

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

Nicola L.C. Talbot

Vincent Belaïche

[www.dickimaw-books.com](http://www.dickimaw-books.com)

2020-02-24 (version 3.07)

## 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>15</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	<code>fc-british.def</code> . . . . .	19
10.1.4	<code>fc-english.def</code> . . . . .	20
10.1.5	<code>fc-francais.def</code> . . . . .	30

10.1.6 fc-french.def . . . . .	30
10.1.7 fc-frenchb.def . . . . .	62
10.1.8 fc-german.def . . . . .	63
10.1.9 fc-germanb.def . . . . .	73
10.1.10fc-italian . . . . .	73
10.1.11fc-ngerman.def . . . . .	74
10.1.12fc-ngermanb.def . . . . .	75
10.1.13fc-portuges.def . . . . .	75
10.1.14fc-portuguese.def . . . . .	90
10.1.15fc-spanish.def . . . . .	90
10.1.16fc-UKenglish.def . . . . .	108
10.1.17fc-USenglish.def . . . . .	108
10.2 fcnumparser.sty . . . . .	109
10.3 fcprefix.sty . . . . .	119
10.4 fmtcount.sty . . . . .	129
10.4.1 Multilingual Definitions . . . . .	154

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

This will print the value of a  $\LaTeX$  counter  $\langle counter \rangle$  as an ordinal, where the macro

`\fmtord` `\fmtord{\langle text \rangle}`

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  $\langle gender \rangle$  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 [`<gender>`] 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{<n>}[<gender>]
```

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

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

`\Numberstring`

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

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

`\NUMBERstring`

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

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

`\numberstringnum`

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

`\Numberstringnum`

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

`\NUMBERstringnum`

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

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

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`).

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

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

`RDINALstringnum` `\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>]
preordinalstring	\storeordinalstring{<label>}{<counter>}[<gender>]
preOrdinalstring	\storeOrdinalstring{<label>}{<counter>}[<gender>]
preORDINALstring	\storeORDINALstring{<label>}{<counter>}[<gender>]
prenumberstring	\storenumberstring{<label>}{<counter>}[<gender>]
preNumberstring	\storeNumberstring{<label>}{<counter>}[<gender>]
preNUMBERstring	\storeNUMBERstring{<label>}{<counter>}[<gender>]
storeordinalnum	\storeordinalnum{<label>}{<number>}[<gender>]
preordinalstringnum	\storeordinalstring{<label>}{<number>}[<gender>]
preOrdinalstringnum	\storeOrdinalstringnum{<label>}{<number>}[<gender>]
preORDINALstringnum	\storeORDINALstringnum{<label>}{<number>}[<gender>]
prenumberstringnum	\storenumberstring{<label>}{<number>}[<gender>]
preNumberstringnum	\storeNumberstring{<label>}{<number>}[<gender>]
preNUMBERstringnum	\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` `\binary{<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:

`\fmtcountsetoptions` `\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`, and `italian`.

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 `reformed` 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 $\langle n \rangle$ illion et $\langle n \rangle$ illiard 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 $\langle n \rangle$ illion ou un $\langle n \rangle$ illiard; ceci est la convention en vigueur pour le pluriel de “vingt” et de “cent” lorsque le nombre formaté a une valeur cardinale,

- multiple lng-last pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 et est *localement* mais *non globalement* en dernière position, où “localement” et *globalement* on la même signification que pour les options multiple g-last et multiple l-last; 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*’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’inferè être en vigueur pour les nombres de la forme  $\langle n \rangle$ illion et  $\langle n \rangle$ 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 :

$\langle x \rangle$ dans “ $\langle x \rangle$ 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  $\langle dash or space \rangle$  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, actuel-  
 lement 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 `<scale>` 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 `<n>`illion où `<n>` est remplacé par "bi" pour 2, "tri" pour 3, etc. et  $10^{6 \times n + 3}$  donne un `<n>`illiard avec la même convention pour `<n>`. L'option `long` est correcte en Europe, par contre j'ignore l'usage au Québec.  
`short`  $10^{6 \times n}$  donne un `<n>`illion où `<n>` 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=<n-illiard upto>}}`

Cette option n'a de sens que si `scale` vaut `long`. Certaines personnes préfèrent dire "mille `<n>`illions" qu'un "`<n>`illiard". Mettre l'option `n-illiard upto` à :

`infinity` pour que  $10^{6 \times n + 3}$  donne `<n>`illiards 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 `<n>`illions" lorsque  $n > k$ , et donne "`<n>`illiard" sinon

mil plural mark `\fmtcountsetoptions{french={mil plural mark=<any text>}}`

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{<counter>}[<prefix options>]`

latinnumeralstringnum `\latinnumeralstringnum{<number>}[<prefix options>]`

## 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 typeset 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 `\numbstring` 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

There is a FAQ available at: <http://theoval.cmp.uea.ac.uk/~nlct/latex/packages/faq/>.

Bug reporting should be done via the Github issue manager at: <https://github.com/nlct/fmtcount/issues/>.

Local Variables: coding: utf-8 compile-command: "make -C ../dist fmtcount.pdf" End:

## 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.

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

Load fc-portuges.def if not already loaded.

```
19 \FCloadlang{portuges}%
```

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

```
20 \global\let\@ordinalMbrazilian=\@ordinalMportuges
21 \global\let\@ordinalFbrazilian=\@ordinalFportuges
22 \global\let\@ordinalNbrazilian=\@ordinalNportuges
23 \global\let\@ordinalstringFbrazilian\@ordinalstringFportuges
24 \global\let\@ordinalstringMbrazilian\@ordinalstringMportuges
25 \global\let\@ordinalstringNbrazilian\@ordinalstringMportuges
26 \global\let\@OrdinalstringMbrazilian\@OrdinalstringMportuges
27 \global\let\@OrdinalstringFbrazilian\@OrdinalstringFportuges
28 \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):

```
29 \newcommand*{\@teenstringbrazilian[1]}{%
30   \ifcase#1\relax
31     dez%
32     \or onze%
33     \or doze%
34     \or treze%
35     \or quatorze%
```

```

36 \or quinze%
37 \or dezesesseis%
38 \or dezessete%
39 \or dezoito%
40 \or dezenove%
41 \fi
42 }%
43 \global\let\@@teenstringbrazilian\@@teenstringbrazilian

```

Teens (with initial letter in upper case):

```

44 \newcommand*\@@Teenstringbrazilian[1]{%
45 \ifcase#1\relax
46 Dez%
47 \or Onze%
48 \or Doze%
49 \or Treze%
50 \or Quatorze%
51 \or Quinze%
52 \or Dezesesseis%
53 \or Dezessete%
54 \or Dezoito%
55 \or Dezenove%
56 \fi
57 }%
58 \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.)

```

59 \newcommand*\@@numberstringMbrazilian}[2]{%
60 \let\@unitstring=\@unitstringportuges
61 \let\@teenstring=\@teenstringbrazilian
62 \let\@tenstring=\@tenstringportuges
63 \let\@hundredstring=\@hundredstringportuges
64 \def\@hundred{cem}\def\@thousand{mil}%
65 \def\@andname{e}%
66 \@@numberstringportuges{#1}{#2}%
67 }%
68 \global\let\@numberstringMbrazilian\@numberstringMbrazilian

```

As above, but feminine form:

```

69 \newcommand*\@@numberstringFbrazilian}[2]{%
70 \let\@unitstring=\@unitstringFportuges
71 \let\@teenstring=\@teenstringbrazilian
72 \let\@tenstring=\@tenstringportuges
73 \let\@hundredstring=\@hundredstringFportuges
74 \def\@hundred{cem}\def\@thousand{mil}%
75 \def\@andname{e}%
76 \@@numberstringportuges{#1}{#2}%
77 }%
78 \global\let\@numberstringFbrazilian\@numberstringFbrazilian

```

Make neuter same as masculine:

```
79 \global\let\@numberstringNbrazilian\@numberstringMbrazilian
```

As above, but initial letters in upper case:

```
80 \newcommand*{\@NumberstringMbrazilian}[2]{%
81   \let\@unitstring=\@unitstringportuges
82   \let\@teenstring=\@Teenstringbrazilian
83   \let\@tenstring=\@Tenstringportuges
84   \let\@hundredstring=\@hundredstringportuges
85   \def\@hundred{Cem}\def\@thousand{Mil}%
86   \def\@andname{e}%
87   \@numberstringportuges{#1}{#2}%
88 }%
89 \global\let\@NumberstringMbrazilian\@NumberstringMbrazilian
```

As above, but feminine form:

```
90 \newcommand*{\@NumberstringFbrazilian}[2]{%
91   \let\@unitstring=\@UnitstringFportuges
92   \let\@teenstring=\@Teenstringbrazilian
93   \let\@tenstring=\@Tenstringportuges
94   \let\@hundredstring=\@HundredstringFportuges
95   \def\@hundred{Cem}\def\@thousand{Mil}%
96   \def\@andname{e}%
97   \@numberstringportuges{#1}{#2}%
98 }%
99 \global\let\@NumberstringFbrazilian\@NumberstringFbrazilian
```

Make neuter same as masculine:

```
100 \global\let\@NumberstringNbrazilian\@NumberstringMbrazilian
```

### 10.1.3 fc-british.def

British definitions

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

Load fc-english.def, if not already loaded

```
102 \FCloadlang{english}%
```

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

```
103 \global\let\@ordinalMbritish\@ordinalMenglish
104 \global\let\@ordinalFbritish\@ordinalMenglish
105 \global\let\@ordinalNbritish\@ordinalMenglish
106 \global\let\@numberstringMbritish\@numberstringMenglish
107 \global\let\@numberstringFbritish\@numberstringMenglish
108 \global\let\@numberstringNbritish\@numberstringMenglish
109 \global\let\@NumberstringMbritish\@NumberstringMenglish
110 \global\let\@NumberstringFbritish\@NumberstringMenglish
111 \global\let\@NumberstringNbritish\@NumberstringMenglish
112 \global\let\@ordinalstringMbritish\@ordinalstringMenglish
113 \global\let\@ordinalstringFbritish\@ordinalstringMenglish
114 \global\let\@ordinalstringNbritish\@ordinalstringMenglish
```

```

115 \global\let\@OrdinalstringMbritish\@OrdinalstringMenglish
116 \global\let\@OrdinalstringFbritish\@OrdinalstringMenglish
117 \global\let\@OrdinalstringNbritish\@OrdinalstringMenglish

```

#### 10.1.4 fc-english.def

English definitions

```
118 \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.

```

119 \newcommand*\@ordinalMenglish[2]{%
120 \def\@fc@ord{}%
121 \@orgargctr=#1\relax
122 \@ordinalctr=#1%
123 \@FCmodulo{\@ordinalctr}{100}%
124 \ifnum\@ordinalctr=11\relax
125 \def\@fc@ord{th}%
126 \else
127 \ifnum\@ordinalctr=12\relax
128 \def\@fc@ord{th}%
129 \else
130 \ifnum\@ordinalctr=13\relax
131 \def\@fc@ord{th}%
132 \else
133 \@FCmodulo{\@ordinalctr}{10}%
134 \ifcase\@ordinalctr
135 \def\@fc@ord{th}% case 0
136 \or \def\@fc@ord{st}% case 1
137 \or \def\@fc@ord{nd}% case 2
138 \or \def\@fc@ord{rd}% case 3
139 \else
140 \def\@fc@ord{th}% default case
141 \fi
142 \fi
143 \fi
144 \fi
145 \edef#2{\number#1\relax\noexpand\fmtord{\@fc@ord}}%
146 }%
147 \global\let\@ordinalMenglish\@ordinalMenglish

```

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

```

148 \global\let\@ordinalFenglish=\@ordinalMenglish
149 \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.

```
150 \newcommand*\@@unitstringenglish[1]{%
```

```

151 \ifcase#1\relax
152   zero%
153   \or one%
154   \or two%
155   \or three%
156   \or four%
157   \or five%
158   \or six%
159   \or seven%
160   \or eight%
161   \or nine%
162 \fi
163 }%
164 \global\let\@@unitstringenglish\@@unitstringenglish

```

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

```

165 \newcommand*\@@tenstringenglish[1]{%
166   \ifcase#1\relax
167   \or ten%
168   \or twenty%
169   \or thirty%
170   \or forty%
171   \or fifty%
172   \or sixty%
173   \or seventy%
174   \or eighty%
175   \or ninety%
176   \fi
177 }%
178 \global\let\@@tenstringenglish\@@tenstringenglish

```

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

```

179 \newcommand*\@@teenstringenglish[1]{%
180   \ifcase#1\relax
181   ten%
182   \or eleven%
183   \or twelve%
184   \or thirteen%
185   \or fourteen%
186   \or fifteen%
187   \or sixteen%
188   \or seventeen%
189   \or eighteen%
190   \or nineteen%
191   \fi
192 }%
193 \global\let\@@teenstringenglish\@@teenstringenglish

```

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

```

194 \newcommand*\@@Unitstringenglish[1]{%
195   \ifcase#1\relax

```

```

196   Zero%
197   \or One%
198   \or Two%
199   \or Three%
200   \or Four%
201   \or Five%
202   \or Six%
203   \or Seven%
204   \or Eight%
205   \or Nine%
206   \fi
207 }%
208 \global\let\@@Unitstringenglish\@@Unitstringenglish

```

The tens:

```

209 \newcommand*\@@Tenstringenglish[1]{%
210   \ifcase#1\relax
211     \or Ten%
212     \or Twenty%
213     \or Thirty%
214     \or Forty%
215     \or Fifty%
216     \or Sixty%
217     \or Seventy%
218     \or Eighty%
219     \or Ninety%
220   \fi
221 }%
222 \global\let\@@Tenstringenglish\@@Tenstringenglish

```

The teens:

```

223 \newcommand*\@@Teenstringenglish[1]{%
224   \ifcase#1\relax
225     Ten%
226     \or Eleven%
227     \or Twelve%
228     \or Thirteen%
229     \or Fourteen%
230     \or Fifteen%
231     \or Sixteen%
232     \or Seventeen%
233     \or Eighteen%
234     \or Nineteen%
235   \fi
236 }%
237 \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.)

```

238 \newcommand*{\@@numberstringenglish[2]}{%
239 \ifnum#1>99999
240 \PackageError{fmtcount}{Out of range}%
241 {This macro only works for values less than 100000}%
242 \else
243 \ifnum#1<0
244 \PackageError{fmtcount}{Negative numbers not permitted}%
245 {This macro does not work for negative numbers, however
246 you can try typing "minus" first, and then pass the modulus of
247 this number}%
248 \fi
249 \fi
250 \def#2{}%
251 \@strctr=#1\relax \divide\@strctr by 1000\relax
252 \ifnum\@strctr>9
253 \divide\@strctr by 10
254 \ifnum\@strctr>1\relax
255 \let\@@fc@numstr#2\relax
256 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
257 \@strctr=#1 \divide\@strctr by 1000\relax
258 \@FCmodulo{\@strctr}{10}%
259 \ifnum\@strctr>0\relax
260 \let\@@fc@numstr#2\relax
261 \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
262 \fi
263 \else
264 \@strctr=#1\relax
265 \divide\@strctr by 1000\relax
266 \@FCmodulo{\@strctr}{10}%
267 \let\@@fc@numstr#2\relax
268 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
269 \fi
270 \let\@@fc@numstr#2\relax
271 \edef#2{\@@fc@numstr\ \@thousand}%
272 \else
273 \ifnum\@strctr>0\relax
274 \let\@@fc@numstr#2\relax
275 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@thousand}%
276 \fi
277 \fi
278 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
279 \divide\@strctr by 100
280 \ifnum\@strctr>0\relax
281 \ifnum#1>1000\relax
282 \let\@@fc@numstr#2\relax
283 \edef#2{\@@fc@numstr\ }%
284 \fi
285 \let\@@fc@numstr#2\relax
286 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@hundred}%

```

```

287 \fi
288 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
289 \ifnum#1>100\relax
290 \ifnum\@strctr>0\relax
291 \let\@fc@numstr#2\relax
292 \edef#2{\@fc@numstr \@andname\ }%
293 \fi
294 \fi
295 \ifnum\@strctr>19\relax
296 \divide\@strctr by 10\relax
297 \let\@fc@numstr#2\relax
298 \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
299 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
300 \ifnum\@strctr>0\relax
301 \let\@fc@numstr#2\relax
302 \edef#2{\@fc@numstr-\@unitstring{\@strctr}}%
303 \fi
304 \else
305 \ifnum\@strctr<10\relax
306 \ifnum\@strctr=0\relax
307 \ifnum#1<100\relax
308 \let\@fc@numstr#2\relax
309 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
310 \fi
311 \else
312 \let\@fc@numstr#2\relax
313 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
314 \fi
315 \else
316 \@FCmodulo{\@strctr}{10}%
317 \let\@fc@numstr#2\relax
318 \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
319 \fi
320 \fi
321 }%
322 \global\let\@numberstringenglish\@numberstringenglish

```

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

```

323 \newcommand*{\@numberstringMenglish}[2]{%
324 \let\@unitstring=\@unitstringenglish
325 \let\@teenstring=\@teenstringenglish
326 \let\@tenstring=\@tenstringenglish
327 \def\@hundred{hundred}\def\@thousand{thousand}%
328 \def\@andname{and}%
329 \@numberstringenglish{#1}{#2}%
330 }%
331 \global\let\@numberstringMenglish\@numberstringMenglish

```

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

```

332 \global\let\@numberstringFenglish=\@numberstringMenglish

```



```
333 \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.

```
334 \newcommand*\@NumberstringMenglish[2]{%
335   \let\@unitstring=\@Unitstringenglish
336   \let\@teenstring=\@Teenstringenglish
337   \let\@tenstring=\@Tenstringenglish
338   \def\@hundred{Hundred}\def\@thousand{Thousand}%
339   \def\@andname{and}%
340   \@numberstringenglish{#1}{#2}%
341 }%
342 \global\let\@NumberstringMenglish\@NumberstringMenglish
```

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

```
343 \global\let\@NumberstringFenglish=\@NumberstringMenglish
344 \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:

```
345 \newcommand*\@unitthstringenglish[1]{%
346   \ifcase#1\relax
347     zeroth%
348     \or first%
349     \or second%
350     \or third%
351     \or fourth%
352     \or fifth%
353     \or sixth%
354     \or seventh%
355     \or eighth%
356     \or ninth%
357   \fi
358 }%
359 \global\let\@unitthstringenglish\@unitthstringenglish
```

Next the tens:

```
360 \newcommand*\@tenthstringenglish[1]{%
361   \ifcase#1\relax
362     \or tenth%
363     \or twentieth%
364     \or thirtieth%
365     \or fortieth%
366     \or fiftieth%
367     \or sixtieth%
368     \or seventieth%
369     \or eightieth%
370     \or ninetieth%
371   \fi
372 }%
373 \global\let\@tenthstringenglish\@tenthstringenglish
```

The teens:

```
374 \newcommand*{\@@teenthstringenglish[1]}{%
375   \ifcase#1\relax
376     tenth%
377     \or eleventh%
378     \or twelfth%
379     \or thirteenth%
380     \or fourteenth%
381     \or fifteenth%
382     \or sixteenth%
383     \or seventeenth%
384     \or eighteenth%
385     \or nineteenth%
386   \fi
387 }%
388 \global\let\@@teenthstringenglish\@@teenthstringenglish
```

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

```
389 \newcommand*{\@@Unitthstringenglish[1]}{%
390   \ifcase#1\relax
391     Zeroth%
392     \or First%
393     \or Second%
394     \or Third%
395     \or Fourth%
396     \or Fifth%
397     \or Sixth%
398     \or Seventh%
399     \or Eighth%
400     \or Ninth%
401   \fi
402 }%
403 \global\let\@@Unitthstringenglish\@@Unitthstringenglish
```

The tens:

```
404 \newcommand*{\@@Tenthstringenglish[1]}{%
405   \ifcase#1\relax
406     \or Tenth%
407     \or Twentieth%
408     \or Thirtieth%
409     \or Fortieth%
410     \or Fiftieth%
411     \or Sixtieth%
412     \or Seventieth%
413     \or Eightieth%
414     \or Ninetieth%
415   \fi
416 }%
417 \global\let\@@Tenthstringenglish\@@Tenthstringenglish
```

The teens:

```

418 \newcommand*{\@@Teenthstringenglish[1]}{
419   \ifcase#1\relax
420     Tenth%
421     \or Eleventh%
422     \or Twelfth%
423     \or Thirteenth%
424     \or Fourteenth%
425     \or Fifteenth%
426     \or Sixteenth%
427     \or Seventeenth%
428     \or Eighteenth%
429     \or Nineteenth%
430   \fi
431 }%
432 \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.

```

433 \newcommand*{\@@ordinalstringenglish[2]}{
434   \@strctr=#1\relax
435   \ifnum#1>99999
436     \PackageError{fmtcount}{Out of range}%
437     {This macro only works for values less than 100000 (value given: \number\@strctr)}%
438   \else
439     \ifnum#1<0
440       \PackageError{fmtcount}{Negative numbers not permitted}%
441       {This macro does not work for negative numbers, however
442       you can try typing "minus" first, and then pass the modulus of
443       this number}%
444     \fi
445     \def#2{}%
446     \fi
447     \@strctr=#1\relax \divide\@strctr by 1000\relax
448     \ifnum\@strctr>9\relax
449       #1 is greater or equal to 10000
450       \divide\@strctr by 10
451       \ifnum\@strctr>1\relax
452         \let\@@fc@ordstr#2\relax
453         \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
454         \@strctr=#1\relax
455         \divide\@strctr by 1000\relax
456         \@FCmodulo{\@strctr}{10}%
457         \ifnum\@strctr>0\relax
458           \let\@@fc@ordstr#2\relax
459           \edef#2{\@@fc@ordstr-\@unitstring{\@strctr}}%
460         \fi
461       \else
462         \@strctr=#1\relax \divide\@strctr by 1000\relax

```

```

462 \@FCmodulo{\@strctr}{10}%
463 \let\@fc@ordstr#2\relax
464 \edef#2{\@fc@ordstr\@teenstring{\@strctr}}%
465 \fi
466 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
467 \ifnum\@strctr=0\relax
468 \let\@fc@ordstr#2\relax
469 \edef#2{\@fc@ordstr\ \@thousandth}%
470 \else
471 \let\@fc@ordstr#2\relax
472 \edef#2{\@fc@ordstr\ \@thousand}%
473 \fi
474 \else
475 \ifnum\@strctr>0\relax
476 \let\@fc@ordstr#2\relax
477 \edef#2{\@fc@ordstr\@unitstring{\@strctr}}%
478 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
479 \let\@fc@ordstr#2\relax
480 \ifnum\@strctr=0\relax
481 \edef#2{\@fc@ordstr\ \@thousandth}%
482 \else
483 \edef#2{\@fc@ordstr\ \@thousand}%
484 \fi
485 \fi
486 \fi
487 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
488 \divide\@strctr by 100
489 \ifnum\@strctr>0\relax
490 \ifnum#1>1000\relax
491 \let\@fc@ordstr#2\relax
492 \edef#2{\@fc@ordstr\ }%
493 \fi
494 \let\@fc@ordstr#2\relax
495 \edef#2{\@fc@ordstr\@unitstring{\@strctr}}%
496 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
497 \let\@fc@ordstr#2\relax
498 \ifnum\@strctr=0\relax
499 \edef#2{\@fc@ordstr\ \@hundredth}%
500 \else
501 \edef#2{\@fc@ordstr\ \@hundred}%
502 \fi
503 \fi
504 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
505 \ifnum#1>100\relax
506 \ifnum\@strctr>0\relax
507 \let\@fc@ordstr#2\relax
508 \edef#2{\@fc@ordstr\ \@andname\ }%
509 \fi
510 \fi

```

```

511 \ifnum \@strctr>19\relax
512   \@tmpstrctr=\@strctr
513   \divide \@strctr by 10\relax
514   \@FCmodulo{\@tmpstrctr}{10}%
515   \let \@fc@ordstr#2\relax
516   \ifnum \@tmpstrctr=0\relax
517     \edef#2{\@fc@ordstr\@tenthstring{\@strctr}}%
518   \else
519     \edef#2{\@fc@ordstr\@tenstring{\@strctr}}%
520   \fi
521   \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
522   \ifnum \@strctr>0\relax
523     \let \@fc@ordstr#2\relax
524     \edef#2{\@fc@ordstr-\@unitthstring{\@strctr}}%
525   \fi
526 \else
527   \ifnum \@strctr<10\relax
528     \ifnum \@strctr=0\relax
529       \ifnum#1<100\relax
530         \let \@fc@ordstr#2\relax
531         \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
532       \fi
533     \else
534       \let \@fc@ordstr#2\relax
535       \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
536     \fi
537   \else
538     \@FCmodulo{\@strctr}{10}%
539     \let \@fc@ordstr#2\relax
540     \edef#2{\@fc@ordstr\@teenthstring{\@strctr}}%
541   \fi
542 \fi
543 }%
544 \global\let \@ordinalstringenglish \@ordinalstringenglish

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

545 \newcommand*{\@ordinalstringMenglish}[2]{%
546   \let \@unitthstring=\@unitthstringenglish
547   \let \@teenthstring=\@teenthstringenglish
548   \let \@tenthstring=\@tenthstringenglish
549   \let \@unitstring=\@unitstringenglish
550   \let \@teenstring=\@teenstringenglish
551   \let \@tenstring=\@tenstringenglish
552   \def \@andname{and}%
553   \def \@hundred{hundred}\def \@thousand{thousand}%
554   \def \@hundredth{hundredth}\def \@thousandth{thousandth}%
555   \@ordinalstringenglish{#1}{#2}%
556 }%
557 \global\let \@ordinalstringMenglish \@ordinalstringMenglish

```

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

```
558 \global\let\@ordinalstringFenglish=\@ordinalstringMenglish
559 \global\let\@ordinalstringNenglish=\@ordinalstringMenglish
```

First letter of each word in upper case:

```
560 \newcommand*{\@OrdinalstringMenglish}[2]{%
561   \let\@unitthstring=\@Unitthstringenglish
562   \let\@teenthstring=\@Teenthstringenglish
563   \let\@tenthstring=\@Tenthstringenglish
564   \let\@unitstring=\@Unitstringenglish
565   \let\@teenstring=\@Teenstringenglish
566   \let\@tenstring=\@Tenstringenglish
567   \def\@andname{and}%
568   \def\@hundred{Hundred}\def\@thousand{Thousand}%
569   \def\@hundredth{Hundredth}\def\@thousandth{Thousandth}%
570   \@ordinalstringenglish{#1}{#2}%
571 }%
572 \global\let\@OrdinalstringMenglish\@OrdinalstringMenglish
```

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

```
573 \global\let\@OrdinalstringFenglish=\@OrdinalstringMenglish
574 \global\let\@OrdinalstringNenglish=\@OrdinalstringMenglish
```

### 10.1.5 fc-francais.def

```
575 \ProvidesFCLanguage{francais}[2013/08/17]%
576 \FCloadlang{french}%
```

Set francais to be equivalent to french.

