere in Europe. First we check that the macro is not yet defined.

```
727 \ifcsundef\fc@pot@shortscalefrench\{}-%
728 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
729   'fc@pot@shortscalefrench'}}
```

Arguments as follows — same interface as for `\fc@pot@longscalefrench`:

- #1 input, plural value of  $d$ , that is to say: let  $d$  be the number multiplying the considered power of ten, then the plural value #2 is expected to be 0 if  $d = 0$ , 1 if  $d = 1$ , or  $> 1$  if  $d > 1$
- #2 output, counter, maybe 0 when power of ten is 1, 1 when power of ten starts with “mil(le)”, or 2 when power of ten is a “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)”
- #3 output, macro into which to place the formatted power of ten

Implicit arguments as follows:

`\count0` input, counter giving the weight  $w$ , this is expected to be multiple of 3

```
730 \def\fc@pot@shortscalefrench#1#2#3{%
731   {%
```

First save input arguments #1, #2, and #3 into local macros respectively `\@tempa`, `\@tempb`, `\@tempc` and `\@tempd`.

```
732   \edef\@tempb{\number#1}%
```

And let `\count1` be the plural value.

```
733   \count1=\@tempb
```



Now, let  $\backslash\text{count2}$  be the integer  $n$  generating the pseudo latin prefix, i.e.  $n$  is such that  $w = 3 \times n + 3$ .

```
734 \count2\count0 %
735 \divide\count2 by 3 %
736 \advance\count2 by -1 %
```

Here is the real job, the formatted power of ten will go to  $\backslash\text{@tempe}$ , and its power type will go to  $\backslash\text{@temph}$ . Please remember that the power type is an index in  $[0..2]$  indicating whether  $10^w$  is formatted as *<nothing>*, “mil(le)” or “*<n>*illion(s)|*<n>*illiard(s)”.

```
737 \ifnum\count0>0 % If weight>=3, i.e we do have to append thousand or n-illion(s)/n-illiard(
738 \ifnum\count1>0 % we have at least one thousand/n-illion/n-illiard
739 \ifnum\count2=0 %
740 \def\@temph{1}%
741 \count1=\fc@frenchoptions@mil@plural\space
742 \edef\@tempe{%
743 mil%
744 \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
745 }%
746 \else
747 \def\@temph{2}%
748 % weight >= 6
749 \expandafter\fc@@latin@cardinal@pefix\expandafter{\the\count2}\@tempg
750 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
751 \edef\@tempe{%
752 \noexpand\fc@wcase
753 \@tempg
754 illion%
755 \fc@@do@plural@mark s%
756 \noexpand\@nil
757 }%
758 \fi
759 \else
```

Here we have  $d = 0$ , so nothing is to be formatted for  $d \times 10^w$ .

```
760 \def\@temph{0}%
761 \let\@tempe\@empty
762 \fi
763 \else
```

Here  $w = 0$ .

```
764 \def\@temph{0}%
765 \let\@tempe\@empty
766 \fi
767 % now place into \@cs{@tempa} the assignment of results \cs{@temph} and \cs{@tempe} to to \text
768 % \texttt{#3} for further propagation after closing brace.
769 % \begin{macrocode}
770 \expandafter\toks\expandafter1\expandafter{\@tempe}%
771 \toks0{#2}%
772 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
773 \expandafter
```

```

774 } \@tempa
775 } %
776 \global \let \fc@@pot@shortscalefrench \fc@@pot@shortscalefrench

```

Macro `\fc@@pot@recursivefrench` is used to produce power of tens that are of the form “million de milliards de milliards” for  $10^{24}$ . First we check that the macro is not yet defined.

```

777 \ifcsundef{fc@@pot@recursivefrench}{-}{%
778 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
779 'fc@@pot@recursivefrench'}}

```

The arguments are as follows — same interface as for `\fc@@pot@longscalefrench`:

- #1 input, plural value of  $d$ , that is to say: let  $d$  be the number multiplying the considered power of ten, then the plural value #2 is expected to be 0 if  $d = 0$ , 1 if  $d = 1$ , or  $> 1$  if  $d > 1$
- #2 output, counter, maybe 0 when power of ten is 1, 1 when power of ten starts with “mil(le)”, or 2 when power of ten is a “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)”
- #3 output, macro into which to place the formatted power of ten

Implicit arguments as follows:

`\count0` input, counter giving the weight  $w$ , this is expected to be multiple of 3

```

780 \def \fc@@pot@recursivefrench#1#2#3{%
781 {%

```

First the input arguments are saved into local objects: #1 and #1 are respectively saved into `\@tempa` and `\@tempb`.

```

782 \edef \@tempb{\number#1}%
783 \let \@tempa \@tempa

```

New get the inputs #1 and #1 into counters `\count0` and `\count1` as this is more practical.

```

784 \count1=\@tempb\space

```

Now compute into `\count2` how many times “de milliards” has to be repeated.

```

785 \ifnum \count1 > 0 %
786 \count2 \count0 %
787 \divide \count2 by 9 %
788 \advance \count2 by -1 %
789 \let \@tempe \@empty
790 \edef \@tempf{\fc@frenchoptions@supermillion@dos
791 de\fc@frenchoptions@supermillion@dos\fc@wcase milliards \@nil}%
792 \count11 \count0 %
793 \ifnum \count2 > 0 %
794 \count3 \count2 %
795 \count3 - \count3 %
796 \multiply \count3 by 9 %
797 \advance \count11 by \count3 %
798 \loop
799 % (\count2, \count3) <- (\count2 div 2, \count2 mod 2)
800 \count3 \count2 %
801 \divide \count3 by 2 %
802 \multiply \count3 by 2 %
803 \count3 - \count3 %
804 \advance \count3 by \count2 %

```

```

805         \divide\count2 by 2 %
806         \ifnum\count3=1 %
807             \let\@tempg\@tempe
808             \edef\@tempe{\@tempg\@tempf}%
809         \fi
810         \let\@tempg\@tempf
811         \edef\@tempf{\@tempg\@tempg}%
812         \ifnum\count2>0 %
813     \repeat
814 \fi
815 \divide\count11 by 3 %
816 \ifcase\count11 % 0 .. 5
817     % 0 => d milliard(s) (de milliards)*
818     \def\@temp{2}%
819     \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
820 \or % 1 => d mille milliard(s) (de milliards)*
821     \def\@temp{1}%
822     \count10=\fc@frenchoptions@mil@plural\space
823 \or % 2 => d million(s) (de milliards)*
824     \def\@temp{2}%
825     \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
826 \or % 3 => d milliard(s) (de milliards)*
827     \def\@temp{2}%
828     \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
829 \or % 4 => d mille milliards (de milliards)*
830     \def\@temp{1}%
831     \count10=\fc@frenchoptions@mil@plural\space
832 \else % 5 => d million(s) (de milliards)*
833     \def\@temp{2}%
834     \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
835 \fi
836 \let\@tempg\@tempe
837 \edef\@tempf{%
838     \ifcase\count11 % 0 .. 5
839     \or
840         mil\fc@@do@plural@mark \fc@frenchoptions@mil@plural@mark
841     \or
842         million\fc@@do@plural@mark s%
843     \or
844         milliard\fc@@do@plural@mark s%
845     \or
846         mil\fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
847         \noexpand\@nil\fc@frenchoptions@supermillion@dos
848         \noexpand\fc@wcase milliards% 4
849     \or
850         million\fc@@do@plural@mark s%
851         \noexpand\@nil\fc@frenchoptions@supermillion@dos
852         de\fc@frenchoptions@supermillion@dos\noexpand\fc@wcase  milliards% 5
853 \fi

```

```

854     }%
855     \edef\@tempe{%
856         \ifx\@tempf\@empty\else
857         \expandafter\fc@wcase\@tempf\@nil
858         \fi
859         \@tempg
860     }%
861 \else
862     \def\@temph{0}%
863     \let\@tempe\@empty
864 \fi

```

Now place into `cs@tempa` the assignment of results `\@temph` and `\@tempe` to #2 and #3 for further propagation after closing brace.

```

865     \expandafter\toks\expandafter1\expandafter{\@tempe}%
866     \toks0{#2}%
867     \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
868     \expandafter
869 } \@tempa
870 }%
871 \global\let\fc@@pot@recursivefrench\fc@pot@recursivefrench

```

`fc@muladdfrench`

Macro `\fc@muladdfrench` is used to format the sum of a number  $a$  and the product of a number  $d$  by a power of ten  $10^w$ . Number  $d$  is made of three consecutive digits  $d_{w+2}d_{w+1}d_w$  of respective weights  $w + 2$ ,  $w + 1$ , and  $w$ , while number  $a$  is made of all digits with weight  $w' > w + 2$  that have already been formatted. First check that the macro is not yet defined.

```

872 \ifcsundef{fc@muladdfrench}{}%
873 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
874   'fc@muladdfrench'}}

```

Arguments as follows:

- #2 input, plural indicator for number  $d$
- #3 input, formatted number  $d$
- #5 input, formatted number  $10^w$ , i.e. power of ten which is multiplied by  $d$

Implicit arguments from context:

- `\@tempa` input, formatted number  $a$   
output, macro to which place the mul-add result
- `\count8` input, power type indicator for  $10^{w'}$ , where  $w'$  is a weight of  $a$ , this is an index in  $[0..2]$  that reflects whether  $10^{w'}$  is formatted by “mil(le)” — for index = 1 — or by “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)” — for index = 2
- `\count9` input, power type indicator for  $10^w$ , this is an index in  $[0..2]$  that reflect whether the weight  $w$  of  $d$  is formatted by “metanothing” — for index = 0, “mil(le)” — for index = 1 — or by “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)” — for index = 2

```

875 \def\fc@muladdfrench#1#2#3{%
876   {%

```

First we save input arguments #1 – #3 to local macros `\@tempc`, `\@tempd` and `\@tempf`.

```

877     \edef\@tempc{#1}%
878     \edef\@tempd{#2}%

```

```

879 \edef\@tempf{#3}%
880 \let\@tempc\@tempc
881 \let\@tempd\@tempd

```

First we want to do the “multiplication” of  $d \Rightarrow \backslash\@tempd$  and of  $10^w \Rightarrow \backslash\@tempf$ . So, prior to this we do some preprocessing of  $d \Rightarrow \backslash\@tempd$ : we force  $\backslash\@tempd$  to *empty* if both  $d = 1$  and  $10^w \Rightarrow$  “mil(le)”, this is because we, French, we do not say “un mil”, but just “mil”.

```

882 \ifnum\@tempc=1 %
883 \ifnum\count9=1 %
884 \let\@tempd\@empty
885 \fi
886 \fi

```

Now we do the “multiplication” of  $d = \backslash\@tempd$  and of  $10^w = \backslash\@tempf$ , and place the result into  $\backslash\@tempg$ .

```

887 \edef\@tempg{%
888 \@tempd
889 \ifx\@tempd\@empty\else
890 \ifx\@tempf\@empty\else
891 \ifcase\count9 %
892 \or
893 \fc@frenchoptions@submillion@dos
894 \or
895 \fc@frenchoptions@supermillion@dos
896 \fi
897 \fi
898 \fi
899 \@tempf
900 }%

```

Now to the “addition” of  $a \Rightarrow \backslash\@tempa$  and  $d \times 10^w \Rightarrow \backslash\@tempg$ , and place the results into  $\backslash\@temp$ .

```

901 \edef\@temp{%
902 \@tempa
903 \ifx\@tempa\@empty\else
904 \ifx\@tempg\@empty\else
905 \ifcase\count8 %
906 \or
907 \fc@frenchoptions@submillion@dos
908 \or
909 \fc@frenchoptions@supermillion@dos
910 \fi
911 \fi
912 \fi
913 \@tempg
914 }%

```

Now propagate the result — i.e. the expansion of  $\backslash\@temp$  — into macro  $\backslash\@tempa$  after closing brace.

```

915 \def\@tempb##1{\def\@tempa{\def\@tempa{##1}}}%

```

```

916 \expandafter\@tempb\expandafter{\@tempb}%
917 \expandafter
918 }\@tempa
919 }%
920 \global\let\fc@muladdfrench\fc@muladdfrench

```

Macro `\fc@lthundredstringfrench` is used to format a number in interval  $[0..99]$ . First we check that it is not already defined.

```

921 \ifcsundef{fc@lthundredstringfrench}{\}%
922 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
923 'fc@lthundredstringfrench'}}

```

The number to format is not passed as an argument to this macro, instead each digits of it is in a `\fc@digit@<w>` macro after this number has been parsed. So the only thing that `\fc@lthundredstringfrench` needs is to know  $\langle w \rangle$  which is passed as `\count0` for the less significant digit.

#1 input/output macro to which append the result

Implicit input arguments as follows:

`\count0` weight  $w$  of least significant digit  $d_w$ .

The formatted number is appended to the content of #1, and the result is placed into #1.

```

924 \def\fc@lthundredstringfrench#1{%
925   {%

```

First save arguments into local temporary macro.

```

926 \let\@tempc#1%

```

Read units  $d_w$  to `\count1`.

```

927 \fc@read@unit{\count1}{\count0}%

```

Read tens  $d_{w+1}$  to `\count2`.

```

928 \count3\count0 %
929 \advance\count3 1 %
930 \fc@read@unit{\count2}{\count3}%

```

Now do the real job, set macro `\@tempa` to #1 followed by  $d_{w+1}d_w$  formatted.

```

931 \edef\@tempa{%
932   \@tempc
933   \ifnum\count2>1 %
934     % 20 .. 99
935     \ifnum\count2>6 %
936       % 70 .. 99
937       \ifnum\count2<8 %
938         % 70 .. 79
939         \@seventies{\count1}%
940     \else
941       % 80..99
942       \ifnum\count2<9 %
943         % 80 .. 89
944         \@eighties{\count1}%
945     \else
946       % 90 .. 99

```

```

947         \@nineties{\count1}%
948     \fi
949     \fi
950 \else
951     % 20..69
952     \@tenstring{\count2}%
953     \ifnum\count1>0 %
954         % x1 .. x0
955         \ifnum\count1=1 %
956             % x1
957             \fc@frenchoptions@submillion@dos\andname\fc@frenchoptions@submillion@dos
958         \else
959             % x2 .. x9
960             -%
961         \fi
962     \@unitstring{\count1}%
963 \fi
964 \fi
965 \else
966     % 0 .. 19
967     \ifnum\count2=0 % when tens = 0
968         % 0 .. 9
969         \ifnum\count1=0 % when units = 0
970             % \count3=1 when #1 = 0, i.e. only for the unit of the top level number
971             \ifnum\count3=1 %
972                 \ifnum\fc@max@weight=0 %
973                     \@unitstring{0}%
974                 \fi
975             \fi
976         \else
977             % 1 .. 9
978             \@unitstring{\count1}%
979         \fi
980     \else
981         % 10 .. 19
982         \@teenstring{\count1}%
983     \fi
984 \fi
985 }%

```

Now propagate the expansion of \@tempa into #1 after closing brace.

```

986 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
987 \expandafter\@tempb\expandafter{\@tempa}%
988 \expandafter
989 }\@tempa
990 }%
991 \global\let\fc@lthundredstringfrench\fc@lthundredstringfrench

```

\fc@lthousandstringfrench \fc@lthousandstringfrench is used to format a number in interval [0..999]. First we check that it is not already defined.

```

992 \ifcsundef{fc@ltthousandstringfrench}{}{%
993   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
994     'fc@ltthousandstringfrench'}}

```

Output is empty for 0. Arguments as follows:

#2 output, macro, formatted number  $d = d_{w+2}d_{w+1}d_w$

Implicit input arguments as follows:

\count0 input weight  $10^w$  of number  $d_{w+2}d_{w+1}d_w$  to be formatted.

\count5 least weight of formatted number with a non null digit.

\count9 input, power type indicator of  $10^w$   $0 \Rightarrow \emptyset$ ,  $1 \Rightarrow$  “mil(le)”,  $2 \Rightarrow$   
 $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)

```

995 \def\fc@ltthousandstringfrench#1{%
996   {%

```

Set counter \count2 to digit  $d_{w+2}$ , i.e. hundreds.

```

997   \count4\count0 %
998   \advance\count4 by 2 %
999   \fc@read@unit{\count2 }{\count4 }%

```

Check that the two subsequent digits  $d_{w+1}d_w$  are non zero, place check-result into \@tempa.

```

1000  \advance\count4 by -1 %
1001  \count3\count4 %
1002  \advance\count3 by -1 %
1003  \fc@check@nonzeros{\count3 }{\count4 }\@tempa

```

Compute plural mark of ‘cent’ into \@temps.

```

1004  \edef\@temps{%
1005    \ifcase\fc@frenchoptions@cent@plural\space
1006    % 0 => always
1007    s%
1008    \or
1009    % 1 => never
1010    \or
1011    % 2 => multiple
1012    \ifnum\count2>1s\fi
1013    \or
1014    % 3 => multiple g-last
1015    \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count0=\count6s\fi\fi\fi
1016    \or
1017    % 4 => multiple l-last
1018    \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count9=0s\else\ifnum\count9=2s\fi\fi\fi\fi
1019    \fi
1020  }%
1021  % compute spacing after cent(s?) into \@tempb
1022  \expandafter\let\expandafter\@tempb
1023    \ifnum\@tempa>0 \fc@frenchoptions@submillion@dos\else\@empty\fi
1024  % now place into \@tempa the hundreds
1025  \edef\@tempa{%
1026    \ifnum\count2=0 %
1027    \else

```



```

1028     \ifnum\count2=1 %
1029     \expandafter\fc@wcase\@hundred\@nil
1030     \else
1031     \unitstring{\count2}\fc@frenchoptions@submillion@dos
1032     \noexpand\fc@wcase\@hundred\@temps\noexpand\@nil
1033     \fi
1034     \@tempb
1035     \fi
1036 }%
1037 % now append to \@tempa the ten and unit
1038 \fc@lthundredstringfrench\@tempa

```

Propagate expansion of \@tempa into macro #1 after closing brace.

```

1039 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
1040 \expandafter\@tempb\expandafter{\@tempa}%
1041 \expandafter
1042 }\@tempa
1043 }%
1044 \global\let\fc@ltthousandstringfrench\fc@ltthousandstringfrench

```

numberstringfrenchMacro \@@numberstringfrench is the main engine for formatting cadinal numbers in French. First we check that the control sequence is not yet defined.

```

1045 \ifcsundef{@@numberstringfrench}{-}{%
1046 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro ‘@@numberstringfrench’}}

```

Arguments are as follows:

- #1 number to convert to string
- #2 macro into which to place the result

```

1047 \def\@@numberstringfrench#1#2{%
1048 {%

```

First parse input number to be formatted and do some error handling.

```

1049 \edef\@tempa{#1}%
1050 \expandafter\fc@number@parser\expandafter{\@tempa}%
1051 \ifnum\fc@min@weight<0 %
1052 \PackageError{fmtcount}{Out of range}%
1053 {This macro does not work with fractional numbers}%
1054 \fi

```

In the sequel, \@tempa is used to accumulate the formatted number. Please note that \space after \fc@sign@case is eaten by preceding number collection. This \space is needed so that when \fc@sign@case expands to ‘0’, then \@tempa is defined to ‘’ (i.e. empty) rather than to ‘\relax’.

```

1055 \edef\@tempa{\ifcase\fc@sign@case\space\or\fc@wcase plus\@nil\or\fc@wcase moins\@nil\fi}%
1056 \fc@nbrstr@preamble
1057 \fc@@nbrstrfrench@inner
1058 \fc@nbrstr@postamble

```

Propagate the result — i.e. expansion of \@tempa — into macro #2 after closing brace.

```

1059 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
1060 \expandafter\@tempb\expandafter{\@tempa}%

```

```

1061 \expandafter
1062 }\@tempa
1063 }%
1064 \global\let\@@numberstringfrench\@@numberstringfrench

```

`\@@numberstringfrench@inner` Common part of `\@@numberstringfrench` and `\@@ordinalstringfrench`. Arguments are as follows:

`\@tempa` input/output, macro to which the result is to be aggregated, initially empty or contains the sign indication.

```

1065 \def\fc@@nbrstrfrench@inner{%

```

Now loop, first we compute starting weight as  $3 \times \left\lfloor \frac{\text{\fc@max@weight}}{3} \right\rfloor$  into `\count0`.

```

1066 \count0=\fc@max@weight
1067 \divide\count0 by 3 %
1068 \multiply\count0 by 3 %

```

Now we compute final weight into `\count5`, and round down to multiple of 3 into `\count6`.

Warning: `\count6` is an implicit input argument to macro `\fc@ltthousandstringfrench`.

```

1069 \fc@intpart@find@last{\count5 }%
1070 \count6\count5 %
1071 \divide\count6 3 %
1072 \multiply\count6 3 %
1073 \count8=0 %
1074 \loop

```

First we check whether digits in weight interval  $[w..(w+2)]$  are all zero and place check result into macro `\@tempt`.

```

1075 \count1\count0 %
1076 \advance\count1 by 2 %
1077 \fc@check@nonzeros{\count0 }{\count1 }\@tempt

```

Now we generate the power of ten  $10^w$ , formatted power of ten goes to `\@tempb`, while power type indicator goes to `\count9`.

```

1078 \fc@poweroften\@tempt{\count9 }\@tempb

```

Now we generate the formatted number  $d$  into macro `\@tempd` by which we need to multiply  $10^w$ . Implicit input argument is `\count9` for power type of  $10^9$ , and `\count6`

```

1079 \fc@ltthousandstringfrench\@tempd

```

Finally do the multiplication-addition. Implicit arguments are `\@tempa` for input/output growing formatted number, `\count8` for input previous power type, i.e. power type of  $10^{w+3}$ , `\count9` for input current power type, i.e. power type of  $10^w$ .

```

1080 \fc@muladdfrench\@tempt\@tempd\@tempb

```

Then iterate.

```

1081 \count8\count9 %
1082 \advance\count0 by -3 %
1083 \ifnum\count6>\count0 \else
1084 \repeat
1085 }%

```

```

1086 \global\let\fc@@nbrstrfrench@inner\fc@@nbrstrfrench@inner

```

rdinalstringfrench Macro \@ordinalstringfrench is the main engine for formatting ordinal numbers in French. First check it is not yet defined.

```
1087 \ifcsundef{@@ordinalstringfrench}{-}{%
1088   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1089     '@@ordinalstringfrench'}}
```

Arguments are as follows:

- #1 number to convert to string
- #2 macro into which to place the result

```
1090 \def\@@ordinalstringfrench#1#2{%
1091   {%
```

First parse input number to be formatted and do some error handling.

```
1092   \edef\@tempa{#1}%
1093   \expandafter\fc@number@parser\expandafter{\@tempa}%
1094   \ifnum\fc@min@weight<0 %
1095     \PackageError{fmtcount}{Out of range}%
1096     {This macro does not work with fractional numbers}%
1097   \fi
1098   \ifnum\fc@sign@case>0 %
1099     \PackageError{fmtcount}{Out of range}%
1100     {This macro does with negative or explicitly marked as positive numbers}%
1101   \fi
```

Now handle the special case of first. We set \count0 to 1 if we are in this case, and to 0 otherwise

```
1102   \ifnum\fc@max@weight=0 %
1103     \ifnum\csname fc@digit@0\endcsname=1 %
1104       \count0=1 %
1105     \else
1106       \count0=0 %
1107     \fi
1108   \else
1109     \count0=0 %
1110   \fi
1111   \ifnum\count0=1 %
1112     \expandafter\@firstoftwo
1113   \else
1114     \expandafter\@secondoftwo
1115   \fi
1116   {%
1117   \protected@edef\@tempa{\expandafter\fc@wcase\fc@first\@nil}%
1118   }%
```

Now we tamper a little bit with the plural handling options to ensure that there is no final plural mark.

```
1119   {%
1120   \def\@tempa##1{%
1121     \expandafter\edef\csname fc@frenchoptions@##1@plural\endcsname{%
```

```

1122     \ifcase\csname fc@frenchoptions@##1@plural\endcsname\space
1123     0% 0: always => always
1124     \or
1125     1% 1: never => never
1126     \or
1127     6% 2: multiple => multiple ng-last
1128     \or
1129     1% 3: multiple g-last => never
1130     \or
1131     5% 4: multiple l-last => multiple lng-last
1132     \or
1133     5% 5: multiple lng-last => multiple lng-last
1134     \or
1135     6% 6: multiple ng-last => multiple ng-last
1136     \fi
1137   }%
1138 }%
1139 \@tempa{vingt}%
1140 \@tempa{cent}%
1141 \@tempa{mil}%
1142 \@tempa{n-illion}%
1143 \@tempa{n-illiard}%

```

Now make \fc@wcase and \@nil non expandable

```

1144     \let\fc@wcase@save\fc@wcase
1145     \def\fc@wcase{\noexpand\fc@wcase}%
1146     \def\@nil{\noexpand\@nil}%

```

In the sequel, \@tempa is used to accumulate the formatted number.

```

1147     \let\@tempa\@empty
1148     \fc@@nbrstrfrench@inner

```

Now restore \fc@wcase

```

1149     \let\fc@wcase\fc@wcase@save

```

Now we add the “ième” ending

```

1150     \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
1151     \expandafter\fc@get@last@letter\expandafter{\@tempc}\@tempd\@tempe
1152     \def\@tempf{e}%
1153     \ifx\@tempe\@tempf
1154         \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd i\protect\'eme\@nil}%
1155     \else
1156         \def\@tempf{q}%
1157         \ifx\@tempe\@tempf
1158             \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd qui\protect\'eme\@nil}%
1159         \else
1160             \def\@tempf{f}%
1161             \ifx\@tempe\@tempf
1162                 \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd vi\protect\'eme\@nil}%
1163             \else
1164                 \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempc i\protect\'eme\@nil}%

```

```

1165         \fi
1166     \fi
1167     \fi
1168 }%

```

Apply `\fc@gc` to the result.

```
1169 \fc@apply@gc
```

Propagate the result — i.e. expansion of `\@tempa` — into macro #2 after closing brace.

```

1170 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
1171 \expandafter\@tempb\expandafter{\@tempa}%
1172 \expandafter
1173 }\@tempa
1174 }%