```
577 \global\let\@ordinalMfrancais=\@ordinalMfrench
578 \global\let\@ordinalFfrancais=\@ordinalFfrench
579 \global\let\@ordinalNfrancais=\@ordinalNfrench
580 \global\let\@numberstringMfrancais=\@numberstringMfrench
581 \global\let\@numberstringFfrancais=\@numberstringFfrench
582 \global\let\@numberstringNfrancais=\@numberstringNfrench
583 \global\let\@NumberstringMfrancais=\@NumberstringMfrench
584 \global\let\@NumberstringFfrancais=\@NumberstringFfrench
585 \global\let\@NumberstringNfrancais=\@NumberstringNfrench
586 \global\let\@ordinalstringMfrancais=\@ordinalstringMfrench
587 \global\let\@ordinalstringFfrancais=\@ordinalstringFfrench
588 \global\let\@ordinalstringNfrancais=\@ordinalstringNfrench
589 \global\let\@OrdinalstringMfrancais=\@OrdinalstringMfrench
590 \global\let\@OrdinalstringFfrancais=\@OrdinalstringFfrench
591 \global\let\@OrdinalstringNfrancais=\@OrdinalstringNfrench
```

### 10.1.6 fc-french.def

Definitions for French.

```
592 \ProvidesFCLanguage{french}[2020/02/24]%
```

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.

```
593 \ifcsundef{fc@gl@let}{\global\let\fc@gl@let\let}{\PackageError{fmtcount}{Command already defined.}}
594 \protect\fc@gl@let\space already defined.}}
595 \ifcsundef{fc@gl@def}{\global\let\fc@gl@def\def}{\PackageError{fmtcount}{Command already defined.}}
596 \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'.
597 \gdef\fc@french@set@plural#1#2#3#4#5#6{%
598   \ifthenelse{\equal{#2}{reformed}}{%
599     \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#3}%
600   }{%
601     \ifthenelse{\equal{#2}{traditional}}{%
602       \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#4}%
603     }{%
604       \ifthenelse{\equal{#2}{reformed o}}{%
605         \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#5}%
606       }{%
607         \ifthenelse{\equal{#2}{traditional o}}{%
608           \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#6}%
609         }{%
610           \ifthenelse{\equal{#2}{always}}{%
611             \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{0}%
612           }{%
613             \ifthenelse{\equal{#2}{never}}{%
614               \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{1}%
615             }{%
616               \ifthenelse{\equal{#2}{multiple}}{%
617                 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{2}%
618               }{%
619                 \ifthenelse{\equal{#2}{multiple g-last}}{%
620                   \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{3}%
621                 }{%
622                   \ifthenelse{\equal{#2}{multiple l-last}}{%
623                     \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{4}%
624                   }{%
```

```

625 \ifthenelse{\equal{#2}{multiple lng-last}}{%
626 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{5}%
627 }{%
628 \ifthenelse{\equal{#2}{multiple ng-last}}{%
629 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{6}%
630 }{%
631 \PackageError{fmtcount}{Unexpected argument}{%
632 '#2' was unexpected: french option '#1 plural' expects 'reformed', 't
633 'reformed o', 'traditional o', 'always', 'never', 'multiple', 'multip
634 'multiple l-last', 'multiple lng-last', or 'multiple ng-last'.%
635 }}}}

```

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’.

```

636 \def\@tempa#1#2#3{%
637 \define@key{fcfrench}{#1 plural}[reformed]{%
638 \fc@french@set@plural{#1}{##1}{#2}{#3}{#3}%
639 }%

```

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.

```

640 \expandafter\@tempb\csname KV@fcfrench@#1 plural\endcsname
641 }%
642 \def\@tempb#1{%
643 \global\let#1#1
644 }%
645 \@tempa{vingt}{4}{5}
646 \@tempa{cent}{4}{5}
647 \@tempa{mil}{0}{0}
648 \@tempa{n-illion}{2}{6}
649 \@tempa{n-illiard}{2}{6}

```

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

```

650 \define@key{fcfrench}{all plural}[reformed]{%
651 \csname KV@fcfrench@vingt plural\endcsname{#1}%
652 \csname KV@fcfrench@cent plural\endcsname{#1}%
653 \csname KV@fcfrench@mil plural\endcsname{#1}%
654 \csname KV@fcfrench@n-illion plural\endcsname{#1}%
655 \csname KV@fcfrench@n-illiard plural\endcsname{#1}%
656 }%
657 \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

```

658 \define@key{fcfrench}{dash or space}[reformed]{%
659   \ifthenelse{\equal{#1}{traditional}}{%
660     \let\fc@frenchoptions@supermillion@dos\space%
661     \let\fc@frenchoptions@submillion@dos\space
662   }{%
663     \ifthenelse{\equal{#1}{reformed}\or\equal{#1}{1990}}{%
664       \let\fc@frenchoptions@supermillion@dos\space
665       \def\fc@frenchoptions@submillion@dos{-}%
666     }{%
667       \ifthenelse{\equal{#1}{always}}{%
668         \def\fc@frenchoptions@supermillion@dos{-}%
669         \def\fc@frenchoptions@submillion@dos{-}%
670       }{%
671         \PackageError{fmtcount}{Unexpected argument}{%
672           French option ‘dash or space’ expects ‘always’, ‘reformed’ or ‘traditional’
673         }
674       }%
675     }%
676   }%
677 }%
  
```

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

```

678 \define@key{fcfrench}{scale}[recursive]{%
679   \ifthenelse{\equal{#1}{long}}{%
680     \let\fc@poweroften\fc@@pot@longscalefrench
681   }{%
682     \ifthenelse{\equal{#1}{recursive}}{%
683       \let\fc@poweroften\fc@@pot@recursivefrench
684     }{%
685       \ifthenelse{\equal{#1}{short}}{%
686         \let\fc@poweroften\fc@@pot@shortscalefrench
687       }{%
688         \PackageError{fmtcount}{Unexpected argument}{%
689           French option ‘scale’ expects ‘long’, ‘recursive’ or ‘short’
690         }
691       }%
692     }%
693   }%
694 }%
  
```

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”

```

695 \define@key{fcfrench}{n-illiard upto}[infinity]{%
696   \ifthenelse{\equal{#1}{infinity}}{%
697     \def\fc@longscale@nilliard@upto{0}%
698   }{%
699     \ifthenelse{\equal{#1}{infty}}{%
700       \def\fc@longscale@nilliard@upto{0}%
701     }{%
702       \if Q\ifnum9<1#1Q\fi\else
703         \PackageError{fmtcount}{Unexpected argument}{%
704           French option ‘milliard threshold’ expects ‘infinity’, or equivalently ‘infty’, or a no
705           integer.}%
706       \fi
707       \def\fc@longscale@nilliard@upto{#1}%
708     }%
709 }%

```

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.

```

710 \def\@tempa#1{%
711   \define@key{fcfrench}{#1}[]{%
712     \PackageError{fmtcount}{Unexpected argument}{French option with key ‘#1’ does not take
713     any value}}%
714   \csgdef{KV@fcfrench@#1@default}{%
715     \fc@gl@def\fmtcount@french{#1}}%
716 }%
717 \@tempa{france}\@tempa{swiss}\@tempa{belgian}%

```

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

```

718 \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.

```

719 \define@key{fcfrench}{dialect}[france]{%
720   \ifthenelse{\equal{#1}{france}
721     \or\equal{#1}{swiss}
722     \or\equal{#1}{belgian}}{%
723     \def\fmtcount@french{#1}}{%
724     \PackageError{fmtcount}{Invalid value ‘#1’ to french option dialect key}
725     {Option ‘french’ can only take the values ‘france’,
726     ‘belgian’ or ‘swiss’}}%
727 \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’.

```

728 \define@key{fcfrench}{mil plural mark}[le]{%
729   \def\fc@frenchoptions@mil@plural@mark{#1}}
730 \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.

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

```

731 \gdef\fc@UpperCaseFirstLetter#1#2\@nil{%
732   \uppercase{#1}#2}

```

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

```

733 \gdef\fc@CaseIden#1\@nil{%
734   #1%
735 }%

```

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

```

736 \gdef\fc@UpperCaseAll#1\@nil{%
737   \uppercase{#1}%
738 }%

```

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

```

739 \global\let\fc@wcase\fc@CaseIden

```

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

```

740 \global\let\fc@gcase\fc@CaseIden

```

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

```

741 \gdef\fc@apply@gcase{%

```

First of all we expand whatever `\fc@wcase... \@nil` found within `\@tempa`.

```

742   \protected@edef\@tempa{\@tempa}%
743   \protected@edef\@tempa{\expandafter\fc@gcase\@tempa\@nil}%
744 }

```

`\ordinalMfrench`

```

745 \newcommand*{\@ordinalMfrench}[2]{%
746   \iffmtord@abbrv
747   \ifnum#1=1 %
748     \edef#2{\number#1\relax\noexpand\fmtord{er}}%
749   \else
750     \edef#2{\number#1\relax\noexpand\fmtord{e}}%
751   \fi
752 \else
753   \PackageWarning{fmtcount}{Non abbreviated ordinal finals ('eme) are
754     considered incorrect in French.}%

```

```

755 \ifnum#1=1 %
756   \edef#2{\number#1\relax\noexpand\fmtord{er}}%
757 \else
758   \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'eme}}%
759 \fi
760 \fi}
761 \global\let\@ordinalMfrench\@ordinalMfrench

```

@ordinalFfrench

```

762 \newcommand*{\@ordinalFfrench}[2]{%
763 \iffmtord@abbrv
764 \ifnum#1=1 %
765   \edef#2{\number#1\relax\noexpand\fmtord{re}}%
766 \else
767   \edef#2{\number#1\relax\noexpand\fmtord{e}}%
768 \fi
769 \else
770 \PackageWarning{fmtcount}{Non abbreviated ordinal finals ('eme) are
771   considered incorrect in French.}%
772 \ifnum#1=1 %
773   \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'ere}}%
774 \else
775   \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'eme}}%
776 \fi
777 \fi}
778 \global\let\@ordinalFfrench\@ordinalFfrench

```

In French neutral gender and masculine gender are formally identical.

```

779 \global\let\@ordinalNfrench\@ordinalMfrench

```

unitstringfrench

```

780 \newcommand*{\@unitstringfrench}[1]{%
781 \noexpand\fc@wcase
782 \ifcase#1 %
783 z\'ero%
784 or un%
785 or deux%
786 or trois%
787 or quatre%
788 or cinq%
789 or six%
790 or sept%
791 or huit%
792 or neuf%
793 \fi
794 \noexpand\@nil
795 }%
796 \global\let\@unitstringfrench\@unitstringfrench

```

tenstringfrench

```

797 \newcommand*{\@tenstringfrench}[1]{%

```

```

798 \noexpand\fc@wcase
799 \ifcase#1 %
800 \or dix%
801 \or vingt%
802 \or trente%
803 \or quarante%
804 \or cinquante%
805 \or soixante%
806 \or septante%
807 \or huitante%
808 \or nonante%
809 \or cent%
810 \fi
811 \noexpand\@nil
812 }%
813 \global\let\@@tenstringfrench\@@tenstringfrench

```

teenstringfrench

```

814 \newcommand*\@@teenstringfrench}[1]{%
815 \noexpand\fc@wcase
816 \ifcase#1 %
817   dix%
818 \or onze%
819 \or douze%
820 \or treize%
821 \or quatorze%
822 \or quinze%
823 \or seize%
824 \or dix\noexpand\@nil-\noexpand\fc@wcase sept%
825 \or dix\noexpand\@nil-\noexpand\fc@wcase huit%
826 \or dix\noexpand\@nil-\noexpand\fc@wcase neuf%
827 \fi
828 \noexpand\@nil
829 }%
830 \global\let\@@teenstringfrench\@@teenstringfrench

```

seventiesfrench

```

831 \newcommand*\@@seventiesfrench}[1]{%
832 \@tenstring{6}%
833 \ifnum#1=1 %
834 \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
835 \else
836 -%
837 \fi
838 \@teenstring{#1}%
839 }%
840 \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".

```
841 \newcommand*{\@eightiesfrench[1]}{%
842 \fc@wcase quatre\@nil-\noexpand\fc@wcase vingt%
843 \ifnum#1>0 %
844 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
845 s%
846 \fi
847 \noexpand\@nil
848 -\@unitstring{#1}%
849 \else
850 \ifcase\fc@frenchoptions@vingt@plural\space
851 s% 0: always
852 \or
853 % 1: never
854 \or
855 s% 2: multiple
856 \or
857 % 3: multiple g-last
858 \ifnum\count0=\count6\ifnum\count9=0 s\fi\fi
859 \or
860 % 4: multiple l-last
861 \ifnum\count9=1 %
862 \else
863 s%
864 \fi
865 \or
866 % 5: multiple lng-last
867 \ifnum\count9=1 %
868 \else
869 \ifnum\count0>0 %
870 s%
871 \fi
872 \fi
873 \or
874 % or 6: multiple ng-last
875 \ifnum\count0>0 %
876 s%
877 \fi
878 \fi
879 \noexpand\@nil
880 \fi
881 }%
```

```

882 \global\let\@@eightiesfrench\@@eightiesfrench
883 \newcommand*\@@ninetiesfrench}[1]{%
884 \fc@wcase quatre\@nil-\noexpand\fc@wcase vingt%
885 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
886 s%
887 \fi
888 \noexpand\@nil
889 -\@teenstring{#1}%
890 }%
891 \global\let\@@ninetiesfrench\@@ninetiesfrench
892 \newcommand*\@@seventiesfrenchswiss}[1]{%
893 \@tenstring{7}%
894 \ifnum#1=1\ \@andname\ \fi
895 \ifnum#1>1-\fi
896 \ifnum#1>0 \@unitstring{#1}\fi
897 }%
898 \global\let\@@seventiesfrenchswiss\@@seventiesfrenchswiss
899 \newcommand*\@@eightiesfrenchswiss}[1]{%
900 \@tenstring{8}%
901 \ifnum#1=1\ \@andname\ \fi
902 \ifnum#1>1-\fi
903 \ifnum#1>0 \@unitstring{#1}\fi
904 }%
905 \global\let\@@eightiesfrenchswiss\@@eightiesfrenchswiss
906 \newcommand*\@@ninetiesfrenchswiss}[1]{%
907 \@tenstring{9}%
908 \ifnum#1=1\ \@andname\ \fi
909 \ifnum#1>1-\fi
910 \ifnum#1>0 \@unitstring{#1}\fi
911 }%
912 \global\let\@@ninetiesfrenchswiss\@@ninetiesfrenchswiss

```

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

```

913 \newcommand*\fc@french@common{%
914 \let\fc@wcase\fc@CaseIden
915 \let\@unitstring=\@unitstringfrench
916 \let\@teenstring=\@teenstringfrench
917 \let\@tenstring=\@tenstringfrench
918 \def\@hundred{cent}%
919 \def\@andname{et}%
920 }%
921 \global\let\fc@french@common\fc@french@common
922 \newcommand*\@numberstringMfrenchswiss}[2]{%
923 \fc@french@common
924 \let\fc@gcase\fc@CaseIden
925 \let\@seventies=\@seventiesfrenchswiss
926 \let\@eighties=\@eightiesfrenchswiss
927 \let\@nineties=\@ninetiesfrenchswiss

```

```

928 \let\fc@nbrstr@preamble\@empty
929 \let\fc@nbrstr@postamble\@empty
930 \@@numberstringfrench{#1}{#2}}
931 \global\let\@numberstringMfrenchswiss\@numberstringMfrenchswiss
932 \newcommand*{\@numberstringMfrenchfrance}[2]{%
933 \fc@french@common
934 \let\fc@gcase\fc@CaseIden
935 \let\@seventies=\@seventiesfrench
936 \let\@eighties=\@eightiesfrench
937 \let\@nineties=\@ninetiesfrench
938 \let\fc@nbrstr@preamble\@empty
939 \let\fc@nbrstr@postamble\@empty
940 \@@numberstringfrench{#1}{#2}}
941 \global\let\@numberstringMfrenchfrance\@numberstringMfrenchfrance
942 \newcommand*{\@numberstringMfrenchbelgian}[2]{%
943 \fc@french@common
944 \let\fc@gcase\fc@CaseIden
945 \let\@seventies=\@seventiesfrenchswiss
946 \let\@eighties=\@eightiesfrench
947 \let\@nineties=\@ninetiesfrench
948 \let\fc@nbrstr@preamble\@empty
949 \let\fc@nbrstr@postamble\@empty
950 \@@numberstringfrench{#1}{#2}}
951 \global\let\@numberstringMfrenchbelgian\@numberstringMfrenchbelgian
952 \let\@numberstringMfrench=\@numberstringMfrenchfrance
953 \newcommand*{\@numberstringFfrenchswiss}[2]{%
954 \fc@french@common
955 \let\fc@gcase\fc@CaseIden
956 \let\@seventies=\@seventiesfrenchswiss
957 \let\@eighties=\@eightiesfrenchswiss
958 \let\@nineties=\@ninetiesfrenchswiss
959 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
960 \let\fc@nbrstr@postamble\@empty
961 \@@numberstringfrench{#1}{#2}}
962 \global\let\@numberstringFfrenchswiss\@numberstringFfrenchswiss
963 \newcommand*{\@numberstringFfrenchfrance}[2]{%
964 \fc@french@common
965 \let\fc@gcase\fc@CaseIden
966 \let\@seventies=\@seventiesfrench
967 \let\@eighties=\@eightiesfrench
968 \let\@nineties=\@ninetiesfrench
969 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
970 \let\fc@nbrstr@postamble\@empty
971 \@@numberstringfrench{#1}{#2}}
972 \global\let\@numberstringFfrenchfrance\@numberstringFfrenchfrance
973 \newcommand*{\@numberstringFfrenchbelgian}[2]{%
974 \fc@french@common
975 \let\fc@gcase\fc@CaseIden
976 \let\@seventies=\@seventiesfrenchswiss

```



```

977 \let\@eighties=\@eightiesfrench
978 \let\@nineties=\@ninetiesfrench
979 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
980 \let\fc@nbrstr@postamble\@empty
981 \@@numberstringfrench{#1}{#2}}
982 \global\let\@numberstringFfrenchbelgian\@numberstringFfrenchbelgian
983 \global\let\@numberstringFfrench=\@numberstringFfrenchfrance
984 \global\let\@ordinalstringNfrench\@ordinalstringMfrench
985 \newcommand*\@NumberstringMfrenchswiss}[2]{%
986 \fc@french@common
987 \let\fc@gcase\fc@UpperCaseFirstLetter
988 \let\@seventies=\@seventiesfrenchswiss
989 \let\@eighties=\@eightiesfrenchswiss
990 \let\@nineties=\@ninetiesfrenchswiss
991 \let\fc@nbrstr@preamble\@empty
992 \let\fc@nbrstr@postamble\fc@apply@gcase
993 \@@numberstringfrench{#1}{#2}}
994 \global\let\@NumberstringMfrenchswiss\@NumberstringMfrenchswiss
995 \newcommand*\@NumberstringMfrenchfrance}[2]{%
996 \fc@french@common
997 \let\fc@gcase\fc@UpperCaseFirstLetter
998 \let\@seventies=\@seventiesfrench
999 \let\@eighties=\@eightiesfrench
1000 \let\@nineties=\@ninetiesfrench
1001 \let\fc@nbrstr@preamble\@empty
1002 \let\fc@nbrstr@postamble\fc@apply@gcase
1003 \@@numberstringfrench{#1}{#2}}
1004 \global\let\@NumberstringMfrenchfrance\@NumberstringMfrenchfrance
1005 \newcommand*\@NumberstringMfrenchbelgian}[2]{%
1006 \fc@french@common
1007 \let\fc@gcase\fc@UpperCaseFirstLetter
1008 \let\@seventies=\@seventiesfrenchswiss
1009 \let\@eighties=\@eightiesfrench
1010 \let\@nineties=\@ninetiesfrench
1011 \let\fc@nbrstr@preamble\@empty
1012 \let\fc@nbrstr@postamble\fc@apply@gcase
1013 \@@numberstringfrench{#1}{#2}}
1014 \global\let\@NumberstringMfrenchbelgian\@NumberstringMfrenchbelgian
1015 \global\let\@NumberstringMfrench=\@NumberstringMfrenchfrance
1016 \newcommand*\@NumberstringFfrenchswiss}[2]{%
1017 \fc@french@common
1018 \let\fc@gcase\fc@UpperCaseFirstLetter
1019 \let\@seventies=\@seventiesfrenchswiss
1020 \let\@eighties=\@eightiesfrenchswiss
1021 \let\@nineties=\@ninetiesfrenchswiss
1022 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
1023 \let\fc@nbrstr@postamble\fc@apply@gcase
1024 \@@numberstringfrench{#1}{#2}}
1025 \global\let\@NumberstringFfrenchswiss\@NumberstringFfrenchswiss

```

```

1026 \newcommand*{\@NumberstringFfrenchfrance}[2]{%
1027 \fc@french@common
1028 \let\fc@gcase\fc@UpperCaseFirstLetter
1029 \let\@seventies=\@seventiesfrench
1030 \let\@eighties=\@eightiesfrench
1031 \let\@nineties=\@ninetiesfrench
1032 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
1033 \let\fc@nbrstr@postamble\fc@apply@gcase
1034 \@@numberstringfrench{#1}{#2}}
1035 \global\let\@NumberstringFfrenchfrance\@NumberstringFfrenchfrance
1036 \newcommand*{\@NumberstringFfrenchbelgian}[2]{%
1037 \fc@french@common
1038 \let\fc@gcase\fc@UpperCaseFirstLetter
1039 \let\@seventies=\@seventiesfrenchswiss
1040 \let\@eighties=\@eightiesfrench
1041 \let\@nineties=\@ninetiesfrench
1042 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
1043 \let\fc@nbrstr@postamble\fc@apply@gcase
1044 \@@numberstringfrench{#1}{#2}}
1045 \global\let\@NumberstringFfrenchbelgian\@NumberstringFfrenchbelgian
1046 \global\let\@NumberstringFfrench=\@NumberstringFfrenchfrance
1047 \global\let\@NumberstringNfrench\@NumberstringMfrench
1048 \newcommand*{\@ordinalstringMfrenchswiss}[2]{%
1049 \fc@french@common
1050 \let\fc@gcase\fc@CaseIden
1051 \let\fc@first\fc@@firstfrench
1052 \let\@seventies=\@seventiesfrenchswiss
1053 \let\@eighties=\@eightiesfrenchswiss
1054 \let\@nineties=\@ninetiesfrenchswiss
1055 \@@ordinalstringfrench{#1}{#2}%
1056 }%
1057 \global\let\@ordinalstringMfrenchswiss\@ordinalstringMfrenchswiss
1058 \newcommand*\fc@@firstfrench{premier}
1059 \global\let\fc@@firstfrench\fc@@firstfrench

1060 \newcommand*\fc@@firstFfrench{premi\protect\`ere}
1061 \global\let\fc@@firstFfrench\fc@@firstFfrench
1062 \newcommand*{\@ordinalstringMfrenchfrance}[2]{%
1063 \fc@french@common
1064 \let\fc@gcase\fc@CaseIden
1065 \let\fc@first=\fc@@firstfrench
1066 \let\@seventies=\@seventiesfrench
1067 \let\@eighties=\@eightiesfrench
1068 \let\@nineties=\@ninetiesfrench
1069 \@@ordinalstringfrench{#1}{#2}}
1070 \global\let\@ordinalstringMfrenchfrance\@ordinalstringMfrenchfrance
1071 \newcommand*{\@ordinalstringMfrenchbelgian}[2]{%
1072 \fc@french@common
1073 \let\fc@gcase\fc@CaseIden
1074 \let\fc@first=\fc@@firstfrench

```

```

1075 \let\@seventies=\@seventiesfrench
1076 \let\@eighties=\@eightiesfrench
1077 \let\@nineties=\@ninetiesfrench
1078 \@ordinalstringfrench{#1}{#2}%
1079 }%
1080 \global\let\@ordinalstringMfrenchbelgian\@ordinalstringMfrenchbelgian
1081 \global\let\@ordinalstringMfrench=\@ordinalstringMfrenchfrance
1082 \newcommand*\@ordinalstringFfrenchswiss}[2]{%
1083 \fc@french@common
1084 \let\fc@gcase\fc@CaseIden
1085 \let\fc@first\fc@@firstFfrench
1086 \let\@seventies=\@seventiesfrenchswiss
1087 \let\@eighties=\@eightiesfrenchswiss
1088 \let\@nineties=\@ninetiesfrenchswiss
1089 \@ordinalstringfrench{#1}{#2}%
1090 }%
1091 \global\let\@ordinalstringFfrenchswiss\@ordinalstringFfrenchswiss
1092 \newcommand*\@ordinalstringFfrenchfrance}[2]{%
1093 \fc@french@common
1094 \let\fc@gcase\fc@CaseIden
1095 \let\fc@first=\fc@@firstFfrench
1096 \let\@seventies=\@seventiesfrench
1097 \let\@eighties=\@eightiesfrench
1098 \let\@nineties=\@ninetiesfrench
1099 \@ordinalstringfrench{#1}{#2}%
1100 }%
1101 \global\let\@ordinalstringFfrenchfrance\@ordinalstringFfrenchfrance
1102 \newcommand*\@ordinalstringFfrenchbelgian}[2]{%
1103 \fc@french@common
1104 \let\fc@gcase\fc@CaseIden
1105 \let\fc@first=\fc@@firstFfrench
1106 \let\@seventies=\@seventiesfrench
1107 \let\@eighties=\@eightiesfrench
1108 \let\@nineties=\@ninetiesfrench
1109 \@ordinalstringfrench{#1}{#2}%
1110 }%
1111 \global\let\@ordinalstringFfrenchbelgian\@ordinalstringFfrenchbelgian
1112 \global\let\@ordinalstringFfrench=\@ordinalstringFfrenchfrance
1113 \global\let\@ordinalstringNfrench\@ordinalstringMfrench
1114 \newcommand*\@OrdinalstringMfrenchswiss}[2]{%
1115 \fc@french@common
1116 \let\fc@gcase\fc@UpperCaseFirstLetter
1117 \let\fc@first=\fc@@firstfrench
1118 \let\@seventies=\@seventiesfrenchswiss
1119 \let\@eighties=\@eightiesfrenchswiss
1120 \let\@nineties=\@ninetiesfrenchswiss
1121 \@ordinalstringfrench{#1}{#2}%
1122 }%
1123 \global\let\@OrdinalstringMfrenchswiss\@OrdinalstringMfrenchswiss

```

```

1124 \newcommand*{\@OrdinalstringMfrenchfrance}[2]{%
1125 \fc@french@common
1126 \let\fc@gcase\fc@UpperCaseFirstLetter
1127 \let\fc@first\fc@@firstfrench
1128 \let\@seventies=\@seventiesfrench
1129 \let\@eighties=\@eightiesfrench
1130 \let\@nineties=\@ninetiesfrench
1131 \@ordinalstringfrench{#1}{#2}%
1132 }%
1133 \global\let\@OrdinalstringMfrenchfrance\@OrdinalstringMfrenchfrance
1134 \newcommand*{\@OrdinalstringMfrenchbelgian}[2]{%
1135 \fc@french@common
1136 \let\fc@gcase\fc@UpperCaseFirstLetter
1137 \let\fc@first\fc@@firstfrench
1138 \let\@seventies=\@seventiesfrench
1139 \let\@eighties=\@eightiesfrench
1140 \let\@nineties=\@ninetiesfrench
1141 \@ordinalstringfrench{#1}{#2}%
1142 }%
1143 \global\let\@OrdinalstringMfrenchbelgian\@OrdinalstringMfrenchbelgian
1144 \global\let\@OrdinalstringMfrench=\@OrdinalstringMfrenchfrance
1145 \newcommand*{\@OrdinalstringFfrenchswiss}[2]{%
1146 \fc@french@common
1147 \let\fc@gcase\fc@UpperCaseFirstLetter
1148 \let\fc@first\fc@@firstfrench
1149 \let\@seventies=\@seventiesfrenchswiss
1150 \let\@eighties=\@eightiesfrenchswiss
1151 \let\@nineties=\@ninetiesfrenchswiss
1152 \@ordinalstringfrench{#1}{#2}%
1153 }%
1154 \global\let\@OrdinalstringFfrenchswiss\@OrdinalstringFfrenchswiss
1155 \newcommand*{\@OrdinalstringFfrenchfrance}[2]{%
1156 \fc@french@common
1157 \let\fc@gcase\fc@UpperCaseFirstLetter
1158 \let\fc@first\fc@@firstFfrench
1159 \let\@seventies=\@seventiesfrench
1160 \let\@eighties=\@eightiesfrench
1161 \let\@nineties=\@ninetiesfrench
1162 \@ordinalstringfrench{#1}{#2}%
1163 }%
1164 \global\let\@OrdinalstringFfrenchfrance\@OrdinalstringFfrenchfrance
1165 \newcommand*{\@OrdinalstringFfrenchbelgian}[2]{%
1166 \fc@french@common
1167 \let\fc@gcase\fc@UpperCaseFirstLetter
1168 \let\fc@first\fc@@firstFfrench
1169 \let\@seventies=\@seventiesfrench
1170 \let\@eighties=\@eightiesfrench
1171 \let\@nineties=\@ninetiesfrench
1172 \@ordinalstringfrench{#1}{#2}%

```

```

1173 }%
1174 \global\let\@OrdinalstringFfrenchbelgian\@OrdinalstringFfrenchbelgian
1175 \global\let\@OrdinalstringFfrench=\@OrdinalstringFfrenchfrance
1176 \global\let\@OrdinalstringNfrench\@OrdinalstringMfrench

```

`\fc@@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.

```

1177 \ifcsundef{fc@@do@plural@mark}{}%
1178 {\PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1179   '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.

```

1180 \def\fc@@do@plural@mark#1{%
1181   \ifcase\count10 %
1182     #1% 0=always
1183     \or% 1=never
1184     \or% 2=multiple
1185     \ifnum\count1>1 %
1186       #1%
1187       \fi
1188     \or% 3= multiple g-last
1189     \ifnum\count1>1 %
1190       \ifnum\count0=\count6 %
1191         #1%
1192         \fi
1193       \fi
1194     \or% 4= multiple l-last
1195     \ifnum\count1>1 %
1196       \ifnum\count9=1 %
1197         \else
1198         #1%
1199         \fi
1200       \fi
1201     \or% 5= multiple lng-last
1202     \ifnum\count1>1 %
1203       \ifnum\count9=1 %
1204         \else
1205         \if\count0>\count6 %
1206         #1%
1207         \fi

```

```

1208     \fi
1209     \fi
1210 \or% 6= multiple ng-last
1211     \ifnum\count1>1 %
1212         \ifnum\count0>\count6 %
1213             #1%
1214     \fi
1215     \fi
1216 \fi
1217}%
1218 \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.

```

1219 \ifcsundef\fc@@nbrstr@Fpreamble\{}{%
1220 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1221 'fc@@nbrstr@Fpreamble'}}

```

`\@nbrstr@Fpreamble`

```

1222 \def\fc@@nbrstr@Fpreamble{%
1223 \fc@read@unit{\count1}{0}%
1224 \ifnum\count1=1 %
1225     \let\fc@wcase@save\fc@wcase
1226     \def\fc@wcase{\noexpand\fc@wcase}%
1227     \def\@nil{\noexpand\@nil}%
1228     \let\fc@nbrstr@postamble\fc@@nbrstr@Fpostamble
1229 \fi
1230}%
1231 \global\let\fc@@nbrstr@Fpreamble\fc@@nbrstr@Fpreamble

```

`\@nbrstr@Fpostamble`

```

1232 \def\fc@@nbrstr@Fpostamble{%
1233 \let\fc@wcase\fc@wcase@save
1234 \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
1235 \def\@tempd{un}%
1236 \ifx\@tempc\@tempd
1237     \let\@tempc\@tempa
1238     \edef\@tempa{\@tempb\fc@wcase une\@nil}%
1239 \fi
1240}%
1241 \global\let\fc@@nbrstr@Fpostamble\fc@@nbrstr@Fpostamble

```

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

```

1242 \ifcsundef\fc@@pot@longscalefrench\{}{%
1243 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1244 '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

```

1245 \def\fc@pot@longscalefrench#1#2#3{%
1246   {%

```

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

```

1247   \edef\@tempb{\number#1}%

```

Let `\count1` be the plural value.

```

1248   \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  $r$ .

```

1249   \count2\count0 %
1250   \divide\count2 by 6 %
1251   \count3\count2 %
1252   \multiply\count3 by 6 %
1253   \count3-\count3 %
1254   \advance\count3 by \count0 %
1255   \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)”.

```

1256   \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.

```

1257   \edef\@temp{%
1258   \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.

```

1259       1%
1260   \else
1261       \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`.

```

1262       \ifnum\fc@longscale@nilliard@upto=0 %

```

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

```

1263           2%
1264   \else

```

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

```

1265         \ifnum\count2>\fc@longscale@nilliard@upto
1266             1%
1267         \else
1268             2%
1269         \fi
1270     \fi
1271 \else
1272     2%
1273 \fi
1274 \fi
1275 }%
1276 \ifnum\@temph=1 %

```

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

```

1277     \count10=\fc@frenchoptions@mil@plural\space
1278     \edef\@tempe{%
1279         \noexpand\fc@wcase
1280         mil%
1281         \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
1282         \noexpand\@nil
1283     }%
1284 \else
1285     % weight >= 6
1286     \expandafter\fc@@latin@cardinal@prefix\expandafter{\the\count2}\@tempg
1287     % now form the xxx-illion(s) or xxx-illiard(s) word
1288     \ifnum\count3>2 %
1289         \toks10{illiard}%
1290         \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
1291     \else
1292         \toks10{illion}%
1293         \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1294     \fi
1295     \edef\@tempe{%
1296         \noexpand\fc@wcase
1297         \@tempg
1298         \the\toks10 %
1299         \fc@@do@plural@mark s%
1300         \noexpand\@nil
1301     }%
1302 \fi
1303 \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.

```

1304     \let\@tempe\@empty
1305     \def\@temph{0}%
1306 \fi
1307 \else

```



Case of  $w = 0$ .

```
1308 \let\@tempe\@empty
1309 \def\@temph{0}%
1310 \fi
```

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

```
1311 \expandafter\toks\expandafter1\expandafter{\@tempe}%
1312 \toks0{#2}%
1313 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
1314 \expandafter
1315 }\@tempa
1316 }%
1317 \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.

```
1318 \ifcsundef\fc@pot@shortscalefrench\{}{%
1319 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1320 '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

```
1321 \def\fc@pot@shortscalefrench#1#2#3{%
1322 {%
```

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

```
1323 \edef\@tempb{\number#1}%
```

And let `\count1` be the plural value.

```
1324 \count1=\@tempb
```

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

```
1325 \count2\count0 %
1326 \divide\count2 by 3 %
1327 \advance\count2 by -1 %
```

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

```
1328 \ifnum\count0>0 % If weight>=3, i.e we do have to append thousand or n-illion(s)/n-illiard(s)
```

```

1329 \ifnum\count1>0 % we have at least one thousand/n-illion/n-illiard
1330 \ifnum\count2=0 %
1331 \def\@temph{1}%
1332 \count1=\fc@frenchoptions@mil@plural\space
1333 \edef\@tempe{%
1334 mil%
1335 \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
1336 }%
1337 \else
1338 \def\@temph{2}%
1339 % weight >= 6
1340 \expandafter\fc@@latin@cardinal@prefix\expandafter{\the\count2}\@tempg
1341 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1342 \edef\@tempe{%
1343 \noexpand\fc@wcase
1344 \@tempg
1345 illion%
1346 \fc@@do@plural@mark s%
1347 \noexpand\@nil
1348 }%
1349 \fi
1350 \else

```

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

```

1351 \def\@temph{0}%
1352 \let\@tempe\@empty
1353 \fi
1354 \else

```

Here  $w = 0$ .

```

1355 \def\@temph{0}%
1356 \let\@tempe\@empty
1357 \fi
1358 % now place into \@tempa the assignment of results \cs{@temph} and \cs{@tempe} to to \text
1359 % \texttt{\#3} for further propagation after closing brace.
1360 % \begin{macrocode}
1361 \expandafter\toks\expandafter1\expandafter{\@tempe}%
1362 \toks0{\#2}%
1363 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
1364 \expandafter
1365 }\@tempa
1366 }%
1367 \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.

```

1368 \ifcsundef{fc@@pot@recursivefrench}{\%
1369 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1370 '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

```
1371 \def\fc@pot@recursivefrench#1#2#3{%
1372   {%
```

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

```
1373   \edef\@tempb{\number#1}%
1374   \let\@tempa\@tempa
```

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

```
1375   \count1=\@tempb\space
```

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

```
1376   \ifnum\count1>0 %
1377     \count2\count0 %
1378     \divide\count2 by 9 %
1379     \advance\count2 by -1 %
1380     \let\@tempe\@empty
1381     \edef\@tempf{\fc@frenchoptions@supermillion@dos
1382       de\fc@frenchoptions@supermillion@dos\fc@wcase milliards\@nil}%
1383     \count11\count0 %
1384     \ifnum\count2>0 %
1385       \count3\count2 %
1386       \count3-\count3 %
1387       \multiply\count3 by 9 %
1388       \advance\count11 by \count3 %
1389       \loop
1390         % (\count2, \count3) <- (\count2 div 2, \count2 mod 2)
1391         \count3\count2 %
1392         \divide\count3 by 2 %
1393         \multiply\count3 by 2 %
1394         \count3-\count3 %
1395         \advance\count3 by \count2 %
1396         \divide\count2 by 2 %
1397         \ifnum\count3=1 %
1398           \let\@tempg\@tempe
1399           \edef\@tempe{\@tempg\@tempf}%
1400         \fi
1401         \let\@tempg\@tempf
1402         \edef\@tempf{\@tempg\@tempg}%
1403         \ifnum\count2>0 %
1404           \repeat
1405         \fi
1406       \divide\count11 by 3 %
```

```

1407 \ifcase\count11 % 0 .. 5
1408   % 0 => d milliard(s) (de milliards)*
1409   \def\@temph{2}%
1410   \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
1411 \or % 1 => d mille milliard(s) (de milliards)*
1412   \def\@temph{1}%
1413   \count10=\fc@frenchoptions@mil@plural\space
1414 \or % 2 => d million(s) (de milliards)*
1415   \def\@temph{2}%
1416   \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1417 \or % 3 => d milliard(s) (de milliards)*
1418   \def\@temph{2}%
1419   \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
1420 \or % 4 => d mille milliards (de milliards)*
1421   \def\@temph{1}%
1422   \count10=\fc@frenchoptions@mil@plural\space
1423 \else % 5 => d million(s) (de milliards)*
1424   \def\@temph{2}%
1425   \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1426 \fi
1427 \let\@tempg\@tempe
1428 \edef\@tempf{%
1429   \ifcase\count11 % 0 .. 5
1430   \or
1431     mil\fc@@do@plural@mark \fc@frenchoptions@mil@plural@mark
1432   \or
1433     million\fc@@do@plural@mark s%
1434   \or
1435     milliard\fc@@do@plural@mark s%
1436   \or
1437     mil\fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
1438     \noexpand\@nil\fc@frenchoptions@supermillion@dos
1439     \noexpand\fc@wcase milliards% 4
1440   \or
1441     million\fc@@do@plural@mark s%
1442     \noexpand\@nil\fc@frenchoptions@supermillion@dos
1443     de\fc@frenchoptions@supermillion@dos\noexpand\fc@wcase  milliards% 5
1444   \fi
1445 }%
1446 \edef\@tempe{%
1447   \ifx\@tempf\@empty\else
1448     \expandafter\fc@wcase\@tempf\@nil
1449   \fi
1450 \@tempg
1451 }%
1452 \else
1453   \def\@temph{0}%
1454   \let\@tempe\@empty
1455 \fi

```

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

```

1456   \expandafter\toks\expandafter1\expandafter{\@tempb}%
1457   \toks0{#2}%
1458   \edef\@tempa{\the\toks0 \@tempa \def\noexpand#3{\the\toks1}}%
1459   \expandafter
1460   }\@tempa
1461 }%
1462 \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.

```

1463 \ifcsundef{fc@muladdfrench}{}%
1464 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1465   '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

```

1466 \def\fc@muladdfrench#1#2#3{%
1467   {%

```

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

```

1468   \edef\@tempc{#1}%
1469   \edef\@tempd{#2}%
1470   \edef\@tempf{#3}%
1471   \let\@tempc\@tempc
1472   \let\@tempd\@tempd

```

First we want to do the “multiplication” of  $d \Rightarrow \@tempd$  and of  $10^w \Rightarrow \@tempf$ . So, prior to this we do some preprocessing of  $d \Rightarrow \@tempd$ : we force `\@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”.

```

1473   \ifnum\@tempc=1 %
1474     \ifnum\count9=1 %
1475       \let\@tempd\empty
1476     \fi
1477   \fi

```

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

```

1478 \edef\@tempg{%
1479   \@tempd
1480   \ifx\@tempd\@empty\else
1481     \ifx\@tempf\@empty\else
1482       \ifcase\count9 %
1483         \or
1484         \fc@frenchoptions@submillion@dos
1485       \or
1486       \fc@frenchoptions@supermillion@dos
1487     \fi
1488   \fi
1489 \@tempf
1491 }%
```

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

```

1492 \edef\@temph{%
1493   \@tempa
1494   \ifx\@tempa\@empty\else
1495     \ifx\@tempg\@empty\else
1496       \ifcase\count8 %
1497         \or
1498         \fc@frenchoptions@submillion@dos
1499       \or
1500       \fc@frenchoptions@supermillion@dos
1501     \fi
1502   \fi
1503 \@tempg
1504 }%
```

Now propagate the result — i.e. the expansion of  $\text{\@temph}$  — into macro  $\text{\@tempa}$  after closing brace.

```

1506 \def\@tempb##1{\def\@tempa{\def\@tempa{##1}}}%
1507 \expandafter\@tempb\expandafter{\@temph}%
1508 \expandafter
1509 }\@tempa
1510 }%
```

```

1511 \global\let\fc@muladdfrench\fc@muladdfrench
```

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

```

1512 \ifcsundef\fc@lthundredstringfrench\{}{%
1513   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1514     ‘\fc@lthundredstringfrench’}}
```

The number to format is not passed as an argument to this macro, instead each digit of it is in a  $\text{\fc@digit@<w>}$  macro after this number has been parsed. So the only thing that

`\fc@lthundredstringfrench` needs 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.

```
1515 \def\fc@lthundredstringfrench#1{%
1516   {%
```

First save arguments into local temporary macro.

```
1517   \let\@tempc#1%
```

Read units  $d_w$  to `\count1`.

```
1518   \fc@read@unit{\count1}{\count0}%
```

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

```
1519   \count3\count0 %
```

```
1520   \advance\count3 1 %
```

```
1521   \fc@read@unit{\count2}{\count3}%
```

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

```
1522   \edef\@tempa{%
```

```
1523     \@tempc
```

```
1524     \ifnum\count2>1 %
```

```
1525       % 20 .. 99
```

```
1526       \ifnum\count2>6 %
```

```
1527         % 70 .. 99
```

```
1528         \ifnum\count2<8 %
```

```
1529           % 70 .. 79
```

```
1530           \@seventies{\count1}%
```

```
1531         \else
```

```
1532           % 80..99
```

```
1533           \ifnum\count2<9 %
```

```
1534             % 80 .. 89
```

```
1535             \@eighties{\count1}%
```

```
1536           \else
```

```
1537             % 90 .. 99
```

```
1538             \@nineties{\count1}%
```

```
1539           \fi
```

```
1540         \fi
```

```
1541       \else
```

```
1542         % 20..69
```

```
1543         \@tenstring{\count2}%
```

```
1544         \ifnum\count1>0 %
```

```
1545           % x1 .. x0
```

```
1546           \ifnum\count1=1 %
```

```
1547             % x1
```

```
1548             \fc@frenchoptions@submillion@dos\andname\fc@frenchoptions@submillion@dos
```

```
1549           \else
```

```
1550             % x2 .. x9
```

```

1551         -%
1552         \fi
1553         \@unitstring{\count1}%
1554     \fi
1555     \fi
1556 \else
1557     % 0 .. 19
1558     \ifnum\count2=0 % when tens = 0
1559     % 0 .. 9
1560     \ifnum\count1=0 % when units = 0
1561     % \count3=1 when #1 = 0, i.e. only for the unit of the top level number
1562     \ifnum\count3=1 %
1563     \ifnum\fc@max@weight=0 %
1564     \@unitstring{0}%
1565     \fi
1566     \fi
1567     \else
1568     % 1 .. 9
1569     \@unitstring{\count1}%
1570     \fi
1571     \else
1572     % 10 .. 19
1573     \@teenstring{\count1}%
1574     \fi
1575 \fi
1576 }%

```

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

```

1577 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
1578 \expandafter\@tempb\expandafter{\@tempa}%
1579 \expandafter
1580 }\@tempa
1581 }%
1582 \global\let\fc@lthundredstringfrench\fc@lthundredstringfrench

```

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

```

1583 \ifcsundef\fc@lthousandstringfrench}{%
1584 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1585 'fc@lthousandstringfrench'}}

```

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)

```

1586 \def\fc@lthousandstringfrench#1{%
1587 {%

```



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

```
1588 \count4\count0 %
1589 \advance\count4 by 2 %
1590 \fc@read@unit{\count2 }{\count4 }%
```

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

```
1591 \advance\count4 by -1 %
1592 \count3\count4 %
1593 \advance\count3 by -1 %
1594 \fc@check@nonzeros{\count3 }{\count4 }\@tempa
```

Compute plural mark of 'cent' into \@temps.

```
1595 \edef\@temps{%
1596 \ifcase\fc@frenchoptions@cent@plural\space
1597 % 0 => always
1598 s%
1599 \or
1600 % 1 => never
1601 \or
1602 % 2 => multiple
1603 \ifnum\count2>1s\fi
1604 \or
1605 % 3 => multiple g-last
1606 \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count0=\count6s\fi\fi\fi
1607 \or
1608 % 4 => multiple l-last
1609 \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count9=0s\else\ifnum\count9=2s\fi\fi\fi\fi
1610 \fi
1611 }%
1612 % compute spacing after cent(s?) into \@tempb
1613 \expandafter\let\expandafter\@tempb
1614 \ifnum\@tempa>0 \fc@frenchoptions@submillion@dos\else\@empty\fi
1615 % now place into \@tempa the hundreds
1616 \edef\@tempa{%
1617 \ifnum\count2=0 %
1618 \else
1619 \ifnum\count2=1 %
1620 \expandafter\fc@wcase\@hundred\@nil
1621 \else
1622 \@unitstring{\count2}\fc@frenchoptions@submillion@dos
1623 \noexpand\fc@wcase\@hundred\@temps\noexpand\@nil
1624 \fi
1625 \@tempb
1626 \fi
1627 }%
1628 % now append to \@tempa the ten and unit
1629 \fc@lthundredstringfrench\@tempa
```

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

```
1630 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
```

```

1631 \expandafter\@tempb\expandafter{\@tempa}%
1632 \expandafter
1633 }\@tempa
1634 }%
1635 \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.

```

1636 \ifcsundef{@@numberstringfrench}{}%
1637 \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

```

1638 \def\@@numberstringfrench#1#2{%
1639  {%

```

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

```

1640 \edef\@tempa{#1}%
1641 \expandafter\fc@number@parser\expandafter{\@tempa}%
1642 \ifnum\fc@min@weight<0 %
1643 \PackageError{fmtcount}{Out of range}%
1644 {This macro does not work with fractional numbers}%
1645 \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'.

```

1646 \edef\@tempa{\ifcase\fc@sign@case\space\or\fc@wcase plus\@nil\or\fc@wcase moins\@nil\fi}%
1647 \fc@nbrstr@preamble
1648 \fc@@nbrstrfrench@inner
1649 \fc@nbrstr@postamble

```

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

```

1650 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
1651 \expandafter\@tempb\expandafter{\@tempa}%
1652 \expandafter
1653 }\@tempa
1654 }%
1655 \global\let\@@numberstringfrench\@@numberstringfrench

```

@@nbrstrfrench@innerCommon 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.

```

1656 \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.

```

1657 \count0=\fc@max@weight
1658 \divide\count0 by 3 %
1659 \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`.

```
1660 \fc@intpart@find@last{\count5 }%
1661 \count6\count5 %
1662 \divide\count6 3 %
1663 \multiply\count6 3 %
1664 \count8=0 %
1665 \loop
```

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

```
1666 \count1\count0 %
1667 \advance\count1 by 2 %
1668 \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`.

```
1669 \fc@powerof ten \@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`

```
1670 \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  $10^w$ .

```
1671 \fc@muladdfrench \@tempt \@tempd \@tempb
```

Then iterate.

```
1672 \count8\count9 %
1673 \advance\count0 by -3 %
1674 \ifnum\count6>\count0 \else
1675 \repeat
1676 }%
1677 \global\let\fc@nbrstrfrench@inner\fc@nbrstrfrench@inner
```

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

```
1678 \ifcsundef{\@@ordinalstringfrench}{\}%
1679 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1680 '@@ordinalstringfrench'}}
```

Arguments are as follows:

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

```
1681 \def\@@ordinalstringfrench#1#2{%
1682 {%
```

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

```
1683 \edef\@tempa{#1}%
1684 \expandafter\fc@number@parser\expandafter{\@tempa}%
```

```

1685 \ifnum\fc@min@weight<0 %
1686   \PackageError{fmtcount}{Out of range}%
1687   {This macro does not work with fractional numbers}%
1688 \fi
1689 \ifnum\fc@sign@case>0 %
1690   \PackageError{fmtcount}{Out of range}%
1691   {This macro does with negative or explicitly marked as positive numbers}%
1692 \fi

```

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

```

1693 \ifnum\fc@max@weight=0 %
1694   \ifnum\csname fc@digit@0\endcsname=1 %
1695     \count0=1 %
1696   \else
1697     \count0=0 %
1698   \fi
1699 \else
1700   \count0=0 %
1701 \fi
1702 \ifnum\count0=1 %
1703   \expandafter\@firstoftwo
1704 \else
1705   \expandafter\@secondoftwo
1706 \fi

```

{%  
`\protected@edef\@tempa{\expandafter\fc@wcase\fc@first\@nil}`%  
}%

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

```

1710 {%
1711   \def\@tempa##1{%
1712     \expandafter\edef\csname fc@frenchoptions@##1@plural\endcsname{%
1713       \ifcase\csname fc@frenchoptions@##1@plural\endcsname\space
1714       0% 0: always => always
1715       \or
1716       1% 1: never => never
1717       \or
1718       6% 2: multiple => multiple ng-last
1719       \or
1720       1% 3: multiple g-last => never
1721       \or
1722       5% 4: multiple l-last => multiple lng-last
1723       \or
1724       5% 5: multiple lng-last => multiple lng-last
1725       \or
1726       6% 6: multiple ng-last => multiple ng-last
1727     \fi

```

```

1728     }%
1729   }%
1730   \@tempa{vingt}%
1731   \@tempa{cent}%
1732   \@tempa{mil}%
1733   \@tempa{n-illion}%
1734   \@tempa{n-illiard}%

```

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

```

1735   \let\fc@wcase@save\fc@wcase
1736   \def\fc@wcase{\noexpand\fc@wcase}%
1737   \def\@nil{\noexpand\@nil}%

```

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

```

1738   \let\@tempa\@empty
1739   \fc@@nbrstrfrench@inner

```

Now restore `\fc@wcase`

```

1740   \let\fc@wcase\fc@wcase@save

```

Now we add the “*ième*” ending

```

1741   \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
1742   \expandafter\fc@get@last@letter\expandafter{\@tempc}\@tempd\@tempe
1743   \def\@tempf{e}%
1744   \ifx\@tempe\@tempf
1745     \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd i\protect\'eme\@nil}%
1746   \else
1747     \def\@tempf{q}%
1748     \ifx\@tempe\@tempf
1749       \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd qui\protect\'eme\@nil}%
1750     \else
1751       \def\@tempf{f}%
1752       \ifx\@tempe\@tempf
1753         \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd vi\protect\'eme\@nil}%
1754       \else
1755         \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempc i\protect\'eme\@nil}%
1756       \fi
1757     \fi
1758   \fi
1759 }%

```

Apply `\fc@gcase` to the result.