```

```
1175 \global\let\@@ordinalstringfrench\@@ordinalstringfrench
```

Macro `\fc@frenchoptions@setdefaults` allows to set all options to default for the French.

```

1176 \newcommand*\fc@frenchoptions@setdefaults{%
1177     \csname KV@fcfrench@all plural\endcsname{reformed}%
1178     \fc@gl@def\fc@frenchoptions@submillion@dos{-}%
1179     \fc@gl@let\fc@frenchoptions@supermillion@dos\space
1180     \fc@gl@let\fc@u@in@duo\@empty% Could be ‘u’
1181     % \fc@gl@let\fc@poweroften\fc@pot@longscalefrench
1182     \fc@gl@let\fc@poweroften\fc@pot@recursivefrench
1183     \fc@gl@def\fc@longscale@nilliard@upto{0}% infinity
1184     \fc@gl@def\fc@frenchoptions@mil@plural@mark{le}%
1185 }%
1186 \global\let\fc@frenchoptions@setdefaults\fc@frenchoptions@setdefaults
1187 {%
1188     \let\fc@gl@def\gdef
1189     \def\fc@gl@let{\global\let}%
1190     \fc@frenchoptions@setdefaults
1191 }%

```

Make some indirection to call the current French dialect corresponding macro.

```

1192 \gdef\@ordinalstringMfrench{\csuse{\@ordinalstringMfrench\fmtcount@french}}%
1193 \gdef\@ordinalstringFfrench{\csuse{\@ordinalstringFfrench\fmtcount@french}}%
1194 \gdef\@OrdinalstringMfrench{\csuse{\@OrdinalstringMfrench\fmtcount@french}}%
1195 \gdef\@OrdinalstringFfrench{\csuse{\@OrdinalstringFfrench\fmtcount@french}}%
1196 \gdef\@numberstringMfrench{\csuse{\@numberstringMfrench\fmtcount@french}}%
1197 \gdef\@numberstringFfrench{\csuse{\@numberstringFfrench\fmtcount@french}}%
1198 \gdef\@NumberstringMfrench{\csuse{\@NumberstringMfrench\fmtcount@french}}%
1199 \gdef\@NumberstringFfrench{\csuse{\@NumberstringFfrench\fmtcount@french}}%

```

### 10.1.8 fc-frenchb.def

```

1 \ProvidesFCLanguage{frenchb}[2013/08/17]%
2 \FCloadlang{french}%

```

Set `frenchb` to be equivalent to `french`.

```
3 \global\let\@ordinalMfrenchb=\@ordinalMfrench
```

```

4 \global\let\@ordinalFfrenchb=\@ordinalFfrench
5 \global\let\@ordinalNfrenchb=\@ordinalNfrench
6 \global\let\@numberstringMfrenchb=\@numberstringMfrench
7 \global\let\@numberstringFfrenchb=\@numberstringFfrench
8 \global\let\@numberstringNfrenchb=\@numberstringNfrench
9 \global\let\@NumberstringMfrenchb=\@NumberstringMfrench
10 \global\let\@NumberstringFfrenchb=\@NumberstringFfrench
11 \global\let\@NumberstringNfrenchb=\@NumberstringNfrench
12 \global\let\@ordinalstringMfrenchb=\@ordinalstringMfrench
13 \global\let\@ordinalstringFfrenchb=\@ordinalstringFfrench
14 \global\let\@ordinalstringNfrenchb=\@ordinalstringNfrench
15 \global\let\@OrdinalstringMfrenchb=\@OrdinalstringMfrench
16 \global\let\@OrdinalstringFfrenchb=\@OrdinalstringFfrench
17 \global\let\@OrdinalstringNfrenchb=\@OrdinalstringNfrench

```

### 10.1.9 fc-german.def

German definitions (thank you to K. H. Fricke for supplying this information)

```
1 \ProvidesFCLanguage{german}[2018/06/17]%
```

Define macro that converts a number or count register (first argument) to an ordinal, and stores the result in the second argument, which must be a control sequence. Masculine:

```

2 \newcommand{\@ordinalMgerman}[2]{%
3   \edef#2{\number#1\relax.}%
4 }%
5 \global\let\@ordinalMgerman\@ordinalMgerman

```

Feminine:

```

6 \newcommand{\@ordinalFgerman}[2]{%
7   \edef#2{\number#1\relax.}%
8 }%
9 \global\let\@ordinalFgerman\@ordinalFgerman

```

Neuter:

```

10 \newcommand{\@ordinalNgerman}[2]{%
11   \edef#2{\number#1\relax.}%
12 }%
13 \global\let\@ordinalNgerman\@ordinalNgerman

```

Convert a number to text. The easiest way to do this is to break it up into units, tens and teens. Units (argument must be a number from 0 to 9, 1 on its own (eins) is dealt with separately):

```

14 \newcommand*{\@unitstringgerman}[1]{%
15   \ifcase#1%
16     null%
17     \or ein%
18     \or zwei%
19     \or drei%
20     \or vier%
21     \or fünf%
22     \or sechs%
23     \or sieben%

```

```

24 \or acht%
25 \or neun%
26 \fi
27 }%
28 \global\let\@@unitstringgerman\@@unitstringgerman

```

Tens (argument must go from 1 to 10):

```

29 \newcommand*\@@tenstringgerman[1]{%
30 \ifcase#1%
31 \or zehn%
32 \or zwanzig%
33 \or dreißig%
34 \or vierzig%
35 \or fünfzig%
36 \or sechzig%
37 \or siebzig%
38 \or achtzig%
39 \or neunzig%
40 \or einhundert%
41 \fi
42 }%
43 \global\let\@@tenstringgerman\@@tenstringgerman

```

`\einhundert` is set to `einhundert` by default, user can redefine this command to just `hundert` if required, similarly for `\eintausend`.

```

44 \providecommand*\einhundert{\einhundert}%
45 \providecommand*\eintausend{\eintausend}%
46 \global\let\einhundert\einhundert
47 \global\let\eintausend\eintausend

```

Teens:

```

48 \newcommand*\@@teenstringgerman[1]{%
49 \ifcase#1%
50 zehn%
51 \or elf%
52 \or zwölf%
53 \or dreizehn%
54 \or vierzehn%
55 \or fünfzehn%
56 \or sechzehn%
57 \or siebzehn%
58 \or achtzehn%
59 \or neunzehn%
60 \fi
61 }%
62 \global\let\@@teenstringgerman\@@teenstringgerman

```

The results are stored in the second argument, but doesn't display anything.

```

63 \newcommand*\@numberstringMgerman[2]{%
64 \let\@unitstring=\@@unitstringgerman
65 \let\@teenstring=\@@teenstringgerman

```

```

66 \let\@tenstring=\@tenstringgerman
67 \@numberstringgerman{#1}{#2}%
68 }%
69 \global\let\@numberstringMgerman\@numberstringMgerman
Feminine and neuter forms:
70 \global\let\@numberstringFgerman=\@numberstringMgerman
71 \global\let\@numberstringNgerman=\@numberstringMgerman
As above, but initial letters in upper case:
72 \newcommand*{\@NumberstringMgerman}[2]{%
73 \@numberstringMgerman{#1}{\@num@str}%
74 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
75 }%
76 \global\let\@NumberstringMgerman\@NumberstringMgerman
Feminine and neuter form:
77 \global\let\@NumberstringFgerman=\@NumberstringMgerman
78 \global\let\@NumberstringNgerman=\@NumberstringMgerman
As above, but for ordinals.
79 \newcommand*{\@ordinalstringMgerman}[2]{%
80 \let\@unitthstring=\@unitthstringMgerman
81 \let\@teenthstring=\@teenthstringMgerman
82 \let\@tenthstring=\@tenthstringMgerman
83 \let\@unitstring=\@unitstringgerman
84 \let\@teenstring=\@teenstringgerman
85 \let\@tenstring=\@tenstringgerman
86 \def\@thousandth{tausendster}%
87 \def\@hundredth{hundertster}%
88 \@ordinalstringgerman{#1}{#2}%
89 }%
90 \global\let\@ordinalstringMgerman\@ordinalstringMgerman
Feminine form:
91 \newcommand*{\@ordinalstringFgerman}[2]{%
92 \let\@unitthstring=\@unitthstringFgerman
93 \let\@teenthstring=\@teenthstringFgerman
94 \let\@tenthstring=\@tenthstringFgerman
95 \let\@unitstring=\@unitstringgerman
96 \let\@teenstring=\@teenstringgerman
97 \let\@tenstring=\@tenstringgerman
98 \def\@thousandth{tausendste}%
99 \def\@hundredth{hundertste}%
100 \@ordinalstringgerman{#1}{#2}%
101 }%
102 \global\let\@ordinalstringFgerman\@ordinalstringFgerman
Neuter form:
103 \newcommand*{\@ordinalstringNgerman}[2]{%
104 \let\@unitthstring=\@unitthstringNgerman
105 \let\@teenthstring=\@teenthstringNgerman

```



```

106 \let\@tenthstring=\@tenthstringNgerman
107 \let\@unitstring=\@unitstringgerman
108 \let\@teenstring=\@teenstringgerman
109 \let\@tenstring=\@tenstringgerman
110 \def\@thousandth{tausendstes}%
111 \def\@hundredth{hunderstes}%
112 \@ordinalstringgerman{#1}{#2}%
113 }%
114 \global\let\@ordinalstringNgerman\@ordinalstringNgerman

```

As above, but with initial letters in upper case.

```

115 \newcommand*\@OrdinalstringMgerman}[2]{%
116 \@ordinalstringMgerman{#1}{\@num@str}%
117 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
118 }%
119 \global\let\@OrdinalstringMgerman\@OrdinalstringMgerman

```

Feminine form:

```

120 \newcommand*\@OrdinalstringFgerman}[2]{%
121 \@ordinalstringFgerman{#1}{\@num@str}%
122 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
123 }%
124 \global\let\@OrdinalstringFgerman\@OrdinalstringFgerman

```

Neuter form:

```

125 \newcommand*\@OrdinalstringNgerman}[2]{%
126 \@ordinalstringNgerman{#1}{\@num@str}%
127 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
128 }%
129 \global\let\@OrdinalstringNgerman\@OrdinalstringNgerman

```

Code for converting numbers into textual ordinals. As before, it is easier to split it into units, tens and teens. Units:

```

130 \newcommand*\@unitthstringMgerman[1]{%
131 \ifcase#1%
132 nullter%
133 \or erster%
134 \or zweiter%
135 \or dritter%
136 \or vierter%
137 \or fünfter%
138 \or sechster%
139 \or siebter%
140 \or achter%
141 \or neunter%
142 \fi
143 }%
144 \global\let\@unitthstringMgerman\@unitthstringMgerman

```

Tens:

```

145 \newcommand*\@tenthstringMgerman[1]{%

```

```

146 \ifcase#1%
147 \or zehnter%
148 \or zwanzigster%
149 \or dreißigster%
150 \or vierzigster%
151 \or fünfzigster%
152 \or sechzigster%
153 \or siebzigster%
154 \or achtzigster%
155 \or neunzigster%
156 \fi
157 }%
158 \global\let\@@tenthstringMgerman\@@tenthstringMgerman
    Teens:
159 \newcommand*\@@teenthstringMgerman[1]{%
160 \ifcase#1%
161 zehnter%
162 \or elfter%
163 \or zwölfter%
164 \or dreizehnter%
165 \or vierzehnter%
166 \or fünfzehnter%
167 \or sechzehnter%
168 \or siebzehnter%
169 \or achtzehnter%
170 \or neunzehnter%
171 \fi
172 }%
173 \global\let\@@teenthstringMgerman\@@teenthstringMgerman
    Units (feminine):
174 \newcommand*\@@unitthstringFgerman[1]{%
175 \ifcase#1%
176 nullte%
177 \or erste%
178 \or zweite%
179 \or dritte%
180 \or vierte%
181 \or fünfte%
182 \or sechste%
183 \or siebte%
184 \or achte%
185 \or neunte%
186 \fi
187 }%
188 \global\let\@@unitthstringFgerman\@@unitthstringFgerman
    Tens (feminine):
189 \newcommand*\@@tenthstringFgerman[1]{%
190 \ifcase#1%

```

```

191 \or zehnte%
192 \or zwanzigste%
193 \or dreißigste%
194 \or vierzigste%
195 \or fünfzigste%
196 \or sechzigste%
197 \or siebzigste%
198 \or achtzigste%
199 \or neunzigste%
200 \fi
201 }%
202 \global\let\@@tenthstringFgerman\@@tenthstringFgerman
    Teens (feminine)
203 \newcommand*\@@teenthstringFgerman[1]{%
204 \ifcase#1%
205 zehnte%
206 \or elfte%
207 \or zwölfte%
208 \or dreizehnte%
209 \or vierzehnte%
210 \or fünfzehnte%
211 \or sechzehnte%
212 \or siebzehnte%
213 \or achtzehnte%
214 \or neunzehnte%
215 \fi
216 }%
217 \global\let\@@teenthstringFgerman\@@teenthstringFgerman
    Units (neuter):
218 \newcommand*\@@unitthstringNgerman[1]{%
219 \ifcase#1%
220 nulltes%
221 \or erstes%
222 \or zweites%
223 \or drittes%
224 \or viertes%
225 \or fünftes%
226 \or sechstes%
227 \or siebtes%
228 \or achttes%
229 \or neuntes%
230 \fi
231 }%
232 \global\let\@@unitthstringNgerman\@@unitthstringNgerman
    Tens (neuter):
233 \newcommand*\@@tenthstringNgerman[1]{%
234 \ifcase#1%
235 \or zehntes%

```

```

236 \or zwanzigstes%
237 \or dreißigstes%
238 \or vierzigstes%
239 \or fünfzigstes%
240 \or sechzigstes%
241 \or siebzigstes%
242 \or achtzigstes%
243 \or neunzigstes%
244 \fi
245 }%
246 \global\let\@@tenthstringNgerman\@@tenthstringNgerman

```

Teens (neuter)

```

247 \newcommand*\@@teenthstringNgerman[1]{%
248 \ifcase#1%
249 zehntes%
250 \or elftes%
251 \or zwölftes%
252 \or dreizehntes%
253 \or vierzehntes%
254 \or fünfzehntes%
255 \or sechzehntes%
256 \or siebzehntes%
257 \or achtzehntes%
258 \or neunzehntes%
259 \fi
260 }%
261 \global\let\@@teenthstringNgerman\@@teenthstringNgerman

```

This appends the results to \#2 for number \#2 (in range 0 to 100.) null and eins are dealt with separately in \@@numberstringgerman.

```

262 \newcommand*\@@numberunderhundredgerman[2]{%
263 \ifnum#1<10\relax
264 \ifnum#1>0\relax
265 \eappto#2{\@unitstring{#1}}%
266 \fi
267 \else
268 \@tmpstrctr=#1\relax
269 \@FCmodulo{\@tmpstrctr}{10}%
270 \ifnum#1<20\relax
271 \eappto#2{\@teenstring{\@tmpstrctr}}%
272 \else
273 \ifnum\@tmpstrctr=0\relax
274 \else
275 \eappto#2{\@unitstring{\@tmpstrctr}und}%
276 \fi
277 \@tmpstrctr=#1\relax
278 \divide\@tmpstrctr by 10\relax
279 \eappto#2{\@tenstring{\@tmpstrctr}}%
280 \fi

```

```

281 \fi
282 }%
283 \global\let\@@numberunderhundredgerman\@@numberunderhundredgerman
    This stores the results in the second argument (which must be a control sequence), but it
    doesn't display anything.
284 \newcommand*\@@numberstringgerman[2]{%
285 \ifnum#1>99999\relax
286   \PackageError{fmtcount}{Out of range}%
287   {This macro only works for values less than 100000}%
288 \else
289   \ifnum#1<0\relax
290     \PackageError{fmtcount}{Negative numbers not permitted}%
291     {This macro does not work for negative numbers, however
292     you can try typing "minus" first, and then pass the modulus of
293     this number}%
294   \fi
295 \fi
296 \def#2{}%
297 \@strctr=#1\relax \divide\@strctr by 1000\relax
298 \ifnum\@strctr>1\relax
    #1 is  $\geq 2000$ , \@strctr now contains the number of thousands
299 \@@numberunderhundredgerman{\@strctr}{#2}%
300 \appto#2{tausend}%
301 \else
    #1 lies in range [1000,1999]
302 \ifnum\@strctr=1\relax
303   \eappto#2{\eintausend}%
304 \fi
305 \fi
306 \@strctr=#1\relax
307 \@FCmodulo{\@strctr}{1000}%
308 \divide\@strctr by 100\relax
309 \ifnum\@strctr>1\relax
    now dealing with number in range [200,999]
310 \eappto#2{\@unitstring{\@strctr}hundert}%
311 \else
312   \ifnum\@strctr=1\relax
    dealing with number in range [100,199]
313     \ifnum#1>1000\relax
    if original number > 1000, use einhundert
314       \appto#2{einhundert}%
315     \else
    otherwise use \einhundert
316       \eappto#2{\einhundert}%
317     \fi

```

```

318 \fi
319 \fi
320 \@strctr=#1\relax
321 \@FCmodulo{\@strctr}{100}%
322 \ifnum#1=0\relax
323 \def#2{null}%
324 \else
325 \ifnum\@strctr=1\relax
326 \appto#2{eins}%
327 \else
328 \@numberunderhundredgerman{\@strctr}{#2}%
329 \fi
330 \fi
331 }%
332 \global\let\@numberstringgerman\@numberstringgerman

```

As above, but for ordinals

```

333 \newcommand*\@numberunderhundredthgerman[2]{%
334 \ifnum#1<10\relax
335 \eappto#2{\@unitthstring{#1}}%
336 \else
337 \@tmpstrctr=#1\relax
338 \@FCmodulo{\@tmpstrctr}{10}%
339 \ifnum#1<20\relax
340 \eappto#2{\@teenthstring{\@tmpstrctr}}%
341 \else
342 \ifnum\@tmpstrctr=0\relax
343 \else
344 \eappto#2{\@unitstring{\@tmpstrctr}und}%
345 \fi
346 \@tmpstrctr=#1\relax
347 \divide\@tmpstrctr by 10\relax
348 \eappto#2{\@tenthstring{\@tmpstrctr}}%
349 \fi
350 \fi
351 }%
352 \global\let\@numberunderhundredthgerman\@numberunderhundredthgerman

353 \newcommand*\@ordinalstringgerman[2]{%
354 \@orgargctr=#1\relax
355 \ifnum\@orgargctr>99999\relax
356 \PackageError{fmtcount}{Out of range}%
357 {This macro only works for values less than 100000}%
358 \else
359 \ifnum\@orgargctr<0\relax
360 \PackageError{fmtcount}{Negative numbers not permitted}%
361 {This macro does not work for negative numbers, however
362 you can try typing "minus" first, and then pass the modulus of
363 this number}%
364 \fi

```

```

365 \fi
366 \def#2{}%
367 \@strctr=\@orgargctr\divide\@strctr by 1000\relax
368 \ifnum\@strctr>1\relax

#1 is  $\geq 2000$ , \@strctr now contains the number of thousands
369 \@numberunderhundredgerman{\@strctr}{#2}%

is that it, or is there more?
370 \@tmpstrctr=\@orgargctr\@FCmodulo{\@tmpstrctr}{1000}%
371 \ifnum\@tmpstrctr=0\relax
372 \eappto#2{\@thousandth}%
373 \else
374 \appto#2{tausend}%
375 \fi
376 \else

#1 lies in range [1000,1999]
377 \ifnum\@strctr=1\relax
378 \ifnum\@orgargctr=1000\relax
379 \eappto#2{\@thousandth}%
380 \else
381 \eappto#2{\eintausend}%
382 \fi
383 \fi
384 \fi
385 \@strctr=\@orgargctr
386 \@FCmodulo{\@strctr}{1000}%
387 \divide\@strctr by 100\relax
388 \ifnum\@strctr>1\relax

now dealing with number in range [200,999] is that it, or is there more?
389 \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{100}%
390 \ifnum\@tmpstrctr=0\relax
391 \ifnum\@strctr=1\relax
392 \eappto#2{\@hundredth}%
393 \else
394 \eappto#2{\@unitstring{\@strctr}\@hundredth}%
395 \fi
396 \else
397 \eappto#2{\@unitstring{\@strctr}hundert}%
398 \fi
399 \else
400 \ifnum\@strctr=1\relax

dealing with number in range [100,199] is that it, or is there more?
401 \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{100}%
402 \ifnum\@tmpstrctr=0\relax
403 \eappto#2{\@hundredth}%
404 \else
405 \ifnum\@orgargctr>1000\relax

```

```

406     \appto#2{einhundert}%
407   \else
408     \eappto#2{\einhundert}%
409   \fi
410 \fi
411 \fi
412 \fi
413 \@strctr=\@orgargctr
414 \@FCmodulo{\@strctr}{100}%
415 \ifthenelse{\@strctr=0 \and \@orgargctr>0 }{}{%
416 \@numberunderhundredthgerman{\@strctr}{#2}%
417 }%
418 }%
419 \global\let\@ordinalstringgerman\@ordinalstringgerman
    Load fc-germanb.def if not already loaded
420 \FCloadlang{germanb}%

```

### 10.1.10 fc-germanb.def

```
1 \ProvidesFCLanguage{germanb}[2013/08/17]%
```

Load fc-german.def if not already loaded

```
2 \FCloadlang{german}%
```

Set germanb to be equivalent to german.

```

3 \global\let\@ordinalMgermanb=\@ordinalMgerman
4 \global\let\@ordinalFgermanb=\@ordinalFgerman
5 \global\let\@ordinalNgermanb=\@ordinalNgerman
6 \global\let\@numberstringMgermanb=\@numberstringMgerman
7 \global\let\@numberstringFgermanb=\@numberstringFgerman
8 \global\let\@numberstringNgermanb=\@numberstringNgerman
9 \global\let\@NumberstringMgermanb=\@NumberstringMgerman
10 \global\let\@NumberstringFgermanb=\@NumberstringFgerman
11 \global\let\@NumberstringNgermanb=\@NumberstringNgerman
12 \global\let\@ordinalstringMgermanb=\@ordinalstringMgerman
13 \global\let\@ordinalstringFgermanb=\@ordinalstringFgerman
14 \global\let\@ordinalstringNgermanb=\@ordinalstringNgerman
15 \global\let\@OrdinalstringMgermanb=\@OrdinalstringMgerman
16 \global\let\@OrdinalstringFgermanb=\@OrdinalstringFgerman
17 \global\let\@OrdinalstringNgermanb=\@OrdinalstringNgerman

```

### 10.1.11 fc-italian.def

Italian support is now handled by interfacing to Enrico Gregorio's itnumpar package.

```

1 \ProvidesFCLanguage{italian}[2013/08/17]
2
3 \RequirePackage{itnumpar}
4
5 \newcommand{\@numberstringMitalian}[2]{%
6   \begingroup

```



```

7   \def\np@oa{o}%
8   \count@=#1
9   \edef\@tempa{\def\noexpand#2{\@numeroinparole{\count@}}}%
10  \expandafter
11  \endgroup\@tempa
12 }
13 \global\let\@numberstringMitalian\@numberstringMitalian
14
15 \newcommand{\@numberstringFitalian}[2]{%
16  \begingroup
17  \def\np@oa{a}%
18  \count@=#1
19  \edef\@tempa{\def\noexpand#2{\@numeroinparole{\count@}}}%
20  \expandafter
21  \endgroup\@tempa
22 }
23
24 \global\let\@numberstringFitalian\@numberstringFitalian
25
26 \newcommand{\@NumberstringMitalian}[2]{%
27  \begingroup
28  \def\np@oa{o}%
29  \count@=#1
30  \edef\@tempa{\def\noexpand#2{\@Numeroinparole{\count@}}}%
31  \expandafter
32  \endgroup\@tempa
33 }
34 \global\let\@NumberstringMitalian\@NumberstringMitalian
35
36 \newcommand{\@NumberstringFitalian}[2]{%
37  \begingroup
38  \def\np@oa{a}%
39  \count@=#1
40  \edef\@tempa{\def\noexpand#2{\@Numeroinparole{\count@}}}%
41  \expandafter
42  \endgroup\@tempa
43 }
44 \global\let\@NumberstringFitalian\@NumberstringFitalian
45
46 \newcommand{\@ordinalstringMitalian}[2]{%
47  \begingroup
48  \count@=#1
49  \edef\@tempa{\def\noexpand#2{\@ordinalem{\count@}}}%
50  \expandafter
51  \endgroup\@tempa
52 }
53 \global\let\@ordinalstringMitalian\@ordinalstringMitalian
54
55 \newcommand{\@ordinalstringFitalian}[2]{%

```

```

56 \begingroup
57   \count@=#1
58   \edef\@tempa{\def\noexpand#2{\@ordinalef{\count@}}}%
59   \expandafter
60 \endgroup\@tempa
61 }
62 \global\let\@ordinalstringFitalian\@ordinalstringFitalian
63
64 \newcommand{\@OrdinalstringMitalian}[2]{%
65   \begingroup
66     \count@=#1
67     \edef\@tempa{\def\noexpand#2{\@Ordinalem{\count@}}}%
68     \expandafter
69 \endgroup\@tempa
70 }
71 \global\let\@OrdinalstringMitalian\@OrdinalstringMitalian
72
73 \newcommand{\@ORDINALSTRINGMitalian}[2]{%
74   \begingroup
75     \count@=#1
76     \edef\@tempa{\def\noexpand#2{\MakeUppercase{\@ordinalem{\count@}}}}%
77     \expandafter
78 \endgroup\@tempa
79 }
80 \global\let\@ORDINALSTRINGMitalian\@ORDINALSTRINGMitalian
81
82 \newcommand{\@OrdinalstringFitalian}[2]{%
83   \begingroup
84     \count@=#1
85     \edef\@tempa{\def\noexpand#2{\@ordinalef{\count@}}}%
86     \expandafter
87 \endgroup\@tempa
88 }
89 \global\let\@OrdinalstringFitalian\@OrdinalstringFitalian
90
91 \newcommand{\@ORDINALSTRINGFitalian}[2]{%
92   \begingroup
93     \count@=#1
94     \edef\@tempa{\def\noexpand#2{\MakeUppercase{\@ordinalef{\count@}}}}%
95     \expandafter
96 \endgroup\@tempa
97 }
98 \global\let\@ORDINALSTRINGFitalian\@ORDINALSTRINGFitalian
99
100 \newcommand{\@ordinalMitalian}[2]{%
101   \edef#2{#1\relax\noexpand\fmtord{o}}
102
103 \global\let\@ordinalMitalian\@ordinalMitalian
104

```

```

105 \newcommand{\@OrdinalFitalian}[2]{%
106 \edef#2{#1\relax\noexpand\fmtord{a}}%
107 \global\let\@OrdinalFitalian\@OrdinalFitalian

```

### 10.1.12 fc-ngerman.def

```

1 \ProvidesFCLanguage{ngerman}[2012/06/18]%
2 \FCloadlang{german}%
3 \FCloadlang{ngermanb}%

```

Set `ngerman` to be equivalent to `german`. Is it okay to do this? (I don't know the difference between the two.)

```

4 \global\let\@OrdinalMngerman=\@OrdinalMgerman
5 \global\let\@OrdinalFngerman=\@OrdinalFgerman
6 \global\let\@OrdinalNngerman=\@OrdinalNgerman
7 \global\let\@NumberstringMngerman=\@NumberstringMgerman
8 \global\let\@NumberstringFngerman=\@NumberstringFgerman
9 \global\let\@NumberstringNngerman=\@NumberstringNgerman
10 \global\let\@NumberstringMngerman=\@NumberstringMgerman
11 \global\let\@NumberstringFngerman=\@NumberstringFgerman
12 \global\let\@NumberstringNngerman=\@NumberstringNgerman
13 \global\let\@OrdinalstringMngerman=\@OrdinalstringMgerman
14 \global\let\@OrdinalstringFngerman=\@OrdinalstringFgerman
15 \global\let\@OrdinalstringNngerman=\@OrdinalstringNgerman
16 \global\let\@OrdinalstringMngerman=\@OrdinalstringMgerman
17 \global\let\@OrdinalstringFngerman=\@OrdinalstringFgerman
18 \global\let\@OrdinalstringNngerman=\@OrdinalstringNgerman

```

### 10.1.13 fc-ngermanb.def

```

1 \ProvidesFCLanguage{ngermanb}[2013/08/17]%
2 \FCloadlang{german}%

```

Set `ngermanb` to be equivalent to `german`. Is it okay to do this? (I don't know the difference between the two.)

```

3 \global\let\@OrdinalMngermanb=\@OrdinalMgerman
4 \global\let\@OrdinalFngermanb=\@OrdinalFgerman
5 \global\let\@OrdinalNngermanb=\@OrdinalNgerman
6 \global\let\@NumberstringMngermanb=\@NumberstringMgerman
7 \global\let\@NumberstringFngermanb=\@NumberstringFgerman
8 \global\let\@NumberstringNngermanb=\@NumberstringNgerman
9 \global\let\@NumberstringMngermanb=\@NumberstringMgerman
10 \global\let\@NumberstringFngermanb=\@NumberstringFgerman
11 \global\let\@NumberstringNngermanb=\@NumberstringNgerman
12 \global\let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
13 \global\let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
14 \global\let\@OrdinalstringNngermanb=\@OrdinalstringNgerman
15 \global\let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
16 \global\let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
17 \global\let\@OrdinalstringNngermanb=\@OrdinalstringNgerman

```

Load `fc-ngerman.def` if not already loaded

```
18 \FCloadlang{ngerman}%
```

#### 10.1.14 fc-portuges.def

Portuguese definitions

```
1 \ProvidesFCLanguage{portuges}[2017/12/26]%
```

Define macro that converts a number or count register (first argument) to an ordinal, and stores the result in the second argument, which should be a control sequence. Masculine:

```
2 \newcommand*\@OrdinalMportuges[2]{%
3   \ifnum#1=0\relax
4     \edef#2{\number#1}%
5   \else
6     \edef#2{\number#1\relax\noexpand\fmtord{o}}%
7   \fi
8 }%
9 \global\let\@OrdinalMportuges\@OrdinalMportuges
```

Feminine:

```
10 \newcommand*\@OrdinalFportuges[2]{%
11   \ifnum#1=0\relax
12     \edef#2{\number#1}%
13   \else
14     \edef#2{\number#1\relax\noexpand\fmtord{a}}%
15   \fi
16 }%
17 \global\let\@OrdinalFportuges\@OrdinalFportuges
```

Make neuter same as masculine:

```
18 \global\let\@OrdinalNportuges\@OrdinalMportuges
```

Convert a number to a textual representation. To make it easier, split it up into units, tens, teens and hundreds. Units (argument must be a number from 0 to 9):

```
19 \newcommand*\@@unitstringportuges[1]{%
20   \ifcase#1\relax
21     zero%
22     \or um%
23     \or dois%
24     \or tr\^es%
25     \or quatro%
26     \or cinco%
27     \or seis%
28     \or sete%
29     \or oito%
30     \or nove%
31   \fi
32 }%
33 \global\let\@@unitstringportuges\@@unitstringportuges
34 % \end{macrocode}
35 % As above, but for feminine:
36 % \begin{macrocode}
```

```

37 \newcommand*{\@@unitstringFportuges[1]}{
38   \ifcase#1\relax
39     zero%
40     \or uma%
41     \or duas%
42     \or tr\^es%
43     \or quatro%
44     \or cinco%
45     \or seis%
46     \or sete%
47     \or oito%
48     \or nove%
49   \fi
50 }%
51 \global\let\@@unitstringFportuges\@@unitstringFportuges
Tens (argument must be a number from 0 to 10):
52 \newcommand*{\@@tenstringportuges[1]}{
53   \ifcase#1\relax
54     \or dez%
55     \or vinte%
56     \or trinta%
57     \or quarenta%
58     \or cinquenta%
59     \or sessenta%
60     \or setenta%
61     \or oitenta%
62     \or noventa%
63     \or cem%
64   \fi
65 }%
66 \global\let\@@tenstringportuges\@@tenstringportuges
Teens (argument must be a number from 0 to 9):
67 \newcommand*{\@@teenstringportuges[1]}{
68   \ifcase#1\relax
69     dez%
70     \or onze%
71     \or doze%
72     \or treze%
73     \or catorze%
74     \or quinze%
75     \or dezasseis%
76     \or dezassete%
77     \or dezoito%
78     \or dezanove%
79   \fi
80 }%
81 \global\let\@@teenstringportuges\@@teenstringportuges
Hundreds:

```

```

82 \newcommand*\@@hundredstringportuges [1] {%
83   \ifcase#1\relax
84     \or cento%
85     \or duzentos%
86     \or trezentos%
87     \or quatrocentos%
88     \or quinhentos%
89     \or seiscentos%
90     \or setecentos%
91     \or oitocentos%
92     \or novecentos%
93   \fi
94 }%
95 \global\let\@@hundredstringportuges\@@hundredstringportuges

Hundreds (feminine):
96 \newcommand*\@@hundredstringFportuges [1] {%
97   \ifcase#1\relax
98     \or cento%
99     \or duzentas%
100    \or trezentas%
101    \or quatrocentas%
102    \or quinhentas%
103    \or seiscentas%
104    \or setecentas%
105    \or oitocentas%
106    \or novecentas%
107  \fi
108 }%
109 \global\let\@@hundredstringFportuges\@@hundredstringFportuges

Units (initial letter in upper case):
110 \newcommand*\@@Unitstringportuges [1] {%
111   \ifcase#1\relax
112     Zero%
113     \or Um%
114     \or Dois%
115     \or Tr\`es%
116     \or Quatro%
117     \or Cinco%
118     \or Seis%
119     \or Sete%
120     \or Oito%
121     \or Nove%
122   \fi
123 }%
124 \global\let\@@Unitstringportuges\@@Unitstringportuges

As above, but feminine:
125 \newcommand*\@@UnitstringFportuges [1] {%
126   \ifcase#1\relax

```

```

127     Zera%
128     \or Uma%
129     \or Duas%
130     \or Tr\~es%
131     \or Quatro%
132     \or Cinco%
133     \or Seis%
134     \or Sete%
135     \or Oito%
136     \or Nove%
137 \fi
138 }%
139 \global\let\@@UnitstringFportuges\@@UnitstringFportuges

    Tens (with initial letter in upper case):
140 \newcommand*\@@Tenstringportuges[1]{%
141 \ifcase#1\relax
142 \or Dez%
143 \or Vinte%
144 \or Trinta%
145 \or Quarenta%
146 \or Cinquenta%
147 \or Sessenta%
148 \or Setenta%
149 \or Oitenta%
150 \or Noventa%
151 \or Cem%
152 \fi
153 }%
154 \global\let\@@Tenstringportuges\@@Tenstringportuges

    Teens (with initial letter in upper case):
155 \newcommand*\@@Teenstringportuges[1]{%
156 \ifcase#1\relax
157     Dez%
158     \or Onze%
159     \or Doze%
160     \or Treze%
161     \or Catorze%
162     \or Quinze%
163     \or Dezasseis%
164     \or Dezassete%
165     \or Dezoito%
166     \or Dezanove%
167 \fi
168 }%
169 \global\let\@@Teenstringportuges\@@Teenstringportuges

    Hundreds (with initial letter in upper case):
170 \newcommand*\@@Hundredstringportuges[1]{%
171 \ifcase#1\relax

```

```

172 \or Cento%
173 \or Duzentos%
174 \or Trezentos%
175 \or Quatrocentos%
176 \or Quinhentos%
177 \or Seiscentos%
178 \or Setecentos%
179 \or Oitocentos%
180 \or Novecentos%
181 \fi
182 }%
183 \global\let\@@Hundredstringportuges\@@Hundredstringportuges

```

As above, but feminine:

```

184 \newcommand*\@@HundredstringFportuges[1]{%
185 \ifcase#1\relax
186 \or Cento%
187 \or Duzentas%
188 \or Trezentas%
189 \or Quatrocentas%
190 \or Quinhentas%
191 \or Seiscentas%
192 \or Setecentas%
193 \or Oitocentas%
194 \or Novecentas%
195 \fi
196 }%
197 \global\let\@@HundredstringFportuges\@@HundredstringFportuges

```

This has changed in version 1.08, so that it now stores the result in the second argument, but doesn't display anything. Since it only affects internal macros, it shouldn't affect documents created with older versions. (These internal macros are not meant for use in documents.)

```

198 \newcommand*\@@numberstringMportuges}[2]{%
199 \let\@unitstring=\@@unitstringportuges
200 \let\@teenstring=\@@teenstringportuges
201 \let\@tenstring=\@@tenstringportuges
202 \let\@hundredstring=\@@hundredstringportuges
203 \def\@hundred{cem}\def\@thousand{mil}%
204 \def\@andname{e}%
205 \@@numberstringportuges{#1}{#2}%
206 }%
207 \global\let\@numberstringMportuges\@numberstringMportuges

```

As above, but feminine form:

```

208 \newcommand*\@@numberstringFportuges}[2]{%
209 \let\@unitstring=\@@unitstringFportuges
210 \let\@teenstring=\@@teenstringportuges
211 \let\@tenstring=\@@tenstringportuges
212 \let\@hundredstring=\@@hundredstringFportuges
213 \def\@hundred{cem}\def\@thousand{mil}%

```



```

214 \def\@andname{e}%
215 \@numberstringportuges{#1}{#2}%
216 }%
217 \global\let\@numberstringFportuges\@numberstringFportuges
    Make neuter same as masculine:
218 \global\let\@numberstringNportuges\@numberstringMportuges
    As above, but initial letters in upper case:
219 \newcommand*\@NumberstringMportuges}[2]{%
220 \let\@unitstring=\@Unitstringportuges
221 \let\@teenstring=\@Teenstringportuges
222 \let\@tenstring=\@Tenstringportuges
223 \let\@hundredstring=\@Hundredstringportuges
224 \def\@hundred{Cem}\def\@thousand{Mil}%
225 \def\@andname{e}%
226 \@numberstringportuges{#1}{#2}%
227 }%
228 \global\let\@NumberstringMportuges\@NumberstringMportuges
    As above, but feminine form:
229 \newcommand*\@NumberstringFportuges}[2]{%
230 \let\@unitstring=\@UnitstringFportuges
231 \let\@teenstring=\@Teenstringportuges
232 \let\@tenstring=\@Tenstringportuges
233 \let\@hundredstring=\@HundredstringFportuges
234 \def\@hundred{Cem}\def\@thousand{Mil}%
235 \def\@andname{e}%
236 \@numberstringportuges{#1}{#2}%
237 }%
238 \global\let\@NumberstringFportuges\@NumberstringFportuges
    Make neuter same as masculine:
239 \global\let\@NumberstringNportuges\@NumberstringMportuges
    As above, but for ordinals.
240 \newcommand*\@ordinalstringMportuges}[2]{%
241 \let\@unitthstring=\@Unitthstringportuges
242 \let\@unitstring=\@Unitstringportuges
243 \let\@teenthstring=\@Teenthstringportuges
244 \let\@tenthstring=\@Tenthstringportuges
245 \let\@hundredthstring=\@Hundredthstringportuges
246 \def\@thousandth{mil'esimo}%
247 \@ordinalstringportuges{#1}{#2}%
248 }%
249 \global\let\@ordinalstringMportuges\@ordinalstringMportuges
    Feminine form:
250 \newcommand*\@ordinalstringFportuges}[2]{%
251 \let\@unitthstring=\@UnitthstringFportuges
252 \let\@unitstring=\@UnitstringFportuges
253 \let\@teenthstring=\@Teenthstringportuges

```

```

254 \let\@tenthstring=\@tenthstringFportuges
255 \let\@hundredthstring=\@hundredthstringFportuges
256 \def\@thousandth{mil\`esima}%
257 \@ordinalstringportuges{#1}{#2}%
258 }%
259 \global\let\@ordinalstringFportuges\@ordinalstringFportuges

  Make neuter same as masculine:
260 \global\let\@ordinalstringNportuges\@ordinalstringMportuges

  As above, but initial letters in upper case (masculine):
261 \newcommand*\@OrdinalstringMportuges}[2]{%
262 \let\@unitthstring=\@Unitthstringportuges
263 \let\@unitstring=\@Unitstringportuges
264 \let\@teenthstring=\@Teenthstringportuges
265 \let\@tenthstring=\@Tenthstringportuges
266 \let\@hundredthstring=\@Hundredthstringportuges
267 \def\@thousandth{Mil\`esimo}%
268 \@ordinalstringportuges{#1}{#2}%
269 }%
270 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges

  Feminine form:
271 \newcommand*\@OrdinalstringFportuges}[2]{%
272 \let\@unitthstring=\@UnitthstringFportuges
273 \let\@unitstring=\@UnitstringFportuges
274 \let\@teenthstring=\@Teenthstringportuges
275 \let\@tenthstring=\@TenthstringFportuges
276 \let\@hundredthstring=\@HundredthstringFportuges
277 \def\@thousandth{Mil\`esima}%
278 \@ordinalstringportuges{#1}{#2}%
279 }%
280 \global\let\@OrdinalstringFportuges\@OrdinalstringFportuges

  Make neuter same as masculine:
281 \global\let\@OrdinalstringNportuges\@OrdinalstringMportuges

  In order to do the ordinals, split into units, teens, tens and hundreds. Units:
282 \newcommand*\@unitthstringportuges[1]{%
283 \ifcase#1\relax
284   zero%
285   \or primeiro%
286   \or segundo%
287   \or terceiro%
288   \or quarto%
289   \or quinto%
290   \or sexto%
291   \or s\`etimo%
292   \or oitavo%
293   \or nono%
294 \fi

```

```

295 }%
296 \global\let\@@unitthstringportuges\@@unitthstringportuges
    Tens:
297 \newcommand*\@@tenthstringportuges[1]{%
298   \ifcase#1\relax
299     \or d\’esimo%
300     \or vig\’esimo%
301     \or trig\’esimo%
302     \or quadrag\’esimo%
303     \or quinquag\’esimo%
304     \or sexag\’esimo%
305     \or setuag\’esimo%
306     \or octog\’esimo%
307     \or nonag\’esimo%
308   \fi
309 }%
310 \global\let\@@tenthstringportuges\@@tenthstringportuges
    Teens:
311 \newcommand*\@@teenthstringportuges[1]{%
312   \@tenthstring{1}%
313   \ifnum#1>0\relax
314     -\@unitthstring{#1}%
315   \fi
316 }%
317 \global\let\@@teenthstringportuges\@@teenthstringportuges
    Hundreds:
318 \newcommand*\@@hundredthstringportuges[1]{%
319   \ifcase#1\relax
320     \or cent\’esimo%
321     \or ducent\’esimo%
322     \or trecent\’esimo%
323     \or quadringent\’esimo%
324     \or quingent\’esimo%
325     \or seiscent\’esimo%
326     \or setingent\’esimo%
327     \or octingent\’esimo%
328     \or nongent\’esimo%
329   \fi
330 }%
331 \global\let\@@hundredthstringportuges\@@hundredthstringportuges
    Units (feminine):
332 \newcommand*\@@unitthstringFportuges[1]{%
333   \ifcase#1\relax
334     zero%
335     \or primeira%
336     \or segunda%
337     \or terceira%

```

```

338 \or quarta%
339 \or quinta%
340 \or sexta%
341 \or s\'etima%
342 \or oitava%
343 \or nona%
344 \fi
345 }%
346 \global\let\@@unitthstringFportuges\@@unitthstringFportuges
    Tens (feminine):
347 \newcommand*\@@tenthstringFportuges[1]{%
348 \ifcase#1\relax
349 \or d\'esima%
350 \or vig\'esima%
351 \or trig\'esima%
352 \or quadrag\'esima%
353 \or quinquag\'esima%
354 \or sexag\'esima%
355 \or setuag\'esima%
356 \or octog\'esima%
357 \or nonag\'esima%
358 \fi
359 }%
360 \global\let\@@tenthstringFportuges\@@tenthstringFportuges
    Hundreds (feminine):
361 \newcommand*\@@hundredthstringFportuges[1]{%
362 \ifcase#1\relax
363 \or cent\'esima%
364 \or ducent\'esima%
365 \or trecent\'esima%
366 \or quadringent\'esima%
367 \or quingent\'esima%
368 \or seiscent\'esima%
369 \or setingent\'esima%
370 \or octingent\'esima%
371 \or nongent\'esima%
372 \fi
373 }%
374 \global\let\@@hundredthstringFportuges\@@hundredthstringFportuges
    As above, but with initial letter in upper case. Units:
375 \newcommand*\@@Unitthstringportuges[1]{%
376 \ifcase#1\relax
377 Zero%
378 \or Primeiro%
379 \or Segundo%
380 \or Terceiro%
381 \or Quarto%
382 \or Quinto%

```

```

383 \or Sexto%
384 \or S'etimo%
385 \or Oitavo%
386 \or Nono%
387 \fi
388 }%
389 \global\let\@@Unitthstringportuges\@@Unitthstringportuges

```

Tens:

```

390 \newcommand*\@@Tenthstringportuges[1]{%
391 \ifcase#1\relax
392 \or D'ecimo%
393 \or Vig'esimo%
394 \or Trig'esimo%
395 \or Quadrag'esimo%
396 \or Quinquag'esimo%
397 \or Sexag'esimo%
398 \or Setuag'esimo%
399 \or Octog'esimo%
400 \or Nonag'esimo%
401 \fi
402 }%
403 \global\let\@@Tenthstringportuges\@@Tenthstringportuges

```

Hundreds:

```

404 \newcommand*\@@Hundredthstringportuges[1]{%
405 \ifcase#1\relax
406 \or Cent'esimo%
407 \or Ducent'esimo%
408 \or Trecent'esimo%
409 \or Quadringent'esimo%
410 \or Quingent'esimo%
411 \or Seiscent'esimo%
412 \or Setingent'esimo%
413 \or Octingent'esimo%
414 \or Nongent'esimo%
415 \fi
416 }%
417 \global\let\@@Hundredthstringportuges\@@Hundredthstringportuges

```

As above, but feminine. Units:

```

418 \newcommand*\@@UnitthstringFportuges[1]{%
419 \ifcase#1\relax
420 Zera%
421 \or Primeira%
422 \or Segunda%
423 \or Terceira%
424 \or Quarta%
425 \or Quinta%
426 \or Sexta%
427 \or S'etima%

```

```

428   \or Oitava%
429   \or Nona%
430   \fi
431 }%
432 \global\let\@@UnitthstringFportuges\@@UnitthstringFportuges
    Tens (feminine);
433 \newcommand*\@@TenthstringFportuges[1]{%
434   \ifcase#1\relax
435     \or D\'esima%
436     \or Vig\'esima%
437     \or Trig\'esima%
438     \or Quadrag\'esima%
439     \or Quinquag\'esima%
440     \or Sexag\'esima%
441     \or Setuag\'esima%
442     \or Octog\'esima%
443     \or Nonag\'esima%
444   \fi
445 }%
446 \global\let\@@TenthstringFportuges\@@TenthstringFportuges
    Hundreds (feminine):
447 \newcommand*\@@HundredthstringFportuges[1]{%
448   \ifcase#1\relax
449     \or Cent\'esima%
450     \or Ducent\'esima%
451     \or Trecent\'esima%
452     \or Quadringent\'esima%
453     \or Quingent\'esima%
454     \or Seiscent\'esima%
455     \or Setingent\'esima%
456     \or Octingent\'esima%
457     \or Nongent\'esima%
458   \fi
459 }%
460 \global\let\@@HundredthstringFportuges\@@HundredthstringFportuges
    This has changed in version 1.09, so that it now stores the result in the second argument
    (a control sequence), but it doesn't display anything. Since it only affects internal macros,
    it shouldn't affect documents created with older versions. (These internal macros are not
    meant for use in documents.)
461 \newcommand*\@@numberstringportuges[2]{%
462 \ifnum#1>99999\relax
463   \PackageError{fmtcount}{Out of range}%
464   {This macro only works for values less than 100000}%
465 \else
466   \ifnum#1<0\relax
467     \PackageError{fmtcount}{Negative numbers not permitted}%
468     {This macro does not work for negative numbers, however

```

```

469     you can try typing "minus" first, and then pass the modulus of
470     this number}%
471 \fi
472 \fi
473 \def#2{}%
474 \@strctr=#1\relax \divide\@strctr by 1000\relax
475 \ifnum\@strctr>9\relax
    #1 is greater or equal to 10000
476 \divide\@strctr by 10\relax
477 \ifnum\@strctr>1\relax
478 \let\@fc@numstr#2\relax
479 \protected@edef#2{\@fc@numstr\@tenstring{\@strctr}}%
480 \@strctr=#1 \divide\@strctr by 1000\relax
481 \@FCmodulo{\@strctr}{10}%
482 \ifnum\@strctr>0
483 \let\@fc@numstr#2\relax
484 \protected@edef#2{\@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
485 \fi
486 \else
487 \@strctr=#1\relax
488 \divide\@strctr by 1000\relax
489 \@FCmodulo{\@strctr}{10}%
490 \let\@fc@numstr#2\relax
491 \protected@edef#2{\@fc@numstr\@teenstring{\@strctr}}%
492 \fi
493 \let\@fc@numstr#2\relax
494 \protected@edef#2{\@fc@numstr\ \@thousand}%
495 \else
496 \ifnum\@strctr>0\relax
497 \ifnum\@strctr>1\relax
498 \let\@fc@numstr#2\relax
499 \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}\ }%
500 \fi
501 \let\@fc@numstr#2\relax
502 \protected@edef#2{\@fc@numstr\@thousand}%
503 \fi
504 \fi
505 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
506 \divide\@strctr by 100\relax
507 \ifnum\@strctr>0\relax
508 \ifnum#1>1000 \relax
509 \let\@fc@numstr#2\relax
510 \protected@edef#2{\@fc@numstr\ \@andname\ }%
511 \fi
512 \@tmpstrctr=#1\relax
513 \@FCmodulo{\@tmpstrctr}{1000}%
514 \let\@fc@numstr#2\relax
515 \ifnum\@tmpstrctr=100\relax
516 \protected@edef#2{\@fc@numstr\@tenstring{10}}%

```

```

517 \else
518   \protected@edef#2{\@fc@numstr\@hundredstring{\@strctr}}%
519 \fi%
520 \fi
521 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
522 \ifnum#1>100\relax
523   \ifnum\@strctr>0\relax
524     \let\@fc@numstr#2\relax
525     \protected@edef#2{\@fc@numstr\ \@andname\ }%
526 \fi
527 \fi
528 \ifnum\@strctr>19\relax
529   \divide\@strctr by 10\relax
530   \let\@fc@numstr#2\relax
531   \protected@edef#2{\@fc@numstr\@tenstring{\@strctr}}%
532   \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
533   \ifnum\@strctr>0
534     \let\@fc@numstr#2\relax
535     \protected@edef#2{\@fc@numstr\ \@andname}%
536     \let\@fc@numstr#2\relax
537     \protected@edef#2{\@fc@numstr\ \@unitstring{\@strctr}}%
538 \fi
539 \else
540   \ifnum\@strctr<10\relax
541     \ifnum\@strctr=0\relax
542       \ifnum#1<100\relax
543         \let\@fc@numstr#2\relax
544         \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}}%
545       \fi
546     \else %(>0,<10)
547       \let\@fc@numstr#2\relax
548       \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}}%
549     \fi
550   \else%>10
551     \@FCmodulo{\@strctr}{10}%
552     \let\@fc@numstr#2\relax
553     \protected@edef#2{\@fc@numstr\@teenstring{\@strctr}}%
554   \fi
555 \fi
556 }%
557 \global\let\@numberstringportuges\@numberstringportuges

```

As above, but for ordinals.

```

558 \newcommand*\@ordinalstringportuges [2] {%
559 \@strctr=#1\relax
560 \ifnum#1>99999
561 \PackageError{fmtcount}{Out of range}%
562 {This macro only works for values less than 100000}%
563 \else
564 \ifnum#1<0