```

1760   \fc@apply@gcase

```

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

```

1761   \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
1762   \expandafter\@tempb\expandafter{\@tempa}%
1763   \expandafter
1764 } \@tempa
1765 }%
1766 \global\let\@@ordinalstringfrench\@@ordinalstringfrench

```

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

```
1767 \newcommand*\fc@frenchoptions@setdefaults{%
1768   \csname KV@fcfrench@all plural\endcsname{reformed}%
1769   \fc@gl@def\fc@frenchoptions@submillion@dos{-}%
1770   \fc@gl@let\fc@frenchoptions@supermillion@dos\space
1771   \fc@gl@let\fc@u@in@duo\@empty% Could be ‘u’
1772   % \fc@gl@let\fc@poweroften\fc@@pot@longscalefrench
1773   \fc@gl@let\fc@poweroften\fc@@pot@recursivefrench
1774   \fc@gl@def\fc@longscale@nilliard@upto{0}% infinity
1775   \fc@gl@def\fc@frenchoptions@mil@plural@mark{le}%
1776 }%
1777 \global\let\fc@frenchoptions@setdefaults\fc@frenchoptions@setdefaults
1778 {%
1779   \let\fc@gl@def\gdef
1780   \def\fc@gl@let{\global\let}%
1781   \fc@frenchoptions@setdefaults
1782 }%
```

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

```
1783 \gdef\@ordinalstringMfrench{\csuse{@ordinalstringMfrench\fmtcount@french}}%
1784 \gdef\@ordinalstringFfrench{\csuse{@ordinalstringFfrench\fmtcount@french}}%
1785 \gdef\@OrdinalstringMfrench{\csuse{@OrdinalstringMfrench\fmtcount@french}}%
1786 \gdef\@OrdinalstringFfrench{\csuse{@OrdinalstringFfrench\fmtcount@french}}%
1787 \gdef\@numberstringMfrench{\csuse{@numberstringMfrench\fmtcount@french}}%
1788 \gdef\@numberstringFfrench{\csuse{@numberstringFfrench\fmtcount@french}}%
1789 \gdef\@NumberstringMfrench{\csuse{@NumberstringMfrench\fmtcount@french}}%
1790 \gdef\@NumberstringFfrench{\csuse{@NumberstringFfrench\fmtcount@french}}%
```

### 10.1.7 fc-frenchb.def

```
1791 \ProvidesFCLanguage{frenchb}[2013/08/17]%
1792 \FCloadlang{french}%
```

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

```
1793 \global\let\@ordinalMfrenchb=\@ordinalMfrench
1794 \global\let\@ordinalFfrenchb=\@ordinalFfrench
1795 \global\let\@ordinalNfrenchb=\@ordinalNfrench
1796 \global\let\@numberstringMfrenchb=\@numberstringMfrench
1797 \global\let\@numberstringFfrenchb=\@numberstringFfrench
1798 \global\let\@numberstringNfrenchb=\@numberstringNfrench
1799 \global\let\@NumberstringMfrenchb=\@NumberstringMfrench
1800 \global\let\@NumberstringFfrenchb=\@NumberstringFfrench
1801 \global\let\@NumberstringNfrenchb=\@NumberstringNfrench
1802 \global\let\@ordinalstringMfrenchb=\@ordinalstringMfrench
1803 \global\let\@ordinalstringFfrenchb=\@ordinalstringFfrench
1804 \global\let\@ordinalstringNfrenchb=\@ordinalstringNfrench
1805 \global\let\@OrdinalstringMfrenchb=\@OrdinalstringMfrench
1806 \global\let\@OrdinalstringFfrenchb=\@OrdinalstringFfrench
1807 \global\let\@OrdinalstringNfrenchb=\@OrdinalstringNfrench
```

### 10.1.8 fc-german.def

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

```
1808 \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:

```
1809 \newcommand{\@ordinalMgerman}[2]{%
1810   \edef#2{\number#1\relax.}%
1811 }%
1812 \global\let\@ordinalMgerman\@ordinalMgerman
```

Feminine:

```
1813 \newcommand{\@ordinalFgerman}[2]{%
1814   \edef#2{\number#1\relax.}%
1815 }%
1816 \global\let\@ordinalFgerman\@ordinalFgerman
```

Neuter:

```
1817 \newcommand{\@ordinalNgerman}[2]{%
1818   \edef#2{\number#1\relax.}%
1819 }%
1820 \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):

```
1821 \newcommand*{\@unitstringgerman}[1]{%
1822   \ifcase#1%
1823     null%
1824     \or ein%
1825     \or zwei%
1826     \or drei%
1827     \or vier%
1828     \or f"unf%
1829     \or sechs%
1830     \or sieben%
1831     \or acht%
1832     \or neun%
1833   \fi
1834 }%
1835 \global\let\@unitstringgerman\@unitstringgerman
```

Tens (argument must go from 1 to 10):

```
1836 \newcommand*{\@tenstringgerman}[1]{%
1837   \ifcase#1%
1838     \or zehn%
1839     \or zwanzig%
1840     \or drei{\ss}ig%
1841     \or vierzig%
1842     \or f"unfzig%
1843     \or sechzig%
```

```

1844 \or siebzig%
1845 \or achtzig%
1846 \or neunzig%
1847 \or einhundert%
1848 \fi
1849 }%
1850 \global\let\@@tenstringgerman\@@tenstringgerman
    \einhundert is set to einhundert by default, user can redefine this command to just
    hundert if required, similarly for \eintausend.
1851 \providecommand*\einhundert{\einhundert}%
1852 \providecommand*\eintausend{\eintausend}%
1853 \global\let\einhundert\einhundert
1854 \global\let\eintausend\eintausend
    Teens:
1855 \newcommand*\@@teenstringgerman[1]{%
1856 \ifcase#1%
1857 zehn%
1858 \or elf%
1859 \or zw"olf%
1860 \or dreizehn%
1861 \or vierzehn%
1862 \or f"unfzehn%
1863 \or sechzehn%
1864 \or siebzehn%
1865 \or achtzehn%
1866 \or neunzehn%
1867 \fi
1868 }%
1869 \global\let\@@teenstringgerman\@@teenstringgerman
    The results are stored in the second argument, but doesn't display anything.
1870 \newcommand*\@numberstringMgerman}[2]{%
1871 \let\@unitstring=\@unitstringgerman
1872 \let\@teenstring=\@teenstringgerman
1873 \let\@tenstring=\@tenstringgerman
1874 \@numberstringgerman{#1}{#2}%
1875 }%
1876 \global\let\@numberstringMgerman\@numberstringMgerman
    Feminine and neuter forms:
1877 \global\let\@numberstringFgerman=\@numberstringMgerman
1878 \global\let\@numberstringNgerman=\@numberstringMgerman
    As above, but initial letters in upper case:
1879 \newcommand*\@NumberstringMgerman}[2]{%
1880 \@numberstringMgerman{#1}{\@num@str}%
1881 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1882 }%
1883 \global\let\@NumberstringMgerman\@NumberstringMgerman

```



Feminine and neuter form:

```
1884 \global\let\@NumberstringFgerman=\@NumberstringMgerman
1885 \global\let\@NumberstringNgerman=\@NumberstringMgerman
```

As above, but for ordinals.

```
1886 \newcommand*\@ordinalstringMgerman}[2]{%
1887 \let\@unitthstring=\@unitthstringMgerman
1888 \let\@teenthstring=\@teenthstringMgerman
1889 \let\@tenthstring=\@tenthstringMgerman
1890 \let\@unitstring=\@unitstringgerman
1891 \let\@teenstring=\@teenstringgerman
1892 \let\@tenstring=\@tenstringgerman
1893 \def\@thousandth{tausendster}%
1894 \def\@hundredth{hundertster}%
1895 \@ordinalstringgerman{#1}{#2}%
1896 }%
1897 \global\let\@ordinalstringMgerman\@ordinalstringMgerman
```

Feminine form:

```
1898 \newcommand*\@ordinalstringFgerman}[2]{%
1899 \let\@unitthstring=\@unitthstringFgerman
1900 \let\@teenthstring=\@teenthstringFgerman
1901 \let\@tenthstring=\@tenthstringFgerman
1902 \let\@unitstring=\@unitstringgerman
1903 \let\@teenstring=\@teenstringgerman
1904 \let\@tenstring=\@tenstringgerman
1905 \def\@thousandth{tausendste}%
1906 \def\@hundredth{hundertste}%
1907 \@ordinalstringgerman{#1}{#2}%
1908 }%
1909 \global\let\@ordinalstringFgerman\@ordinalstringFgerman
```

Neuter form:

```
1910 \newcommand*\@ordinalstringNgerman}[2]{%
1911 \let\@unitthstring=\@unitthstringNgerman
1912 \let\@teenthstring=\@teenthstringNgerman
1913 \let\@tenthstring=\@tenthstringNgerman
1914 \let\@unitstring=\@unitstringgerman
1915 \let\@teenstring=\@teenstringgerman
1916 \let\@tenstring=\@tenstringgerman
1917 \def\@thousandth{tausendstes}%
1918 \def\@hundredth{hunderstes}%
1919 \@ordinalstringgerman{#1}{#2}%
1920 }%
1921 \global\let\@ordinalstringNgerman\@ordinalstringNgerman
```

As above, but with initial letters in upper case.

```
1922 \newcommand*\@OrdinalstringMgerman}[2]{%
1923 \@ordinalstringMgerman{#1}{\@num@str}%
1924 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1925 }%
```

1926 \global\let\@OrdinalstringMgerman\@OrdinalstringMgerman

Feminine form:

```
1927 \newcommand*\@OrdinalstringFgerman}[2]{%
1928 \@OrdinalstringFgerman{#1}{\@num@str}%
1929 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1930 }%
1931 \global\let\@OrdinalstringFgerman\@OrdinalstringFgerman
```

Neuter form:

```
1932 \newcommand*\@OrdinalstringNgerman}[2]{%
1933 \@OrdinalstringNgerman{#1}{\@num@str}%
1934 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1935 }%
1936 \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:

```
1937 \newcommand*\@unitthstringMgerman[1]{%
1938 \ifcase#1%
1939 nullter%
1940 \or erster%
1941 \or zweiter%
1942 \or dritter%
1943 \or vierter%
1944 \or f"unfter%
1945 \or sechster%
1946 \or siebter%
1947 \or achter%
1948 \or neunter%
1949 \fi
1950 }%
1951 \global\let\@unitthstringMgerman\@unitthstringMgerman
```

Tens:

```
1952 \newcommand*\@tenthstringMgerman[1]{%
1953 \ifcase#1%
1954 \or zehnter%
1955 \or zwanzigster%
1956 \or drei{\ss}igster%
1957 \or vierzigster%
1958 \or f"unzigster%
1959 \or sechzigster%
1960 \or siebzigster%
1961 \or achtzigster%
1962 \or neunzigster%
1963 \fi
1964 }%
1965 \global\let\@tenthstringMgerman\@tenthstringMgerman
```

Teens:

```

1966 \newcommand*\@@teenthstringMgerman[1]{%
1967   \ifcase#1%
1968     zehnter%
1969     \or elfter%
1970     \or zw"olfter%
1971     \or dreizehnter%
1972     \or vierzehnter%
1973     \or f"unfzehnter%
1974     \or sechzehnter%
1975     \or siebzehnter%
1976     \or achtzehnter%
1977     \or neunzehnter%
1978   \fi
1979 }%
1980 \global\let\@@teenthstringMgerman\@@teenthstringMgerman

```

Units (feminine):

```

1981 \newcommand*\@@unitthstringFgerman[1]{%
1982   \ifcase#1%
1983     nullte%
1984     \or erste%
1985     \or zweite%
1986     \or dritte%
1987     \or vierte%
1988     \or f"unfte%
1989     \or sechste%
1990     \or siebte%
1991     \or achte%
1992     \or neunte%
1993   \fi
1994 }%
1995 \global\let\@@unitthstringFgerman\@@unitthstringFgerman

```

Tens (feminine):

```

1996 \newcommand*\@@tenthstringFgerman[1]{%
1997   \ifcase#1%
1998     \or zehnte%
1999     \or zwanzigste%
2000     \or drei{\ss}igste%
2001     \or vierzigste%
2002     \or f"unfzigste%
2003     \or sechzigste%
2004     \or siebzigste%
2005     \or achtzigste%
2006     \or neunzigste%
2007   \fi
2008 }%
2009 \global\let\@@tenthstringFgerman\@@tenthstringFgerman

```

Teens (feminine)

```

2010 \newcommand*\@@teenthstringFgerman[1]{%

```

```

2011 \ifcase#1%
2012     zehnte%
2013     \or elfte%
2014     \or zw"olfte%
2015     \or dreizehnte%
2016     \or vierzehnte%
2017     \or f"unfzehnte%
2018     \or sechzehnte%
2019     \or siebzehnte%
2020     \or achtzehnte%
2021     \or neunzehnte%
2022 \fi
2023 }%
2024 \global\let\@@teenthstringFgerman\@@teenthstringFgerman

```

Units (neuter):

```

2025 \newcommand*\@@unitthstringNgerman[1]{%
2026     \ifcase#1%
2027         nulltes%
2028         \or erstes%
2029         \or zweites%
2030         \or drittes%
2031         \or viertes%
2032         \or f"unftes%
2033         \or sechstes%
2034         \or siebtes%
2035         \or achttes%
2036         \or neuntes%
2037 \fi
2038 }%
2039 \global\let\@@unitthstringNgerman\@@unitthstringNgerman

```

Tens (neuter):

```

2040 \newcommand*\@@tenthstringNgerman[1]{%
2041     \ifcase#1%
2042         \or zehntes%
2043         \or zwanzigstes%
2044         \or drei{\ss}igstes%
2045         \or vierzigstes%
2046         \or f"unfzigstes%
2047         \or sechzigstes%
2048         \or siebzigstes%
2049         \or achtzigstes%
2050         \or neunzigstes%
2051 \fi
2052 }%
2053 \global\let\@@tenthstringNgerman\@@tenthstringNgerman

```

Teens (neuter)

```

2054 \newcommand*\@@teenthstringNgerman[1]{%
2055     \ifcase#1%

```

```

2056     zehntes%
2057     \or elftes%
2058     \or zw\"olftes%
2059     \or dreizehntes%
2060     \or vierzehntes%
2061     \or f\"unfzehntes%
2062     \or sechzehntes%
2063     \or siebzehntes%
2064     \or achtzehntes%
2065     \or neunzehntes%
2066 \fi
2067 }%
2068 \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.

```

2069 \newcommand*\@@numberunderhundredgerman[2]{%
2070 \ifnum#1<10\relax
2071   \ifnum#1>0\relax
2072     \eappto#2{\@unitstring{#1}}%
2073   \fi
2074 \else
2075   \@tmpstrctr=#1\relax
2076   \@FCmodulo{\@tmpstrctr}{10}%
2077   \ifnum#1<20\relax
2078     \eappto#2{\@teenstring{\@tmpstrctr}}%
2079   \else
2080     \ifnum\@tmpstrctr=0\relax
2081       \else
2082         \eappto#2{\@unitstring{\@tmpstrctr}und}%
2083       \fi
2084     \@tmpstrctr=#1\relax
2085     \divide\@tmpstrctr by 10\relax
2086     \eappto#2{\@tenstring{\@tmpstrctr}}%
2087   \fi
2088 \fi
2089 }%
2090 \global\let\@@numberunderhundredgerman\@@numberunderhundredgerman

```

This stores the results in the second argument (which must be a control sequence), but it doesn't display anything.

```

2091 \newcommand*\@@numberstringgerman[2]{%
2092 \ifnum#1>99999\relax
2093   \PackageError{fmtcount}{Out of range}%
2094   {This macro only works for values less than 100000}%
2095 \else
2096   \ifnum#1<0\relax
2097     \PackageError{fmtcount}{Negative numbers not permitted}%
2098     {This macro does not work for negative numbers, however
2099     you can try typing "minus" first, and then pass the modulus of

```

```

2100     this number}%
2101   \fi
2102 \fi
2103 \def#2{}%
2104 \@strctr=#1\relax \divide\@strctr by 1000\relax
2105 \ifnum\@strctr>1\relax

    #1 is  $\geq 2000$ , \@strctr now contains the number of thousands
2106   \@@numberunderhundredgerman{\@strctr}{#2}%
2107   \appto#2{tausend}%
2108 \else

    #1 lies in range [1000,1999]
2109   \ifnum\@strctr=1\relax
2110     \eappto#2{\eintausend}%
2111   \fi
2112 \fi
2113 \@strctr=#1\relax
2114 \@FCmodulo{\@strctr}{1000}%
2115 \divide\@strctr by 100\relax
2116 \ifnum\@strctr>1\relax

    now dealing with number in range [200,999]
2117   \eappto#2{\@unitstring{\@strctr}hundert}%
2118 \else
2119   \ifnum\@strctr=1\relax

    dealing with number in range [100,199]
2120     \ifnum#1>1000\relax

    if original number > 1000, use einhundert
2121       \appto#2{einhundert}%
2122     \else

    otherwise use \einhundert
2123       \eappto#2{\einhundert}%
2124     \fi
2125   \fi
2126 \fi
2127 \@strctr=#1\relax
2128 \@FCmodulo{\@strctr}{100}%
2129 \ifnum#1=0\relax
2130   \def#2{null}%
2131 \else
2132   \ifnum\@strctr=1\relax
2133     \appto#2{eins}%
2134   \else
2135     \@@numberunderhundredgerman{\@strctr}{#2}%
2136   \fi
2137 \fi
2138 }%
2139 \global\let\@@numberstringgerman\@@numberstringgerman

```

As above, but for ordinals

```
2140 \newcommand*{\@numberunderhundredthgerman[2]}{%
2141 \ifnum#1<10\relax
2142 \eappto#2{\@unitthstring{#1}}%
2143 \else
2144 \@tmpstrctr=#1\relax
2145 \@FCmodulo{\@tmpstrctr}{10}%
2146 \ifnum#1<20\relax
2147 \eappto#2{\@teenthstring{\@tmpstrctr}}%
2148 \else
2149 \ifnum\@tmpstrctr=0\relax
2150 \else
2151 \eappto#2{\@unitstring{\@tmpstrctr}und}%
2152 \fi
2153 \@tmpstrctr=#1\relax
2154 \divide\@tmpstrctr by 10\relax
2155 \eappto#2{\@tenthstring{\@tmpstrctr}}%
2156 \fi
2157 \fi
2158 }%
2159 \global\let\@numberunderhundredthgerman\@numberunderhundredthgerman

2160 \newcommand*{\@ordinalstringgerman[2]}{%
2161 \@orgargctr=#1\relax
2162 \ifnum\@orgargctr>99999\relax
2163 \PackageError{fmtcount}{Out of range}%
2164 {This macro only works for values less than 100000}%
2165 \else
2166 \ifnum\@orgargctr<0\relax
2167 \PackageError{fmtcount}{Negative numbers not permitted}%
2168 {This macro does not work for negative numbers, however
2169 you can try typing "minus" first, and then pass the modulus of
2170 this number}%
2171 \fi
2172 \fi
2173 \def#2{}%
2174 \@strctr=\@orgargctr\divide\@strctr by 1000\relax
2175 \ifnum\@strctr>1\relax

#1 is  $\geq 2000$ , \@strctr now contains the number of thousands
2176 \@numberunderhundredthgerman{\@strctr}{#2}%

is that it, or is there more?
2177 \@tmpstrctr=\@orgargctr\@FCmodulo{\@tmpstrctr}{1000}%
2178 \ifnum\@tmpstrctr=0\relax
2179 \eappto#2{\@thousandth}%
2180 \else
2181 \appto#2{tausend}%
2182 \fi
2183 \else
```

```

#1 lies in range [1000,1999]
2184 \ifnum\@strctr=1\relax
2185 \ifnum\@orgargctr=1000\relax
2186 \eappto#2{\@thousandth}%
2187 \else
2188 \eappto#2{\eintausend}%
2189 \fi
2190 \fi
2191 \fi
2192 \@strctr=\@orgargctr
2193 \@FCmodulo{\@strctr}{1000}%
2194 \divide\@strctr by 100\relax
2195 \ifnum\@strctr>1\relax

now dealing with number in range [200,999] is that it, or is there more?
2196 \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{100}%
2197 \ifnum\@tmpstrctr=0\relax
2198 \ifnum\@strctr=1\relax
2199 \eappto#2{\@hundredth}%
2200 \else
2201 \eappto#2{\@unitstring{\@strctr}\@hundredth}%
2202 \fi
2203 \else
2204 \eappto#2{\@unitstring{\@strctr}hundert}%
2205 \fi
2206 \else
2207 \ifnum\@strctr=1\relax

dealing with number in range [100,199] is that it, or is there more?
2208 \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{100}%
2209 \ifnum\@tmpstrctr=0\relax
2210 \eappto#2{\@hundredth}%
2211 \else
2212 \ifnum\@orgargctr>1000\relax
2213 \appto#2{einhundert}%
2214 \else
2215 \eappto#2{\einhundert}%
2216 \fi
2217 \fi
2218 \fi
2219 \fi
2220 \@strctr=\@orgargctr
2221 \@FCmodulo{\@strctr}{100}%
2222 \ifthenelse{\@strctr=0 \and \@orgargctr>0 }{ }{%
2223 \@@numberunderhundredthgerman{\@strctr}{#2}%
2224 }%
2225 }%
2226 \global\let\@@ordinalstringgerman\@@ordinalstringgerman

Load fc-germanb.def if not already loaded
2227 \FCloadlang{germanb}%

```



### 10.1.9 fc-germanb.def

```
2228 \ProvidesFCLanguage{germanb}[2013/08/17]%
```

Load fc-german.def if not already loaded

```
2229 \FCloadlang{german}%
```

Set germanb to be equivalent to german.

```
2230 \global\let\@ordinalMgermanb=\@ordinalMgerman
```

```
2231 \global\let\@ordinalFgermanb=\@ordinalFgerman
```

```
2232 \global\let\@ordinalNgermanb=\@ordinalNgerman
```

```
2233 \global\let\@numberstringMgermanb=\@numberstringMgerman
```

```
2234 \global\let\@numberstringFgermanb=\@numberstringFgerman
```

```
2235 \global\let\@numberstringNgermanb=\@numberstringNgerman
```

```
2236 \global\let\@NumberstringMgermanb=\@NumberstringMgerman
```

```
2237 \global\let\@NumberstringFgermanb=\@NumberstringFgerman
```

```
2238 \global\let\@NumberstringNgermanb=\@NumberstringNgerman
```

```
2239 \global\let\@ordinalstringMgermanb=\@ordinalstringMgerman
```

```
2240 \global\let\@ordinalstringFgermanb=\@ordinalstringFgerman
```

```
2241 \global\let\@ordinalstringNgermanb=\@ordinalstringNgerman
```

```
2242 \global\let\@OrdinalstringMgermanb=\@OrdinalstringMgerman
```

```
2243 \global\let\@OrdinalstringFgermanb=\@OrdinalstringFgerman
```

```
2244 \global\let\@OrdinalstringNgermanb=\@OrdinalstringNgerman
```

### 10.1.10 fc-italian

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

```
2245 \ProvidesFCLanguage{italian}[2013/08/17]
```

```
2246
```

```
2247 \RequirePackage{itnumpar}
```

```
2248
```

```
2249 \newcommand{\@numberstringMitalian}[2]{%
```

```
2250   \edef#2{\noexpand\printnumeroinparole{#1}}%
```

```
2251 }
```

```
2252 \global\let\@numberstringMitalian\@numberstringMitalian
```

```
2253
```

```
2254 \newcommand{\@numberstringFitalian}[2]{%
```

```
2255   \edef#2{\noexpand\printnumeroinparole{#1}}}
```

```
2256
```

```
2257 \global\let\@numberstringFitalian\@numberstringFitalian
```

```
2258
```

```
2259 \newcommand{\@NumberstringMitalian}[2]{%
```

```
2260   \edef#2{\noexpand\printNumeroinparole{#1}}%
```

```
2261 }
```

```
2262 \global\let\@NumberstringMitalian\@NumberstringMitalian
```

```
2263
```

```
2264 \newcommand{\@NumberstringFitalian}[2]{%
```

```
2265   \edef#2{\noexpand\printNumeroinparole{#1}}%
```

```
2266 }
```

```
2267 \global\let\@NumberstringFitalian\@NumberstringFitalian
```

```
2268
```

```

2269 \newcommand{\@OrdinalstringMitalian}[2]{%
2270   \edef#2{\noexpand\printordinalem{#1}}%
2271 }
2272 \global\let\@OrdinalstringMitalian\@OrdinalstringMitalian
2273
2274 \newcommand{\@OrdinalstringFitalian}[2]{%
2275   \edef#2{\noexpand\printordinalef{#1}}%
2276 }
2277 \global\let\@OrdinalstringFitalian\@OrdinalstringFitalian
2278
2279 \newcommand{\@OrdinalstringMitalian}[2]{%
2280   \edef#2{\noexpand\printOrdinalem{#1}}%
2281 }
2282 \global\let\@OrdinalstringMitalian\@OrdinalstringMitalian
2283
2284 \newcommand{\@OrdinalstringFitalian}[2]{%
2285   \edef#2{\noexpand\printOrdinalef{#1}}%
2286 }
2287 \global\let\@OrdinalstringFitalian\@OrdinalstringFitalian
2288
2289 \newcommand{\@OrdinalMitalian}[2]{%
2290   \edef#2{#1\relax\noexpand\fmtord{o}}%
2291 }
2292 \global\let\@OrdinalMitalian\@OrdinalMitalian
2293
2294 \newcommand{\@OrdinalFitalian}[2]{%
2295   \edef#2{#1\relax\noexpand\fmtord{a}}%
2296 }
2297 \global\let\@OrdinalFitalian\@OrdinalFitalian

```

### 10.1.11 fc-ngerman.def

```

2297 \ProvidesFCLanguage{ngerman}[2012/06/18]%
2298 \FCloadlang{german}%
2299 \FCloadlang{ngermanb}%

```

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

```

2300 \global\let\@OrdinalMngerman=\@OrdinalMgerman
2301 \global\let\@OrdinalFngerman=\@OrdinalFgerman
2302 \global\let\@OrdinalNngerman=\@OrdinalNgerman
2303 \global\let\@NumberstringMngerman=\@NumberstringMgerman
2304 \global\let\@NumberstringFngerman=\@NumberstringFgerman
2305 \global\let\@NumberstringNngerman=\@NumberstringNgerman
2306 \global\let\@NumberstringMngerman=\@NumberstringMgerman
2307 \global\let\@NumberstringFngerman=\@NumberstringFgerman
2308 \global\let\@NumberstringNngerman=\@NumberstringNgerman
2309 \global\let\@OrdinalstringMngerman=\@OrdinalstringMgerman
2310 \global\let\@OrdinalstringFngerman=\@OrdinalstringFgerman
2311 \global\let\@OrdinalstringNngerman=\@OrdinalstringNgerman
2312 \global\let\@OrdinalstringMngerman=\@OrdinalstringMgerman

```

```

2313 \global\let\@OrdinalstringFngerman=\@OrdinalstringFgerman
2314 \global\let\@OrdinalstringNngerman=\@OrdinalstringNgerman

```

### 10.1.12 fc-ngermanb.def

```

2315 \ProvidesFCLanguage{ngermanb}[2013/08/17]%
2316 \FCloadlang{ngermanb}%

```

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

```

2317 \global\let\@OrdinalMngermanb=\@OrdinalMgerman
2318 \global\let\@OrdinalFngermanb=\@OrdinalFgerman
2319 \global\let\@OrdinalNngermanb=\@OrdinalNgerman
2320 \global\let\@NumberstringMngermanb=\@NumberstringMgerman
2321 \global\let\@NumberstringFngermanb=\@NumberstringFgerman
2322 \global\let\@NumberstringNngermanb=\@NumberstringNgerman
2323 \global\let\@NumberstringMngermanb=\@NumberstringMgerman
2324 \global\let\@NumberstringFngermanb=\@NumberstringFgerman
2325 \global\let\@NumberstringNngermanb=\@NumberstringNgerman
2326 \global\let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
2327 \global\let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
2328 \global\let\@OrdinalstringNngermanb=\@OrdinalstringNgerman
2329 \global\let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
2330 \global\let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
2331 \global\let\@OrdinalstringNngermanb=\@OrdinalstringNgerman

```

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

```

2332 \FCloadlang{ngermanb}%

```

### 10.1.13 fc-portuges.def

Portuguese definitions

```

2333 \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:

```

2334 \newcommand*\@ordinalMportuges[2] {%
2335   \ifnum#1=0\relax
2336     \edef#2{\number#1}%
2337   \else
2338     \edef#2{\number#1\relax\noexpand\fmtord{o}}%
2339   \fi
2340 }%
2341 \global\let\@ordinalMportuges\@ordinalMportuges

```

Feminine:

```

2342 \newcommand*\@ordinalFportuges[2] {%
2343   \ifnum#1=0\relax
2344     \edef#2{\number#1}%
2345   \else
2346     \edef#2{\number#1\relax\noexpand\fmtord{a}}%
2347   \fi
2348 }%

```

```

2349 \global\let\@OrdinalFportuges\@OrdinalFportuges
    Make neuter same as masculine:
2350 \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):
2351 \newcommand*\@unitstringportuges[1]{%
2352   \ifcase#1\relax
2353     zero%
2354     \or um%
2355     \or dois%
2356     \or tr\^es%
2357     \or quatro%
2358     \or cinco%
2359     \or seis%
2360     \or sete%
2361     \or oito%
2362     \or nove%
2363   \fi
2364 }%
2365 \global\let\@unitstringportuges\@unitstringportuges
2366 % \end{macrocode}
2367 % As above, but for feminine:
2368 % \begin{macrocode}
2369 \newcommand*\@unitstringFportuges[1]{%
2370   \ifcase#1\relax
2371     zero%
2372     \or uma%
2373     \or duas%
2374     \or tr\^es%
2375     \or quatro%
2376     \or cinco%
2377     \or seis%
2378     \or sete%
2379     \or oito%
2380     \or nove%
2381   \fi
2382 }%
2383 \global\let\@unitstringFportuges\@unitstringFportuges
    Tens (argument must be a number from 0 to 10):
2384 \newcommand*\@tenstringportuges[1]{%
2385   \ifcase#1\relax
2386     \or dez%
2387     \or vinte%
2388     \or trinta%
2389     \or quarenta%
2390     \or cinq\"uenta%
2391     \or sessenta%
2392     \or setenta%