```



```

565 \PackageError{fmtcount}{Negative numbers not permitted}%
566 {This macro does not work for negative numbers, however
567 you can try typing "minus" first, and then pass the modulus of
568 this number}%
569 \else
570 \def#2{}%
571 \ifnum \@strctr>999\relax
572   \divide \@strctr by 1000\relax
573   \ifnum \@strctr>1\relax
574     \ifnum \@strctr>9\relax
575       \@tmpstrctr=\@strctr
576       \ifnum \@strctr<20
577         \FCmodulo{\@tmpstrctr}{10}%
578         \let\@@fc@ordstr#2\relax
579         \protected@edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
580       \else
581         \divide \@tmpstrctr by 10\relax
582         \let\@@fc@ordstr#2\relax
583         \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
584         \@tmpstrctr=\@strctr
585         \FCmodulo{\@tmpstrctr}{10}%
586         \ifnum \@tmpstrctr>0\relax
587           \let\@@fc@ordstr#2\relax
588           \protected@edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
589         \fi
590       \fi
591     \else
592       \let\@@fc@ordstr#2\relax
593       \protected@edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
594     \fi
595   \fi
596   \let\@@fc@ordstr#2\relax
597   \protected@edef#2{\@@fc@ordstr\@thousandth}%
598 \fi
599 \@strctr=#1\relax
600 \FCmodulo{\@strctr}{1000}%
601 \ifnum \@strctr>99\relax
602   \@tmpstrctr=\@strctr
603   \divide \@tmpstrctr by 100\relax
604   \ifnum#1>1000\relax
605     \let\@@fc@ordstr#2\relax
606     \protected@edef#2{\@@fc@ordstr-}%
607   \fi
608   \let\@@fc@ordstr#2\relax
609   \protected@edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
610 \fi
611 \FCmodulo{\@strctr}{100}%
612 \ifnum#1>99\relax
613   \ifnum \@strctr>0\relax

```

```

614 \let\@fc@ordstr#2\relax
615 \protected@edef#2{\@fc@ordstr-}%
616 \fi
617 \fi
618 \ifnum\@strctr>9\relax
619 \@tmpstrctr=\@strctr
620 \divide\@tmpstrctr by 10\relax
621 \let\@fc@ordstr#2\relax
622 \protected@edef#2{\@fc@ordstr\@tenthstring{\@tmpstrctr}}%
623 \@tmpstrctr=\@strctr
624 \@FCmodulo{\@tmpstrctr}{10}%
625 \ifnum\@tmpstrctr>0\relax
626 \let\@fc@ordstr#2\relax
627 \protected@edef#2{\@fc@ordstr-\@unitthstring{\@tmpstrctr}}%
628 \fi
629 \else
630 \ifnum\@strctr=0\relax
631 \ifnum#1=0\relax
632 \let\@fc@ordstr#2\relax
633 \protected@edef#2{\@fc@ordstr\@unitstring{0}}%
634 \fi
635 \else
636 \let\@fc@ordstr#2\relax
637 \protected@edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
638 \fi
639 \fi
640 \fi
641 \fi
642 }%
643 \global\let\@ordinalstringportuges\@ordinalstringportuges

```

### 10.1.15 fc-portuguese.def

```
1 \ProvidesFCLanguage{portuguese}[2014/06/09]%
```

Load fc-portuges.def if not already loaded.

```
2 \FCloadlang{portuges}%
```

Set portuguese to be equivalent to portuges.

```

3 \global\let\@ordinalMportuguese=\@ordinalMportuges
4 \global\let\@ordinalFportuguese=\@ordinalFportuges
5 \global\let\@ordinalNportuguese=\@ordinalNportuges
6 \global\let\@numberstringMportuguese=\@numberstringMportuges
7 \global\let\@numberstringFportuguese=\@numberstringFportuges
8 \global\let\@numberstringNportuguese=\@numberstringNportuges
9 \global\let\@NumberstringMportuguese=\@NumberstringMportuges
10 \global\let\@NumberstringFportuguese=\@NumberstringFportuges
11 \global\let\@NumberstringNportuguese=\@NumberstringNportuges
12 \global\let\@ordinalstringMportuguese=\@ordinalstringMportuges
13 \global\let\@ordinalstringFportuguese=\@ordinalstringFportuges

```

```

14 \global\let\@OrdinalstringNportuguese=\@OrdinalstringNportuges
15 \global\let\@OrdinalstringMportuguese=\@OrdinalstringMportuges
16 \global\let\@OrdinalstringFportuguese=\@OrdinalstringFportuges
17 \global\let\@OrdinalstringNportuguese=\@OrdinalstringNportuges

```

### 10.1.16 fc-spanish.def

Spanish definitions

```
1 \ProvidesFCLanguage{spanish}[2016/01/12]%
```

Define macro that converts a number or count register (first argument) to an ordinal, and stores the result in the second argument, which must be a control sequence. Masculine:

```

2 \newcommand*\@OrdinalMspanish[2]{%
3   \edef#2{\number#1\relax\noexpand\fmtord{o}}%
4 }%
5 \global\let\@OrdinalMspanish\@OrdinalMspanish

```

Feminine:

```

6 \newcommand*\@OrdinalFspanish[2]{%
7   \edef#2{\number#1\relax\noexpand\fmtord{a}}%
8 }%
9 \global\let\@OrdinalFspanish\@OrdinalFspanish

```

Make neuter same as masculine:

```
10 \global\let\@OrdinalNspanish\@OrdinalMspanish
```

Convert a number to text. The easiest way to do this is to break it up into units, tens, teens, twenties and hundreds. Units (argument must be a number from 0 to 9):

```

11 \newcommand*\@@unitstringspanish[1]{%
12   \ifcase#1\relax
13     cero%
14     \or uno%
15     \or dos%
16     \or tres%
17     \or cuatro%
18     \or cinco%
19     \or seis%
20     \or siete%
21     \or ocho%
22     \or nueve%
23   \fi
24 }%
25 \global\let\@@unitstringspanish\@@unitstringspanish

```

Feminine:

```

26 \newcommand*\@@unitstringFspanish[1]{%
27   \ifcase#1\relax
28     cera%
29     \or una%
30     \or dos%
31     \or tres%
32     \or cuatro%

```

```

33   \or cinco%
34   \or seis%
35   \or siete%
36   \or ocho%
37   \or nueve%
38   \fi
39 }%
40 \global\let\@@unitstringFspanish\@@unitstringFspanish
Tens (argument must go from 1 to 10):
41 \newcommand*\@@tenstringspanish[1]{%
42   \ifcase#1\relax
43   \or diez%
44   \or veinte%
45   \or treinta%
46   \or cuarenta%
47   \or cincuenta%
48   \or sesenta%
49   \or setenta%
50   \or ochenta%
51   \or noventa%
52   \or cien%
53   \fi
54 }%
55 \global\let\@@tenstringspanish\@@tenstringspanish
Teens:
56 \newcommand*\@@teenstringspanish[1]{%
57   \ifcase#1\relax
58   diez%
59   \or once%
60   \or doce%
61   \or trece%
62   \or catorce%
63   \or quince%
64   \or dieciséis%
65   \or diecisiete%
66   \or dieciocho%
67   \or diecinueve%
68   \fi
69 }%
70 \global\let\@@teenstringspanish\@@teenstringspanish
Twenties:
71 \newcommand*\@@twentystringspanish[1]{%
72   \ifcase#1\relax
73   veinte%
74   \or veintiuno%
75   \or veintidós%
76   \or veintitrés%
77   \or veinticuatro%

```

```

78 \or veinticinco%
79 \or veintiséis%
80 \or veintisiete%
81 \or veintiocho%
82 \or veintinueve%
83 \fi
84 }%
85 \global\let\@@twentystringspanish\@@twentystringspanish

```

Feminine form:

```

86 \newcommand*\@@twentystringFspanish[1]{%
87 \ifcase#1\relax
88 veinte%
89 \or veintiuna%
90 \or veintidós%
91 \or veintitrés%
92 \or veinticuatro%
93 \or veinticinco%
94 \or veintiséis%
95 \or veintisiete%
96 \or veintiocho%
97 \or veintinueve%
98 \fi
99 }%
100 \global\let\@@twentystringFspanish\@@twentystringFspanish

```

Hundreds:

```

101 \newcommand*\@@hundredstringspanish[1]{%
102 \ifcase#1\relax
103 \or ciento%
104 \or doscientos%
105 \or trescientos%
106 \or cuatrocientos%
107 \or quinientos%
108 \or seiscientos%
109 \or setecientos%
110 \or ochocientos%
111 \or novecientos%
112 \fi
113 }%
114 \global\let\@@hundredstringspanish\@@hundredstringspanish

```

Feminine form:

```

115 \newcommand*\@@hundredstringFspanish[1]{%
116 \ifcase#1\relax
117 \or cienta%
118 \or doscientas%
119 \or trescientas%
120 \or cuatrocientas%
121 \or quinientas%
122 \or seiscientas%

```

```

123 \or setecientas%
124 \or ochocientas%
125 \or novecientas%
126 \fi
127 }%
128 \global\let\@@hundredstringFspanish\@@hundredstringFspanish

```

As above, but with initial letter uppercase:

```

129 \newcommand*\@@Unitstringspanish[1]{%
130 \ifcase#1\relax
131 Cero%
132 \or Uno%
133 \or Dos%
134 \or Tres%
135 \or Cuatro%
136 \or Cinco%
137 \or Seis%
138 \or Siete%
139 \or Ocho%
140 \or Nueve%
141 \fi
142 }%
143 \global\let\@@Unitstringspanish\@@Unitstringspanish

```

Feminine form:

```

144 \newcommand*\@@UnitstringFspanish[1]{%
145 \ifcase#1\relax
146 Cera%
147 \or Una%
148 \or Dos%
149 \or Tres%
150 \or Cuatro%
151 \or Cinco%
152 \or Seis%
153 \or Siete%
154 \or Ocho%
155 \or Nueve%
156 \fi
157 }%
158 \global\let\@@UnitstringFspanish\@@UnitstringFspanish

```

Tens:

```

159 %\changes{2.0}{2012-06-18}{fixed spelling mistake (correction
160 %provided by Fernando Maldonado)}
161 \newcommand*\@@Tenstringspanish[1]{%
162 \ifcase#1\relax
163 \or Diez%
164 \or Veinte%
165 \or Treinta%
166 \or Cuarenta%
167 \or Cincuenta%

```

```

168 \or Sesenta%
169 \or Setenta%
170 \or Ochenta%
171 \or Noventa%
172 \or Cien%
173 \fi
174 }%
175 \global\let\@@Tenstringspanish\@@Tenstringspanish
    Teens:
176 \newcommand*\@@Teenstringspanish[1]{%
177 \ifcase#1\relax
178 Diez%
179 \or Once%
180 \or Doce%
181 \or Trece%
182 \or Catorce%
183 \or Quince%
184 \or Dieciséis%
185 \or Diecisiete%
186 \or Dieciocho%
187 \or Diecinueve%
188 \fi
189 }%
190 \global\let\@@Teenstringspanish\@@Teenstringspanish
    Twenties:
191 \newcommand*\@@Twentystringspanish[1]{%
192 \ifcase#1\relax
193 Veinte%
194 \or Veintiuno%
195 \or Veintidós%
196 \or Veintitrés%
197 \or Veinticuatro%
198 \or Veinticinco%
199 \or Veintiséis%
200 \or Veintisiete%
201 \or Veintiocho%
202 \or Veintinueve%
203 \fi
204 }%
205 \global\let\@@Twentystringspanish\@@Twentystringspanish
    Feminine form:
206 \newcommand*\@@TwentystringFspanish[1]{%
207 \ifcase#1\relax
208 Veinte%
209 \or Veintiuna%
210 \or Veintidós%
211 \or Veintitrés%
212 \or Veinticuatro%

```

```

213 \or Veinticinco%
214 \or Veintiséis%
215 \or Veintisiete%
216 \or Veintiocho%
217 \or Veintinueve%
218 \fi
219 }%
220 \global\let\@@TwentystringFspanish\@@TwentystringFspanish

```

Hundreds:

```

221 \newcommand*\@@Hundredstringspanish[1]{%
222 \ifcase#1\relax
223 \or Ciento%
224 \or Doscientos%
225 \or Trescientos%
226 \or Cuatrocientos%
227 \or Quinientos%
228 \or Seiscientos%
229 \or Setecientos%
230 \or Ochocientos%
231 \or Novecientos%
232 \fi
233 }%
234 \global\let\@@Hundredstringspanish\@@Hundredstringspanish

```

Feminine form:

```

235 \newcommand*\@@HundredstringFspanish[1]{%
236 \ifcase#1\relax
237 \or Cienta%
238 \or Doscientas%
239 \or Trescientas%
240 \or Cuatrocientas%
241 \or Quinientas%
242 \or Seiscientas%
243 \or Setecientas%
244 \or Ochocientas%
245 \or Novecientas%
246 \fi
247 }%
248 \global\let\@@HundredstringFspanish\@@HundredstringFspanish

```

This has changed in version 1.09, so that it now stores the result in the second argument, but doesn't display anything. Since it only affects internal macros, it shouldn't affect documents created with older versions. (These internal macros are not meant for use in documents.)

```

249 \newcommand*\@@numberstringMspanish}[2]{%
250 \let\@unitstring=\@@unitstringspanish
251 \let\@teenstring=\@@teenstringspanish
252 \let\@tenstring=\@@tenstringspanish
253 \let\@twentystring=\@@twentystringspanish
254 \let\@hundredstring=\@@hundredstringspanish

```



```

255 \def\@hundred{cien}\def\@thousand{mil}%
256 \def\@andname{y}%
257 \@numberstringspanish{#1}{#2}%
258 }%
259 \global\let\@numberstringMspanish\@numberstringMspanish

Feminine form:
260 \newcommand*\@numberstringFspanish}[2]{%
261 \let\@unitstring=\@unitstringFspanish
262 \let\@teenstring=\@teenstringspanish
263 \let\@tenstring=\@tenstringspanish
264 \let\@twentystring=\@twentystringFspanish
265 \let\@hundredstring=\@hundredstringFspanish
266 \def\@hundred{cien}\def\@thousand{mil}%
267 \def\@andname{b}%
268 \@numberstringspanish{#1}{#2}%
269 }%
270 \global\let\@numberstringFspanish\@numberstringFspanish

Make neuter same as masculine:
271 \global\let\@numberstringNspanish\@numberstringMspanish

As above, but initial letters in upper case:
272 \newcommand*\@NumberstringMspanish}[2]{%
273 \let\@unitstring=\@Unitstringspanish
274 \let\@teenstring=\@Teenstringspanish
275 \let\@tenstring=\@Tenstringspanish
276 \let\@twentystring=\@Twentystringspanish
277 \let\@hundredstring=\@Hundredstringspanish
278 \def\@andname{Y}%
279 \def\@hundred{Cien}\def\@thousand{Mil}%
280 \@numberstringspanish{#1}{#2}%
281 }%
282 \global\let\@NumberstringMspanish\@NumberstringMspanish

Feminine form:
283 \newcommand*\@NumberstringFspanish}[2]{%
284 \let\@unitstring=\@UnitstringFspanish
285 \let\@teenstring=\@Teenstringspanish
286 \let\@tenstring=\@Tenstringspanish
287 \let\@twentystring=\@TwentystringFspanish
288 \let\@hundredstring=\@HundredstringFspanish
289 \def\@andname{b}%
290 \def\@hundred{Cien}\def\@thousand{Mil}%
291 \@numberstringspanish{#1}{#2}%
292 }%
293 \global\let\@NumberstringFspanish\@NumberstringFspanish

Make neuter same as masculine:
294 \global\let\@NumberstringNspanish\@NumberstringMspanish

As above, but for ordinals.

```

```

295 \newcommand*{\@OrdinalstringMspanish}[2]{%
296 \let\@unitthstring=\@unitthstringspanish
297 \let\@unitstring=\@unitstringspanish
298 \let\@teenthstring=\@teenthstringspanish
299 \let\@tenthstring=\@tenthstringspanish
300 \let\@hundredthstring=\@hundredthstringspanish
301 \def\@thousandth{milésimo}%
302 \@Ordinalstringspanish{#1}{#2}%
303 }%
304 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish

Feminine form:
305 \newcommand*{\@OrdinalstringFspanish}[2]{%
306 \let\@unitthstring=\@unitthstringFspanish
307 \let\@unitstring=\@unitstringFspanish
308 \let\@teenthstring=\@teenthstringFspanish
309 \let\@tenthstring=\@tenthstringFspanish
310 \let\@hundredthstring=\@hundredthstringFspanish
311 \def\@thousandth{milésima}%
312 \@Ordinalstringspanish{#1}{#2}%
313 }%
314 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish

Make neuter same as masculine:
315 \global\let\@OrdinalstringNspanish\@OrdinalstringMspanish

As above, but with initial letters in upper case.
316 \newcommand*{\@OrdinalstringMspanish}[2]{%
317 \let\@unitthstring=\@Unitthstringspanish
318 \let\@unitstring=\@Unitstringspanish
319 \let\@teenthstring=\@Teenthstringspanish
320 \let\@tenthstring=\@Tenthstringspanish
321 \let\@hundredthstring=\@Hundredthstringspanish
322 \def\@thousandth{Milésimo}%
323 \@Ordinalstringspanish{#1}{#2}%
324 }
325 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish

Feminine form:
326 \newcommand*{\@OrdinalstringFspanish}[2]{%
327 \let\@unitthstring=\@UnitthstringFspanish
328 \let\@unitstring=\@UnitstringFspanish
329 \let\@teenthstring=\@TeenthstringFspanish
330 \let\@tenthstring=\@TenthstringFspanish
331 \let\@hundredthstring=\@HundredthstringFspanish
332 \def\@thousandth{Milésima}%
333 \@Ordinalstringspanish{#1}{#2}%
334 }%
335 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish

Make neuter same as masculine:
336 \global\let\@OrdinalstringNspanish\@OrdinalstringMspanish

```

Code for convert numbers into textual ordinals. As before, it is easier to split it into units, tens, teens and hundreds. Units:

```
337 \newcommand*\@@unitthstringspanish[1]{%
338   \ifcase#1\relax
339     cero%
340     \or primero%
341     \or segundo%
342     \or tercero%
343     \or cuarto%
344     \or quinto%
345     \or sexto%
346     \or séptimo%
347     \or octavo%
348     \or noveno%
349   \fi
350 }%
351 \global\let\@@unitthstringspanish\@@unitthstringspanish
```

Tens:

```
352 \newcommand*\@@tenthstringspanish[1]{%
353   \ifcase#1\relax
354     \or décimo%
355     \or vigésimo%
356     \or trigésimo%
357     \or cuadragésimo%
358     \or quincuagésimo%
359     \or sexagésimo%
360     \or septuagésimo%
361     \or octogésimo%
362     \or nonagésimo%
363   \fi
364 }%
365 \global\let\@@tenthstringspanish\@@tenthstringspanish
```

Teens:

```
366 \newcommand*\@@teenthstringspanish[1]{%
367   \ifcase#1\relax
368     décimo%
369     \or undécimo%
370     \or duodécimo%
371     \or decimotercero%
372     \or decimocuarto%
373     \or decimoquinto%
374     \or decimosexto%
375     \or decimoséptimo%
376     \or decimoctavo%
377     \or decimonoveno%
378   \fi
379 }%
380 \global\let\@@teenthstringspanish\@@teenthstringspanish
```

### Hundreds:

```
381 \newcommand*{\@@hundredthstringspanish[1]}{%
382   \ifcase#1\relax
383     \or centésimo%
384     \or ducentésimo%
385     \or tricentésimo%
386     \or cuadingentésimo%
387     \or quingentésimo%
388     \or sexcentésimo%
389     \or septingésimo%
390     \or octingentésimo%
391     \or noningentésimo%
392   \fi
393 }%
394 \global\let\@@hundredthstringspanish\@@hundredthstringspanish
```

### Units (feminine):

```
395 \newcommand*{\@@unitthstringFspanish[1]}{%
396   \ifcase#1\relax
397     cera%
398     \or primera%
399     \or segunda%
400     \or tercera%
401     \or cuarta%
402     \or quinta%
403     \or sexta%
404     \or séptima%
405     \or octava%
406     \or novena%
407   \fi
408 }%
409 \global\let\@@unitthstringFspanish\@@unitthstringFspanish
```

### Tens (feminine):

```
410 \newcommand*{\@@tenthstringFspanish[1]}{%
411   \ifcase#1\relax
412     \or décima%
413     \or vigésima%
414     \or trigésima%
415     \or cuadragésima%
416     \or quincuagésima%
417     \or sexagésima%
418     \or septuagésima%
419     \or octogésima%
420     \or nonagésima%
421   \fi
422 }%
423 \global\let\@@tenthstringFspanish\@@tenthstringFspanish
```

### Teens (feminine)

```

424 \newcommand*{\@@teenthstringFspanish[1]}{
425   \ifcase#1\relax
426     décima%
427     \or undécima%
428     \or duodécima%
429     \or decimotercera%
430     \or decimocuarta%
431     \or decimoquinta%
432     \or decimosexta%
433     \or decimoséptima%
434     \or decimoctava%
435     \or decimonovena%
436   \fi
437 }%
438 \global\let\@@teenthstringFspanish\@@teenthstringFspanish

Hundreds (feminine)
439 \newcommand*{\@@hundredthstringFspanish[1]}{
440   \ifcase#1\relax
441     \or centésima%
442     \or ducentésima%
443     \or tricentésima%
444     \or cuadringentésima%
445     \or quingentésima%
446     \or sexcentésima%
447     \or septingésima%
448     \or octingentésima%
449     \or noningentésima%
450   \fi
451 }%
452 \global\let\@@hundredthstringFspanish\@@hundredthstringFspanish

As above, but with initial letters in upper case
453 \newcommand*{\@@Unitthstringspanish[1]}{
454   \ifcase#1\relax
455     Cero%
456     \or Primero%
457     \or Segundo%
458     \or Tercero%
459     \or Cuarto%
460     \or Quinto%
461     \or Sexto%
462     \or Séptimo%
463     \or Octavo%
464     \or Noveno%
465   \fi
466 }%
467 \global\let\@@Unitthstringspanish\@@Unitthstringspanish

Tens:
468 \newcommand*{\@@Tenthstringspanish[1]}{

```

```

469 \ifcase#1\relax
470 \or Décimo%
471 \or Vigésimo%
472 \or Trigésimo%
473 \or Cuadragésimo%
474 \or Quincuagésimo%
475 \or Sexagésimo%
476 \or Septuagésimo%
477 \or Octogésimo%
478 \or Nonagésimo%
479 \fi
480 }%
481 \global\let\@@Tenthstringspanish\@@Tenthstringspanish

```

#### Teens:

```

482 \newcommand*\@@Teenthstringspanish[1]{%
483 \ifcase#1\relax
484 Décimo%
485 \or Undécimo%
486 \or Duodécimo%
487 \or Decimotercero%
488 \or Decimocuarto%
489 \or Decimoquinto%
490 \or Decimosexto%
491 \or Decimoséptimo%
492 \or Decimooctavo%
493 \or Decimonoveno%
494 \fi
495 }%
496 \global\let\@@Teenthstringspanish\@@Teenthstringspanish

```

#### Hundreds

```

497 \newcommand*\@@Hundredthstringspanish[1]{%
498 \ifcase#1\relax
499 \or Centésimo%
500 \or Ducentésimo%
501 \or Tricentésimo%
502 \or Cuadringentésimo%
503 \or Quingentésimo%
504 \or Sexcentésimo%
505 \or Septingésimo%
506 \or Octingentésimo%
507 \or Noningentésimo%
508 \fi
509 }%
510 \global\let\@@Hundredthstringspanish\@@Hundredthstringspanish

```

#### As above, but feminine.

```

511 \newcommand*\@@UnitthstringFspanish[1]{%
512 \ifcase#1\relax
513 Cera%

```

514 \or Primera%  
515 \or Segunda%  
516 \or Tercera%  
517 \or Cuarta%  
518 \or Quinta%  
519 \or Sexta%  
520 \or Séptima%  
521 \or Octava%  
522 \or Novena%  
523 \fi  
524 }%  
525 \global\let\@@UnitthstringFspanish\@@UnitthstringFspanish  
Tens (feminine)  
526 \newcommand\*\@@TenthstringFspanish[1]{%  
527 \ifcase#1\relax  
528 \or Décima%  
529 \or Vigésima%  
530 \or Trigésima%  
531 \or Cuadragésima%  
532 \or Quincuagésima%  
533 \or Sexagésima%  
534 \or Septuagésima%  
535 \or Octogésima%  
536 \or Nonagésima%  
537 \fi  
538 }%  
539 \global\let\@@TenthstringFspanish\@@TenthstringFspanish  
Teens (feminine):  
540 \newcommand\*\@@TeenthstringFspanish[1]{%  
541 \ifcase#1\relax  
542 Décima%  
543 \or Undécima%  
544 \or Duodécima%  
545 \or Decimotercera%  
546 \or Decimocuarta%  
547 \or Decimoquinta%  
548 \or Decimosexta%  
549 \or Decimoséptima%  
550 \or Decimooctava%  
551 \or Decimonovena%  
552 \fi  
553 }%  
554 \global\let\@@TeenthstringFspanish\@@TeenthstringFspanish  
Hundreds (feminine):  
555 \newcommand\*\@@HundredthstringFspanish[1]{%  
556 \ifcase#1\relax  
557 \or Centésima%  
558 \or Ducentésima%

```

559 \or Tricentésima%
560 \or Cuadringentésima%
561 \or Quingentésima%
562 \or Sexcentésima%
563 \or Septingésima%
564 \or Octingentésima%
565 \or Noningentésima%
566 \fi
567 }%
568 \global\let\@@HundredthstringFspanish\@@HundredthstringFspanish

```

This has changed in version 1.09, so that it now stores the results in the second argument (which must be a control sequence), but it doesn't display anything. Since it only affects internal macros, it shouldn't affect documents created with older versions. (These internal macros are not meant for use in documents.)

```

569 \newcommand*\@@numberstringspanish[2]{%
570 \ifnum#1>99999
571 \PackageError{fmtcount}{Out of range}%
572 {This macro only works for values less than 100000}%
573 \else
574 \ifnum#1<0
575 \PackageError{fmtcount}{Negative numbers not permitted}%
576 {This macro does not work for negative numbers, however
577 you can try typing "minus" first, and then pass the modulus of
578 this number}%
579 \fi
580 \fi
581 \def#2{}%
582 \@strctr=#1\relax \divide\@strctr by 1000\relax
583 \ifnum\@strctr>9
#1 is greater or equal to 1000
584 \divide\@strctr by 10
585 \ifnum\@strctr>1
586 \let\@@fc@numstr#2\relax
587 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
588 \@strctr=#1 \divide\@strctr by 1000\relax
589 \@FCmodulo{\@strctr}{10}%
590 \ifnum\@strctr>0\relax
591 \let\@@fc@numstr#2\relax
592 \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
593 \fi
594 \else
595 \@strctr=#1\relax
596 \divide\@strctr by 1000\relax
597 \@FCmodulo{\@strctr}{10}%
598 \let\@@fc@numstr#2\relax
599 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
600 \fi
601 \let\@@fc@numstr#2\relax

```



```

602 \edef#2{\@fc@numstr\ \@thousand}%
603 \else
604 \ifnum\@strctr>0\relax
605 \ifnum\@strctr>1\relax
606 \let\@fc@numstr#2\relax
607 \edef#2{\@fc@numstr\@unitstring{\@strctr}\ }%
608 \fi
609 \let\@fc@numstr#2\relax
610 \edef#2{\@fc@numstr\@thousand}%
611 \fi
612 \fi
613 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
614 \divide\@strctr by 100\relax
615 \ifnum\@strctr>0\relax
616 \ifnum#1>1000\relax
617 \let\@fc@numstr#2\relax
618 \edef#2{\@fc@numstr\ }%
619 \fi
620 \@tmpstrctr=#1\relax
621 \@FCmodulo{\@tmpstrctr}{1000}%
622 \ifnum\@tmpstrctr=100\relax
623 \let\@fc@numstr#2\relax
624 \edef#2{\@fc@numstr\@tenstring{10}}%
625 \else
626 \let\@fc@numstr#2\relax
627 \edef#2{\@fc@numstr\@hundredstring{\@strctr}}%
628 \fi
629 \fi
630 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
631 \ifnum#1>100\relax
632 \ifnum\@strctr>0\relax
633 \let\@fc@numstr#2\relax
634 \edef#2{\@fc@numstr\ }%
635 \fi
636 \fi
637 \ifnum\@strctr>29\relax
638 \divide\@strctr by 10\relax
639 \let\@fc@numstr#2\relax
640 \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
641 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
642 \ifnum\@strctr>0\relax
643 \let\@fc@numstr#2\relax
644 \edef#2{\@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
645 \fi
646 \else
647 \ifnum\@strctr<10\relax
648 \ifnum\@strctr=0\relax
649 \ifnum#1<100\relax
650 \let\@fc@numstr#2\relax

```

```

651     \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
652     \fi
653   \else
654     \let\@@fc@numstr#2\relax
655     \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
656     \fi
657   \else
658     \ifnum\@strctr>19\relax
659       \@FCmodulo{\@strctr}{10}%
660       \let\@@fc@numstr#2\relax
661       \edef#2{\@@fc@numstr\@twentystring{\@strctr}}%
662     \else
663       \@FCmodulo{\@strctr}{10}%
664       \let\@@fc@numstr#2\relax
665       \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
666     \fi
667   \fi
668 \fi
669 }%
670 \global\let\@@numberstringspanish\@numberstringspanish

```

As above, but for ordinals

```

671 \newcommand*\@@ordinalstringspanish[2]{%
672 \@strctr=#1\relax
673 \ifnum#1>99999
674 \PackageError{fmtcount}{Out of range}%
675 {This macro only works for values less than 100000}%
676 \else
677 \ifnum#1<0
678 \PackageError{fmtcount}{Negative numbers not permitted}%
679 {This macro does not work for negative numbers, however
680 you can try typing "minus" first, and then pass the modulus of
681 this number}%
682 \else
683 \def#2{}%
684 \ifnum\@strctr>999\relax
685   \divide\@strctr by 1000\relax
686   \ifnum\@strctr>1\relax
687     \ifnum\@strctr>9\relax
688       \@tmpstrctr=\@strctr
689       \ifnum\@strctr<20
690         \@FCmodulo{\@tmpstrctr}{10}%
691         \let\@@fc@ordstr#2\relax
692         \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
693       \else
694         \divide\@tmpstrctr by 10\relax
695         \let\@@fc@ordstr#2\relax
696         \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
697         \@tmpstrctr=\@strctr
698         \@FCmodulo{\@tmpstrctr}{10}%

```

```

699     \ifnum \@tmpstrctr>0\relax
700     \let\@@fc@ordstr#2\relax
701     \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
702     \fi
703     \fi
704     \else
705     \let\@@fc@ordstr#2\relax
706     \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
707     \fi
708     \fi
709     \let\@@fc@ordstr#2\relax
710     \edef#2{\@@fc@ordstr\@thousandth}%
711     \fi
712     \@strctr=#1\relax
713     \@FCmodulo{\@strctr}{1000}%
714     \ifnum \@strctr>99\relax
715     \@tmpstrctr=\@strctr
716     \divide \@tmpstrctr by 100\relax
717     \ifnum#1>1000\relax
718     \let\@@fc@ordstr#2\relax
719     \edef#2{\@@fc@ordstr\ }%
720     \fi
721     \let\@@fc@ordstr#2\relax
722     \edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
723     \fi
724     \@FCmodulo{\@strctr}{100}%
725     \ifnum#1>99\relax
726     \ifnum \@strctr>0\relax
727     \let\@@fc@ordstr#2\relax
728     \edef#2{\@@fc@ordstr\ }%
729     \fi
730     \fi
731     \ifnum \@strctr>19\relax
732     \@tmpstrctr=\@strctr
733     \divide \@tmpstrctr by 10\relax
734     \let\@@fc@ordstr#2\relax
735     \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
736     \@tmpstrctr=\@strctr
737     \@FCmodulo{\@tmpstrctr}{10}%
738     \ifnum \@tmpstrctr>0\relax
739     \let\@@fc@ordstr#2\relax
740     \edef#2{\@@fc@ordstr\ \@unitthstring{\@tmpstrctr}}%
741     \fi
742     \else
743     \ifnum \@strctr>9\relax
744     \@FCmodulo{\@strctr}{10}%
745     \let\@@fc@ordstr#2\relax
746     \edef#2{\@@fc@ordstr\@teenthstring{\@strctr}}%
747     \else

```

```

748 \ifnum\@strctr=0\relax
749 \ifnum#1=0\relax
750 \let\@fc@ordstr#2\relax
751 \edef#2{\@fc@ordstr\@unitstring{0}}%
752 \fi
753 \else
754 \let\@fc@ordstr#2\relax
755 \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
756 \fi
757 \fi
758 \fi
759 \fi
760 \fi
761 }%
762 \global\let\@ordinalstringspanish\@ordinalstringspanish

```

### 10.1.17 fc-UKenglish.def

English definitions

```
1 \ProvidesFCLanguage{UKenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
2 \FCloadlang{english}%
```

These are all just synonyms for the commands provided by fc-english.def.

```

3 \global\let\@ordinalMUKenglish\@ordinalMenglish
4 \global\let\@ordinalFUKenglish\@ordinalMenglish
5 \global\let\@ordinalNUKenglish\@ordinalMenglish
6 \global\let\@numberstringMUKenglish\@numberstringMenglish
7 \global\let\@numberstringFUKenglish\@numberstringMenglish
8 \global\let\@numberstringNUKenglish\@numberstringMenglish
9 \global\let\@NumberstringMUKenglish\@NumberstringMenglish
10 \global\let\@NumberstringFUKenglish\@NumberstringMenglish
11 \global\let\@NumberstringNUKenglish\@NumberstringMenglish
12 \global\let\@ordinalstringMUKenglish\@ordinalstringMenglish
13 \global\let\@ordinalstringFUKenglish\@ordinalstringMenglish
14 \global\let\@ordinalstringNUKenglish\@ordinalstringMenglish
15 \global\let\@OrdinalstringMUKenglish\@OrdinalstringMenglish
16 \global\let\@OrdinalstringFUKenglish\@OrdinalstringMenglish
17 \global\let\@OrdinalstringNUKenglish\@OrdinalstringMenglish

```

### 10.1.18 fc-USenglish.def

US English definitions

```
1 \ProvidesFCLanguage{USenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
2 \FCloadlang{english}%
```

These are all just synonyms for the commands provided by fc-english.def. (This needs fixing as there are some differences between UK and US number strings.)

```

3 \global\let\@ordinalMUSenglish\@ordinalMenglish
4 \global\let\@ordinalFUSenglish\@ordinalMenglish
5 \global\let\@ordinalNUSenglish\@ordinalMenglish
6 \global\let\@numberstringMUSenglish\@numberstringMenglish
7 \global\let\@numberstringFUSenglish\@numberstringMenglish
8 \global\let\@numberstringNUSenglish\@numberstringMenglish
9 \global\let\@NumberstringMUSenglish\@NumberstringMenglish
10 \global\let\@NumberstringFUSenglish\@NumberstringMenglish
11 \global\let\@NumberstringNUSenglish\@NumberstringMenglish
12 \global\let\@ordinalstringMUSenglish\@ordinalstringMenglish
13 \global\let\@ordinalstringFUSenglish\@ordinalstringMenglish
14 \global\let\@ordinalstringNUSenglish\@ordinalstringMenglish
15 \global\let\@OrdinalstringMUSenglish\@OrdinalstringMenglish
16 \global\let\@OrdinalstringFUSenglish\@OrdinalstringMenglish
17 \global\let\@OrdinalstringNUSenglish\@OrdinalstringMenglish

```

## 10.2 fcnumparser.sty

```

1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{fcnumparser}[2017/06/15]

```

\fc@counter@parser is just a shorthand to parse a number held in a counter.

```

3 \def\fc@counter@parser#1{%
4   \expandafter\fc@number@parser\expandafter{\the#1.}%
5 }
6 \newcount\fc@digit@counter
7
8 \def\fc@end@{\fc@end}

```

number@analysis First of all we need to separate the number between integer and fractional part. Number to be analysed is in '#1'. Decimal separator may be . or , whichever first. At end of this macro, integer part goes to \fc@integer@part and fractional part goes to \fc@fractional@part.

```

9 \def\fc@number@analysis#1\fc@nil{%

```

First check for the presence of a decimal point in the number.

```

10 \def\@tempb##1.##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%
11 \@tempb#1.\fc@end\fc@nil
12 \ifx\@tempa\fc@end@

```

Here \@tempa is \ifx-equal to \fc@end, which means that the number does not contain any decimal point. So we do the same trick to search for a comma.

```

13 \def\@tempb##1,##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%
14 \@tempb#1,\fc@end\fc@nil
15 \ifx\@tempa\fc@end@

```

No comma either, so fractional part is set empty.

```

16 \def\fc@fractional@part{}%
17 \else

```

Comma has been found, so we just need to drop ‘, \fc@end’ from the end of \@tempa to get the fractional part.

```
18     \def\@tempb##1,\fc@end{\def\fc@fractional@part{##1}}%
19     \expandafter\@tempb\@tempa
20     \fi
21     \else
```

Decimal point has been found, so we just need to drop ‘. \fc@end’ from the end \@tempa to get the fractional part.

```
22     \def\@tempb##1.\fc@end{\def\fc@fractional@part{##1}}%
23     \expandafter\@tempb\@tempa
24     \fi
25 }
```

c@number@parser

Macro \fc@number@parser is the main engine to parse a number. Argument ‘#1’ is input and contains the number to be parsed. At end of this macro, each digit is stored separately in a \fc@digit@<n>, and macros \fc@min@weight and \fc@max@weight are set to the bounds for <n>.

```
26 \def\fc@number@parser#1{%
```

First remove all the spaces in #1, and place the result into \@tempa.

```
27     \let\@tempa\@empty
28     \def\@tempb##1##2\fc@nil{%
29         \def\@tempc{##1}%
30         \ifx\@tempc\space
31             \else
32                 \expandafter\def\expandafter\@tempa\expandafter{\@tempa ##1}%
33             \fi
34         \def\@tempc{##2}%
35         \ifx\@tempc\@empty
36             \expandafter\@gobble
37         \else
38             \expandafter\@tempb
39         \fi
40         ##2\fc@nil
41     }%
42     \@tempb#1\fc@nil
```

Get the sign into \fc@sign and the unsigned number part into \fc@number.

```
43     \def\@tempb##1##2\fc@nil{\def\fc@sign{##1}\def\fc@number{##2}}%
44     \expandafter\@tempb\@tempa\fc@nil
45     \expandafter\if\fc@sign+%
46         \def\fc@sign@case{1}%
47     \else
48         \expandafter\if\fc@sign-%
49             \def\fc@sign@case{2}%
50         \else
51             \def\fc@sign{}%
52             \def\fc@sign@case{0}%
53         \let\fc@number\@tempa
54     \fi
```

```

55 \fi
56 \ifx\fc@number\@empty
57   \PackageError{fcnumparser}{Invalid number}{Number must contain at least one non blank
58     character after sign}%
59 \fi

```

Now, split `\fc@number` into `\fc@integer@part` and `\fc@fractional@part`.

```
60 \expandafter\fc@number@analysis\fc@number\fc@nil
```

Now, split `\fc@integer@part` into a sequence of `\fc@digit@⟨n⟩` with `⟨n⟩` ranging from `\fc@unit@weight` to `\fc@max@weight`. We will use macro `\fc@parse@integer@digits` for that, but that will place the digits into `\fc@digit@⟨n⟩` with `⟨n⟩` ranging from  $2 \times \text{\fc@unit@weight} - \text{\fc@max@weight}$  upto  $\text{\fc@unit@weight} - 1$ .

```
61 \expandafter\fc@digit@counter\fc@unit@weight
62 \expandafter\fc@parse@integer@digits\fc@integer@part\fc@end\fc@nil
```

First we compute the weight of the most significant digit: after `\fc@parse@integer@digits`, `\fc@digit@counter` is equal to  $\text{\fc@unit@weight} - \text{mw} - 1$  and we want to set `\fc@max@weight` to  $\text{\fc@unit@weight} + \text{mw}$  so we do:

$$\text{\fc@max@weight} \leftarrow (-\text{\fc@digit@counter}) + 2 \times \text{\fc@unit@weight} - 1$$

```
63 \fc@digit@counter -\fc@digit@counter
64 \advance\fc@digit@counter by \fc@unit@weight
65 \advance\fc@digit@counter by \fc@unit@weight
66 \advance\fc@digit@counter by -1 %
67 \edef\fc@max@weight{\the\fc@digit@counter}%

```

Now we loop for  $i = \text{\fc@unit@weight}$  to  $\text{\fc@max@weight}$  in order to copy all the digits from `\fc@digit@⟨i + offset⟩` to `\fc@digit@⟨i⟩`. First we compute offset into `\@temp_i`.

```
68 {%
69   \count0 \fc@unit@weight\relax
70   \count1 \fc@max@weight\relax
71   \advance\count0 by -\count1 %
72   \advance\count0 by -1 %
73   \def\@tempa##1{\def\@tempb{\def\@temp_i{##1}}}%
74   \expandafter\@tempa\expandafter{\the\count0}%
75   \expandafter
76 } \@tempb

```

Now we loop to copy the digits. To do that we define a macro `\@temp_l` for terminal recursion.

```
77 \expandafter\fc@digit@counter\fc@unit@weight
78 \def\@temp_l{%
79   \ifnum\fc@digit@counter>\fc@max@weight
80     \let\next\relax
81   \else

```

Here is the loop body:

```
82   {%
83     \count0 \@temp_i
84     \advance\count0 by \fc@digit@counter
85     \expandafter\def\expandafter\@temp_d\expandafter{\csname fc@digit@\the\count0\endcsname
86     \expandafter\def\expandafter\@temp_e\expandafter{\csname fc@digit@\the\fc@digit@counte

```

```

87     \def\@tempa###1###2{\def\@tempb{\let###1###2}}%
88     \expandafter\expandafter\expandafter\@tempa\expandafter\@tempe\@tempd
89     \expandafter
90     }\@tempb
91     \advance\fc@digit@counter by 1 %
92     \fi
93     \next
94 }%
95 \let\next\@templ
96 \@templ

```

Split \fc@fractional@part into a sequence of \fc@digit@ $\langle n \rangle$  with  $\langle n \rangle$  ranging from \fc@unit@weight - 1 to \fc@min@weight by step of -1. This is much more simpler because we get the digits with the final range of index, so no post-processing loop is needed.

```

97 \expandafter\fc@digit@counter\fc@unit@weight
98 \expandafter\fc@parse@integer@digits\fc@fractional@part\fc@end\fc@nil
99 \edef\fc@min@weight{\the\fc@digit@counter}%
100 }

```

```

Macro \fc@parse@integer@digits is used to
101 \ifcsundef\fc@parse@integer@digits\{}{%
102 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of
103 macro 'fc@parse@integer@digits'}}
104 \def\fc@parse@integer@digits#1#2\fc@nil{%
105 \def\@tempa{#1}%
106 \ifx\@tempa\fc@end@
107 \def\next##1\fc@nil{%
108 \else
109 \let\next\fc@parse@integer@digits
110 \advance\fc@digit@counter by -1
111 \expandafter\def\csname fc@digit@\the\fc@digit@counter\endcsname{#1}%
112 \fi
113 \next#2\fc@nil
114 }
115
116
117 \newcommand*\fc@unit@weight{0}
118

```

Now we have macros to read a few digits from the \fc@digit@ $\langle n \rangle$  array and form a corresponding number.

```

\fc@read@unit \fc@read@unit just reads one digit and form an integer in the range [0..9]. First we check
that the macro is not yet defined.
119 \ifcsundef\fc@read@unit\{}{%
120 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro 'fc@read@unit'}}

```

Arguments as follows:

#1 output counter: into which the read value is placed #2  
#2 input number: unit weight at which reach the value is to be read  
does not need to be comprised between \fc@min@weight and fc@min@weight, if outside this interval, then a zero is read.



```

121 \def\fc@read@unit#1#2{%
122   \ifnum#2>\fc@max@weight
123     #1=0\relax
124   \else
125     \ifnum#2<\fc@min@weight
126       #1=0\relax
127     \else
128       {%
129         \edef\@tempa{\number#2}%
130         \count0=\@tempa
131         \edef\@tempa{\csname fc@digit@the\count0\endcsname}%
132         \def\@tempb##1{\def\@tempa{#1=##1\relax}}%
133         \expandafter\@tempb\expandafter{\@tempa}%
134         \expandafter
135       }\@tempa
136     \fi
137   \fi
138 }

```

`\fc@read@hundred` Macro `\fc@read@hundred` is used to read a pair of digits and form an integer in the range [0..99]. First we check that the macro is not yet defined.

```

139 \ifcsundef{fc@read@hundred}{-}{%
140   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro ‘fc@read@hundred’}}

```

Arguments as follows — same interface as `\fc@read@unit`:

- #1 output counter: into which the read value is placed
- #2 input number: unit weight at which reach the value is to be read

```

141 \def\fc@read@hundred#1#2{%
142   {%
143     \fc@read@unit{\count0}{#2}%
144     \def\@tempa##1{\fc@read@unit{\count1}{##1}}%
145     \count2=#2%
146     \advance\count2 by 1 %
147     \expandafter\@tempa{the\count2}%
148     \multiply\count1 by 10 %
149     \advance\count1 by \count0 %
150     \def\@tempa##1{\def\@tempb{#1=##1\relax}}
151     \expandafter\@tempa\expandafter{the\count1}%
152     \expandafter
153   }\@tempb
154 }

```

`\fc@read@thousand` Macro `\fc@read@thousand` is used to read a trio of digits and form an integer in the range [0..999]. First we check that the macro is not yet defined.

```

155 \ifcsundef{fc@read@thousand}{-}{%
156   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
157     ‘fc@read@thousand’}}

```

Arguments as follows — same interface as `\fc@read@unit`:

- #1 output counter: into which the read value is placed
- #2 input number: unit weight at which reach the value is to be read

```

158 \def\fc@read@thousand#1#2{%
159   {%
160     \fc@read@unit{\count0}{#2}%
161     \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%
162     \count2=#2%
163     \advance\count2 by 1 %
164     \expandafter\@tempa{\the\count2}%
165     \multiply\count1 by 10 %
166     \advance\count1 by \count0 %
167     \def\@tempa##1{\def\@tempb{#1=##1\relax}}
168     \expandafter\@tempa\expandafter{\the\count1}%
169     \expandafter
170   }\@tempb
171 }

```

\fc@read@thousand

Note: one myriad is ten thousand. Macro `\fc@read@myriad` is used to read a quatuor of digits and form an integer in the range [0..9999]. First we check that the macro is not yet defined.

```

172 \ifcsundef{fc@read@myriad}{-}{%
173   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
174     'fc@read@myriad'}}

```

Arguments as follows — same interface as `\fc@read@unit`:

- #1 output counter: into which the read value is placed
- #2 input number: unit weight at which reach the value is to be read

```

175 \def\fc@read@myriad#1#2{%
176   {%
177     \fc@read@hundred{\count0}{#2}%
178     \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%
179     \count2=#2
180     \advance\count2 by 2
181     \expandafter\@tempa{\the\count2}%
182     \multiply\count1 by 100 %
183     \advance\count1 by \count0 %
184     \def\@tempa##1{\def\@tempb{#1=##1\relax}}%
185     \expandafter\@tempa\expandafter{\the\count1}%
186     \expandafter
187   }\@tempb
188 }

```

\fc@check@nonzeros

Macro `\fc@check@nonzeros` is used to check whether the number represented by digits `\fc@digit@<n>`, with  $n$  in some interval, is zero, one, or more than one. First we check that the macro is not yet defined.

```

189 \ifcsundef{fc@check@nonzeros}{-}{%
190   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
191     'fc@check@nonzeros'}}

```

Arguments as follows:

- #1 input number: minimum unit weight at which start to search the non-zeros
- #2 input number: maximum unit weight at which end to search the non-zeros
- #3 output macro: let  $n$  be the number represented by digits the weight of which span from #1 to #2, then #3 is set to the number  $\min(n,9)$ .

Actually `\fc@check@nonzeros` is just a wrapper to collect arguments, and the real job is delegated to `\fc@@check@nonzeros@inner` which is called inside a group.

```
192 \def\fc@check@nonzeros#1#2#3{%
193   {%
```

So first we save inputs into local macros used by `\fc@@check@nonzeros@inner` as input arguments

```
194   \edef\@tempa{\number#1}%
195   \edef\@tempb{\number#2}%
196   \count0=\@tempa
197   \count1=\@tempb\relax
```

Then we do the real job

```
198   \fc@@check@nonzeros@inner
```

And finally, we propagate the output after end of group — i.e. closing brace.

```
199   \def\@tempd##1{\def\@tempa{\def#3{##1}}}%
200   \expandafter\@tempd\expandafter{\@tempc}%
201   \expandafter
202 } \@tempa
203 }
```

**Macro** `\fc@@check@nonzeros@inner` Check whether some part of the parsed value contains some non-zero digit At the call of this macro we expect that:

- `\@tempa` input/output macro:
  - input* minimum unit weight at which start to search the non-zeros
  - output* macro may have been redefined
- `\@tempb` input/output macro:
  - input* maximum unit weight at which end to search the non-zeros
  - output* macro may have been redefined
- `\@tempc` output macro: 0 if all-zeros, 1 if at least one zero is found
- `\count0` output counter: weight + 1 of the first found non zero starting from minimum weight.

```
204 \def\fc@@check@nonzeros@inner{%
205   \ifnum\count0<\fc@min@weight
206     \count0=\fc@min@weight\relax
207   \fi
208   \ifnum\count1>\fc@max@weight\relax
209     \count1=\fc@max@weight
210   \fi
211   \count2\count0 %
212   \advance\count2 by 1 %
213   \ifnum\count0>\count1 %
214     \PackageError{fcnumparser}{Unexpected arguments}{Number 2 of macro
215       'fc@check@nonzeros' must be at least equal to number in argument 1}%
216   \else
```

```

217 \fc@@check@nonzeros@inner@loopbody
218 \ifnum \@tempc>0 %
219 \ifnum \@tempc<9 %
220 \ifnum \count0>\count1 %
221 \else
222 \let \@tempd \@tempc
223 \fc@@check@nonzeros@inner@loopbody
224 \ifnum \@tempc=0 %
225 \let \@tempc \@tempd
226 \else
227 \def \@tempc{9}%
228 \fi
229 \fi
230 \fi
231 \fi
232 \fi
233 }
234 \def \fc@@check@nonzeros@inner@loopbody{%
235 % \@tempc <- digit of weight \count0
236 \expandafter\let\expandafter\@tempc\csname fc@digit@the\count0\endcsname
237 \advance\count0 by 1 %
238 \ifnum \@tempc=0 %
239 \ifnum \count0>\count1 %
240 \let \next \relax
241 \else
242 \let \next \fc@@check@nonzeros@inner@loopbody
243 \fi
244 \else
245 \ifnum \count0>\count2 %
246 \def \@tempc{9}%
247 \fi
248 \let \next \relax
249 \fi
250 \next
251 }

```

`\fc@intpart@find@last` Macro `\fc@intpart@find@last` find the rightmost non zero digit in the integer part. First check that the macro is not yet defined.

```

252 \ifcsundef{fc@intpart@find@last}{-}{%
253 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
254 'fc@intpart@find@last'}}

```

When macro is called, the number of interest is already parsed, that is to say each digit of weight  $w$  is stored in macro `\fc@digit@<w>`. Macro `\fc@intpart@find@last` takes one single argument which is a counter to set to the result.

```

255 \def \fc@intpart@find@last#1{%
256 {%

```

Counter `\count0` will hold the result. So we will loop on `\count0`, starting from  $\min\{u, w_{\min}\}$ , where  $u \triangleq \text{\fc@unit@weight}$ , and  $w_{\min} \triangleq \text{\fc@min@weight}$ . So first set `\count0` to

```

min{u, w_min}:
257   \count0=\fc@unit@weight\space
258   \ifnum\count0<\fc@min@weight\space
259     \count0=\fc@min@weight\space
260   \fi

```

Now the loop. This is done by defining macro \@templ for final recursion.

```

261   \def\@templ{%
262     \ifnum\csname fc@digit@the\count0\endcsname=0 %
263       \advance\count0 by 1 %
264       \ifnum\count0>\fc@max@weight\space
265         \let\next\relax
266       \fi
267     \else
268       \let\next\relax
269     \fi
270   \next
271 }%
272 \let\next\@templ
273 \@templ