```

```

2393     \or oitenta%
2394     \or noventa%
2395     \or cem%
2396     \fi
2397 }%
2398 \global\let\@@tenstringportuges\@@tenstringportuges

```

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

```

2399 \newcommand*\@@teenstringportuges[1]{%
2400   \ifcase#1\relax
2401     dez%
2402     \or onze%
2403     \or doze%
2404     \or treze%
2405     \or catorze%
2406     \or quinze%
2407     \or dezasseis%
2408     \or dezassete%
2409     \or dezoito%
2410     \or dezanove%
2411     \fi
2412 }%
2413 \global\let\@@teenstringportuges\@@teenstringportuges

```

Hundreds:

```

2414 \newcommand*\@@hundredstringportuges[1]{%
2415   \ifcase#1\relax
2416     \or cento%
2417     \or duzentos%
2418     \or trezentos%
2419     \or quatrocentos%
2420     \or quinhentos%
2421     \or seiscentos%
2422     \or setecentos%
2423     \or oitocentos%
2424     \or novecentos%
2425     \fi
2426 }%
2427 \global\let\@@hundredstringportuges\@@hundredstringportuges

```

Hundreds (feminine):

```

2428 \newcommand*\@@hundredstringFportuges[1]{%
2429   \ifcase#1\relax
2430     \or cento%
2431     \or duzentas%
2432     \or trezentas%
2433     \or quatrocentas%
2434     \or quinhentas%
2435     \or seiscentas%
2436     \or setecentas%
2437     \or oitocentas%

```

```

2438 \or novecentas%
2439 \fi
2440 }%
2441 \global\let\@@hundredstringFportuges\@@hundredstringFportuges

```

Units (initial letter in upper case):

```

2442 \newcommand*\@@Unitstringportuges[1]{%
2443 \ifcase#1\relax
2444 Zero%
2445 \or Um%
2446 \or Dois%
2447 \or Tr\^es%
2448 \or Quatro%
2449 \or Cinco%
2450 \or Seis%
2451 \or Sete%
2452 \or Oito%
2453 \or Nove%
2454 \fi
2455 }%
2456 \global\let\@@Unitstringportuges\@@Unitstringportuges

```

As above, but feminine:

```

2457 \newcommand*\@@UnitstringFportuges[1]{%
2458 \ifcase#1\relax
2459 Zera%
2460 \or Uma%
2461 \or Duas%
2462 \or Tr\^es%
2463 \or Quatro%
2464 \or Cinco%
2465 \or Seis%
2466 \or Sete%
2467 \or Oito%
2468 \or Nove%
2469 \fi
2470 }%
2471 \global\let\@@UnitstringFportuges\@@UnitstringFportuges

```

Tens (with initial letter in upper case):

```

2472 \newcommand*\@@Tenstringportuges[1]{%
2473 \ifcase#1\relax
2474 \or Dez%
2475 \or Vinte%
2476 \or Trinta%
2477 \or Quarenta%
2478 \or Cinq\^uentas%
2479 \or Sessenta%
2480 \or Setenta%
2481 \or Oitenta%
2482 \or Noventa%

```

```
2483 \or Cem%
2484 \fi
2485 }%
2486 \global\let\@@Teenstringportuges\@@Teenstringportuges
```

Teens (with initial letter in upper case):

```
2487 \newcommand*\@@Teenstringportuges[1]{%
2488 \ifcase#1\relax
2489 Dez%
2490 \or Onze%
2491 \or Doze%
2492 \or Treze%
2493 \or Catorze%
2494 \or Quinze%
2495 \or Dezasseis%
2496 \or Dezassete%
2497 \or Dezoito%
2498 \or Dezanove%
2499 \fi
2500 }%
2501 \global\let\@@Teenstringportuges\@@Teenstringportuges
```

Hundreds (with initial letter in upper case):

```
2502 \newcommand*\@@Hundredstringportuges[1]{%
2503 \ifcase#1\relax
2504 \or Cento%
2505 \or Duzentos%
2506 \or Trezentos%
2507 \or Quatrocentos%
2508 \or Quinhentos%
2509 \or Seiscentos%
2510 \or Setecentos%
2511 \or Oitocentos%
2512 \or Novecentos%
2513 \fi
2514 }%
2515 \global\let\@@Hundredstringportuges\@@Hundredstringportuges
```

As above, but feminine:

```
2516 \newcommand*\@@HundredstringFportuges[1]{%
2517 \ifcase#1\relax
2518 \or Cento%
2519 \or Duzentas%
2520 \or Trezentas%
2521 \or Quatrocentas%
2522 \or Quinhentas%
2523 \or Seiscentas%
2524 \or Setecentas%
2525 \or Oitocentas%
2526 \or Novecentas%
2527 \fi
```

2528 }%  
 2529 \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.)

2530 \newcommand\*\@numberstringMportuges}[2]{%  
 2531 \let\@unitstring=\@@unitstringportuges  
 2532 \let\@teenstring=\@@teenstringportuges  
 2533 \let\@tenstring=\@@tenstringportuges  
 2534 \let\@hundredstring=\@@hundredstringportuges  
 2535 \def\@hundred{cem}\def\@thousand{mil}%  
 2536 \def\@andname{e}%  
 2537 \@numberstringportuges{#1}{#2}%  
 2538 }%  
 2539 \global\let\@numberstringMportuges\@numberstringMportuges

As above, but feminine form:

2540 \newcommand\*\@numberstringFportuges}[2]{%  
 2541 \let\@unitstring=\@@unitstringFportuges  
 2542 \let\@teenstring=\@@teenstringportuges  
 2543 \let\@tenstring=\@@tenstringportuges  
 2544 \let\@hundredstring=\@@hundredstringFportuges  
 2545 \def\@hundred{cem}\def\@thousand{mil}%  
 2546 \def\@andname{e}%  
 2547 \@numberstringportuges{#1}{#2}%  
 2548 }%  
 2549 \global\let\@numberstringFportuges\@numberstringFportuges

Make neuter same as masculine:

2550 \global\let\@numberstringNportuges\@numberstringMportuges

As above, but initial letters in upper case:

2551 \newcommand\*\@NumberstringMportuges}[2]{%  
 2552 \let\@unitstring=\@@Unitstringportuges  
 2553 \let\@teenstring=\@@Teenstringportuges  
 2554 \let\@tenstring=\@@Tenstringportuges  
 2555 \let\@hundredstring=\@@Hundredstringportuges  
 2556 \def\@hundred{Cem}\def\@thousand{Mil}%  
 2557 \def\@andname{e}%  
 2558 \@numberstringportuges{#1}{#2}%  
 2559 }%  
 2560 \global\let\@NumberstringMportuges\@NumberstringMportuges

As above, but feminine form:

2561 \newcommand\*\@NumberstringFportuges}[2]{%  
 2562 \let\@unitstring=\@@UnitstringFportuges  
 2563 \let\@teenstring=\@@Teenstringportuges  
 2564 \let\@tenstring=\@@Tenstringportuges  
 2565 \let\@hundredstring=\@@HundredstringFportuges  
 2566 \def\@hundred{Cem}\def\@thousand{Mil}%



```

2567 \def\@andname{e}%
2568 \@numberstringportuges{#1}{#2}%
2569 }%
2570 \global\let\@NumberstringFportuges\@NumberstringFportuges
    Make neuter same as masculine:
2571 \global\let\@NumberstringNportuges\@NumberstringMportuges
    As above, but for ordinals.
2572 \newcommand*\@ordinalstringMportuges}[2]{%
2573 \let\@unitthstring=\@unitthstringportuges
2574 \let\@unitstring=\@unitstringportuges
2575 \let\@teenthstring=\@teenthstringportuges
2576 \let\@tenthstring=\@tenthstringportuges
2577 \let\@hundredthstring=\@hundredthstringportuges
2578 \def\@thousandth{mil'esimo}%
2579 \@ordinalstringportuges{#1}{#2}%
2580 }%
2581 \global\let\@ordinalstringMportuges\@ordinalstringMportuges
    Feminine form:
2582 \newcommand*\@ordinalstringFportuges}[2]{%
2583 \let\@unitthstring=\@unitthstringFportuges
2584 \let\@unitstring=\@unitstringFportuges
2585 \let\@teenthstring=\@teenthstringportuges
2586 \let\@tenthstring=\@tenthstringFportuges
2587 \let\@hundredthstring=\@hundredthstringFportuges
2588 \def\@thousandth{mil'esima}%
2589 \@ordinalstringportuges{#1}{#2}%
2590 }%
2591 \global\let\@ordinalstringFportuges\@ordinalstringFportuges
    Make neuter same as masculine:
2592 \global\let\@ordinalstringNportuges\@ordinalstringMportuges
    As above, but initial letters in upper case (masculine):
2593 \newcommand*\@OrdinalstringMportuges}[2]{%
2594 \let\@unitthstring=\@Unitthstringportuges
2595 \let\@unitstring=\@Unitstringportuges
2596 \let\@teenthstring=\@teenthstringportuges
2597 \let\@tenthstring=\@Tenthstringportuges
2598 \let\@hundredthstring=\@Hundredthstringportuges
2599 \def\@thousandth{Mil'esimo}%
2600 \@ordinalstringportuges{#1}{#2}%
2601 }%
2602 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges
    Feminine form:
2603 \newcommand*\@OrdinalstringFportuges}[2]{%
2604 \let\@unitthstring=\@UnitthstringFportuges
2605 \let\@unitstring=\@UnitstringFportuges
2606 \let\@teenthstring=\@teenthstringportuges

```

```

2607 \let\@tenthstring=\@TenthstringFportuges
2608 \let\@hundredthstring=\@HundredthstringFportuges
2609 \def\@thousandth{Mil\`esima}%
2610 \@ordinalstringportuges{#1}{#2}%
2611 }%
2612 \global\let\@OrdinalstringFportuges\@OrdinalstringFportuges

  Make neuter same as masculine:
2613 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges

  In order to do the ordinals, split into units, teens, tens and hundreds. Units:
2614 \newcommand*\@unitthstringportuges[1]{%
2615   \ifcase#1\relax
2616     zero%
2617     \or primeiro%
2618     \or segundo%
2619     \or terceiro%
2620     \or quarto%
2621     \or quinto%
2622     \or sexto%
2623     \or s\`etimo%
2624     \or oitavo%
2625     \or nono%
2626   \fi
2627 }%
2628 \global\let\@unitthstringportuges\@unitthstringportuges

  Tens:
2629 \newcommand*\@tenthstringportuges[1]{%
2630   \ifcase#1\relax
2631     \or d\`ecimo%
2632     \or vig\`esimo%
2633     \or trig\`esimo%
2634     \or quadrag\`esimo%
2635     \or q\`uinquag\`esimo%
2636     \or sexag\`esimo%
2637     \or setuag\`esimo%
2638     \or octog\`esimo%
2639     \or nonag\`esimo%
2640   \fi
2641 }%
2642 \global\let\@tenthstringportuges\@tenthstringportuges

  Teens:
2643 \newcommand*\@teenthstringportuges[1]{%
2644   \@tenthstring{1}%
2645   \ifnum#1>0\relax
2646     -\@unitthstring{#1}%
2647   \fi
2648 }%
2649 \global\let\@teenthstringportuges\@teenthstringportuges

```

Hundreds:

```
2650 \newcommand*{\@@hundredthstringportuges[1]}{%
2651   \ifcase#1\relax
2652     \or cent\'esimo%
2653     \or ducent\'esimo%
2654     \or trecent\'esimo%
2655     \or quadringent\'esimo%
2656     \or q\uingent\'esimo%
2657     \or seiscent\'esimo%
2658     \or setingent\'esimo%
2659     \or octingent\'esimo%
2660     \or nongent\'esimo%
2661   \fi
2662 }%
2663 \global\let\@@hundredthstringportuges\@@hundredthstringportuges
```

Units (feminine):

```
2664 \newcommand*{\@@unitthstringFportuges[1]}{%
2665   \ifcase#1\relax
2666     zero%
2667     \or primeira%
2668     \or segunda%
2669     \or terceira%
2670     \or quarta%
2671     \or quinta%
2672     \or sexta%
2673     \or s\'etima%
2674     \or oitava%
2675     \or nona%
2676   \fi
2677 }%
2678 \global\let\@@unitthstringFportuges\@@unitthstringFportuges
```

Tens (feminine):

```
2679 \newcommand*{\@@tenthstringFportuges[1]}{%
2680   \ifcase#1\relax
2681     \or d\'ecima%
2682     \or vig\'esima%
2683     \or trig\'esima%
2684     \or quadrag\'esima%
2685     \or q\inquag\'esima%
2686     \or sexag\'esima%
2687     \or setuag\'esima%
2688     \or octog\'esima%
2689     \or nonag\'esima%
2690   \fi
2691 }%
2692 \global\let\@@tenthstringFportuges\@@tenthstringFportuges
```

Hundreds (feminine):

```

2693 \newcommand*\@@hundredthstringFportuges[1]{%
2694   \ifcase#1\relax
2695     \or cent\'esima%
2696     \or ducent\'esima%
2697     \or trecent\'esima%
2698     \or quadringent\'esima%
2699     \or q\'uingent\'esima%
2700     \or seiscent\'esima%
2701     \or setingent\'esima%
2702     \or octingent\'esima%
2703     \or nongent\'esima%
2704   \fi
2705 }%
2706 \global\let\@@hundredthstringFportuges\@@hundredthstringFportuges

```

As above, but with initial letter in upper case. Units:

```

2707 \newcommand*\@@Unitthstringportuges[1]{%
2708   \ifcase#1\relax
2709     Zero%
2710     \or Primeiro%
2711     \or Segundo%
2712     \or Terceiro%
2713     \or Quarto%
2714     \or Quinto%
2715     \or Sexto%
2716     \or S\'etimo%
2717     \or Oitavo%
2718     \or Nono%
2719   \fi
2720 }%
2721 \global\let\@@Unitthstringportuges\@@Unitthstringportuges

```

Tens:

```

2722 \newcommand*\@@Tenthstringportuges[1]{%
2723   \ifcase#1\relax
2724     \or D\'ecimo%
2725     \or Vig\'esimo%
2726     \or Trig\'esimo%
2727     \or Quadrag\'esimo%
2728     \or Q\'uinquag\'esimo%
2729     \or Sexag\'esimo%
2730     \or Setuag\'esimo%
2731     \or Octog\'esimo%
2732     \or Nonag\'esimo%
2733   \fi
2734 }%
2735 \global\let\@@Tenthstringportuges\@@Tenthstringportuges

```

Hundreds:

```

2736 \newcommand*\@@Hundredthstringportuges[1]{%
2737   \ifcase#1\relax

```

2738 \or Cent\'esimo%  
 2739 \or Ducent\'esimo%  
 2740 \or Trecent\'esimo%  
 2741 \or Quadringent\'esimo%  
 2742 \or Q"uingent\'esimo%  
 2743 \or Seiscent\'esimo%  
 2744 \or Setingent\'esimo%  
 2745 \or Octingent\'esimo%  
 2746 \or Nongent\'esimo%  
 2747 \fi  
 2748 }%  
 2749 \global\let\@@Hundredthstringportuges\@@Hundredthstringportuges

As above, but feminine. Units:

2750 \newcommand\*\@@UnitthstringFportuges[1]{%  
 2751 \ifcase#1\relax  
 2752 Zera%  
 2753 \or Primeira%  
 2754 \or Segunda%  
 2755 \or Terceira%  
 2756 \or Quarta%  
 2757 \or Quinta%  
 2758 \or Sexta%  
 2759 \or S\'etima%  
 2760 \or Oitava%  
 2761 \or Nona%  
 2762 \fi  
 2763 }%  
 2764 \global\let\@@UnitthstringFportuges\@@UnitthstringFportuges

Tens (feminine);

2765 \newcommand\*\@@TenthstringFportuges[1]{%  
 2766 \ifcase#1\relax  
 2767 \or D\'ecima%  
 2768 \or Vig\'esima%  
 2769 \or Trig\'esima%  
 2770 \or Quadrag\'esima%  
 2771 \or Q"uinquag\'esima%  
 2772 \or Sexag\'esima%  
 2773 \or Setuag\'esima%  
 2774 \or Octog\'esima%  
 2775 \or Nonag\'esima%  
 2776 \fi  
 2777 }%  
 2778 \global\let\@@TenthstringFportuges\@@TenthstringFportuges

Hundreds (feminine):

2779 \newcommand\*\@@HundredthstringFportuges[1]{%  
 2780 \ifcase#1\relax  
 2781 \or Cent\'esima%  
 2782 \or Ducent\'esima%

```

2783 \or Trecent\'esima%
2784 \or Quadringent\'esima%
2785 \or Q"uingent\'esima%
2786 \or Seiscent\'esima%
2787 \or Setingent\'esima%
2788 \or Octingent\'esima%
2789 \or Nongent\'esima%
2790 \fi
2791 }%
2792 \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.)

```

2793 \newcommand*\@@numberstringportuges[2]{%
2794 \ifnum#1>99999\relax
2795 \PackageError{fmtcount}{Out of range}%
2796 {This macro only works for values less than 100000}%
2797 \else
2798 \ifnum#1<0\relax
2799 \PackageError{fmtcount}{Negative numbers not permitted}%
2800 {This macro does not work for negative numbers, however
2801 you can try typing "minus" first, and then pass the modulus of
2802 this number}%
2803 \fi
2804 \fi
2805 \def#2{}%
2806 \@strctr=#1\relax \divide\@strctr by 1000\relax
2807 \ifnum\@strctr>9\relax
    #1 is greater or equal to 10000
2808 \divide\@strctr by 10\relax
2809 \ifnum\@strctr>1\relax
2810 \let\@@fc@numstr#2\relax
2811 \protected@edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
2812 \@strctr=#1 \divide\@strctr by 1000\relax
2813 \@FCmodulo{\@strctr}{10}%
2814 \ifnum\@strctr>0
2815 \ifnum\@strctr=1\relax
2816 \let\@@fc@numstr#2\relax
2817 \protected@edef#2{\@@fc@numstr\ \@andname}%
2818 \fi
2819 \let\@@fc@numstr#2\relax
2820 \protected@edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
2821 \fi
2822 \else
2823 \@strctr=#1\relax
2824 \divide\@strctr by 1000\relax
2825 \@FCmodulo{\@strctr}{10}%

```

```

2826 \let\@@fc@numstr#2\relax
2827 \protected@edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
2828 \fi
2829 \let\@@fc@numstr#2\relax
2830 \protected@edef#2{\@@fc@numstr\ \@thousand}%
2831 \else
2832 \ifnum\@strctr>0\relax
2833 \ifnum\@strctr>1\relax
2834 \let\@@fc@numstr#2\relax
2835 \protected@edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
2836 \fi
2837 \let\@@fc@numstr#2\relax
2838 \protected@edef#2{\@@fc@numstr\@thousand}%
2839 \fi
2840 \fi
2841 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
2842 \divide\@strctr by 100\relax
2843 \ifnum\@strctr>0\relax
2844 \ifnum#1>1000 \relax
2845 \let\@@fc@numstr#2\relax
2846 \protected@edef#2{\@@fc@numstr\ }%
2847 \fi
2848 \@tmpstrctr=#1\relax
2849 \@FCmodulo{\@tmpstrctr}{1000}%
2850 \let\@@fc@numstr#2\relax
2851 \ifnum\@tmpstrctr=100\relax
2852 \protected@edef#2{\@@fc@numstr\@tenstring{10}}%
2853 \else
2854 \protected@edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
2855 \fi%
2856 \fi
2857 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
2858 \ifnum#1>100\relax
2859 \ifnum\@strctr>0\relax
2860 \let\@@fc@numstr#2\relax
2861 \protected@edef#2{\@@fc@numstr\ \@andname\ }%
2862 \fi
2863 \fi
2864 \ifnum\@strctr>19\relax
2865 \divide\@strctr by 10\relax
2866 \let\@@fc@numstr#2\relax
2867 \protected@edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
2868 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
2869 \ifnum\@strctr>0
2870 \ifnum\@strctr=1\relax
2871 \let\@@fc@numstr#2\relax
2872 \protected@edef#2{\@@fc@numstr\ \@andname}%
2873 \else
2874 \ifnum#1>100\relax

```

```

2875     \let\@fc@numstr#2\relax
2876     \protected@edef#2{\@fc@numstr\ \@andname}%
2877     \fi
2878     \fi
2879     \let\@fc@numstr#2\relax
2880     \protected@edef#2{\@fc@numstr\ \@unitstring{\@strctr}}%
2881     \fi
2882 \else
2883     \ifnum\@strctr<10\relax
2884     \ifnum\@strctr=0\relax
2885         \ifnum#1<100\relax
2886             \let\@fc@numstr#2\relax
2887             \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}}%
2888             \fi
2889         \else %(>0,<10)
2890             \let\@fc@numstr#2\relax
2891             \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}}%
2892             \fi
2893         \else%>10
2894             \@FCmodulo{\@strctr}{10}%
2895             \let\@fc@numstr#2\relax
2896             \protected@edef#2{\@fc@numstr\@teenstring{\@strctr}}%
2897             \fi
2898     \fi
2899 }%
2900 \global\let\@numberstringportuges\@numberstringportuges

```

As above, but for ordinals.

```

2901 \newcommand*\@ordinalstringportuges[2]{%
2902 \@strctr=#1\relax
2903 \ifnum#1>99999
2904 \PackageError{fmtcount}{Out of range}%
2905 {This macro only works for values less than 100000}%
2906 \else
2907 \ifnum#1<0
2908 \PackageError{fmtcount}{Negative numbers not permitted}%
2909 {This macro does not work for negative numbers, however
2910 you can try typing "minus" first, and then pass the modulus of
2911 this number}%
2912 \else
2913 \def#2{}%
2914 \ifnum\@strctr>999\relax
2915     \divide\@strctr by 1000\relax
2916     \ifnum\@strctr>1\relax
2917         \ifnum\@strctr>9\relax
2918             \@tmpstrctr=\@strctr
2919             \ifnum\@strctr<20
2920                 \@FCmodulo{\@tmpstrctr}{10}%
2921                 \let\@fc@ordstr#2\relax
2922                 \protected@edef#2{\@fc@ordstr\@teenthstring{\@tmpstrctr}}%

```



```

2923     \else
2924         \divide\@tmpstrctr by 10\relax
2925         \let\@@fc@ordstr#2\relax
2926         \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
2927         \@tmpstrctr=\@strctr
2928         \@FCmodulo{\@tmpstrctr}{10}%
2929         \ifnum\@tmpstrctr>0\relax
2930             \let\@@fc@ordstr#2\relax
2931             \protected@edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
2932         \fi
2933     \fi
2934 \else
2935     \let\@@fc@ordstr#2\relax
2936     \protected@edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
2937 \fi
2938 \fi
2939 \let\@@fc@ordstr#2\relax
2940 \protected@edef#2{\@@fc@ordstr\@thousandth}%
2941 \fi
2942 \@strctr=#1\relax
2943 \@FCmodulo{\@strctr}{1000}%
2944 \ifnum\@strctr>99\relax
2945     \@tmpstrctr=\@strctr
2946     \divide\@tmpstrctr by 100\relax
2947     \ifnum#1>1000\relax
2948         \let\@@fc@ordstr#2\relax
2949         \protected@edef#2{\@@fc@ordstr-}%
2950     \fi
2951     \let\@@fc@ordstr#2\relax
2952     \protected@edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
2953 \fi
2954 \@FCmodulo{\@strctr}{100}%
2955 \ifnum#1>99\relax
2956     \ifnum\@strctr>0\relax
2957         \let\@@fc@ordstr#2\relax
2958         \protected@edef#2{\@@fc@ordstr-}%
2959     \fi
2960 \fi
2961 \ifnum\@strctr>9\relax
2962     \@tmpstrctr=\@strctr
2963     \divide\@tmpstrctr by 10\relax
2964     \let\@@fc@ordstr#2\relax
2965     \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
2966     \@tmpstrctr=\@strctr
2967     \@FCmodulo{\@tmpstrctr}{10}%
2968     \ifnum\@tmpstrctr>0\relax
2969         \let\@@fc@ordstr#2\relax
2970         \protected@edef#2{\@@fc@ordstr-\@unitthstring{\@tmpstrctr}}%
2971     \fi

```

```

2972 \else
2973   \ifnum \@strctr=0\relax
2974     \ifnum #1=0\relax
2975       \let \@fc@ordstr#2\relax
2976       \protected@edef#2{\@fc@ordstr\@unitstring{0}}%
2977     \fi
2978   \else
2979     \let \@fc@ordstr#2\relax
2980     \protected@edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
2981   \fi
2982 \fi
2983 \fi
2984 \fi
2985 }%
2986 \global\let \@ordinalstringportuges \@ordinalstringportuges

```

#### 10.1.14 fc-portuguese.def

```

2987 \ProvidesFCLanguage{portuguese}[2014/06/09]%

```

Load fc-portuges.def if not already loaded.

```

2988 \FCloadlang{portuges}%

```

Set portuguese to be equivalent to portuges.