```

Now propagate result after closing bracket into counter #1.

```

274   \toks0{#1}%
275   \edef\@tempa{\the\toks0=\the\count0}%
276   \expandafter
277   }\@tempa\space
278 }

```

c@get@last@word Getting last word. Arguments as follows:

- #1 input: full sequence
- #2 output macro 1: all sequence without last word
- #3 output macro 2: last word

```

279 \ifcsundef{fc@get@last@word}{\PackageError{fcnumparser}{Duplicate definition}{Redefinition
280 of macro 'fc@get@last@word'}}%
281 \def\fc@get@last@word#1#2#3{%
282   {%

```

First we split #1 into two parts: everything that is upto \fc@wcase exclusive goes to \toks0, and evrything from \fc@wcase exclusive upto the final \@nil exclusive goes to \toks1.

```

283   \def\@tempa##1\fc@wcase##2\@nil\fc@end{%
284     \toks0{##1}%

```

Actually a dummy \fc@wcase is appended to \toks1, because that makes easier further checking that it does not contains any other \fc@wcase.

```

285     \toks1{##2\fc@wcase}%
286   }%
287   \@tempa#1\fc@end

```

Now leading part upto last word should be in \toks0, and last word should be in \toks1. However we need to check that this is really the last word, i.e. we need to check that there is no \fc@wcase inside \toks1 other than the tailing dummy one. To that purpose we will

loop while we find that `\toks1` contains some `\fc@wcase`. First we define `\@tempa` to split `\the\toks1` between parts before and after some potential `\fc@wcase`.

```
288 \def\@tempa##1\fc@wcase##2\fc@end{%
289 \toks2{##1}%
290 \def\@tempb{##2}%
291 \toks3{##2}%
292 }%
```

`\@tempt` is just an aliases of `\toks0` to make its handling easier later on.

```
293 \toksdef\@tempt0 %
```

Now the loop itself, this is done by terminal recursion with macro `\@templ`.

```
294 \def\@templ{%
295 \expandafter\@tempa\the\toks1 \fc@end
296 \ifx\@tempb\@empty
```

`\@tempb` empty means that the only `\fc@wcase` found in `\the\toks1` is the dummy one. So we end the loop here, `\toks2` contains the last word.

```
297 \let\next\relax
298 \else
```

`\@tempb` is not empty, first we use

```
299 \expandafter\expandafter\expandafter\@tempt
300 \expandafter\expandafter\expandafter{%
301 \expandafter\the\expandafter\@tempt
302 \expandafter\fc@wcase\the\toks2}%
303 \toks1\toks3 %
304 \fi
305 \next
306 }%
307 \let\next\@templ
308 \@templ
309 \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks2}}%
310 \expandafter
311 }\@tempa
312 }
```

`\fc@get@last@word` Getting last letter. Arguments as follows:

- #1 input: full word
- #2 output macro 1: all word without last letter
- #3 output macro 2: last letter

```
313 \ifcsundef\fc@get@last@letter}{-}{\PackageError{fcnumparser}{Duplicate definition}{Redefinition
314 of macro 'fc@get@last@letter'}}%
315 \def\fc@get@last@letter#1#2#3{%
316 {%
```

First copy input to local `\toks1`. What we are going to do is to bubble one by one letters from `\toks1` which initial contains the whole word, into `\toks0`. At the end of the macro `\toks0` will therefore contain the whole work but the last letter, and the last letter will be in `\toks1`.

```
317 \toks1{#1}%
318 \toks0{}%
319 \toksdef\@tempt0 %
```

We define \@tempa in order to pop the first letter from the remaining of word.

```
320 \def\@tempa##1##2\fc@nil{%
321   \toks2{##1}%
322   \toks3{##2}%
323   \def\@tempb{##2}%
324   }%
```

Now we define \@templ to do the loop by terminal recursion.

```
325 \def\@templ{%
326   \expandafter\@tempa\the\toks1 \fc@nil
327   \ifx\@tempb\@empty
```

Stop loop, as \toks1 has been detected to be one single letter.

```
328   \let\next\relax
329   \else
```

Here we append to \toks0 the content of \toks2, i.e. the next letter.

```
330   \expandafter\expandafter\expandafter\@tempt
331   \expandafter\expandafter\expandafter{%
332   \expandafter\the\expandafter\@tempt
333   \the\toks2}%
```

And the remaining letters go to \toks1 for the next iteration.

```
334   \toks1\toks3 %
335   \fi
336   \next
337   }%
```

Here run the loop.

```
338   \let\next\@templ
339   \next
```

Now propagate the results into macros #2 and #3 after closing brace.

```
340 \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks1}}%
341 \expandafter
342 }\@tempa
343 }%
```

### 10.3 fcprefix.sty

Pseudo-latin prefixes.

```
1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{fcprefix}[2012/09/28]
3 \RequirePackage{ifthen}
4 \RequirePackage{keyval}
5 \RequirePackage{fcnumparser}
```

Option ‘use duode and unde’ is to select whether 18 and suchlikes ( $\langle x \rangle_8$ ,  $\langle x \rangle_9$ ) writes like duodevicies, or like octodecies. For French it should be ‘below 20’. Possible values are ‘below 20’ and ‘never’.

```
6 \define@key{fcprefix}{use duode and unde}[below20]{%
7   \ifthenelse{\equal{#1}{below20}}{%
8     \def\fc@duodeandunde{2}%
9   }{%
```

```

10 \ifthenelse{\equal{#1}{never}}{%
11   \def\fc@duodeandunde{0}%
12 }{%
13   \PackageError{fcprefix}{Unexpected option}{%
14     Option 'use duode and unde' expects 'below 20' or 'never' }%
15 }%
16 }%
17 }

```

Default is 'below 20' like in French.

```
18 \def\fc@duodeandunde{2}
```

Option 'numeral u in duo', this can be 'true' or 'false' and is used to select whether 12 and suchlikes write like dodec(*xxx*) or duodec(*xxx*) for numerals.

```

19 \define@key{fcprefix}{numeral u in duo}[false]{%
20   \ifthenelse{\equal{#1}{false}}{%
21     \let\fc@u@in@duo\@empty
22   }{%
23     \ifthenelse{\equal{#1}{true}}{%
24       \def\fc@u@in@duo{u}%
25     }{%
26       \PackageError{fcprefix}{Unexpected option}{%
27         Option 'numeral u in duo' expects 'true' or 'false' }%
28     }%
29   }%
30 }

```

Option 'e accute', this can be 'true' or 'false' and is used to select whether letter 'e' has an accute accent when it pronounce [e] in French.

```

31 \define@key{fcprefix}{e accute}[false]{%
32   \ifthenelse{\equal{#1}{false}}{%
33     \let\fc@prefix@eaccute\@firstofone
34   }{%
35     \ifthenelse{\equal{#1}{true}}{%
36       \let\fc@prefix@eaccute\'%
37     }{%
38       \PackageError{fcprefix}{Unexpected option}{%
39         Option 'e accute' expects 'true' or 'false' }%
40     }%
41   }%
42 }

```

Default is to set accute accent like in French.

```
43 \let\fc@prefix@eaccute\'%
```

Option 'power of millia' tells how millia is raise to power n. It expects value:

recursive for which millia squared is noted as 'milliamillia'  
arabic for which millia squared is noted as 'millia<sup>2</sup>'  
prefix for which millia squared is noted as 'bismillia'

```

44 \define@key{fcprefix}{power of millia}[prefix]{%
45   \ifthenelse{\equal{#1}{prefix}}{%

```



```

46     \let\fc@power@of@millia@init@gobbletwo
47     \let\fc@power@of@millia\fc@@prefix@millia
48 }{%
49   \ifthenelse{\equal{#1}{arabic}}{%
50     \let\fc@power@of@millia@init@gobbletwo
51     \let\fc@power@of@millia\fc@@arabic@millia
52   }{%
53     \ifthenelse{\equal{#1}{recursive}}{%
54       \let\fc@power@of@millia@init\fc@@recurse@millia@init
55       \let\fc@power@of@millia\fc@@recurse@millia
56     }{%
57       \PackageError{fc@prefix}{Unexpected option}{%
58         Option ‘power of millia’ expects ‘recursive’, ‘arabic’, or ‘prefix’ }%
59     }%
60   }%
61 }%
62 }

```

Arguments as follows:

- #1 output macro
- #2 number with current weight  $w$

```

63 \def\fc@@recurse@millia#1#2{%
64   \let\@temp#1%
65   \edef#1{millia\@temp}%
66 }

```

Arguments as follows — same interface as `\fc@@recurse@millia`:

- #1 output macro
- #2 number with current weight  $w$

```

67 \def\fc@@recurse@millia@init#1#2{%
68   {%

```

Save input argument current weight  $w$  into local macro `\@tempb`.

```

69   \edef\@tempb{\number#2}%

```

Now main loop from 0 to  $w$ . Final value of `\@tempa` will be the result.

```

70   \count0=0 %
71   \let\@tempa\@empty
72   \loop
73     \ifnum\count0<\@tempb
74     \advance\count0 by 1 %
75     \expandafter\def
76     \expandafter\@tempa\expandafter{\@tempa millia}%
77   \repeat

```

Now propagate the expansion of `\@tempa` into #1 after closing base.

```

78   \edef\@tempb{\def\noexpand#1{\@tempa}}%
79   \expandafter
80 } \@tempb
81 }

```

Arguments as follows — same interface as `\fc@@recurse@millia`:

```
#1 output macro
#2 number with current weight  $w$ 
82 \def\fc@@arabic@millia#1#2{%
83 \ifnum#2=0 %
84 \let#1\@empty
85 \else
86 \edef#1{millia\^{\the#2}}%
87 \fi
88 }
```

Arguments as follows — same interface as `\fc@@recurse@millia`:

```
#1 output macro
#2 number with current weight  $w$ 
89 \def\fc@@prefix@millia#1#2{%
90 \fc@@latin@numeral@pefix{#2}{#1}%
91 }
```

Default value of option ‘power of millia’ is ‘prefix’:

```
92 \let\fc@power@of@millia@init\@gobbletwo
93 \let\fc@power@of@millia\fc@@prefix@millia
```

`\fc@@latin@cardinal@pefix` Compute a cardinal prefix for n-illion, like  $1 \Rightarrow$  ‘m’,  $2 \Rightarrow$  ‘bi’,  $3 \Rightarrow$  ‘tri’. The algorithm to derive this prefix is that of Russ Rowlett I founds its documentation on Alain Lassine’s site: [http://www.alain.be/Boece/grands\\_nombres.html](http://www.alain.be/Boece/grands_nombres.html). First check that macro is not yet defined.

```
94 \ifcsundef{fc@@latin@cardinal@pefix}{}%
95 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro ‘fc@@latin@cardinal@pefix
```

Arguments as follows:

```
#1 input number to be formatted
#2 outut macro name into which to place the formatted result
96 \def\fc@@latin@cardinal@pefix#1#2{%
97 {%
```

First we put input argument into local macro `@cs@tempa` with full expansion.

```
98 \edef\@tempa{\number#1}%
```

Now parse number from expanded input.

```
99 \expandafter\fc@number@parser\expandafter{\@tempa}%
100 \count2=0 %
```

`\@tempt` will hold the optional final t, `\@tempu` is used to initialize `\@tempt` to ‘t’ when the first non-zero 3digit group is met, which is the job made by `\@tempu`.

```
101 \let\@tempt\@empty
102 \def\@tempu{t}%
```

`\@tempm` will hold the millia <sup>$n \div 3$</sup>

```
103 \let\@tempm\@empty
```

Loop by means of terminal recursion of herinafter defined macro `\@templ`. We loop by group of 3 digits.

```

104   \def\@templ{%
105     \ifnum\count2>\fc@max@weight
106       \let\next\relax
107     \else

```

Loop body. Here we read a group of 3 consecutive digits  $d_2d_1d_0$  and place them respectively into `\count3`, `\count4`, and `\count5`.

```

108     \fc@read@unit{\count3}{\count2}%
109     \advance\count2 by 1 %
110     \fc@read@unit{\count4}{\count2}%
111     \advance\count2 by 1 %
112     \fc@read@unit{\count5}{\count2}%
113     \advance\count2 by 1 %

```

If the 3 considered digits  $d_2d_1d_0$  are not all zero, then set `\@tempt` to ‘t’ for the first time this event is met.

```

114     \edef\@tempn{%
115       \ifnum\count3=0\else 1\fi
116       \ifnum\count4=0\else 1\fi
117       \ifnum\count5=0\else 1\fi
118     }%
119     \ifx\@tempn\@empty\else
120       \let\@tempt\@tempu
121       \let\@tempu\@empty
122     \fi

```

Now process the current group  $d_2d_1d_0$  of 3 digits.

```

123     \let\@temp\@tempa
124     \edef\@tempa{%

```

Here we process  $d_2$  held by `\count5`, that is to say hundreds.

```

125       \ifcase\count5 %
126       \or cen%
127       \or ducen%
128       \or trecen%
129       \or quadringen%
130       \or quingen%
131       \or sescen%
132       \or septigen%
133       \or octingen%
134       \or nongen%
135     \fi

```

Here we process  $d_1d_0$  held by `\count4` & `\count3`, that is to say tens and units.

```

136       \ifnum\count4=0 %
137       % x0(0..9)
138       \ifnum\count2=3 %
139         % Absolute weight zero
140         \ifcase\count3 \@tempt
141         \or m%
142         \or b%

```

```

143         \or tr%
144         \or quadr%
145         \or quin\@tempt
146         \or sex\@tempt
147         \or sep\@tempt
148         \or oc\@tempt
149         \or non%
150         \fi
151     \else

```

Here the weight of `\count3` is  $3 \times n$ , with  $n > 0$ , i.e. this is followed by a  $\text{millia}^n$ .

```

152         \ifcase\count3 %
153         \or \ifnum\count2>\fc@max@weight\else un\fi
154         \or d\fc@u@in@duo o%
155         \or tre%
156         \or quattuor%
157         \or quin%
158         \or sex%
159         \or septen%
160         \or octo%
161         \or novem%
162         \fi
163     \fi
164 \else
165     % x(10..99)
166     \ifcase\count3 %
167     \or un%
168     \or d\fc@u@in@duo o%
169     \or tre%
170     \or quattuor%
171     \or quin%
172     \or sex%
173     \or septen%
174     \or octo%
175     \or novem%
176     \fi
177     \ifcase\count4 %
178     \or dec%
179     \or vigin\@tempt
180     \or trigin\@tempt
181     \or quadragin\@tempt
182     \or quinquagin\@tempt
183     \or sexagin\@tempt
184     \or septuagin\@tempt
185     \or octogin\@tempt
186     \or nonagin\@tempt
187     \fi
188 \fi

```

Insert the  $\text{millia}^{(n+3)}$  only if  $d_2 d_1 d_0 \neq 0$ , i.e. if one of `\count3` `\count4` or `\count5` is non

zero.

```
189     \@tempm
```

And append previous version of \@tempa.

```
190     \@temp
```

```
191     }%
```

“Concatenate” millia to \@tempm, so that \@tempm will expand to  $\text{millia}^{(n+3)+1}$  at the next iteration. Actually whether this is a concatenation or some millia prefixing depends of option ‘power of millia’.

```
192     \fc@power@of@millia\@tempm{\count2}%
```

```
193     \fi
```

```
194     \next
```

```
195     }%
```

```
196     \let\@tempa\@empty
```

```
197     \let\next\@templ
```

```
198     \@templ
```

Propagate expansion of \@tempa into #2 after closing bracket.

```
199     \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
```

```
200     \expandafter\@tempb\expandafter{\@tempa}%
```

```
201     \expandafter
```

```
202     }\@tempa
```

```
203 }
```

`\@latin@numeral@prefix` Compute a numeral prefix like ‘sémel’, ‘bis’, ‘ter’, ‘quater’, etc... I found the algorithm to derive this prefix on Alain Lassine’s site: [http://www.alain.be/Boece/nombres\\_gargantuesques.html](http://www.alain.be/Boece/nombres_gargantuesques.html). First check that the macro is not yet defined.

```
204 \ifcsundef{fc@latin@numeral@prefix}{\}{%
```

```
205 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
```

```
206 ‘fc@latin@numeral@prefix’}}
```

Arguments as follows:

#1 input number to be formatted,

#2 outut macro name into which to place the result

```
207 \def\fc@latin@numeral@prefix#1#2{%
```

```
208 {%
```

```
209     \edef\@tempa{\number#1}%
```

```
210     \def\fc@unit@weight{0}%
```

```
211     \expandafter\fc@number@parser\expandafter{\@tempa}%
```

```
212     \count2=0 %
```

Macro \@tempm will hold the millies<sup>n+3</sup>.

```
213     \let\@tempm\@empty
```

Loop over digits. This is done by defining macro \@templ for terminal recursion.

```
214     \def\@templ{%
```

```
215         \ifnum\count2>\fc@max@weight
```

```
216             \let\next\relax
```

```
217         \else
```

Loop body. Three consecutive digits  $d_2d_1d_0$  are read into counters `\count3`, `\count4`, and `\count5`.

```
218     \fc@read@unit{\count3}{\count2}%
219     \advance\count2 by 1 %
220     \fc@read@unit{\count4}{\count2}%
221     \advance\count2 by 1 %
222     \fc@read@unit{\count5}{\count2}%
223     \advance\count2 by 1 %
```

Check the use of duodevicies instead of octodecies.

```
224     \let\@tempn\@secondoftwo
225     \ifnum\count3>7 %
226         \ifnum\count4<\fc@duodeandunde
227             \ifnum\count4>0 %
228                 \let\@tempn\@firstoftwo
229             \fi
230         \fi
231     \fi
232     \@tempn
233     {% use duodevicies for eighteen
234         \advance\count4 by 1 %
235         \let\@temps\@secondoftwo
236     }{% do not use duodevicies for eighteen
237         \let\@temps\@firstoftwo
238     }%
239     \let\@tempp\@tempa
240     \edef\@tempa{%
241         % hundreds
242         \ifcase\count5 %
243         \expandafter\@gobble
244         \or c%
245         \or duc%
246         \or trec%
247         \or quadring%
248         \or quing%
249         \or sesc%
250         \or septing%
251         \or octing%
252         \or nong%
253         \fi
254         {enties}%
255         \ifnum\count4=0 %
```

Here  $d_2d_1d_0$  is such that  $d_1 = 0$ .

```
256         \ifcase\count3 %
257         \or
258         \ifnum\count2=3 %
259             s\fc@prefix@eacute emel%
260         \else
261             \ifnum\count2>\fc@max@weight\else un\fi
```

```

262         \fi
263         \or bis%
264         \or ter%
265         \or quater%
266         \or quinquies%
267         \or sexies%
268         \or septies%
269         \or octies%
270         \or novies%
271         \fi
272     \else
Here  $d_2d_1d_0$  is such that  $d_1 \geq 1$ .
273         \ifcase\count3 %
274         \or un%
275         \or d\fc@u@in@duo o%
276         \or ter%
277         \or quater%
278         \or quin%
279         \or sex%
280         \or septen%
281         \or \@temps{octo}\{duod\fc@prefix@eaccute e}\% x8 = two before next (x+1)0
282         \or \@temps{novem}\{und\fc@prefix@eaccute e}\% x9 = one before next (x+1)0
283         \fi
284         \ifcase\count4 %
285         % can't get here
286         \or d\fc@prefix@eaccute ec%
287         \or vic%
288         \or tric%
289         \or quadrag%
290         \or quinquag%
291         \or sexag%
292         \or septuag%
293         \or octog%
294         \or nonag%
295         \fi
296         ies%
297     \fi
298     % Insert the millies(n/3) only if one of \count3 \count4 \count5 is non zero
299     \@tempm
300     % add up previous version of \@tempa
301     \@temppp
302 }%
Concatenate millies to \@tempm so that it is equal to milliesn+3 at the next iteration. Here
we just have plain concatenation, contrary to cardinal for which a prefix can be used instead.
303     \let\@temppp\@tempm
304     \edef\@tempm{millies\@temppp}%
305 \fi
306 \next

```

```

307 }%
308 \let\@tempa\@empty
309 \let\next\@templ
310 \@templ

```

Now propagate expansion of tempa into #2 after closing bracket.

```

311 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
312 \expandafter\@tempb\expandafter{\@tempa}%
313 \expandafter
314 }\@tempa
315 }

```

Stuff for calling macros. Construct `\fc@call<some macro>` can be used to pass two arguments to `<some macro>` with a configurable calling convention:

- the calling convention is such that there is one mandatory argument `< marg >` and an optional argument `< oarg >`
- either `\fc@call` is `\let` to be equal to `\fc@call@opt@arg@second`, and then calling convention is that the `< marg >` is first and `< oarg >` is second,
- or `\fc@call` is `\let` to be equal to `\fc@call@opt@arg@first`, and then calling convention is that the `< oarg >` is first and `< aarg >` is second,
- if `< oarg >` is absent, then it is by convention set empty,
- `< some macro >` is supposed to have two mandatory arguments of which `< oarg >` is passed to the first, and `< marg >` is passed to the second, and
- `< some macro >` is called within a group.

```

316 \def\fc@call@opt@arg@second#1#2{%
317 \def\@tempb{%
318 \ifx[\@tempa
319 \def\@tempc[####1]{%
320 {#1{####1}{#2}}%
321 }%
322 \else
323 \def\@tempc{#1}{#2}}%
324 \fi
325 \@tempc
326 }%
327 \futurelet\@tempa
328 \@tempb
329 }

330 \def\fc@call@opt@arg@first#1{%
331 \def\@tempb{%
332 \ifx[\@tempa
333 \def\@tempc[####1]####2{#1{####1}{####2}}%
334 \else

```



```

335     \def\@tempc####1{#{1}{####1}}%
336     \fi
337     \@tempc
338 }%
339 \futurelet\@tempa
340 \@tempb
341 }
342
343 \let\fc@call\fc@call@opt@arg@first

```

User API.

`\latinnumeralstringnum` Macro `\@latinnumeralstringnum`. Arguments as follows:

- #1 local options
- #2 input number

```

344 \newcommand*{\@latinnumeralstringnum}[2]{%
345   \setkeys{fcprefix}{#1}%
346   \fc@latin@numeral@prefix{#2}\@tempa
347   \@tempa
348 }

```

Arguments as follows:

- #1 local options
- #2 input counter

```

349 \newcommand*{\@latinnumeralstring}[2]{%
350   \setkeys{fcprefix}{#1}%
351   \expandafter\let\expandafter
352     \@tempa\expandafter\csname c@#2\endcsname
353   \expandafter\fc@latin@numeral@prefix\expandafter{\the\@tempa}\@tempa
354   \@tempa
355 }

```

```

356 \newcommand*{\latinnumeralstring}{%
357   \fc@call\@latinnumeralstring
358 }

```

```

359 \newcommand*{\latinnumeralstringnum}{%
360   \fc@call\@latinnumeralstringnum
361 }

```

## 10.4 `fmtcount.sty`

This section deals with the code for `fmtcount.sty`.

```

1 \NeedsTeXFormat{LaTeX2e}
2 \ProvidesPackage{fmtcount}[2025/12/02 v3.12 Displaying the values of LaTeX counters (NT,VB,NE)]
3 \RequirePackage{ifthen}

4 \RequirePackage{xkeyval}
5 \RequirePackage{etoolbox}
6 \RequirePackage{fcprefix}

```

Need to use `\new@ifnextchar` instead of `\@ifnextchar` in commands that have a final optional argument (such as `\gls`) so require `amsgen`.

```
7 \RequirePackage{amsgen}
```

These commands need to be defined before the configuration file is loaded.