```

2989 \global\let \@ordinalMportuguese \@ordinalMportuges
2990 \global\let \@ordinalFportuguese \@ordinalFportuges
2991 \global\let \@ordinalNportuguese \@ordinalNportuges
2992 \global\let \@numberstringMportuguese \@numberstringMportuges
2993 \global\let \@numberstringFportuguese \@numberstringFportuges
2994 \global\let \@numberstringNportuguese \@numberstringNportuges
2995 \global\let \@NumberstringMportuguese \@NumberstringMportuges
2996 \global\let \@NumberstringFportuguese \@NumberstringFportuges
2997 \global\let \@NumberstringNportuguese \@NumberstringNportuges
2998 \global\let \@ordinalstringMportuguese \@ordinalstringMportuges
2999 \global\let \@ordinalstringFportuguese \@ordinalstringFportuges
3000 \global\let \@ordinalstringNportuguese \@ordinalstringNportuges
3001 \global\let \@OrdinalstringMportuguese \@OrdinalstringMportuges
3002 \global\let \@OrdinalstringFportuguese \@OrdinalstringFportuges
3003 \global\let \@OrdinalstringNportuguese \@OrdinalstringNportuges

```

#### 10.1.15 fc-spanish.def

Spanish definitions

```

3004 \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:

```

3005 \newcommand* \@ordinalMspanish[2]{%
3006   \edef#2{\number#1\relax\noexpand\fmtord{o}}%
3007 }%
3008 \global\let \@ordinalMspanish \@ordinalMspanish

```

Feminine:

```
3009 \newcommand{\@ordinalFspanish}[2]{%
3010   \edef#2{\number#1\relax\noexpand\fmtord{a}}%
3011 }%
3012 \global\let\@ordinalFspanish\@ordinalFspanish
```

Make neuter same as masculine:

```
3013 \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):

```
3014 \newcommand*\@unitstringspanish[1]{%
3015   \ifcase#1\relax
3016     cero%
3017     \or uno%
3018     \or dos%
3019     \or tres%
3020     \or cuatro%
3021     \or cinco%
3022     \or seis%
3023     \or siete%
3024     \or ocho%
3025     \or nueve%
3026   \fi
3027 }%
3028 \global\let\@unitstringspanish\@unitstringspanish
```

Feminine:

```
3029 \newcommand*\@unitstringFspanish[1]{%
3030   \ifcase#1\relax
3031     cera%
3032     \or una%
3033     \or dos%
3034     \or tres%
3035     \or cuatro%
3036     \or cinco%
3037     \or seis%
3038     \or siete%
3039     \or ocho%
3040     \or nueve%
3041   \fi
3042 }%
3043 \global\let\@unitstringFspanish\@unitstringFspanish
```

Tens (argument must go from 1 to 10):

```
3044 \newcommand*\@tenstringspanish[1]{%
3045   \ifcase#1\relax
3046     \or diez%
3047     \or veinte%
3048     \or treinta%
3049     \or cuarenta%
```

```

3050     \or cincuenta%
3051     \or sesenta%
3052     \or setenta%
3053     \or ochenta%
3054     \or noventa%
3055     \or cien%
3056     \fi
3057 }%
3058 \global\let\@@tenstringspanish\@@tenstringspanish

  Teens:
3059 \newcommand*\@@teenstringspanish[1]{%
3060   \ifcase#1\relax
3061     diez%
3062     \or once%
3063     \or doce%
3064     \or trece%
3065     \or catorce%
3066     \or quince%
3067     \or diecis\'eis%
3068     \or diecisiete%
3069     \or dieciocho%
3070     \or diecinueve%
3071     \fi
3072 }%
3073 \global\let\@@teenstringspanish\@@teenstringspanish

  Twenties:
3074 \newcommand*\@@twentystringspanish[1]{%
3075   \ifcase#1\relax
3076     veinte%
3077     \or veintiuno%
3078     \or veintid\'os%
3079     \or veintitr\'es%
3080     \or veinticuatro%
3081     \or veinticinco%
3082     \or veintis\'eis%
3083     \or veintisiete%
3084     \or veintiocho%
3085     \or veintinueve%
3086     \fi
3087 }%
3088 \global\let\@@twentystringspanish\@@twentystringspanish

  Feminine form:
3089 \newcommand*\@@twentystringFspanish[1]{%
3090   \ifcase#1\relax
3091     veinte%
3092     \or veintiuna%
3093     \or veintid\'os%
3094     \or veintitr\'es%

```

3095 \or veinticuatro%  
 3096 \or veinticinco%  
 3097 \or veintisis\'eis%  
 3098 \or veintisiete%  
 3099 \or veintiocho%  
 3100 \or veintinueve%  
 3101 \fi  
 3102 }%  
 3103 \global\let\@@twentystringFspanish\@@twentystringFspanish

**Hundreds:**

3104 \newcommand\*\@@hundredstringspanish[1]{%  
 3105 \ifcase#1\relax  
 3106 \or ciento%  
 3107 \or doscientos%  
 3108 \or trescientos%  
 3109 \or cuatrocientos%  
 3110 \or quinientos%  
 3111 \or seiscientos%  
 3112 \or setecientos%  
 3113 \or ochocientos%  
 3114 \or novecientos%  
 3115 \fi  
 3116 }%  
 3117 \global\let\@@hundredstringspanish\@@hundredstringspanish

**Feminine form:**

3118 \newcommand\*\@@hundredstringFspanish[1]{%  
 3119 \ifcase#1\relax  
 3120 \or cienta%  
 3121 \or doscientas%  
 3122 \or trescientas%  
 3123 \or cuatrocientas%  
 3124 \or quinientas%  
 3125 \or seiscientas%  
 3126 \or setecientas%  
 3127 \or ochocientas%  
 3128 \or novecientas%  
 3129 \fi  
 3130 }%  
 3131 \global\let\@@hundredstringFspanish\@@hundredstringFspanish

**As above, but with initial letter uppercase:**

3132 \newcommand\*\@@Unitstringspanish[1]{%  
 3133 \ifcase#1\relax  
 3134 Cero%  
 3135 \or Uno%  
 3136 \or Dos%  
 3137 \or Tres%  
 3138 \or Cuatro%  
 3139 \or Cinco%

```

3140 \or Seis%
3141 \or Siete%
3142 \or Ocho%
3143 \or Nueve%
3144 \fi
3145 }%
3146 \global\let\@@Unitstringspanish\@@Unitstringspanish

```

Feminine form:

```

3147 \newcommand*\@@UnitstringFspanish[1]{%
3148 \ifcase#1\relax
3149 Cera%
3150 \or Una%
3151 \or Dos%
3152 \or Tres%
3153 \or Cuatro%
3154 \or Cinco%
3155 \or Seis%
3156 \or Siete%
3157 \or Ocho%
3158 \or Nueve%
3159 \fi
3160 }%
3161 \global\let\@@UnitstringFspanish\@@UnitstringFspanish

```

Tens:

```

3162 %\changes{2.0}{2012-06-18}{fixed spelling mistake (correction
3163 %provided by Fernando Maldonado)}
3164 \newcommand*\@@Tenstringspanish[1]{%
3165 \ifcase#1\relax
3166 \or Diez%
3167 \or Veinte%
3168 \or Treinta%
3169 \or Cuarenta%
3170 \or Cincuenta%
3171 \or Sesenta%
3172 \or Setenta%
3173 \or Ochenta%
3174 \or Noventa%
3175 \or Cien%
3176 \fi
3177 }%
3178 \global\let\@@Tenstringspanish\@@Tenstringspanish

```

Teens:

```

3179 \newcommand*\@@Teenstringspanish[1]{%
3180 \ifcase#1\relax
3181 Diez%
3182 \or Once%
3183 \or Doce%
3184 \or Trece%

```

3185 \or Catorce%  
 3186 \or Quince%  
 3187 \or Diecis\'eis%  
 3188 \or Diecisiete%  
 3189 \or Dieciocho%  
 3190 \or Diecinueve%  
 3191 \fi  
 3192 }%  
 3193 \global\let\@@Teenstringspanish\@@Teenstringspanish

**Twenties:**

3194 \newcommand\*\@@Twentystringspanish[1]{%  
 3195 \ifcase#1\relax  
 3196 Veinte%  
 3197 \or Veintiuno%  
 3198 \or Veintid\'os%  
 3199 \or Veintitr\'es%  
 3200 \or Veinticuatro%  
 3201 \or Veinticinco%  
 3202 \or Veintis\'eis%  
 3203 \or Veintisiete%  
 3204 \or Veintiocho%  
 3205 \or Veintinueve%  
 3206 \fi  
 3207 }%  
 3208 \global\let\@@Twentystringspanish\@@Twentystringspanish

**Feminine form:**

3209 \newcommand\*\@@TwentystringFspanish[1]{%  
 3210 \ifcase#1\relax  
 3211 Veinte%  
 3212 \or Veintiuna%  
 3213 \or Veintid\'os%  
 3214 \or Veintitr\'es%  
 3215 \or Veinticuatro%  
 3216 \or Veinticinco%  
 3217 \or Veintis\'eis%  
 3218 \or Veintisiete%  
 3219 \or Veintiocho%  
 3220 \or Veintinueve%  
 3221 \fi  
 3222 }%  
 3223 \global\let\@@TwentystringFspanish\@@TwentystringFspanish

**Hundreds:**

3224 \newcommand\*\@@Hundredstringspanish[1]{%  
 3225 \ifcase#1\relax  
 3226 \or Ciento%  
 3227 \or Doscientos%  
 3228 \or Trescientos%  
 3229 \or Cuatrocientos%

```

3230 \or Quinientos%
3231 \or Seiscientos%
3232 \or Setecientos%
3233 \or Ochocientos%
3234 \or Novecientos%
3235 \fi
3236 }%
3237 \global\let\@@Hundredstringspanish\@@Hundredstringspanish

```

Feminine form:

```

3238 \newcommand*\@@HundredstringFspanish[1]{%
3239 \ifcase#1\relax
3240 \or Cienta%
3241 \or Doscientas%
3242 \or Trescientas%
3243 \or Cuatrocientas%
3244 \or Quinientas%
3245 \or Seiscientas%
3246 \or Setecientas%
3247 \or Ochocientas%
3248 \or Novecientas%
3249 \fi
3250 }%
3251 \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.)

```

3252 \newcommand*\@numberstringMspanish}[2]{%
3253 \let\@unitstring=\@unitstringspanish
3254 \let\@teenstring=\@teenstringspanish
3255 \let\@tenstring=\@tenstringspanish
3256 \let\@twentystring=\@twentystringspanish
3257 \let\@hundredstring=\@hundredstringspanish
3258 \def\@hundred{cien}\def\@thousand{mil}%
3259 \def\@andname{y}%
3260 \@numberstringspanish{#1}{#2}%
3261 }%
3262 \global\let\@numberstringMspanish\@numberstringMspanish

```

Feminine form:

```

3263 \newcommand*\@numberstringFspanish}[2]{%
3264 \let\@unitstring=\@unitstringFspanish
3265 \let\@teenstring=\@teenstringspanish
3266 \let\@tenstring=\@tenstringspanish
3267 \let\@twentystring=\@twentystringFspanish
3268 \let\@hundredstring=\@hundredstringFspanish
3269 \def\@hundred{cien}\def\@thousand{mil}%
3270 \def\@andname{b}%
3271 \@numberstringspanish{#1}{#2}%

```



```

3272 }%
3273 \global\let\@numberstringFspanish\@numberstringFspanish
    Make neuter same as masculine:
3274 \global\let\@numberstringNspanish\@numberstringMspanish
    As above, but initial letters in upper case:
3275 \newcommand*\@NumberstringMspanish}[2]{%
3276   \let\@unitstring=\@Unitstringspanish
3277   \let\@teenstring=\@Teenstringspanish
3278   \let\@tenstring=\@Tenstringspanish
3279   \let\@twentystring=\@Twentystringspanish
3280   \let\@hundredstring=\@Hundredstringspanish
3281   \def\@andname{y}%
3282   \def\@hundred{Cien}\def\@thousand{Mil}%
3283   \@numberstringspanish{#1}{#2}%
3284 }%
3285 \global\let\@NumberstringMspanish\@NumberstringMspanish
    Feminine form:
3286 \newcommand*\@NumberstringFspanish}[2]{%
3287   \let\@unitstring=\@UnitstringFspanish
3288   \let\@teenstring=\@Teenstringspanish
3289   \let\@tenstring=\@Tenstringspanish
3290   \let\@twentystring=\@TwentystringFspanish
3291   \let\@hundredstring=\@HundredstringFspanish
3292   \def\@andname{b}%
3293   \def\@hundred{Cien}\def\@thousand{Mil}%
3294   \@numberstringspanish{#1}{#2}%
3295 }%
3296 \global\let\@NumberstringFspanish\@NumberstringFspanish
    Make neuter same as masculine:
3297 \global\let\@NumberstringNspanish\@NumberstringMspanish
    As above, but for ordinals.
3298 \newcommand*\@ordinalstringMspanish}[2]{%
3299   \let\@unitthstring=\@unitthstringspanish
3300   \let\@unitstring=\@unitstringspanish
3301   \let\@teenthstring=\@teenthstringspanish
3302   \let\@tenthstring=\@tenthstringspanish
3303   \let\@hundredthstring=\@hundredthstringspanish
3304   \def\@thousandth{mil'esimo}%
3305   \@ordinalstringspanish{#1}{#2}%
3306 }%
3307 \global\let\@ordinalstringMspanish\@ordinalstringMspanish
    Feminine form:
3308 \newcommand*\@ordinalstringFspanish}[2]{%
3309   \let\@unitthstring=\@unitthstringFspanish
3310   \let\@unitstring=\@unitstringFspanish
3311   \let\@teenthstring=\@teenthstringFspanish

```

```

3312 \let\tenthstring=\@tenthstringFspanish
3313 \let\@hundredthstring=\@hundredthstringFspanish
3314 \def\@thousandth{mil\'esima}%
3315 \@ordinalstringspanish{#1}{#2}%
3316 }%
3317 \global\let\@ordinalstringFspanish\@ordinalstringFspanish
    Make neuter same as masculine:
3318 \global\let\@ordinalstringMspanish\@ordinalstringMspanish
    As above, but with initial letters in upper case.
3319 \newcommand*\@OrdinalstringMspanish}[2]{%
3320 \let\@unitthstring=\@Unitthstringspanish
3321 \let\@unitstring=\@Unitstringspanish
3322 \let\@teenthstring=\@Teenthstringspanish
3323 \let\@tenthstring=\@Tenthstringspanish
3324 \let\@hundredthstring=\@Hundredthstringspanish
3325 \def\@thousandth{Mil\'esimo}%
3326 \@ordinalstringspanish{#1}{#2}%
3327 }
3328 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish
    Feminine form:
3329 \newcommand*\@OrdinalstringFspanish}[2]{%
3330 \let\@unitthstring=\@UnitthstringFspanish
3331 \let\@unitstring=\@UnitstringFspanish
3332 \let\@teenthstring=\@TeenthstringFspanish
3333 \let\@tenthstring=\@TenthstringFspanish
3334 \let\@hundredthstring=\@HundredthstringFspanish
3335 \def\@thousandth{Mil\'esima}%
3336 \@ordinalstringspanish{#1}{#2}%
3337 }%
3338 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish
    Make neuter same as masculine:
3339 \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:
3340 \newcommand*\@unitthstringspanish[1]{%
3341 \ifcase#1\relax
3342   cero%
3343   \or primero%
3344   \or segundo%
3345   \or tercero%
3346   \or cuarto%
3347   \or quinto%
3348   \or sexto%
3349   \or s\'eptimo%
3350   \or octavo%
3351   \or noveno%

```

```

3352 \fi
3353 }%
3354 \global\let\@@unitthstringspanish\@@unitthstringspanish

```

Tens:

```

3355 \newcommand*\@@tenthstringspanish[1]{%
3356 \ifcase#1\relax
3357 \or d\'ecimo%
3358 \or vig\'esimo%
3359 \or trig\'esimo%
3360 \or cuadrag\'esimo%
3361 \or quincuag\'esimo%
3362 \or sexag\'esimo%
3363 \or septuag\'esimo%
3364 \or octog\'esimo%
3365 \or nonag\'esimo%
3366 \fi
3367 }%
3368 \global\let\@@tenthstringspanish\@@tenthstringspanish

```

Teens:

```

3369 \newcommand*\@@teenthstringspanish[1]{%
3370 \ifcase#1\relax
3371 d\'ecimo%
3372 \or und\'ecimo%
3373 \or duod\'ecimo%
3374 \or decimotercero%
3375 \or decimocuarto%
3376 \or decimoquinto%
3377 \or decimosexto%
3378 \or decimos\'eptimo%
3379 \or decimoctavo%
3380 \or decimonoveno%
3381 \fi
3382 }%
3383 \global\let\@@teenthstringspanish\@@teenthstringspanish

```

Hundreds:

```

3384 \newcommand*\@@hundredthstringspanish[1]{%
3385 \ifcase#1\relax
3386 \or cent\'esimo%
3387 \or ducent\'esimo%
3388 \or tricent\'esimo%
3389 \or cuadingent\'esimo%
3390 \or quingent\'esimo%
3391 \or sexcent\'esimo%
3392 \or septing\'esimo%
3393 \or octingent\'esimo%
3394 \or noningent\'esimo%
3395 \fi
3396 }%

```

3397 \global\let\@@hundredthstringspanish\@@hundredthstringspanish

Units (feminine):

3398 \newcommand\*\@@unitthstringFspanish[1]{%

3399 \ifcase#1\relax

3400 \cer%

3401 \or primera%

3402 \or segunda%

3403 \or tercera%

3404 \or cuarta%

3405 \or quinta%

3406 \or sexta%

3407 \or s\septima%

3408 \or octava%

3409 \or novena%

3410 \fi

3411 }%

3412 \global\let\@@unitthstringFspanish\@@unitthstringFspanish

Tens (feminine):

3413 \newcommand\*\@@tenthstringFspanish[1]{%

3414 \ifcase#1\relax

3415 \ord\decima%

3416 \or vig\esima%

3417 \or trig\esima%

3418 \or cuadrag\esima%

3419 \or quincuag\esima%

3420 \or sexag\esima%

3421 \or septuag\esima%

3422 \or octog\esima%

3423 \or nonag\esima%

3424 \fi

3425 }%

3426 \global\let\@@tenthstringFspanish\@@tenthstringFspanish

Teens (feminine)

3427 \newcommand\*\@@teenthstringFspanish[1]{%

3428 \ifcase#1\relax

3429 \d\decima%

3430 \or und\decima%

3431 \or duod\decima%

3432 \or decimotercera%

3433 \or decimocuarta%

3434 \or decimoquinta%

3435 \or decimosexta%

3436 \or decimos\septima%

3437 \or decimoctava%

3438 \or decimonovena%

3439 \fi

3440 }%

3441 \global\let\@@teenthstringFspanish\@@teenthstringFspanish

Hundreds (feminine)

```
3442 \newcommand*{\@@hundredthstringFspanish[1]}{%
3443   \ifcase#1\relax
3444     \or cent\'esima%
3445     \or ducent\'esima%
3446     \or tricent\'esima%
3447     \or cuadingent\'esima%
3448     \or quingent\'esima%
3449     \or sexcent\'esima%
3450     \or septing\'esima%
3451     \or octingent\'esima%
3452     \or noningent\'esima%
3453   \fi
3454 }%
3455 \global\let\@@hundredthstringFspanish\@@hundredthstringFspanish
```

As above, but with initial letters in upper case

```
3456 \newcommand*{\@@Unitthstringspanish[1]}{%
3457   \ifcase#1\relax
3458     Cero%
3459     \or Primero%
3460     \or Segundo%
3461     \or Tercero%
3462     \or Cuarto%
3463     \or Quinto%
3464     \or Sexto%
3465     \or S\'eptimo%
3466     \or Octavo%
3467     \or Noveno%
3468   \fi
3469 }%
3470 \global\let\@@Unitthstringspanish\@@Unitthstringspanish
```

Tens:

```
3471 \newcommand*{\@@Tenthstringspanish[1]}{%
3472   \ifcase#1\relax
3473     \or D\'ecimo%
3474     \or Vig\'esimo%
3475     \or Trig\'esimo%
3476     \or Cuadrag\'esimo%
3477     \or Quincuag\'esimo%
3478     \or Sexag\'esimo%
3479     \or Septuag\'esimo%
3480     \or Octog\'esimo%
3481     \or Nonag\'esimo%
3482   \fi
3483 }%
3484 \global\let\@@Tenthstringspanish\@@Tenthstringspanish
```

Teens:

```

3485 \newcommand*{\@@Teenthstringspanish[1]}{
3486   \ifcase#1\relax
3487     D\'ecimo%
3488     \or Und\'ecimo%
3489     \or Duod\'ecimo%
3490     \or Decimotercero%
3491     \or Decimocuarto%
3492     \or Decimoquinto%
3493     \or Decimosexto%
3494     \or Decimos\'eptimo%
3495     \or Decimooctavo%
3496     \or Decimonoveno%
3497   \fi
3498 }%
3499 \global\let\@@Teenthstringspanish\@@Teenthstringspanish

```

### Hundreds

```

3500 \newcommand*{\@@Hundredthstringspanish[1]}{
3501   \ifcase#1\relax
3502     \or Cent\'esimo%
3503     \or Ducent\'esimo%
3504     \or Tricent\'esimo%
3505     \or Cuadringent\'esimo%
3506     \or Quingent\'esimo%
3507     \or Sexcent\'esimo%
3508     \or Septing\'esimo%
3509     \or Octingent\'esimo%
3510     \or Noningent\'esimo%
3511   \fi
3512 }%
3513 \global\let\@@Hundredthstringspanish\@@Hundredthstringspanish

```

As above, but feminine.

```

3514 \newcommand*{\@@UnitthstringFspanish[1]}{
3515   \ifcase#1\relax
3516     Cera%
3517     \or Primera%
3518     \or Segunda%
3519     \or Tercera%
3520     \or Cuarta%
3521     \or Quinta%
3522     \or Sexta%
3523     \or S\'eptima%
3524     \or Octava%
3525     \or Novena%
3526   \fi
3527 }%
3528 \global\let\@@UnitthstringFspanish\@@UnitthstringFspanish

```

Tens (feminine)

```

3529 \newcommand*{\@@TenthstringFspanish[1]}{

```

```

3530 \ifcase#1\relax
3531 \or D\'ecima%
3532 \or Vig\'esima%
3533 \or Trig\'esima%
3534 \or Cuadrag\'esima%
3535 \or Quincuag\'esima%
3536 \or Sexag\'esima%
3537 \or Septuag\'esima%
3538 \or Octog\'esima%
3539 \or Nonag\'esima%
3540 \fi
3541 }%
3542 \global\let\@@TenthstringFspanish\@@TenthstringFspanish

```

Teens (feminine):

```

3543 \newcommand*\@@TeenthstringFspanish[1]{%
3544 \ifcase#1\relax
3545 D\'ecima%
3546 \or Und\'ecima%
3547 \or Duod\'ecima%
3548 \or Decimotercera%
3549 \or Decimocuarta%
3550 \or Decimoquinta%
3551 \or Decimosexta%
3552 \or Decimos\'eptima%
3553 \or Decimoctava%
3554 \or Decimonovena%
3555 \fi
3556 }%
3557 \global\let\@@TeenthstringFspanish\@@TeenthstringFspanish

```

Hundreds (feminine):

```

3558 \newcommand*\@@HundredthstringFspanish[1]{%
3559 \ifcase#1\relax
3560 \or Cent\'esima%
3561 \or Ducent\'esima%
3562 \or Tricent\'esima%
3563 \or Cuadringent\'esima%
3564 \or Quingent\'esima%
3565 \or Sexcent\'esima%
3566 \or Septing\'esima%
3567 \or Octingent\'esima%
3568 \or Noningent\'esima%
3569 \fi
3570 }%
3571 \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 documnets created with older versions. (These internal

macros are not meant for use in documents.)

```
3572 \newcommand*\@@numberstringspanish[2]{%
3573 \ifnum#1>99999
3574 \PackageError{fmtcount}{Out of range}%
3575 {This macro only works for values less than 100000}%
3576 \else
3577 \ifnum#1<0
3578 \PackageError{fmtcount}{Negative numbers not permitted}%
3579 {This macro does not work for negative numbers, however
3580 you can try typing "minus" first, and then pass the modulus of
3581 this number}%
3582 \fi
3583 \fi
3584 \def#2{}%
3585 \@strctr=#1\relax \divide\@strctr by 1000\relax
3586 \ifnum\@strctr>9
    #1 is greater or equal to 10000
3587 \divide\@strctr by 10
3588 \ifnum\@strctr>1
3589 \let\@@fc@numstr#2\relax
3590 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
3591 \@strctr=#1 \divide\@strctr by 1000\relax
3592 \@FCmodulo{\@strctr}{10}%
3593 \ifnum\@strctr>0\relax
3594 \let\@@fc@numstr#2\relax
3595 \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
3596 \fi
3597 \else
3598 \@strctr=#1\relax
3599 \divide\@strctr by 1000\relax
3600 \@FCmodulo{\@strctr}{10}%
3601 \let\@@fc@numstr#2\relax
3602 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
3603 \fi
3604 \let\@@fc@numstr#2\relax
3605 \edef#2{\@@fc@numstr\ \@thousand}%
3606 \else
3607 \ifnum\@strctr>0\relax
3608 \ifnum\@strctr>1\relax
3609 \let\@@fc@numstr#2\relax
3610 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
3611 \fi
3612 \let\@@fc@numstr#2\relax
3613 \edef#2{\@@fc@numstr\@thousand}%
3614 \fi
3615 \fi
3616 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
3617 \divide\@strctr by 100\relax
3618 \ifnum\@strctr>0\relax
```



```

3619 \ifnum#1>1000\relax
3620   \let\@fc@numstr#2\relax
3621   \edef#2{\@fc@numstr\ }%
3622 \fi
3623 \@tmpstrctr=#1\relax
3624 \@FCmodulo{\@tmpstrctr}{1000}%
3625 \ifnum\@tmpstrctr=100\relax
3626   \let\@fc@numstr#2\relax
3627   \edef#2{\@fc@numstr\@tenstring{10}}%
3628 \else
3629   \let\@fc@numstr#2\relax
3630   \edef#2{\@fc@numstr\@hundredstring{\@strctr}}%
3631 \fi
3632 \fi
3633 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
3634 \ifnum#1>100\relax
3635   \ifnum\@strctr>0\relax
3636     \let\@fc@numstr#2\relax
3637     \edef#2{\@fc@numstr\ }%
3638   \fi
3639 \fi
3640 \ifnum\@strctr>29\relax
3641   \divide\@strctr by 10\relax
3642   \let\@fc@numstr#2\relax
3643   \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
3644   \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
3645   \ifnum\@strctr>0\relax
3646     \let\@fc@numstr#2\relax
3647     \edef#2{\@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
3648   \fi
3649 \else
3650   \ifnum\@strctr<10\relax
3651     \ifnum\@strctr=0\relax
3652       \ifnum#1<100\relax
3653         \let\@fc@numstr#2\relax
3654         \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
3655       \fi
3656     \else
3657       \let\@fc@numstr#2\relax
3658       \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
3659     \fi
3660   \else
3661     \ifnum\@strctr>19\relax
3662       \@FCmodulo{\@strctr}{10}%
3663       \let\@fc@numstr#2\relax
3664       \edef#2{\@fc@numstr\@twentystring{\@strctr}}%
3665     \else
3666       \@FCmodulo{\@strctr}{10}%
3667       \let\@fc@numstr#2\relax

```

```

3668     \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
3669     \fi
3670 \fi
3671 \fi
3672 }%
3673 \global\let\@@numberstringspanish\@numberstringspanish

```

As above, but for ordinals

```

3674 \newcommand*\@@ordinalstringspanish[2]{%
3675 \@strctr=#1\relax
3676 \ifnum#1>99999
3677 \PackageError{fmtcount}{Out of range}%
3678 {This macro only works for values less than 100000}%
3679 \else
3680 \ifnum#1<0
3681 \PackageError{fmtcount}{Negative numbers not permitted}%
3682 {This macro does not work for negative numbers, however
3683 you can try typing "minus" first, and then pass the modulus of
3684 this number}%
3685 \else
3686 \def#2{}%
3687 \ifnum\@strctr>999\relax
3688   \divide\@strctr by 1000\relax
3689   \ifnum\@strctr>1\relax
3690     \ifnum\@strctr>9\relax
3691       \@tmpstrctr=\@strctr
3692       \ifnum\@strctr<20
3693         \@FCmodulo{\@tmpstrctr}{10}%
3694         \let\@@fc@ordstr#2\relax
3695         \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
3696       \else
3697         \divide\@tmpstrctr by 10\relax
3698         \let\@@fc@ordstr#2\relax
3699         \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
3700       \@tmpstrctr=\@strctr
3701       \@FCmodulo{\@tmpstrctr}{10}%
3702       \ifnum\@tmpstrctr>0\relax
3703         \let\@@fc@ordstr#2\relax
3704         \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
3705       \fi
3706     \fi
3707   \else
3708     \let\@@fc@ordstr#2\relax
3709     \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
3710   \fi
3711 \fi
3712 \let\@@fc@ordstr#2\relax
3713 \edef#2{\@@fc@ordstr\@thousandth}%
3714 \fi
3715 \@strctr=#1\relax

```