Define the macro to format the `st`, `nd`, `rd` or `th` of an ordinal.

```
\fc@orddef@ult
```

```
8 \providecommand*\fc@orddef@ult}[1]{\fc@textsuperscript{#1}}
```

```
c@ord@multiling
```

```
9 \providecommand*\fc@ord@multiling}[1]{%
10 \ifcsundef{fc@\language@name @alias@of}{%
```

Not a supported language, just use the default setting:

```
11 \fc@orddef@ult{#1}}{%
12 \expandafter\let\expandafter\@tempa\csname fc@\language@name @alias@of\endcsname
13 \ifcsundef{fc@ord@\@tempa}{%
```

Not language specific setting, just use the default setting:

```
14 \fc@orddef@ult{#1}}{%
```

Language with specific setting, use that setting:

```
15 \csname fc@ord@\@tempa\endcsname{#1}}}}
```

```
\padzeroes
```

```
\padzeroes [⟨n⟩]
```

Specifies how many digits should be displayed for commands such as `\decimal` and `\binary`.

```
16 \newcount\c@padzeroesN
17 \c@padzeroesN=1\relax
18 \providecommand*\padzeroes}[1][17]{\c@padzeroesN=#1}
```

```
\FCloadlang
```

```
\FCloadlang{⟨language⟩}
```

Load `fmtcount` language file, `fc-⟨language⟩.def`, unless already loaded. Unfortunately neither `babel` nor `polyglossia` keep a list of loaded dialects, so we can't load all the necessary `def` files in the preamble as we don't know which dialects the user requires. Therefore the dialect definitions get loaded when a command such as `\ordinalnum` is used, if they haven't already been loaded.

```
19 \newcount\fc@tmpcatcode
20 \def\fc@languages{}%
21 \def\fc@mainlang{}%
22 \newcommand*\FCloadlang}[1]{%
23 \@FC@iflangloaded{#1}{}%
24 {%
25 \fc@tmpcatcode=\catcode'\@ \relax
26 \catcode '\@ 11 \relax
```

```

27 \InputIfFileExists{fc-#1.def}%
28 {%
29 \ifdefempty{\fc@languages}%
30 {%
31 \gdef\fc@languages{#1}%
32 }%
33 {%
34 \gappto\fc@languages{,#1}%
35 }%
36 \gdef\fc@mainlang{#1}%
37 }%
38 {}%
39 \catcode '@ \fc@tmpcatcode\relax
40 }%
41 }

```

`\FC@iflangloaded`

```
\FC@iflangloaded{<language>}{<true>}{<false>}
```

If `fntcount` language definition file `fc-<language>.def` has been loaded, do `<true>` otherwise do `<false>`

```

42 \newcommand{\FC@iflangloaded}[3]{%
43 \ifcsundef{ver@fc-#1.def}{#3}{#2}%
44 }

```

`\ProvidesFCLanguage`

Declare `fntcount` language definition file. Adapted from `\ProvidesFile`

```

45 \newcommand*{\ProvidesFCLanguage}[1]{%
46 \ProvidesFile{fc-#1.def}%
47 }

```

We need that flag to remember that a language has been loaded via package option, so that in the end we can set `fntcount` in multiling

```

48 \newif\iffntcount@language@option
49 \fntcount@language@optionfalse

```

`\d@language@list`

Declare list of supported languages, as a comma separated list. No space, no empty items. Each item is a language for which `fntcount` is able to load language specific definitions. Aliases but be *after* their meaning, for instance ‘american’ being an alias of ‘USenglish’, it has to appear after it in the list. The raison d’être of this list is to commonalize iteration on languages for the two following purposes:

- loading language definition as a result of the language being used by `babel/polyglossia`
- loading language definition as a result of package option

These two purposes cannot be handled in the same pass, we need two different passes otherwise there would be some corner cases when a package would be required — as a result of loading language definition for one language — between a `\DeclareOption` and a `\ProcessOption` which is forbidden by  $\LaTeX 2\mathcal{E}$ .

```

50 \newcommand*\fc@supported@language@list{%
51 english,%
52 UKenglish,%
53 brazilian,%
54 british,%
55 USenglish,%
56 american,%
57 spanish,%
58 portuges,%
59 portuguese,%
60 french,%
61 frenchb,%
62 francais,%
63 german,%
64 germanb,%
65 ngerman,%
66 ngermanb,%
67 italian,%
68 dutch}

```

te@on@languages

```
\fc@iterate@on@languages{<body>}
```

Now make some language iterator, note that for the following to work properly `\fc@supported@language@list` must not be empty. `<body>` is a macro that takes one argument, and `\fc@iterate@on@languages` applies it iteratively :

```

69 \newcommand*\fc@iterate@on@languages [1]{%
70 \ifx\fc@supported@language@list\@empty

```

That case should never happen !

```

71   \PackageError{fmtcount}{Macro ‘\protect\fc@iterate@on@languages’ is empty}{You should never
72     Something is broken within \texttt{fmtcount}, please report the issue on
73     \texttt{https://github.com/search?q=fmtcount\&ref=cmdform\&type=Issues}}%
74 \else
75   \let\fc@iterate@on@languages@body#1
76   \expandafter\fc@iterate@on@languages\fc@supported@language@list,\@nil,%
77 \fi
78 }
79 \def\fc@iterate@on@languages#1,{%
80   {%
81     \def\@tempa{#1}%
82     \ifx\@tempa\@nnil
83       \let\@tempa\@empty
84     \else
85       \def\@tempa{%
86         \fc@iterate@on@languages@body{#1}%
87         \@fc@iterate@on@languages
88       }%
89     \fi

```

```

90     \expandafter
91   } \@tempa
92 }%

```

polyglossialdf

```
\@fc@loadifbabelorpolyglossialdf{<language>}
```

Loads fmtcount language file, `fc-<language>.def`, if one of the following condition is met:

- babel language definition file `<language>.ldf` has been loaded — conditionally to compilation with `latex`, not `xelatex`.
- polyglossia language definition file `gloss-<language>.ldf` has been loaded — conditionally to compilation with `xelatex`, not `latex`.
- `<language>` option has been passed to package `fmtcount`.

```

93 \newcommand*\@fc@loadifbabelldf [1]{\ifcsundef{ver@#1.ldf}{}\FCloadlang{#1}}
94 \newcommand*\@fc@loadifbabelorpolyglossialdf [1]{
95   \ifpackageloaded{polyglossia}{%
96     \def\@fc@loadifbabelorpolyglossialdf#1{\IfFileExists{gloss-#1.ldf}{\iflanguageloaded{#1}{\FCloadlang{#1}}{\FCloadlang{#1}}}
97     \@fc@loadifbabelldf{#1}%
98   }%
99 }{\ifpackageloaded{babel}{%
100   \let\@fc@loadifbabelorpolyglossialdf\@fc@loadifbabelldf
101 }{}}

```

Load appropriate language definition files:

```
102 \fc@iterate@on@languages\@fc@loadifbabelorpolyglossialdf
```

By default all languages are unique — i.e. aliases not yet defined.

```

103 \def\fc@iterate@on@languages@body#1{%
104   \expandafter\def\csname fc@#1@alias@of\endcsname{#1}}
105 \expandafter\@fc@iterate@on@languages\fc@supported@language@list,\@nil,%

```

Now define those languages that are aliases of another language. This is done with: `\@tempa{<alias>}{<language>}`

```

106 \def\@tempa#1#2{%
107   \expandafter\def\csname fc@#1@alias@of\endcsname{#2}%
108 }%
109 \@tempa{frenchb}{french}
110 \@tempa{français}{french}
111 \@tempa{germanb}{german}
112 \@tempa{ngermanb}{german}
113 \@tempa{ngerman}{german}
114 \@tempa{british}{english}
115 \@tempa{american}{USenglish}

```

Now, thanks to the aliases, we are going to define one option for each language, so that each language can have its own settings.

```

116 \def\fc@iterate@on@languages@body#1{%
117   \define@key{fmtcount}{#1}[]{%
118     \@FC@iflangloaded{#1}%
119     {%
120       \setkeys{fc\csname fc@#1@alias@of\endcsname}{##1}%
121     }{%
122       \PackageError{fmtcount}%
123       {Language ‘#1’ not defined}%
124       {You need to load \ifxetex polyglossia\else babel\fi\space before loading fmtcount}%
125     }%
126   }%
127   \ifthenelse{\equal{\csname fc@#1@alias@of\endcsname}{#1}}{%
128     \define@key{fc\csname fc@#1@alias@of\endcsname}{fmtord}{%
129       \ifthenelse{\equal{##1}{raise}\or\equal{##1}{level}}{%
130         \expandafter\let\expandafter\@tempa\csname fc@set@ord@as@##1\endcsname
131         \expandafter\@tempa\csname fc@ord@#1\endcsname
132       }{%
133         \ifthenelse{\equal{##1}{undefine}}{%
134           \expandafter\let\csname fc@ord@#1\endcsname\undefined
135         }{%
136           \PackageError{fmtcount}%
137           {Invalid value ‘##1’ to fmtord key}%
138           {Option ‘fmtord’ can only take the values ‘level’, ‘raise’
139            or ‘undefine’}%
140         }%
141       }%
142     }%

```

When the language #1 is an alias, do the same as the language of which it is an alias:

```

143   \expandafter\let\expandafter\@tempa\csname KV@\csname fc@#1@alias@of\endcsname @fmtord\endcsname
144   \expandafter\let\csname KV@#1@fmtord\endcsname\@tempa
145 }%
146 }
147 \expandafter\@fc@iterate@on@languages\fc@supported@language@list,\@nil,%

```

**fmtord** Key to determine how to display the ordinal

```

148 \def\fc@set@ord@as@level#1{%
149   \def#1##1{##1}%
150 }
151 \def\fc@set@ord@as@raise#1{%
152   \let#1\fc@textsuperscript
153 }
154 \define@key{fmtcount}{fmtord}{%
155   \ifthenelse{\equal{#1}{level}
156     \or\equal{#1}{raise}}{%
157     {%
158       \csname fc@set@ord@as@#1\endcsname\fc@orddef@ult
159     }%
160   }%
161   {%

```

```

162   \PackageError{fmtcount}%
163   {Invalid value ‘#1’ to fmtord key}%
164   {Option ‘fmtord’ can only take the values ‘level’ or ‘raise’}%
165 }%
166 }

```

`\iffmtord@abbrv` Key to determine whether the ordinal superscript should be abbreviated (language dependent, currently only affects French ordinals, non-abbreviated French ordinals ending — i.e. ‘ier’ and ‘ième’ — are considered faulty.)

```

167 \newif\iffmtord@abbrv

168 \fmtord@abbrvtrue
169 \define@key{fmtcount}{abbrv}[true]{%
170   \ifthenelse{\equal{#1}{true}\or\equal{#1}{false}}%
171   {%
172     \csname fmtord@abbrv#1\endcsname
173   }%
174   {%
175     \PackageError{fmtcount}%
176     {Invalid value ‘#1’ to fmtord key}%
177     {Option ‘abbrv’ can only take the values ‘true’ or
178     ‘false’}%
179   }%
180 }

```

`prefix`

```

181 \define@key{fmtcount}{prefix}[scale=long]{%
182   \RequirePackage{fmtprefix}%
183   \fmtprefixsetoption{#1}%
184 }

```

`countsetoptions` Define command to set options.

```

185 \def\fmtcountsetoptions{%
186   \def\fmtcount@fmtord{}%
187   \setkeys{fmtcount}}%

```

Load configuration file if it exists. This needs to be done before the package options, to allow the user to override the settings in the configuration file.

```

188 \InputIfFileExists{fmtcount.cfg}%
189 {%
190   \PackageInfo{fmtcount}{Using configuration file fmtcount.cfg}%
191 }%
192 {%
193 }

```

`ption@lang@list`

```

194 \newcommand*{\fmtcount@loaded@by@option@lang@list}{}

```

`\metalinguage` Option *<language>* causes language *<language>* to be registered for loading.

```
195 \newcommand*\fc@declare@language@option[1]{%
196   \DeclareOption{#1}{%
197     \ifx\fmtcount@loaded@by@option@lang@list\@empty
198       \def\fmtcount@loaded@by@option@lang@list{#1}%
199     \else
200       \edef\fmtcount@loaded@by@option@lang@list{\fmtcount@loaded@by@option@lang@list,#1}%
201     \fi
202   }%
203 \fc@iterate@on@languages\fc@declare@language@option
```

level

```
204 \DeclareOption{level}{\def\fmtcount@fmtord{level}%
205   \def\fc@orddef@ult#1{#1}}
```

raise

```
206 \DeclareOption{raise}{\def\fmtcount@fmtord{raise}%
207   \def\fc@orddef@ult#1{\fc@textsuperscript{#1}}}
```

Process package options

```
208 \ProcessOptions\relax
```

Now we do the loading of all languages that have been set by option to be loaded.

```
209 \ifx\fmtcount@loaded@by@option@lang@list\@empty\else
210 \def\fc@iterate@on@languages@body#1{%
211   \@FC@iflangloaded{#1}{-}{%
212     \fmtcount@language@optiontrue
213     \FCloadlang{#1}%
214   }}
215 \expandafter\fc@iterate@on@languages\fmtcount@loaded@by@option@lang@list,\@nil,%
216 \fi
```

```
\@FCmodulo
```

<code>\@FCmodulo{&lt;count reg&gt;}{&lt;n&gt;}</code>
-------------------------------------------------------

Sets the count register to be its value modulo *<n>*. This is used for the date, time, ordinal and numberstring commands. (The `fmtcount` package was originally part of the `datetime` package.)

```
217 \newcount\@DT@modctr
218 \newcommand*\@FCmodulo[2]{%
219   \@DT@modctr=#1\relax
220   \divide \@DT@modctr by #2\relax
221   \multiply \@DT@modctr by #2\relax
222   \advance #1 by -\@DT@modctr
223 }
```

The following registers are needed by `\@ordinal` etc

```
224 \newcount\@ordinalctr
```



```

225 \newcount\@orgargctr
226 \newcount\@strctr
227 \newcount\@tmpstrctr

```

Define commands that display numbers in different bases. Define counters and conditionals needed.

```

228 \newif\if@DT@padzeroes
229 \newcount\@DT@loopN
230 \newcount\@DT@X

```

`\binarynum` Converts a decimal number to binary, and display.

```

231 \newrobustcmd*{\@binary}[1]{%
232   \@DT@padzeroestru
233   \@DT@loopN=17\relax
234   \@strctr=\@DT@loopN
235   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
236   \@strctr=65536\relax
237   \@DT@X=#1\relax
238   \loop
239     \@DT@modctr=\@DT@X
240     \divide\@DT@modctr by \@strctr
241     \ifthenelse{\boolean{@DT@padzeroes}
242       \and \(\@DT@modctr=0\}
243       \and \(\@DT@loopN>\c@padzeroesN\)}%
244     {}%
245     {\the\@DT@modctr}%
246     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
247     \multiply\@DT@modctr by \@strctr
248     \advance\@DT@X by -\@DT@modctr
249     \divide\@strctr by \tw@
250     \advance\@DT@loopN by \m@ne
251     \ifnum\@strctr>\@ne
252     \repeat
253     \the\@DT@X
254 }
255
256 \let\binarynum=\@binary

```

`\octalnum` Converts a decimal number to octal, and displays.

```

257 \newrobustcmd*{\@octal}[1]{%
258   \@DT@X=#1\relax
259   \ifnum\@DT@X>32768
260     \PackageError{fmtcount}%
261     {Value of counter too large for \protect\@octal}
262     {Maximum value 32768}
263   \else
264     \@DT@padzeroestru
265     \@DT@loopN=6\relax
266     \@strctr=\@DT@loopN

```

```

267 \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
268 \@strctr=32768\relax
269 \loop
270   \@DT@modctr=\@DT@X
271   \divide\@DT@modctr by \@strctr
272   \ifthenelse{\boolean{@DT@padzeroes}
273     \and \(\@DT@modctr=0\}
274     \and \(\@DT@loopN>\c@padzeroesN\)}%
275   {\the\@DT@modctr}%
276   \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
277   \multiply\@DT@modctr by \@strctr
278   \advance\@DT@X by -\@DT@modctr
279   \divide\@strctr by \@viiipt
280   \advance\@DT@loopN by \m@ne
281   \ifnum\@strctr>\@ne
282   \repeat
283   \the\@DT@X
284   \fi
285 }
286 \let\octalnum=\@octal

```

`\@@hexadecimal` Converts number from 0 to 15 into lowercase hexadecimal notation.

```

287 \newcommand*{\@@hexadecimal}[1]{%
288   \ifcase#1\or1\or2\or3\or4\or5\or
289   6\or7\or8\or9\or a\or b\or c\or d\or e\or f\fi
290 }

```

`\hexadecimalnum` Converts a decimal number to a lowercase hexadecimal number, and displays it.

```

291 \newrobustcmd*{\hexadecimalnum}{\@hexadecimalengine\@@hexadecimal}

```

`\@@Hexadecimal` Converts number from 0 to 15 into uppercase hexadecimal notation.

```

292 \newcommand*{\@@Hexadecimal}[1]{%
293   \ifcase#1\or1\or2\or3\or4\or5\or6\or
294   7\or8\or9\or A\or B\or C\or D\or E\or F\fi
295 }

```

`\HEXADecimalnum` Uppercase hexadecimal

```

296 \newrobustcmd*{\HEXADecimalnum}{\@hexadecimalengine\@@Hexadecimal}
297 \newcommand*{\@hexadecimalengine}[2]{%
298   \@DT@padzeroestru
299   \@DT@loopN=\@vpt
300   \@strctr=\@DT@loopN
301   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
302   \@strctr=65536\relax
303   \@DT@X=#2\relax
304   \loop
305     \@DT@modctr=\@DT@X
306     \divide\@DT@modctr by \@strctr
307     \ifthenelse{\boolean{@DT@padzeroes}

```

```

308     \and \(\@DT@modctr=0\)
309     \and \(\@DT@loopN>\c@padzeroesN\)}
310     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
311     \multiply\@DT@modctr by \@strctr
312     \advance\@DT@X by -\@DT@modctr
313     \divide\@strctr by 16\relax
314     \advance\@DT@loopN by \m@ne
315     \ifnum\@strctr>\@ne
316     \repeat
317     #1\@DT@X
318 }
319 }
320 \def\Hexadecimalnum{%
321 \PackageWarning{fmtcount}{\string\Hexadecimalnum\space is deprecated, use \string\HEXADecimal
322 instead. The \string\Hexadecimalnum\space control sequence name is confusing as it can misl
323 that only the 1st letter is upper-cased.}%
324 \HEXADecimalnum}

```

`\aaalphnum` Lowercase alphabetical representation (a ... z aa ... zz)

```

325 \newrobustcmd*{\@aaalph}{\fc@aaalph\@alph}
326 \newcommand*\fc@aaalph[2]{%
327 \@DT@loopN=#2\relax
328 \@DT@X\@DT@loopN
329 \advance\@DT@loopN by \m@ne
330 \divide\@DT@loopN by 26\relax
331 \@DT@modctr=\@DT@loopN
332 \multiply\@DT@modctr by 26\relax
333 \advance\@DT@X by \m@ne
334 \advance\@DT@X by -\@DT@modctr
335 \advance\@DT@loopN by \@ne
336 \advance\@DT@X by \@ne
337 \edef\@tempa{#1\@DT@X}%
338 \loop
339 \@tempa
340 \advance\@DT@loopN by \m@ne
341 \ifnum\@DT@loopN>0
342 \repeat
343 }
344
345 \let\aaalphnum=\@aaalph

```

`\AAAalphnum` Uppercase alphabetical representation (a ... z aa ... zz)

```

346 \newrobustcmd*{\@AAAalph}{\fc@aaalph\@Alph}%
347
348 \let\AAAalphnum=\@AAAalph

```

`\abalphnum` Lowercase alphabetical representation

```

349 \newrobustcmd*{\@abalph}{\fc@abalph\@alph}%
350 \newcommand*\fc@abalph[2]{%

```

```

351 \DT@X=#2\relax
352 \ifnum\DT@X>17576\relax
353   \ifx#1\@alph\def\@tempa{\@abalph}%
354   \else\def\@tempa{\@ABAlph}\fi
355   \PackageError{fmtcount}%
356   {Value of counter too large for \expandafter\protect\@tempa}%
357   {Maximum value 17576}%
358 \else
359   \@DT@padzeroestruer
360   \@strctr=17576\relax
361   \advance\DT@X by \m@ne
362   \loop
363     \@DT@modctr=\DT@X
364     \divide\@DT@modctr by \@strctr
365     \ifthenelse{\boolean{\@DT@padzeroes}
366       \and \(\@DT@modctr=1\)}%
367     {\#1\@DT@modctr}%
368     \ifnum\@DT@modctr=\@ne\else\@DT@padzeroesfalse\fi
369     \multiply\@DT@modctr by \@strctr
370     \advance\DT@X by -\@DT@modctr
371     \divide\@strctr by 26\relax
372   \ifnum\@strctr>\@ne
373     \repeat
374     \advance\DT@X by \@ne
375     #1\@DT@X
376   \fi
377 }
378
379 \let\abalphnum=\@abalph

```

**\ABAlphnum** Uppercase alphabetical representation

```

380 \newrobustcmd*{\@ABAlph}{\fc@abalph\@Alph}%
381 \let\ABAlphnum=\@ABAlph

```

**\@fmtc@count** Recursive command to count number of characters in argument. \@strctr should be set to zero before calling it.

```

382 \def\@fmtc@count#1#2\relax{%
383   \if\relax#1%
384   \else
385     \advance\@strctr by 1\relax
386     \@fmtc@count#2\relax
387   \fi
388 }

```

**\@decimal** Format number as a decimal, possibly padded with zeroes in front.

```

389 \newrobustcmd*{\@decimal}[1]{%
390   \@strctr=0\relax
391   \expandafter\@fmtc@count\number#1\relax
392   \@DT@loopN=\c@padzeroesN

```

```

393 \advance\@DT@loopN by -\@strctr
394 \ifnum\@DT@loopN>0\relax
395   \@strctr=0\relax
396   \whiledo{\@strctr < \@DT@loopN}{0\advance\@strctr by 1\relax}%
397 \fi
398 \number#1\relax
399 }
400
401 \let\decimalnum=\@decimal

```

`\FCordinal`

```
\FCordinal{<number>}
```

This is a bit cumbersome. Previously `\@ordinal` was defined in a similar way to `\abalph` etc. This ensured that the actual value of the counter was written in the new label stuff in the `.aux` file. However adding in an optional argument to determine the gender for multilingual compatibility messed things up somewhat. This was the only work around I could get to keep the the cross-referencing stuff working, which is why the optional argument comes *after* the compulsory argument, instead of the usual manner of placing it before. Note however, that putting the optional argument means that any spaces will be ignored after the command if the optional argument is omitted. Version 1.04 changed `\ordinal` to `\FCordinal` to prevent it clashing with the memoir class.

```

402 \newcommand{\FCordinal}[1]{%
403   \ordinalnum{%
404     \the\value{#1}}%
405 }

```

`\ordinal` If `\ordinal` isn't defined make `\ordinal` a synonym for `\FCordinal` to maintain compatibility with previous versions.

```

406 \ifcsundef{ordinal}
407 {\let\ordinal\FCordinal}%
408 {%
409   \PackageWarning{fmtcount}%
410   {\protect\ordinal \space already defined use
411     \protect\FCordinal \space instead.}
412 }

```

`\ordinalnum` Display ordinal where value is given as a number or count register instead of a counter:

```

413 \newrobustcmd*{\ordinalnum}[1]{%
414   \new@ifnextchar[%
415     {\@ordinalnum{#1}}%
416     {\@ordinalnum{#1}[m]}%
417 }

```

`\@ordinalnum` Display ordinal according to gender (neuter added in v1.1, `\xspace` added in v1.2, and removed in v1.3<sup>7</sup>):

---

<sup>7</sup>I couldn't get it to work consistently both with and without the optional argument

```

418 \def\@ordinalnum#1[#2]{%
419   {%
420     \ifthenelse{\equal{#2}{f}}%
421     {%
422       \protect\@ordinalF{#1}{\@fc@ordstr}%
423     }%
424     {%
425       \ifthenelse{\equal{#2}{n}}%
426       {%
427         \protect\@ordinalN{#1}{\@fc@ordstr}%
428       }%
429       {%
430         \ifthenelse{\equal{#2}{m}}%
431         {}%
432         {%
433           \PackageError{fmtcount}%
434             {Invalid gender option ‘#2’}%
435             {Available options are m, f or n}%
436         }%
437         \protect\@ordinalM{#1}{\@fc@ordstr}%
438       }%
439     }%
440   \@fc@ordstr
441 }%
442 }

```

`\storeordinal` Store the ordinal (first argument is identifying name, second argument is a counter.)

```

443 \newcommand*\storeordinal}[2]{%
444   {%
445     \toks0{\storeordinalnum{#1}}%
446     \expandafter
447   }\the\toks0\expandafter{%
448     \the\value{#2}}%
449 }

```

`storeordinalnum` Store ordinal (first argument is identifying name, second argument is a number or count register.)

```

450 \newrobustcmd*\storeordinalnum}[2]{%
451   \@ifnextchar[%
452   {\@storeordinalnum{#1}{#2}}%
453   {\@storeordinalnum{#1}{#2}[m]}%
454 }

```

`storeordinalnum` Store ordinal according to gender:

```

455 \def\@storeordinalnum#1#2[#3]{%
456   \ifthenelse{\equal{#3}{f}}%
457   {%
458     \protect\@ordinalF{#2}{\@fc@ord}
459   }%

```

```

460 {%
461   \ifthenelse{\equal{#3}{n}}%
462   {%
463     \protect\@ordinalN{#2}{\@fc@ord}%
464   }%
465   {%
466     \ifthenelse{\equal{#3}{m}}%
467     {}%
468     {%
469       \PackageError{fmtcount}%
470       {Invalid gender option ‘#3’}%
471       {Available options are m or f}%
472     }%
473     \protect\@ordinalM{#2}{\@fc@ord}%
474   }%
475 }%
476 \expandafter\let\csname @fcs@#1\endcsname\@fc@ord
477 }

```

`\FMCuse` Get stored information:

```
478 \newcommand*\@FMCuse[1]{\csname @fcs@#1\endcsname}
```

`\ordinalstring` Display ordinal as a string (argument is a counter)

```

479 \newcommand*\@ordinalstring[1]{%
480   \ordinalstringnum{\expandafter\expandafter\expandafter
481     \the\value{#1}}%
482 }

```

`\ordinalstringnum` Display ordinal as a string (argument is a count register or number.)

```

483 \newrobustcmd*\@ordinalstringnum[1]{%
484   \new@ifnextchar[%
485     {\@ordinal@string{#1}}%
486     {\@ordinal@string{#1}[m]}%
487 }

```

`\@ordinal@string` Display ordinal as a string according to gender.

```

488 \def\@ordinal@string#1[#2]{%
489   {%
490     \ifthenelse{\equal{#2}{f}}%
491     {%
492       \protect\@ordinalstringF{#1}{\@fc@ordstr}%
493     }%
494     {%
495       \ifthenelse{\equal{#2}{n}}%
496       {%
497         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
498       }%
499       {%
500         \ifthenelse{\equal{#2}{m}}%

```

```

501     {}%
502     {%
503         \PackageError{fmtcount}%
504         {Invalid gender option ‘#2’ to \protect\ordinalstring}%
505         {Available options are m, f or n}%
506     }%
507     \protect\@ordinalstringM{#1}{\@fc@ordstr}%
508 }%
509 }%
510 \@fc@ordstr
511 }%
512 }

```

`\storeordinalstring` Store textual representation of number. First argument is identifying name, second argument is the counter set to the required number.

```

513 \newcommand*\storeordinalstring}[2]{%
514   {%
515     \toks0{\storeordinalstringnum{#1}}%
516     \expandafter
517   }\the\toks0\expandafter{\the\value{#2}}%
518 }

```

`\storeordinalstringnum` Store textual representation of number. First argument is identifying name, second argument is a count register or number.

```

519 \newrobustcmd*\storeordinalstringnum}[2]{%
520   \@ifnextchar[%
521   {\@store@ordinal@string{#1}{#2}}%
522   {\@store@ordinal@string{#1}{#2}[m]}%
523 }

```

`\@store@ordinal@string` Store textual representation of number according to gender.

```

524 \def\@store@ordinal@string#1#2[#3]{%
525   \ifthenelse{\equal{#3}{f}}%
526   {%
527     \protect\@ordinalstringF{#2}{\@fc@ordstr}%
528   }%
529   {%
530     \ifthenelse{\equal{#3}{n}}%
531     {%
532       \protect\@ordinalstringN{#2}{\@fc@ordstr}%
533     }%
534     {%
535       \ifthenelse{\equal{#3}{m}}%
536       {}%
537     }%
538     \PackageError{fmtcount}%
539     {Invalid gender option ‘#3’ to \protect\ordinalstring}%
540     {Available options are m, f or n}%
541   }%