```

3716 \@FCmodulo{\@strctr}{1000}%
3717 \ifnum \@strctr > 99 \relax
3718   \@tmpstrctr = \@strctr
3719   \divide \@tmpstrctr by 100 \relax
3720   \ifnum #1 > 1000 \relax
3721     \let @@fc@ordstr #2 \relax
3722     \edef #2 {\@fc@ordstr \ }%
3723   \fi
3724   \let @@fc@ordstr #2 \relax
3725   \edef #2 {\@fc@ordstr \@hundredthstring{\@tmpstrctr}}%
3726 \fi
3727 \@FCmodulo{\@strctr}{100}%
3728 \ifnum #1 > 99 \relax
3729   \ifnum \@strctr > 0 \relax
3730     \let @@fc@ordstr #2 \relax
3731     \edef #2 {\@fc@ordstr \ }%
3732   \fi
3733 \fi
3734 \ifnum \@strctr > 19 \relax
3735   \@tmpstrctr = \@strctr
3736   \divide \@tmpstrctr by 10 \relax
3737   \let @@fc@ordstr #2 \relax
3738   \edef #2 {\@fc@ordstr \@tenthstring{\@tmpstrctr}}%
3739   \@tmpstrctr = \@strctr
3740   \@FCmodulo{\@tmpstrctr}{10}%
3741   \ifnum \@tmpstrctr > 0 \relax
3742     \let @@fc@ordstr #2 \relax
3743     \edef #2 {\@fc@ordstr \@unitthstring{\@tmpstrctr}}%
3744   \fi
3745 \else
3746   \ifnum \@strctr > 9 \relax
3747     \@FCmodulo{\@strctr}{10}%
3748     \let @@fc@ordstr #2 \relax
3749     \edef #2 {\@fc@ordstr \@teenthstring{\@strctr}}%
3750   \else
3751     \ifnum \@strctr = 0 \relax
3752       \ifnum #1 = 0 \relax
3753         \let @@fc@ordstr #2 \relax
3754         \edef #2 {\@fc@ordstr \@unitstring{0}}%
3755       \fi
3756     \else
3757       \let @@fc@ordstr #2 \relax
3758       \edef #2 {\@fc@ordstr \@unitthstring{\@strctr}}%
3759     \fi
3760   \fi
3761 \fi
3762 \fi
3763 \fi
3764 }%

```

3765 \global\let\@@ordinalstringspanish\@@ordinalstringspanish

### 10.1.16 fc-UKenglish.def

English definitions

3766 \ProvidesFCLanguage{UKenglish}[2013/08/17]%

Loaded fc-english.def if not already loaded

3767 \FCloadlang{english}%

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

3768 \global\let\@ordinalMUKenglish\@ordinalMenglish

3769 \global\let\@ordinalFUKenglish\@ordinalMenglish

3770 \global\let\@ordinalNUKenglish\@ordinalMenglish

3771 \global\let\@numberstringMUKenglish\@numberstringMenglish

3772 \global\let\@numberstringFUKenglish\@numberstringMenglish

3773 \global\let\@numberstringNUKenglish\@numberstringMenglish

3774 \global\let\@NumberstringMUKenglish\@NumberstringMenglish

3775 \global\let\@NumberstringFUKenglish\@NumberstringMenglish

3776 \global\let\@NumberstringNUKenglish\@NumberstringMenglish

3777 \global\let\@ordinalstringMUKenglish\@ordinalstringMenglish

3778 \global\let\@ordinalstringFUKenglish\@ordinalstringMenglish

3779 \global\let\@ordinalstringNUKenglish\@ordinalstringMenglish

3780 \global\let\@OrdinalstringMUKenglish\@OrdinalstringMenglish

3781 \global\let\@OrdinalstringFUKenglish\@OrdinalstringMenglish

3782 \global\let\@OrdinalstringNUKenglish\@OrdinalstringMenglish

### 10.1.17 fc-USenglish.def

US English definitions

3783 \ProvidesFCLanguage{USenglish}[2013/08/17]%

Loaded fc-english.def if not already loaded

3784 \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.)

3785 \global\let\@ordinalMUSenglish\@ordinalMenglish

3786 \global\let\@ordinalFUSenglish\@ordinalMenglish

3787 \global\let\@ordinalNUSenglish\@ordinalMenglish

3788 \global\let\@numberstringMUSenglish\@numberstringMenglish

3789 \global\let\@numberstringFUSenglish\@numberstringMenglish

3790 \global\let\@numberstringNUSenglish\@numberstringMenglish

3791 \global\let\@NumberstringMUSenglish\@NumberstringMenglish

3792 \global\let\@NumberstringFUSenglish\@NumberstringMenglish

3793 \global\let\@NumberstringNUSenglish\@NumberstringMenglish

3794 \global\let\@ordinalstringMUSenglish\@ordinalstringMenglish

3795 \global\let\@ordinalstringFUSenglish\@ordinalstringMenglish

3796 \global\let\@ordinalstringNUSenglish\@ordinalstringMenglish

3797 \global\let\@OrdinalstringMUSenglish\@OrdinalstringMenglish

```

3798 \global\let\@OrdinalstringFUSenglish\@OrdinalstringMenglish
3799 \global\let\@OrdinalstringNUSenglish\@OrdinalstringMenglish

```

## 10.2 fcnumparser.sty

```

3800 \NeedsTeXFormat{LaTeX2e}
3801 \ProvidesPackage{fcnumparser}[2017/06/15]

```

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

```

3802 \def\fc@counter@parser#1{%
3803   \expandafter\fc@number@parser\expandafter{\the#1.}%
3804 }
3805 \newcount\fc@digit@counter
3806
3807 \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.

```

3808 \def\fc@number@analysis#1\fc@nil{%

```

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

```

3809   \def\@tempb##1.##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%
3810   \@tempb#1.\fc@end\fc@nil
3811   \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.

```

3812   \def\@tempb##1,##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%
3813   \@tempb#1,\fc@end\fc@nil
3814   \ifx\@tempa\fc@end@

```

No comma either, so fractional part is set empty.

```

3815       \def\fc@fractional@part{}%
3816   \else

```

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

```

3817       \def\@tempb##1,\fc@end{\def\fc@fractional@part{##1}}%
3818       \expandafter\@tempb\@tempa
3819   \fi
3820 \else

```

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

```

3821       \def\@tempb##1.\fc@end{\def\fc@fractional@part{##1}}%
3822       \expandafter\@tempb\@tempa
3823   \fi
3824 }

```

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  $\text{\fc@digit@}\langle n \rangle$ , and macros  $\text{\fc@min@weight}$  and  $\text{\fc@max@weight}$  are set to the bounds for  $\langle n \rangle$ .

```
3825 \def\fc@number@parser#1{%
```

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

```
3826 \let\@tempa\@empty
3827 \def\@tempb##1##2\fc@nil{%
3828   \def\@tempc{##1}%
3829   \ifx\@tempc\space
3830   \else
3831     \expandafter\def\expandafter\@tempa\expandafter{\@tempa #1}%
3832   \fi
3833   \def\@tempc{##2}%
3834   \ifx\@tempc\@empty
3835     \expandafter\@gobble
3836   \else
3837     \expandafter\@tempb
3838   \fi
3839   ##2\fc@nil
3840 }%
3841 \@tempb#1\fc@nil
```

Get the sign into  $\text{\fc@sign}$  and the unsigned number part into  $\text{\fc@number}$ .

```
3842 \def\@tempb##1##2\fc@nil{\def\fc@sign{##1}\def\fc@number{##2}}%
3843 \expandafter\@tempb\@tempa\fc@nil
3844 \expandafter\if\fc@sign+%
3845   \def\fc@sign@case{1}%
3846 \else
3847   \expandafter\if\fc@sign-%
3848     \def\fc@sign@case{2}%
3849   \else
3850     \def\fc@sign{}%
3851     \def\fc@sign@case{0}%
3852     \let\fc@number\@tempa
3853   \fi
3854 \fi
3855 \ifx\fc@number\@empty
3856   \PackageError{fcnumparser}{Invalid number}{Number must contain at least one non blank
3857     character after sign}%
3858 \fi
```

Now, split  $\text{\fc@number}$  into  $\text{\fc@integer@part}$  and  $\text{\fc@fractional@part}$ .

```
3859 \expandafter\fc@number@analysis\fc@number\fc@nil
```

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

```
3860 \expandafter\fc@digit@counter\fc@unit@weight
3861 \expandafter\fc@parse@integer@digits\fc@integer@part\fc@end\fc@nil
```

First we compute the weight of the most significant digit: after  $\text{\fc@parse@integer@digits}$ ,

`\fc@digit@counter` is equal to `\fc@unit@weight - mw - 1` and we want to set `\fc@max@weight` to `\fc@unit@weight + mw` so we do:

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

```

3862 \fc@digit@counter -\fc@digit@counter
3863 \advance\fc@digit@counter by \fc@unit@weight
3864 \advance\fc@digit@counter by \fc@unit@weight
3865 \advance\fc@digit@counter by -1 %
3866 \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  $\text{\fc@digit@}(i + \text{offset})$  to  $\text{\fc@digit@}(i)$ . First we compute offset into `\@temp_i`.

```

3867 {%
3868   \count0 \fc@unit@weight\relax
3869   \count1 \fc@max@weight\relax
3870   \advance\count0 by -\count1 %
3871   \advance\count0 by -1 %
3872   \def\@tempa##1{\def\@tempb{\def\@temp_i{##1}}}%
3873   \expandafter\@tempa\expandafter{\the\count0}%
3874   \expandafter
3875 } \@tempb

```

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

```

3876 \expandafter\fc@digit@counter\fc@unit@weight
3877 \def\@temp_l{%
3878   \ifnum\fc@digit@counter>\fc@max@weight
3879     \let\next\relax
3880   \else

```

Here is the loop body:

```

3881   {%
3882     \count0 \@temp_i
3883     \advance\count0 by \fc@digit@counter
3884     \expandafter\def\expandafter\@temp_d\expandafter{\csname fc@digit@the\count0\endcsname
3885     \expandafter\def\expandafter\@temp_e\expandafter{\csname fc@digit@the\fc@digit@counte
3886     \def\@tempa###1###2{\def\@tempb{\let###1###2}}%
3887     \expandafter\expandafter\expandafter\@tempa\expandafter\@temp_e\@temp_d
3888     \expandafter
3889     } \@tempb
3890     \advance\fc@digit@counter by 1 %
3891   \fi
3892   \next
3893 }%
3894 \let\next\@temp_l
3895 \@temp_l

```

Split `\fc@fractional@part` into a sequence of `\fc@digit@(n)` with  $\langle n \rangle$  ranging from  $\text{\fc@unit@weight} - 1$  to  $\text{\fc@min@weight}$  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.

```

3896 \expandafter\fc@digit@counter\fc@unit@weight
3897 \expandafter\fc@parse@integer@digits\fc@fractional@part\fc@end\fc@nil

```

```

3898 \edef\fc@min@weight{\the\fc@digit@counter}%
3899 }

```

```

\fc@read@unit Macro \fc@parse@integer@digits is used to
3900 \ifcsundef{fc@parse@integer@digits}{\fi}%
3901 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of
3902 macro 'fc@parse@integer@digits'}}
3903 \def\fc@parse@integer@digits#1#2\fc@nil{%
3904 \def\@tempa{#1}%
3905 \ifx\@tempa\fc@end@
3906 \def\next##1\fc@nil{}}%
3907 \else
3908 \let\next\fc@parse@integer@digits
3909 \advance\fc@digit@counter by -1
3910 \expandafter\def\csname fc@digit@the\fc@digit@counter\endcsname{#1}%
3911 \fi
3912 \next#2\fc@nil
3913 }
3914
3915
3916 \newcommand*{\fc@unit@weight}{0}
3917

```

Now we have macros to read a few digits from the `\fc@digit@<n>` 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.
3918 \ifcsundef{fc@read@unit}{\fi}%
3919 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro 'fc@read@unit'}}

```

Arguments as follows:

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

```

3920 \def\fc@read@unit#1#2{%
3921 \ifnum#2>\fc@max@weight
3922 #1=0\relax
3923 \else
3924 \ifnum#2<\fc@min@weight
3925 #1=0\relax
3926 \else
3927 {%
3928 \edef\@tempa{\number#2}%
3929 \count0=\@tempa
3930 \edef\@tempa{\csname fc@digit@the\count0\endcsname}%
3931 \def\@tempb##1{\def\@tempa{#1=##1\relax}}%
3932 \expandafter\@tempb\expandafter{\@tempa}%
3933 \expandafter
3934 }\@tempa
3935 \fi

```



```
3936 \fi
3937 }
```

`\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.

```
3938 \ifcsundef{fc@read@hundred}{}{%
3939 \PackageError{fncnumparser}{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
3940 \def\fc@read@hundred#1#2{%
3941 {%
3942 \fc@read@unit{\count0}{#2}%
3943 \def\@tempa#1{\fc@read@unit{\count1}{#1}}%
3944 \count2=#2%
3945 \advance\count2 by 1 %
3946 \expandafter\@tempa{\the\count2}%
3947 \multiply\count1 by 10 %
3948 \advance\count1 by \count0 %
3949 \def\@tempa#1{\def\@tempb{#1=#1\relax}}
3950 \expandafter\@tempa\expandafter{\the\count1}%
3951 \expandafter
3952 }\@tempb
3953 }
```

`\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.

```
3954 \ifcsundef{fc@read@thousand}{}{%
3955 \PackageError{fncnumparser}{Duplicate definition}{Redefinition of macro
3956 '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

```
3957 \def\fc@read@thousand#1#2{%
3958 {%
3959 \fc@read@unit{\count0}{#2}%
3960 \def\@tempa#1{\fc@read@hundred{\count1}{#1}}%
3961 \count2=#2%
3962 \advance\count2 by 1 %
3963 \expandafter\@tempa{\the\count2}%
3964 \multiply\count1 by 10 %
3965 \advance\count1 by \count0 %
3966 \def\@tempa#1{\def\@tempb{#1=#1\relax}}
3967 \expandafter\@tempa\expandafter{\the\count1}%
3968 \expandafter
3969 }\@tempb
3970 }
```

`\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.

```
3971 \ifcsundef{fc@read@myriad}{}{%
3972   \PackageError{fncnumparser}{Duplicate definition}{Redefinition of macro
3973     '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

```
3974 \def\fc@read@myriad#1#2{%
3975   {%
3976     \fc@read@hundred{\count0}{#2}%
3977     \def\@tempa#1{\fc@read@hundred{\count1}{#1}}%
3978     \count2=#2
3979     \advance\count2 by 2
3980     \expandafter\@tempa{\the\count2}%
3981     \multiply\count1 by 100 %
3982     \advance\count1 by \count0 %
3983     \def\@tempa#1{\def\@tempb{#1=#1\relax}}%
3984     \expandafter\@tempa\expandafter{\the\count1}%
3985     \expandafter
3986   }\@tempb
3987 }
```

`\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.

```
3988 \ifcsundef{fc@check@nonzeros}{}{%
3989   \PackageError{fncnumparser}{Duplicate definition}{Redefinition of macro
3990     'fc@check@nonzeros'}}}
```

Arguments as follows:

#1 input number: minimum unit unit weight at which start to search the non-zeros

#2 input number: maximum unit weight at which end to seach 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.

```
3991 \def\fc@check@nonzeros#1#2#3{%
3992   {%
```

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

```
3993   \edef\@tempa{\number#1}%
3994   \edef\@tempb{\number#2}%
3995   \count0=\@tempa
3996   \count1=\@tempb\relax
```

Then we do the real job

```
3997   \fc@@check@nonzeros@inner
```

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

```
3998   \def\@tempd##1{\def\@tempa{\def#3{##1}}}%
```

```

3999 \expandafter\@tempd\expandafter{\@tempc}%
4000 \expandafter
4001 }\@tempa
4002 }

```

`\fc@@check@nonzeros@inner` 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.

```

```

4003 \def\fc@@check@nonzeros@inner{%
4004   \ifnum\count0<\fc@min@weight
4005     \count0=\fc@min@weight\relax
4006   \fi
4007   \ifnum\count1>\fc@max@weight\relax
4008     \count1=\fc@max@weight
4009   \fi
4010   \count2\count0 %
4011   \advance\count2 by 1 %
4012   \ifnum\count0>\count1 %
4013     \PackageError{fcnumparser}{Unexpected arguments}{Number in argument 2 of macro
4014       'fc@check@nonzeros' must be at least equal to number in argument 1}%
4015   \else
4016     \fc@@check@nonzeros@inner@loopbody
4017     \ifnum\@tempc>0 %
4018       \ifnum\@tempc<9 %
4019         \ifnum\count0>\count1 %
4020           \else
4021             \let\@tempd\@tempc
4022             \fc@@check@nonzeros@inner@loopbody
4023             \ifnum\@tempc=0 %
4024               \let\@tempc\@tempd
4025             \else
4026               \def\@tempc{9}%
4027             \fi
4028           \fi
4029         \fi
4030       \fi
4031     \fi
4032 }
4033 \def\fc@@check@nonzeros@inner@loopbody{%
4034   % \@tempc <- digit of weight \count0
4035   \expandafter\let\expandafter\@tempc\csname fc@digit@the\count0\endcsname

```

```

4036 \advance\count0 by 1 %
4037 \ifnum\@tempc=0 %
4038   \ifnum\count0>\count1 %
4039     \let\next\relax
4040   \else
4041     \let\next\fc@@check@nonzeros@inner@loopbody
4042   \fi
4043 \else
4044   \ifnum\count0>\count2 %
4045     \def\@tempc{9}%
4046   \fi
4047   \let\next\relax
4048 \fi
4049 \next
4050 }

```

`\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.

```

4051 \ifcsundef\fc@intpart@find@last\{}{%
4052 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
4053   '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@<u>`. Macro `\fc@intpart@find@last` takes one single argument which is a counter to set to the result.

```

4054 \def\fc@intpart@find@last#1{%
4055   {%

```

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  $\min\{u, w_{\min}\}$ :

```

4056   \count0=\fc@unit@weight\space
4057   \ifnum\count0<\fc@min@weight\space
4058     \count0=\fc@min@weight\space
4059   \fi

```

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

```

4060   \def\@templ{%
4061     \ifnum\csname fc@digit@the\count0\endcsname=0 %
4062       \advance\count0 by 1 %
4063       \ifnum\count0>\fc@max@weight\space
4064         \let\next\relax
4065       \fi
4066     \else
4067       \let\next\relax
4068     \fi
4069     \next
4070   }%
4071   \let\next\@templ
4072   \@templ

```

Now propagate result after closing bracket into counter #1.

```

4073     \toks0{#1}%
4074     \edef\@tempa{\the\toks0=\the\count0}%
4075     \expandafter
4076   }\@tempa\space
4077 }

```

`\fc@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

```

4078 \ifcsundef\fc@get@last@word\PackageError{fcnumparser}{Duplicate definition}{Redefinition
4079   of macro 'fc@get@last@word'}}%
4080 \def\fc@get@last@word#1#2#3{%
4081   {%

```

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`.

```

4082   \def\@tempa##1\fc@wcase##2\@nil\fc@end{%
4083     \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`.

```

4084     \toks1{##2\fc@wcase}%
4085   }%
4086   \@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`.

```

4087   \def\@tempa##1\fc@wcase##2\fc@end{%
4088     \toks2{##1}%
4089     \def\@tempb{##2}%
4090     \toks3{##2}%
4091   }%

```

`\@tempt` is just an aliases of `\toks0` to make its handling easier later on.

```

4092   \toksdef\@tempt0 %

```

Now the loop itself, this is done by terminal recursion with macro `\@templ`.

```

4093   \def\@templ{%
4094     \expandafter\@tempa\the\toks1 \fc@end
4095     \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.

```

4096       \let\next\relax
4097     \else

```

`\@tempb` is not empty, first we use

```

4098       \expandafter\expandafter\expandafter\@tempt
4099       \expandafter\expandafter\expandafter{%

```

```

4100         \expandafter\the\expandafter\@tempt
4101         \expandafter\fc@wcase\the\toks2}%
4102     \toks1\toks3 %
4103     \fi
4104     \next
4105 }%
4106 \let\next\@templ
4107 \@templ
4108 \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks2}}%
4109 \expandafter
4110 }\@tempa
4111 }

```

`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

```

4112 \ifcsundef{fc@get@last@letter}{\PackageError{fcnumparser}{Duplicate definition}{Redefinition
4113 of macro 'fc@get@last@letter'}}%
4114 \def\fc@get@last@letter#1#2#3{%
4115   {%

```

First copy input to local `\toks1`. What we are going to do is to bubble one by one letters from `\toks1` which initially contains the whole word, into `\toks0`. At the end of the macro `\toks0` will therefore contain the whole word but the last letter, and the last letter will be in `\toks1`.

```

4116   \toks1{#1}%
4117   \toks0{}%
4118   \toksdef\@tempt0 %

```

We define `\@tempa` in order to pop the first letter from the remaining of word.

```

4119   \def\@tempa##1##2\fc@nil{%
4120     \toks2{##1}%
4121     \toks3{##2}%
4122     \def\@tempb{##2}%
4123   }%

```

Now we define `\@templ` to do the loop by terminal recursion.

```

4124   \def\@templ{%
4125     \expandafter\@tempa\the\toks1 \fc@nil
4126     \ifx\@tempb\@empty

```

Stop loop, as `\toks1` has been detected to be one single letter.

```

4127     \let\next\relax
4128   \else

```

Here we append to `\toks0` the content of `\toks2`, i.e. the next letter.

```

4129     \expandafter\expandafter\expandafter\@tempt
4130     \expandafter\expandafter\expandafter{%
4131     \expandafter\the\expandafter\@tempt
4132     \the\toks2}%

```

And the remaining letters go to `\toks1` for the next iteration.

```

4133     \toks1\toks3 %

```

```

4134     \fi
4135     \next
4136   }%

```

Here run the loop.

```

4137     \let\next\@templ
4138     \next

```

Now propagate the results into macros #2 and #3 after closing brace.

```

4139     \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks1}}%
4140     \expandafter
4141   }\@tempa
4142 }%

```

### 10.3 fcprefix.sty

Pseudo-latin prefixes.

```

4143 \NeedsTeXFormat{LaTeX2e}
4144 \ProvidesPackage{fcprefix}[2012/09/28]
4145 \RequirePackage{ifthen}
4146 \RequirePackage{keyval}
4147 \RequirePackage{fncnumparser}

```

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’.

```

4148 \define@key{fcprefix}{use duode and unde}[below20]{%
4149   \ifthenelse{\equal{#1}{below20}}{%
4150     \def\fc@duodeandunde{2}%
4151   }{%
4152     \ifthenelse{\equal{#1}{never}}{%
4153       \def\fc@duodeandunde{0}%
4154     }{%
4155       \PackageError{fcprefix}{Unexpected option}{%
4156         Option ‘use duode and unde’ expects ‘below 20’ or ‘never’ }%
4157     }%
4158   }%
4159 }

```

Default is ‘below 20’ like in French.

```

4160 \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 $\langle xxx \rangle$  or duodec $\langle xxx \rangle$  for numerals.

```

4161 \define@key{fcprefix}{numeral u in duo}[false]{%
4162   \ifthenelse{\equal{#1}{false}}{%
4163     \let\fc@u@in@duo\@empty
4164   }{%
4165     \ifthenelse{\equal{#1}{true}}{%
4166       \def\fc@u@in@duo{u}%
4167     }{%
4168       \PackageError{fcprefix}{Unexpected option}{%

```

```

4169     Option ‘numeral u in duo’ expects ‘true’ or ‘false’ }%
4170   }%
4171 }%
4172 }

```

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.

```

4173 \define@key{fcprefix}{e accute}[false]{%
4174   \ifthenelse{\equal{#1}{false}}{%
4175     \let\fc@prefix@eaccute\@firstofone
4176   }{%
4177     \ifthenelse{\equal{#1}{true}}{%
4178       \let\fc@prefix@eaccute\'%
4179     }{%
4180       \PackageError{fcprefix}{Unexpected option}{%
4181         Option ‘e accute’ expects ‘true’ or ‘false’ }%
4182     }%
4183   }%
4184 }

```

Default is to set accute accent like in French.

```

4185 \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’

```

4186 \define@key{fcprefix}{power of millia}[prefix]{%
4187   \ifthenelse{\equal{#1}{prefix}}{%
4188     \let\fc@power@of@millia@init\@gobbletwo
4189     \let\fc@power@of@millia\fc@@prefix@millia
4190   }{%
4191     \ifthenelse{\equal{#1}{arabic}}{%
4192       \let\fc@power@of@millia@init\@gobbletwo
4193       \let\fc@power@of@millia\fc@@arabic@millia
4194     }{%
4195       \ifthenelse{\equal{#1}{recursive}}{%
4196         \let\fc@power@of@millia@init\fc@@recurse@millia@init
4197         \let\fc@power@of@millia\fc@@recurse@millia
4198       }{%
4199         \PackageError{fcprefix}{Unexpected option}{%
4200           Option ‘power of millia’ expects ‘recursive’, ‘arabic’, or ‘prefix’ }%
4201         }%
4202       }%
4203     }%
4204 }

```

Arguments as follows:

#1 output macro

#2 number with current weight *w*



```

4205 \def\fc@@recurse@millia#1#2{%
4206   \let\@temp#1%
4207   \edef#1{millia\@temp}%
4208 }

Arguments as follows — same interface as \fc@@recurse@millia:
#1  output macro
#2  number with current weight  $w$ 
4209 \def\fc@@recurse@millia@init#1#2{%
4210   {%

Save input argument current weight  $w$  into local macro \@tempb.
4211   \edef\@tempb{\number#2}%

Now main loop from 0 to  $w$ . Final value of \@tempa will be the result.
4212   \count0=0 %
4213   \let\@tempa\@empty
4214   \loop
4215     \ifnum\count0<\@tempb
4216       \advance\count0 by 1 %
4217       \expandafter\def
4218       \expandafter\@tempa\expandafter{\@tempa millia}%
4219   \repeat

Now propagate the expansion of \@tempa into #1 after closing brace.
4220   \edef\@tempb{\def\noexpand#1{\@tempa}}%
4221   \expandafter
4222   }\@tempb
4223 }

Arguments as follows — same interface as \fc@@recurse@millia:
#1  output macro
#2  number with current weight  $w$ 
4224 \def\fc@@arabic@millia#1#2{%
4225   \ifnum#2=0 %
4226     \let#1\@empty
4227   \else
4228     \edef#1{millia\^{\}\the#2}%
4229   \fi
4230 }

Arguments as follows — same interface as \fc@@recurse@millia:
#1  output macro
#2  number with current weight  $w$ 
4231 \def\fc@@prefix@millia#1#2{%
4232   \fc@@latin@numeral@pefix{#2}{#1}%
4233 }

Default value of option ‘power of millia’ is ‘prefix’:
4234 \let\fc@power@of@millia@init\@gobbletwo
4235 \let\fc@power@of@millia\fc@@prefix@millia

```

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 found 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.

```
4236 \ifcsundef{fc@latin@cardinal@prefix}{}%
4237 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro ‘fc@latin@cardinal@prefix
```

Arguments as follows:

- #1 input number to be formatted
- #2 outut macro name into which to place the formatted result

```
4238 \def\fc@latin@cardinal@prefix#1#2{%
4239   {%
```

First we put input argument into local macro @cs@tempa with full expansion.

```
4240   \edef\@tempa{\number#1}%
```

Now parse number from expanded input.

```
4241   \expandafter\fc@number@parser\expandafter{\@tempa}%
4242   \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.