```



```

542     \protect\@ordinalstringM{#2}{\@fc@ordstr}%
543   }%
544 }%
545 \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
546 }

```

`\Ordinalstring` Display ordinal as a string with initial letters in upper case (argument is a counter)

```

547 \newcommand*{\Ordinalstring}[1]{%
548   \Ordinalstringnum{\expandafter\expandafter\expandafter\the\value{#1}}%
549 }

```

`rdinalstringnum` Display ordinal as a string with initial letters in upper case (argument is a number or count register)

```

550 \newrobustcmd*{\Ordinalstringnum}[1]{%
551   \new@ifnextchar[%
552     {\@Ordinal@string{#1}}%
553     {\@Ordinal@string{#1}[m]}%
554 }

```

`@Ordinal@string` Display ordinal as a string with initial letters in upper case according to gender

```

555 \def\@Ordinal@string#1[#2]{%
556   {%
557     \ifthenelse{equal{#2}{f}}%
558     {%
559       \protect\@OrdinalstringF{#1}{\@fc@ordstr}%
560     }%
561     {%
562       \ifthenelse{equal{#2}{n}}%
563       {%
564         \protect\@OrdinalstringN{#1}{\@fc@ordstr}%
565       }%
566       {%
567         \ifthenelse{equal{#2}{m}}%
568         {}%
569         {%
570           \PackageError{fmtcount}%
571             {Invalid gender option ‘#2’}%
572             {Available options are m, f or n}%
573           }%
574         \protect\@OrdinalstringM{#1}{\@fc@ordstr}%
575       }%
576     }%
577   \@fc@ordstr
578 }%
579 }

```

`reOrdinalstring` Store textual representation of number, with initial letters in upper case. First argument is identifying name, second argument is the counter set to the required number.

```

580 \newcommand*{\storeOrdinalstring}[2]{%
581   {%
582     \toks0{\storeOrdinalstringnum{#1}}%
583     \expandafter
584   }\the\toks0\expandafter{\the\value{#2}}%
585 }

```

`\storeOrdinalstringnum` Store textual representation of number, with initial letters in upper case. First argument is identifying name, second argument is a count register or number.

```

586 \newrobustcmd*{\storeOrdinalstringnum}[2]{%
587   \@ifnextchar[%
588   {\@store@Ordinal@string{#1}{#2}}%
589   {\@store@Ordinal@string{#1}{#2}[m]}}%
590 }

```

`\store@Ordinal@string` Store textual representation of number according to gender, with initial letters in upper case.

```

591 \def\@store@Ordinal@string#1#2[#3]{%
592   \ifthenelse{\equal{#3}{f}}%
593   {%
594     \protect\@OrdinalstringF{#2}{\@fc@ordstr}}%
595   }%
596   {%
597     \ifthenelse{\equal{#3}{n}}%
598     {%
599       \protect\@OrdinalstringN{#2}{\@fc@ordstr}}%
600     }%
601     {%
602       \ifthenelse{\equal{#3}{m}}%
603       {}%
604       {%
605         \PackageError{fmtcount}%
606         {Invalid gender option ‘#3’}%
607         {Available options are m or f}}%
608       }%
609     \protect\@OrdinalstringM{#2}{\@fc@ordstr}}%
610   }%
611 }%
612 \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
613 }

```

`\storeORDINALstring` Store upper case textual representation of ordinal. The first argument is identifying name, the second argument is a counter.

```

614 \newcommand*{\storeORDINALstring}[2]{%
615   {%
616     \toks0{\storeORDINALstringnum{#1}}%
617     \expandafter
618   }\the\toks0\expandafter{\the\value{#2}}%
619 }

```

`ORDINALstringnum` As above, but the second argument is a count register or a number.

```
620 \newrobustcmd*{\storeORDINALstringnum}[2]{%
621   \@ifnextchar[%
622     {\@storeORDINAL@string{#1}{#2}}%
623     {\@storeORDINAL@string{#1}{#2}[m]}}%
624 }
```

`@ORDINAL@string` Gender is specified as an optional argument at the end.

```
625 \def\@storeORDINAL@string#1#2[#3]{%
626   \ifthenelse{\equal{#3}{f}}%
627     {%
628       \protect\@ordinalstringF{#2}{\@fc@ordstr}}%
629   }%
630   {%
631     \ifthenelse{\equal{#3}{n}}%
632       {%
633         \protect\@ordinalstringN{#2}{\@fc@ordstr}}%
634       }%
635       {%
636         \ifthenelse{\equal{#3}{m}}%
637           {}%
638           {%
639             \PackageError{fmtcount}%
640               {Invalid gender option ‘#3’}%
641               {Available options are m or f}}%
642           }%
643         \protect\@ordinalstringM{#2}{\@fc@ordstr}}%
644   }%
645 }%

646 \expandafter\protected@edef\csname @fcs@#1\endcsname{%
647   \noexpand\MakeUppercase{\@fc@ordstr}}%
648 }%
649 }
```

`\ORDINALstring` Display upper case textual representation of an ordinal. The argument must be a counter.

```
650 \newcommand*{\ORDINALstring}[1]{%
651   \ORDINALstringnum{\expandafter\expandafter\expandafter
652     \the\value{#1}}%
653 }%
654 }
```

`ORDINALstringnum` As above, but the argument is a count register or a number.

```
655 \newrobustcmd*{\ORDINALstringnum}[1]{%
656   \new@ifnextchar[%
657     {\@ORDINAL@string{#1}}%
658     {\@ORDINAL@string{#1}[m]}}%
659 }
```

`@ORDINAL@string` Gender is specified as an optional argument at the end.

```
660 \def\@ORDINAL@string#1[#2]{%
661   {%
662     \ifthenelse{\equal{#2}{f}}%
663     {%
664       \protect\@ordinalstringF{#1}{\@fc@ordstr}%
665     }%
666     {%
667       \ifthenelse{\equal{#2}{n}}%
668       {%
669         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
670       }%
671       {%
672         \ifthenelse{\equal{#2}{m}}%
673         {}%
674         {%
675           \PackageError{fmtcount}%
676             {Invalid gender option ‘#2’}%
677             {Available options are m, f or n}%
678           }%
679         \protect\@ordinalstringM{#1}{\@fc@ordstr}%
680       }%
681     }%
682   \MakeUppercase{\@fc@ordstr}%
683 }%
684 }
```

`storenumberstring` Convert number to textual representation, and store. First argument is the identifying name, second argument is a counter containing the number.

```
685 \newcommand*\@storenumberstring[2]{%
686   \expandafter\protect\expandafter\storenumberstringnum{#1}{%
687     \expandafter\the\value{#2}}%
688 }
```

`numberstringnum` As above, but second argument is a number or count register.

```
689 \newcommand*\@storenumberstringnum[2]{%
690   \@ifnextchar[%
691     {\@store@number@string{#1}{#2}}%
692     {\@store@number@string{#1}{#2}[m]}%
693 }
```

`@number@string` Gender is given as optional argument, *at the end*.

```
694 \def\@store@number@string#1#2[#3]{%
695   \ifthenelse{\equal{#3}{f}}%
696   {%
697     \protect\@numberstringF{#2}{\@fc@numstr}%
698   }%
699   {%
700     \ifthenelse{\equal{#3}{n}}%
```

```

701   {%
702     \protect\@numberstringN{#2}{\@fc@numstr}%
703   }%
704   {%
705     \ifthenelse{\equal{#3}{m}}%
706     {}%
707     {%
708       \PackageError{fmtcount}
709       {Invalid gender option ‘#3’}%
710       {Available options are m, f or n}%
711     }%
712     \protect\@numberstringM{#2}{\@fc@numstr}%
713   }%
714 }%
715 \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
716 }

```

`\numberstring` Display textual representation of a number. The argument must be a counter.

```

717 \newcommand*{\numberstring}[1]{%
718   \numberstringnum{\expandafter\expandafter\expandafter
719     \the\value{#1}}%
720 }

```

`numberstringnum` As above, but the argument is a count register or a number.

```

721 \newrobustcmd*{\numberstringnum}[1]{%
722   \new@ifnextchar[%
723     {\@number@string{#1}}%
724     {\@number@string{#1}[m]}%
725 }

```

`\@number@string` Gender is specified as an optional argument *at the end*.

```

726 \def\@number@string#1[#2]{%
727   {%
728     \ifthenelse{\equal{#2}{f}}%
729     {%
730       \protect\@numberstringF{#1}{\@fc@numstr}%
731     }%
732     {%
733       \ifthenelse{\equal{#2}{n}}%
734       {%
735         \protect\@numberstringN{#1}{\@fc@numstr}%
736       }%
737       {%
738         \ifthenelse{\equal{#2}{m}}%
739         {}%
740         {%
741           \PackageError{fmtcount}%
742           {Invalid gender option ‘#2’}%
743           {Available options are m, f or n}%

```

```

744     }%
745     \protect\@numberstringM{#1}{\@fc@numstr}%
746   }%
747 }%
748 \@fc@numstr
749 }%
750 }

```

`\storeNumberstring` Store textual representation of number. First argument is identifying name, second argument is a counter.

```

751 \newcommand*\storeNumberstring[2]{%
752   {%
753     \toks0{\storeNumberstringnum{#1}}%
754     \expandafter
755   }\the\toks0\expandafter{\the\value{#2}}%
756 }

```

`\Numberstringnum` As above, but second argument is a count register or number.

```

757 \newcommand*\storeNumberstringnum[2]{%
758   \@ifnextchar[%
759     {\@store@Number@string{#1}{#2}}%
760     {\@store@Number@string{#1}{#2}[m]}%
761 }

```

`\@Number@string` Gender is specified as an optional argument *at the end*:

```

762 \def\@store@Number@string#1#2[#3]{%
763   \ifthenelse{\equal{#3}{f}}%
764   {%
765     \protect\@NumberstringF{#2}{\@fc@numstr}%
766   }%
767   {%
768     \ifthenelse{\equal{#3}{n}}%
769     {%
770       \protect\@NumberstringN{#2}{\@fc@numstr}%
771     }%
772     {%
773       \ifthenelse{\equal{#3}{m}}%
774       {}%
775       {%
776         \PackageError{fmtcount}%
777         {Invalid gender option ‘#3’}%
778         {Available options are m, f or n}%
779       }%
780       \protect\@NumberstringM{#2}{\@fc@numstr}%
781     }%
782   }%
783   \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
784 }

```

`\Numberstring` Display textual representation of number. The argument must be a counter.

```
785 \newcommand*{\Numberstring}[1]{%
786   \Numberstringnum{\expandafter\expandafter\expandafter
787     \the\value{#1}}%
788 }
```

`Numberstringnum` As above, but the argument is a count register or number.

```
789 \newrobustcmd*{\Numberstringnum}[1]{%
790   \new@ifnextchar[%
791     {\@Number@string{#1}}%
792     {\@Number@string{#1}[m]}%
793 }
```

`\@Number@string` Gender is specified as an optional argument at the end.

```
794 \def\@Number@string#1[#2]{%
795   {%
796     \ifthenelse{\equal{#2}{f}}%
797     {%
798       \protect\@NumberstringF{#1}{\@fc@numstr}%
799     }%
800     {%
801       \ifthenelse{\equal{#2}{n}}%
802       {%
803         \protect\@NumberstringN{#1}{\@fc@numstr}%
804       }%
805       {%
806         \ifthenelse{\equal{#2}{m}}%
807         {}%
808         {%
809           \PackageError{fmtcount}%
810             {Invalid gender option ‘#2’}%
811             {Available options are m, f or n}%
812         }%
813         \protect\@NumberstringM{#1}{\@fc@numstr}%
814       }%
815     }%
816   \@fc@numstr
817 }%
818 }
```

`storeNUMBERstring` Store upper case textual representation of number. The first argument is identifying name, the second argument is a counter.

```
819 \newcommand{\storeNUMBERstring}[2]{%
820   {%
821     \toks0{\storeNUMBERstringnum{#1}}%
822     \expandafter
823     }\the\toks0\expandafter{\the\value{#2}}%
824 }
```

`NUMBERstringnum` As above, but the second argument is a count register or a number.

```
825 \newcommand{\storeNUMBERstringnum}[2]{%
826   \@ifnextchar[%
827     {\@store@NUMBER@string{#1}{#2}}%
828     {\@store@NUMBER@string{#1}{#2}[m]}}%
829 }
```

`e@NUMBER@string` Gender is specified as an optional argument at the end.

```
830 \def\@store@NUMBER@string#1#2[#3]{%
831   \ifthenelse{\equal{#3}{f}}%
832     {%
833       \protect\@numberstringF{#2}{\@fc@numstr}%
834     }%
835     {%
836       \ifthenelse{\equal{#3}{n}}%
837         {%
838           \protect\@numberstringN{#2}{\@fc@numstr}%
839         }%
840         {%
841           \ifthenelse{\equal{#3}{m}}%
842             {}%
843             {%
844               \PackageError{fmtcount}%
845                 {Invalid gender option ‘#3’}%
846                 {Available options are m or f}%
847             }%
848           \protect\@numberstringM{#2}{\@fc@numstr}%
849         }%
850     }%
851   \expandafter\edef\csname @fcs@#1\endcsname{%
852     \noexpand\MakeUppercase{\@fc@numstr}}%
853   }%
854 }
```

`\NUMBERstring` Display upper case textual representation of a number. The argument must be a counter.

```
855 \newcommand*{\NUMBERstring}[1]{%
856   \NUMBERstringnum{\expandafter\expandafter\expandafter
857     \the\value{#1}}%
858 }
```

`NUMBERstringnum` As above, but the argument is a count register or a number.

```
859 \newrobustcmd*{\NUMBERstringnum}[1]{%
860   \new@ifnextchar[%
861     {\@NUMBER@string{#1}}%
862     {\@NUMBER@string{#1}[m]}}%
863 }
```

`\@NUMBER@string` Gender is specified as an optional argument at the end.



```

864 \def\@NUMBER@string#1[#2]{%
865   {%
866     \ifthenelse{\equal{#2}{f}}%
867     {%
868       \protect\@numberstringF{#1}{\@fc@numstr}%
869     }%
870     {%
871       \ifthenelse{\equal{#2}{n}}%
872       {%
873         \protect\@numberstringN{#1}{\@fc@numstr}%
874       }%
875       {%
876         \ifthenelse{\equal{#2}{m}}%
877         {}%
878         {%
879           \PackageError{fmtcount}%
880             {Invalid gender option ‘#2’}%
881             {Available options are m, f or n}%
882           }%
883           \protect\@numberstringM{#1}{\@fc@numstr}%
884         }%
885       }%
886     \protect\MakeUppercase{\@fc@numstr}%
887   }%
888 }

```

`\binary` Number representations in other bases. Binary:

```

889 \providecommand*\binary[1]{%
890   \@binary{\expandafter\expandafter\expandafter
891     \the\value{#1}}%
892 }

```

`\aaalph` Like `\alph` but goes beyond 26. (a ... z aa ... zz ...)

```

893 \providecommand*\aaalph[1]{%
894   \@aaalph{\expandafter\expandafter\expandafter
895     \the\value{#1}}%
896 }

```

`\AAAph` As before, but upper case.

```

897 \providecommand*\AAAph[1]{%
898   \@AAAph{\expandafter\expandafter\expandafter
899     \the\value{#1}}%
900 }

```

`\abalph` Like `\alph` but goes beyond 26. (a ... z ab ... az ...)

```

901 \providecommand*\abalph[1]{%
902   \@abalph{\expandafter\expandafter\expandafter
903     \the\value{#1}}%
904 }

```

`\ABAlph` As above, but upper case.

```
905 \providecommand*\ABAlph}[1]{%
906   \@ABAlph{\expandafter\expandafter\expandafter
907     \the\value{#1}}%
908 }
```

`\hexadecimal` Hexadecimal:

```
909 \providecommand*\hexadecimal}[1]{%
910   \hexadecimalnum{\expandafter\expandafter\expandafter
911     \the\value{#1}}%
912 }
```

`\HEXADecimal` As above, but in upper case.

```
913 \providecommand*\HEXADecimal}[1]{%
914   \HEXADecimalnum{\expandafter\expandafter\expandafter
915     \the\value{#1}}%
916 }
917 \newrobustcmd*\FC@Hexadecimal@warning{%
918   \PackageWarning{fmtcount}{\string\Hexadecimal\space is deprecated, use \string\HEXADecimal\sp
919     instead. The \string\Hexadecimal\space control sequence name is confusing as it can mislead
920     that only the 1st letter is upper-cased.}%
921 }
922 \def\Hexadecimal{%
923   \FC@Hexadecimal@warning
924   \HEXADecimal}
```

`\octal` Octal:

```
925 \providecommand*\octal}[1]{%
926   \@octal{\expandafter\expandafter\expandafter
927     \the\value{#1}}%
928 }
```

`\decimal` Decimal:

```
929 \providecommand*\decimal}[1]{%
930   \@decimal{\expandafter\expandafter\expandafter
931     \the\value{#1}}%
932 }
```

### 10.4.1 Multilanguage Definitions

Flag `\fc@languagemode@detected` allows to stop scanning for multilingual mode trigger conditions. It is initialized to false as no such scanning as taken place yet.

```
933 \newif\iffc@languagemode@detected
934 \fc@languagemode@detectedfalse
```

`def@ultfmtcount` If multilingual support is provided, make `\@numberstring` etc use the correct language (if defined). Otherwise use English definitions. `\@setdef@ultfmtcount` sets the macros to use English.

```

935 \def\@setdef@ultfmtcount{%
936   \fc@languagemode@detectedtrue
937   \ifcsundef{@ordinalMenglish}{\FCloadlang{english}}{}%
938   \def\@ordinalstringM{\@ordinalstringMenglish}%
939   \let\@ordinalstringF=\@ordinalstringMenglish
940   \let\@ordinalstringN=\@ordinalstringMenglish
941   \def\@OrdinalstringM{\@OrdinalstringMenglish}%
942   \let\@OrdinalstringF=\@OrdinalstringMenglish
943   \let\@OrdinalstringN=\@OrdinalstringMenglish
944   \def\@numberstringM{\@numberstringMenglish}%
945   \let\@numberstringF=\@numberstringMenglish
946   \let\@numberstringN=\@numberstringMenglish
947   \def\@NumberstringM{\@NumberstringMenglish}%
948   \let\@NumberstringF=\@NumberstringMenglish
949   \let\@NumberstringN=\@NumberstringMenglish
950   \def\@ordinalM{\@ordinalMenglish}%
951   \let\@ordinalF=\@ordinalM
952   \let\@ordinalN=\@ordinalM
953   \let\fmtord\fc@orddef@ult
954 }

```

```
\fc@multiling \fc@multiling{<name>}{<gender>}
```

```

955 \newcommand*{\fc@multiling}[2]{%
956   \ifcsundef{@#1#2\languagename}%
957   {% try loading it
958     \FCloadlang{\languagename}%
959   }%
960   {%
961   }%
962   \ifcsundef{@#1#2\languagename}%
963   {%
964     \PackageWarning{fmtcount}%
965     {No support for \expandafter\protect\csname #1\endcsname\space for
966     language '\languagename'}%
967     \ifthenelse{\equal{\languagename}{\fc@mainlang}}{%
968     {%
969       \FCloadlang{english}%
970     }%
971     {%
972     }%
973     \ifcsdef{@#1#2\fc@mainlang}%
974     {%
975       \csuse{@#1#2\fc@mainlang}%
976     }%
977     {%
978       \PackageWarningNoLine{fmtcount}%
979       {No languages loaded at all! Loading english definitions}%
980       \FCloadlang{english}%
981       \def\fc@mainlang{english}%

```

```

982     \csuse{@#1#2english}%
983   }%
984 }%
985 {%
986   \csuse{@#1#2\languagename}%
987 }%
988 }

```

itling@fmtcount This defines the number and ordinal string macros to use \languagename:

```

989 \def\@set@multiling@fmtcount{%
990   \fc@languagemode@detectedtrue

```

The masculine version of \numberstring:

```

991   \def\@numberstringM{%
992     \fc@multiling{numberstring}{M}%
993   }%

```

The feminine version of \numberstring:

```

994   \def\@numberstringF{%
995     \fc@multiling{numberstring}{F}%
996   }%

```

The neuter version of \numberstring:

```

997   \def\@numberstringN{%
998     \fc@multiling{numberstring}{N}%
999   }%

```

The masculine version of \Numberstring:

```

1000  \def\@NumberstringM{%
1001    \fc@multiling{Numberstring}{M}%
1002  }%

```

The feminine version of \Numberstring:

```

1003  \def\@NumberstringF{%
1004    \fc@multiling{Numberstring}{F}%
1005  }%

```

The neuter version of \Numberstring:

```

1006  \def\@NumberstringN{%
1007    \fc@multiling{Numberstring}{N}%
1008  }%

```

The masculine version of \ordinal:

```

1009  \def\@ordinalM{%
1010    \fc@multiling{ordinal}{M}%
1011  }%

```

The feminine version of \ordinal:

```

1012  \def\@ordinalF{%
1013    \fc@multiling{ordinal}{F}%
1014  }%

```

The neuter version of \ordinal:

```
1015 \def\@ordinalN{%
1016   \fc@multiling{ordinal}{N}%
1017 }%
```

The masculine version of \ordinalstring:

```
1018 \def\@ordinalstringM{%
1019   \fc@multiling{ordinalstring}{M}%
1020 }%
```

The feminine version of \ordinalstring:

```
1021 \def\@ordinalstringF{%
1022   \fc@multiling{ordinalstring}{F}%
1023 }%
```

The neuter version of \Ordinalstring:

```
1024 \def\@OrdinalstringN{%
1025   \fc@multiling{Ordinalstring}{N}%
1026 }%
```

The masculine version of \Ordinalstring:

```
1027 \def\@OrdinalstringM{%
1028   \fc@multiling{Ordinalstring}{M}%
1029 }%
```

The feminine version of \Ordinalstring:

```
1030 \def\@OrdinalstringF{%
1031   \fc@multiling{Ordinalstring}{F}%
1032 }%
```

The neuter version of \Ordinalstring:

```
1033 \def\@OrdinalstringN{%
1034   \fc@multiling{Ordinalstring}{N}%
1035 }%
```

Make \fmtord language dependent:

```
1036 \let\fmtord\fc@ord@multiling
1037 }
```

Check to see if babel, polyglossia, mlp, or ngerman packages have been loaded, and if yes set \fmtcount in multiling. First we define some \fc@check@for@multiling macro to do such action where #1 is the package name, and #2 is a callback.

```
1038 \def\fc@check@for@multiling#1:#2\@nil{%
1039   \@ifpackageloaded{#1}{%
1040     #2\@set@multiling@fmtcount
1041   }{}%
1042 }
```

Now we define \fc@loop@on@multiling@pkg as an iterator to scan whether any of babel, polyglossia, mlp, or ngerman packages has been loaded, and if so set multilingual mode.

```
1043 \def\fc@loop@on@multiling@pkg#1,{%
1044   \def\@tempb{#1}%
1045   \ifx\@tempb\@nnil
```

We have reached the end of the loop, so stop here.

```
1046 \let\fc@loop@on@multiling@pkg\@empty
1047 \else
```

Make the `\@ifpackageloaded` test and break the loop if it was positive.

```
1048 \fc@check@for@multiling#1\@nil
1049 \iffc@languagemode@detected
1050 \def\fc@loop@on@multiling@pkg##1\@nil,{}%
1051 \fi
1052 \fi
1053 \fc@loop@on@multiling@pkg
1054 }
```

Now, do the loop itself, we do this at beginning of document not to constrain the order of loading `fmtcount` and the multilingual package `babel`, `polyglossia`, etc.:

```
1055 \AtBeginDocument{%
1056 \fc@loop@on@multiling@pkg babel:,polyglossia:,ngerman:\FCloadlang{ngerman},\@nil,
```

In the case that no multilingual package (such as `babel/polyglossia/ngerman`) has been loaded, then we go to `multiling` if a language has been loaded by package option.

```
1057 \unless\iffc@languagemode@detected\iffmtcount@language@option
```

If the multilingual mode has not been yet activated, but a language option has been passed to `fmtcount`, we should go to multilingual mode. However, first of, we do some sanity check, as this may help the end user understand what is wrong: we check that macro `\languagename` is defined, and activate the multilingual mode only then, and otherwise fall back to default legacy mode.

```
1058 \ifcsundef{languagename}%
1059 {%
1060 \PackageWarning{fmtcount}{%
1061 \protect\languagename' is undefined, you should use a language package such as bab
1062 when loading a language via package option. Reverting to default language.
1063 }%
1064 \@setdef@ultfmtcount
1065 }{%
1066 \@set@multiling@fmtcount
1067
```

Now, some more checking, having activated multilingual mode after a language option has been passed to `fmtcount`, we check that the `fmtcount` language definitions corresponding to `\languagename` have been loaded, and otherwise fall `\languagename` back to the latest `fmtcount` language definition loaded.

```
1068 \@FC@iflangloaded{\languagename}{-}{%
```

The current `\languagename` is not a `fmtcount` language that has been previously loaded. The correction is to have `\languagename` let to `\fc@mainlang`. Please note that, as `\iffmtcount@language@option` is true, we know that `fmtcount` has loaded some language.

```
1069 \PackageWarning{fmtcount}{%
1070 Setting '\protect\languagename' to '\fc@mainlang'.\MessageBreak
1071 Reason is that '\protect\languagename' was '\languagename',\MessageBreak
```

```

1072         but ‘\languagename’ was not loaded by fmtcount,\MessageBreak
1073         whereas ‘\fc@mainlang’ was the last language loaded by fmtcount ;
1074     }%
1075     \let\languagename\fc@mainlang
1076 }%
1077 }%
1078 \else
1079     \setdef@ultfmtcount
1080 \fi\fi

1081 \ifcsundef{FBsupR}{\let\fc@textsuperscript\textsuperscript}{\let\fc@textsuperscript\sup}%
1082 }

```

Backwards compatibility:

```

1083 \let\@ordinal=\@ordinalM
1084 \let\@ordinalstring=\@ordinalstringM
1085 \let\@Ordinalstring=\@OrdinalstringM
1086 \let\@numberstring=\@numberstringM
1087 \let\@Numberstring=\@NumberstringM

```