```
4243   \let\@tempt\@empty
4244   \def\@tempu{t}%
```

\@tempm will hold the millia<sup>n÷3</sup>

```
4245   \let\@tempm\@empty
```

Loop by means of terminal recursion of hereinafter defined macro \@templ. We loop by group of 3 digits.

```
4246   \def\@templ{%
4247     \ifnum\count2>\fc@max@weight
4248     \let\next\relax
4249     \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.

```
4250     \fc@read@unit{\count3}{\count2}%
4251     \advance\count2 by 1 %
4252     \fc@read@unit{\count4}{\count2}%
4253     \advance\count2 by 1 %
4254     \fc@read@unit{\count5}{\count2}%
4255     \advance\count2 by 1 %
```

If the 3 considered digits  $d_2d_1d_0$  are not all zero, then set \@tempn to ‘t’ for the first time this event is met.

```
4256     \edef\@tempn{%
4257       \ifnum\count3=0\else 1\fi
4258       \ifnum\count4=0\else 1\fi
4259       \ifnum\count5=0\else 1\fi
4260     }%
4261     \ifx\@tempn\@empty\else
```

```

4262         \let\@tempt\@tempu
4263         \let\@tempu\@empty
4264     \fi

```

Now process the current group  $d_2d_1d_0$  of 3 digits.

```

4265         \let\@temp\@tempa
4266         \edef\@tempa{%

```

Here we process  $d_2$  held by `\count5`, that is to say hundreds.

```

4267         \ifcase\count5 %
4268         \or cen%
4269         \or ducen%
4270         \or trecen%
4271         \or quadringen%
4272         \or quingen%
4273         \or sescen%
4274         \or septigen%
4275         \or octingen%
4276         \or nongen%
4277     \fi

```

Here we process  $d_1d_0$  held by `\count4` & `\count3`, that is to say tens and units.

```

4278         \ifnum\count4=0 %
4279         % x0(0..9)
4280         \ifnum\count2=3 %
4281         % Absolute weight zero
4282         \ifcase\count3 \@tempt
4283         \or m%
4284         \or b%
4285         \or tr%
4286         \or quadr%
4287         \or quin\@tempt
4288         \or sex\@tempt
4289         \or sep\@tempt
4290         \or oc\@tempt
4291         \or non%
4292         \fi
4293     \else

```

Here the weight of `\count3` is  $3 \times n$ , with  $n > 0$ , i.e. this is followed by a millia<sup>n</sup>.

```

4294         \ifcase\count3 %
4295         \or \ifnum\count2>\fc@max@weight\else un\fi
4296         \or d\fc@u@in@duo o%
4297         \or tre%
4298         \or quattuor%
4299         \or quin%
4300         \or sex%
4301         \or septen%
4302         \or octo%
4303         \or novem%
4304     \fi

```

```

4305         \fi
4306     \else
4307         % x(10..99)
4308         \ifcase\count3 %
4309         \or un%
4310         \or d\fc@u@in@duo o%
4311         \or tre%
4312         \or quattuor%
4313         \or quin%
4314         \or sex%
4315         \or septen%
4316         \or octo%
4317         \or novem%
4318         \fi
4319         \ifcase\count4 %
4320         \or dec%
4321         \or vigin\@tempt
4322         \or trigin\@tempt
4323         \or quadragin\@tempt
4324         \or quinquagin\@tempt
4325         \or sexagin\@tempt
4326         \or septuagin\@tempt
4327         \or octogin\@tempt
4328         \or nonagin\@tempt
4329         \fi
4330     \fi

```

Insert the millia<sup>(n+3)</sup> only if  $d_2 d_1 d_0 \neq 0$ , i.e. if one of \count3 \count4 or \count5 is non zero.

```
4331     \@tempm
```

And append previous version of \@tempa.

```
4332     \@temppp
4333     }%
```

“Concatenate” millia to \@tempm, so that \@tempm will expand to millia<sup>(n+3)+1</sup> at the next iteration. Actually whether this is a concatenation or some millia prefixing depends of option ‘power of millia’.

```

4334     \fc@power@of@millia\@tempm{\count2}%
4335     \fi
4336     \next
4337     }%
4338     \let\@tempa\@empty
4339     \let\@next\@templ
4340     \@templ

```

Propagate expansion of \@tempa into #2 after closing bracket.

```

4341     \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
4342     \expandafter\@tempb\expandafter{\@tempa}%
4343     \expandafter
4344     }\@tempa

```

4345 }

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.

```
4346 \ifcsundef{fc@latin@numeral@pefix}{}{%
4347   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
4348     ‘fc@latin@numeral@pefix’}}
```

Arguments as follows:

- #1 input number to be formatted,
- #2 outut macro name into which to place the result

```
4349 \def\fc@latin@numeral@pefix#1#2{%
4350   {%
4351     \edef\@tempa{\number#1}%
4352     \def\fc@unit@weight{0}%
4353     \expandafter\fc@number@parser\expandafter{\@tempa}%
4354     \count2=0 %
```

Macro \@tempm will hold the millies  $^{n\div 3}$ .

```
4355   \let\@tempm\@empty
```

Loop over digits. This is done by defining macro \@templ for terminal recursion.

```
4356   \def\@templ{%
4357     \ifnum\count2>\fc@max@weight
4358       \let\next\relax
4359     \else
```

Loop body. Three consecutive digits  $d_2d_1d_0$  are read into counters \count3, \count4, and \count5.

```
4360       \fc@read@unit{\count3}{\count2}%
4361       \advance\count2 by 1 %
4362       \fc@read@unit{\count4}{\count2}%
4363       \advance\count2 by 1 %
4364       \fc@read@unit{\count5}{\count2}%
4365       \advance\count2 by 1 %
```

Check the use of duodevicies instead of octodevicies.

```
4366       \let\@tempn\@secondoftwo
4367       \ifnum\count3>7 %
4368         \ifnum\count4<\fc@duodeandunde
4369           \ifnum\count4>0 %
4370             \let\@tempn\@firstoftwo
4371           \fi
4372         \fi
4373       \fi
4374       \@tempn
4375       {% use duodevicies for eighteen
4376         \advance\count4 by 1 %
4377         \let\@temps\@secondoftwo
4378       }{% do not use duodevicies for eighteen
```

```

4379         \let\@temps\@firstoftwo
4380     }%
4381     \let\@tempp\@tempa
4382     \edef\@tempa{%
4383         % hundreds
4384         \ifcase\count5 %
4385         \expandafter\@gobble
4386         \or c%
4387         \or duc%
4388         \or trec%
4389         \or quadring%
4390         \or quing%
4391         \or sesc%
4392         \or septing%
4393         \or octing%
4394         \or nong%
4395         \fi
4396         {enties}%
4397         \ifnum\count4=0 %

```

Here  $d_2d_1d_0$  is such that  $d_1 = 0$ .

```

4398         \ifcase\count3 %
4399         \or
4400         \ifnum\count2=3 %
4401             s\fc@prefix@eaccute emel%
4402         \else
4403             \ifnum\count2>\fc@max@weight\else un\fi
4404         \fi
4405         \or bis%
4406         \or ter%
4407         \or quater%
4408         \or quinquies%
4409         \or sexies%
4410         \or septies%
4411         \or octies%
4412         \or novies%
4413         \fi
4414     \else

```

Here  $d_2d_1d_0$  is such that  $d_1 \geq 1$ .

```

4415         \ifcase\count3 %
4416         \or un%
4417         \or d\fc@u@in@duo o%
4418         \or ter%
4419         \or quater%
4420         \or quin%
4421         \or sex%
4422         \or septen%
4423         \or \@temps{octo}{duod\fc@prefix@eaccute e}% x8 = two before next (x+1)0
4424         \or \@temps{novem}{und\fc@prefix@eaccute e}% x9 = one before next (x+1)0

```

```

4425         \fi
4426         \ifcase\count4 %
4427         % can't get here
4428         \or d\fc@prefix@eacute ec%
4429         \or vic%
4430         \or tric%
4431         \or quadrag%
4432         \or quinquag%
4433         \or sexag%
4434         \or septuag%
4435         \or octog%
4436         \or nonag%
4437         \fi
4438         ies%
4439     \fi
4440     % Insert the millies^(n/3) only if one of \count3 \count4 \count5 is non zero
4441     \@tempm
4442     % add up previous version of \@tempa
4443     \@tempm
4444 }%

```

Concatenate millies to \@tempm so that it is equal to  $\text{millies}^{n+3}$  at the next iteration. Here we just have plain concatenation, contrary to cardinal for which a prefix can be used instead.

```

4445     \let\@tempm\@tempm
4446     \edef\@tempm{\millies\@tempm}%
4447     \fi
4448     \next
4449 }%
4450 \let\@tempa\@empty
4451 \let\next\@templ
4452 \@templ

```

Now propagate expansion of tempa into #2 after closing bracket.

```

4453 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
4454 \expandafter\@tempb\expandafter{\@tempa}%
4455 \expandafter
4456 }\@tempa
4457 }

```

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  $\langle oarg \rangle$  is absent, then it is by convention set empty,
- $\langle some\ macro \rangle$  is supposed to have two mandatory arguments of which  $\langle oarg \rangle$  is passed to the first, and  $\langle marg \rangle$  is passed to the second, and
- $\langle some\ macro \rangle$  is called within a group.

```

4458 \def\fc@call@opt@arg@second#1#2{%
4459   \def\@tempb{%
4460     \ifx[\@tempa
4461       \def\@tempc[####1]{%
4462         {#1{####1}{#2}}%
4463       }%
4464     \else
4465       \def\@tempc{{#1}{#2}}}%
4466     \fi
4467   \@tempc
4468 }%
4469 \futurelet\@tempa
4470 \@tempb
4471 }

4472 \def\fc@call@opt@arg@first#1{%
4473   \def\@tempb{%
4474     \ifx[\@tempa
4475       \def\@tempc[####1]####2{{#1{####1}{####2}}}%
4476     \else
4477       \def\@tempc####1{{#1}{####1}}}%
4478     \fi
4479   \@tempc
4480 }%
4481 \futurelet\@tempa
4482 \@tempb
4483 }
4484
4485 \let\fc@call\fc@call@opt@arg@first

```

#### User API.

$\text{latinnumeralstringnumMacro}$   $\backslash\text{latinnumeralstringnum}$ . Arguments as follows:

- #1 local options
- #2 input number

```

4486 \newcommand*{\@latinnumeralstringnum}[2]{%
4487   \setkeys{fcprefix}{#1}%
4488   \fc@latin@numeral@pefix{#2}\@tempa
4489   \@tempa
4490 }

```

Arguments as follows:

- #1 local options
- #2 input counter



```

4491 \newcommand*{\@latinnumeralstring}[2]{%
4492   \setkeys{fcprefix}{#1}%
4493   \expandafter\let\expandafter
4494     \@tempa\expandafter\csname c@#2\endcsname
4495   \expandafter\fc@latin@numeral@pefix\expandafter{\the\@tempa}\@tempa
4496   \@tempa
4497 }

4498 \newcommand*{\latinnumeralstring}{%
4499   \fc@call\@latinnumeralstring
4500 }

4501 \newcommand*{\latinnumeralstringnum}{%
4502   \fc@call\@latinnumeralstringnum
4503 }

```

## 10.4 fmtcount.sty

This section deals with the code for `fmtcount.sty`

```

4504 \NeedsTeXFormat{LaTeX2e}
4505 \ProvidesPackage{fmtcount}[2020/01/30 v3.07]
4506 \RequirePackage{ifthen}

4507 \RequirePackage{xkeyval}
4508 \RequirePackage{etoolbox}
4509 \RequirePackage{fcprefix}

```

Need to use `\new@ifnextchar` instead of `\@ifnextchar` in commands that have a final optional argument (such as `\gls`) so require `amsgen`.

```
4510 \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
```

```
4511 \providecommand*\fc@orddef@ult}[1]{\fc@textsuperscript{#1}}
```

```
c@ord@multiling
```

```
4512 \providecommand*\fc@ord@multiling}[1]{%
4513   \ifcsundef{fc@\languagenome @alias@of}{%
```

Not a supported language, just use the default setting:

```
4514   \fc@orddef@ult{#1}}{%
4515   \expandafter\let\expandafter\@tempa\csname fc@\languagenome @alias@of\endcsname
4516   \ifcsundef{fc@ord@\@tempa}{%
```

Not language specific setting, just use the default setting:

```
4517   \fc@orddef@ult{#1}}{%
```

Language with specific setting, use that setting:

```
4518 \csname fc@ord@\@tempa\endcsname{#1}}}}
```

`\padzeroes` `\padzeroes[n]`

Specifies how many digits should be displayed for commands such as `\decimal` and `\binary`.

```
4519 \newcount\c@padzeroesN
4520 \c@padzeroesN=1\relax
4521 \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.

```
4522 \newcount\fc@tmpcatcode
4523 \def\fc@languages{}%
4524 \def\fc@mainlang{}%
4525 \newcommand*\FCloadlang}[1]{%
4526   \@FC@iflangloaded{#1}{}%
4527   {%
4528     \fc@tmpcatcode=\catcode'\@ \relax
4529     \catcode '@ 11\relax
4530     \InputIfFileExists{fc-#1.def}%
4531     {%
4532       \ifdefempty{\fc@languages}%
4533       {%
4534         \gdef\fc@languages{#1}%
4535       }%
4536       {%
4537         \gappto\fc@languages{,#1}%
4538       }%
4539       \gdef\fc@mainlang{#1}%
4540     }%
4541   }%
4542   \catcode '@ \fc@tmpcatcode\relax
4543 }%
4544 }
```

`@FC@iflangloaded` `@FC@iflangloaded{language}{true}{false}`

If `fmtcount` language definition file `fc-language.def` has been loaded, do *true* otherwise do *false*

```
4545 \newcommand{@FC@iflangloaded}[3]{%
4546   \ifcsundef{ver@fc-#1.def}{#3}{#2}%
4547 }
```

ProvidesFCLanguage Declare fmtcount language definition file. Adapted from \ProvidesFile.

```
4548 \newcommand*\ProvidesFCLanguage}[1]{%
4549   \ProvidesFile{fc-#1.def}%
4550 }
```

We need that flag to remember that a language has been loaded via package option, so that in the end we can set fmtcount in multiling

```
4551 \newif\iffmtcount@language@option
4552 \fmtcount@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 fmtcount 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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>.

```
4553 \newcommand*\fc@supported@language@list{%
4554   english,%
4555   UKenglish,%
4556   brazilian,%
4557   british,%
4558   USenglish,%
4559   american,%
4560   spanish,%
4561   portuges,%
4562   portuguese,%
4563   french,%
4564   frenchb,%
4565   francais,%
4566   german,%
4567   germanb,%
4568   ngerman,%
4569   ngermanb,%
4570   italian}
```

iterate@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 :

```

4571 \newcommand*\fc@iterate@on@languages[1]{%
4572   \ifx\fc@supported@language@list\@empty
      That case should never happen !
4573   \PackageError{fmtcount}{Macro ‘\protect\fc@iterate@on@languages’ is empty}{You should never
4574     Something is broken within \texttt{fmtcount}, please report the issue on
4575     \texttt{https://github.com/search?q=fmtcount\&ref=cmdform\&type=Issues}}%
4576   \else
4577     \let\fc@iterate@on@languages@body#1
4578     \expandafter\fc@iterate@on@languages\fc@supported@language@list,\@nil,%
4579   \fi
4580 }
4581 \def\fc@iterate@on@languages#1,{%
4582   {%
4583     \def\@tempa{#1}%
4584     \ifx\@tempa\@nnil
4585       \let\@tempa\@empty
4586     \else
4587       \def\@tempa{%
4588         \fc@iterate@on@languages@body{#1}%
4589         \fc@iterate@on@languages
4590       }%
4591     \fi
4592     \expandafter
4593   }\@tempa
4594 }%

```

orpolyglossialdf

```
\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`.

```

4595 \newcommand*\fc@loadifbabelldf[1]{\ifcsundef{ver@#1.ldf}{}\FCloadlang{#1}}
4596 \newcommand*\fc@loadifbabelorpolyglossialdf[1]{
4597   \ifpackageloaded{polyglossia}{%
4598     \def\fc@loadifbabelorpolyglossialdf#1{\IfFileExists{gloss-#1.ldf}{\ifcsundef{#1@loaded}}{\F
4599     \fc@loadifbabelldf{#1}%
4600   }%
4601 }{\ifpackageloaded{babel}{%
4602   \let\fc@loadifbabelorpolyglossialdf\fc@loadifbabelldf
4603 }{}}

```

Load appropriate language definition files:

```
4604 \fc@iterate@on@languages\fc@loadifbabelorpolyglossialdf
```

By default all languages are unique — i.e. aliases not yet defined.

```
4605 \def\fc@iterate@on@languages@body#1{%
```

```
4606   \expandafter\def\csname fc@#1@alias@of\endcsname{#1}}
```

```
4607 \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>}`

```
4608 \def\@tempa#1#2{%
```

```
4609   \expandafter\def\csname fc@#1@alias@of\endcsname{#2}%
```

```
4610 }%
```

```
4611 \@tempa{frenchb}{french}
```

```
4612 \@tempa{français}{french}
```

```
4613 \@tempa{germanb}{german}
```

```
4614 \@tempa{ngermanb}{german}
```

```
4615 \@tempa{ngerman}{german}
```

```
4616 \@tempa{british}{english}
```

```
4617 \@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.

```
4618 \def\fc@iterate@on@languages@body#1{%
```

```
4619   \define@key{fmtcount}{#1}[]{%
```

```
4620     \FC@iflangloaded{#1}%
```

```
4621     {%
```

```
4622       \setkeys{fc\csname fc@#1@alias@of\endcsname}{##1}%
```

```
4623     }{%
```

```
4624       \PackageError{fmtcount}%
```

```
4625         {Language ‘#1’ not defined}%
```

```
4626         {You need to load \ifxetex polyglossia\else babel\fi space before loading fmtcount}%
```

```
4627     }%
```

```
4628 }%
```

```
4629 \ifthenelse{\equal{\csname fc@#1@alias@of\endcsname}{#1}}{%
```

```
4630   \define@key{fc\csname fc@#1@alias@of\endcsname}{fmtord}{%
```

```
4631     \ifthenelse{\equal{##1}{raise}\or\equal{##1}{level}}{%
```

```
4632       \expandafter\let\expandafter\@tempa\csname fc@set@ord@as@##1\endcsname
```

```
4633       \expandafter\@tempa\csname fc@ord@#1\endcsname
```

```
4634     }{%
```

```
4635       \ifthenelse{\equal{##1}{undefine}}{%
```

```
4636         \expandafter\let\csname fc@ord@#1\endcsname\undefined
```

```
4637       }{%
```

```
4638         \PackageError{fmtcount}%
```

```
4639           {Invalid value ‘##1’ to fmtord key}%
```

```
4640           {Option ‘fmtord’ can only take the values ‘level’, ‘raise’
```

```
4641             or ‘undefine’}%
```

```
4642         }}%
```

```
4643     }%
```

```
4644 }{%
```

When the language #1 is an alias, do the same as the language of which it is an alias:

```

4645 \expandafter\let\expandafter\@tempa\csname KV@\csname fc@#1@alias@of\endcsname @fmtord\endc
4646 \expandafter\let\csname KV@#1@fmtord\endcsname\@tempa
4647 }%
4648 }
4649 \expandafter\@fc@iterate@on@languages\fc@supported@language@list,\@nil,%

```

**fmtord** Key to determine how to display the ordinal

```

4650 \def\fc@set@ord@as@level#1{%
4651 \def#1##1{##1}%
4652 }
4653 \def\fc@set@ord@as@raise#1{%
4654 \let#1\fc@textsuperscript
4655 }
4656 \define@key{fmtcount}{fmtord}{%
4657 \ifthenelse{\equal{#1}{level}}
4658 \or\equal{#1}{raise}}%
4659 {%
4660 \csname fc@set@ord@as@#1\endcsname\fc@orddef@ult
4661 \def\fmtcount@fmtord{#1}%
4662 }%
4663 {%
4664 \PackageError{fmtcount}%
4665 {Invalid value ‘#1’ to fmtord key}%
4666 {Option ‘fmtord’ can only take the values ‘level’ or ‘raise’}%
4667 }%
4668 }

```

**\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.)

```

4669 \newif\iffmtord@abbrv

4670 \fmtord@abbrvtrue
4671 \define@key{fmtcount}{abbrv}[true]{%
4672 \ifthenelse{\equal{#1}{true}\or\equal{#1}{false}}%
4673 {%
4674 \csname fmtord@abbrv#1\endcsname
4675 }%
4676 {%
4677 \PackageError{fmtcount}%
4678 {Invalid value ‘#1’ to fmtord key}%
4679 {Option ‘abbrv’ can only take the values ‘true’ or
4680 ‘false’}%
4681 }%
4682 }

```

**prefix**

```

4683 \define@key{fmtcount}{prefix}[scale=long]{%

```

```

4684 \RequirePackage{fmtprefix}%
4685 \fmtprefixsetoption{#1}%
4686 }

```

countsetoptions Define command to set options.

```

4687 \def\fmtcountsetoptions{%
4688 \def\fmtcount@fmtord{}%
4689 \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.

```

4690 \InputIfFileExists{fmtcount.cfg}%
4691 {%
4692 \PackageInfo{fmtcount}{Using configuration file fmtcount.cfg}%
4693 }%
4694 {%
4695 }

```

ption@lang@list

```

4696 \newcommand*{\fmtcount@loaded@by@option@lang@list}{}

```

\metallanguage Option *<language>* causes language *<language>* to be registered for loading.

```

4697 \newcommand*\@fc@declare@language@option[1]{%
4698 \DeclareOption{#1}{%
4699 \ifx\fmtcount@loaded@by@option@lang@list\@empty
4700 \def\fmtcount@loaded@by@option@lang@list{#1}%
4701 \else
4702 \edef\fmtcount@loaded@by@option@lang@list{\fmtcount@loaded@by@option@lang@list,#1}%
4703 \fi
4704 }}%
4705 \fc@iterate@on@languages\@fc@declare@language@option

```

level

```

4706 \DeclareOption{level}{\def\fmtcount@fmtord{level}%
4707 \def\fc@orddef@ult#1{#1}}

```

raise

```

4708 \DeclareOption{raise}{\def\fmtcount@fmtord{raise}%
4709 \def\fc@orddef@ult#1{\fc@textsuperscript{#1}}}

```

Process package options

```

4710 \ProcessOptions\relax

```

Now we do the loading of all languages that have been set by option to be loaded.

```

4711 \ifx\fmtcount@loaded@by@option@lang@list\@empty\else
4712 \def\fc@iterate@on@languages@body#1{%
4713 \@FC@iflangloaded{#1}{}%
4714 \fmtcount@language@optiontrue

```

```

4715     \FCloadlang{#1}%
4716     }}
4717 \expandafter\@fc@iterate@on@languages\fmtcount@loaded@by@option@lang@list,\@nil,%
4718 \fi

```

`\@FCmodulo`    `\@FCmodulo{<count reg>}{<n>}`

Sets the count register to be its value modulo  $\langle n \rangle$ . This is used for the date, time, ordinal and numberstring commands. (The `fmtcount` package was originally part of the `datetime` package.)

```

4719 \newcount\@DT@modctr
4720 \newcommand*{\@FCmodulo}[2]{%
4721   \@DT@modctr=#1\relax
4722   \divide \@DT@modctr by #2\relax
4723   \multiply \@DT@modctr by #2\relax
4724   \advance #1 by -\@DT@modctr
4725 }

```

The following registers are needed by `\@ordinal` etc

```

4726 \newcount\@ordinalctr
4727 \newcount\@orgargctr
4728 \newcount\@strctr
4729 \newcount\@tmpstrctr

```

Define commands that display numbers in different bases. Define counters and conditionals needed.

```

4730 \newif\if@DT@padzeroes
4731 \newcount\@DT@loopN
4732 \newcount\@DT@X

```

`\binarynum`    Converts a decimal number to binary, and display.

```

4733 \newrobustcmd*{\@binary}[1]{%
4734   \@DT@padzeroestrue
4735   \@DT@loopN=17\relax
4736   \@strctr=\@DT@loopN
4737   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
4738   \@strctr=65536\relax
4739   \@DT@X=#1\relax
4740   \loop
4741     \@DT@modctr=\@DT@X
4742     \divide\@DT@modctr by \@strctr
4743     \ifthenelse{\boolean{@DT@padzeroes}
4744       \and \(\@DT@modctr=0\)}
4745       \and \(\@DT@loopN>\c@padzeroesN\)}%
4746     {}%
4747     {\the\@DT@modctr}%
4748     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
4749     \multiply\@DT@modctr by \@strctr

```



```

4750 \advance\DT@X by -\DT@modctr
4751 \divide\@strctr by \tw@
4752 \advance\DT@loopN by \m@ne
4753 \ifnum\@strctr>\@ne
4754 \repeat
4755 \the\DT@X
4756 }
4757
4758 \let\binarynum=\@binary

```

`\octalnum` Converts a decimal number to octal, and displays.

```

4759 \newrobustcmd*{\@octal}[1]{%
4760 \DT@X=#1\relax
4761 \ifnum\DT@X>32768
4762 \PackageError{fmtcount}%
4763 {Value of counter too large for \protect\@octal}
4764 {Maximum value 32768}
4765 \else
4766 \DT@padzeroes\true
4767 \DT@loopN=6\relax
4768 \@strctr=\DT@loopN
4769 \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
4770 \@strctr=32768\relax
4771 \loop
4772 \DT@modctr=\DT@X
4773 \divide\DT@modctr by \@strctr
4774 \ifthenelse{\boolean{DT@padzeroes}
4775 \and \(\DT@modctr=0\}
4776 \and \(\DT@loopN>\c@padzeroesN\)}%
4777 {\the\DT@modctr}%
4778 \ifnum\DT@modctr=0\else\DT@padzeroesfalse\fi
4779 \multiply\DT@modctr by \@strctr
4780 \advance\DT@X by -\DT@modctr
4781 \divide\@strctr by \@viiipt
4782 \advance\DT@loopN by \m@ne
4783 \ifnum\@strctr>\@ne
4784 \repeat
4785 \the\DT@X
4786 \fi
4787 }
4788 \let\octalnum=\@octal

```

`\@@hexadecimal` Converts number from 0 to 15 into lowercase hexadecimal notation.

```

4789 \newcommand*{\@@hexadecimal}[1]{%
4790 \ifcase#1\or1\or2\or3\or4\or5\or
4791 6\or7\or8\or9\or a\or b\or c\or d\or e\or f\fi
4792 }

```

`\hexadecimalnum` Converts a decimal number to a lowercase hexadecimal number, and displays it.

```
4793 \newrobustcmd*{\hexadecimalnum}{\@hexadecimalengine\@hexadecimal}
```

`\@Hexadecimal` Converts number from 0 to 15 into uppercase hexadecimal notation.

```
4794 \newcommand*{\@Hexadecimal}[1]{%
4795   \ifcase#1\or1\or2\or3\or4\or5\or6\or
4796   7\or8\or9\or A\or B\or C\or D\or E\or F\fi
4797 }
```

`\HEXADecimalnum` Uppercase hexadecimal

```
4798 \newrobustcmd*{\HEXADecimalnum}{\@hexadecimalengine\@Hexadecimal}
4799 \newcommand*{\@hexadecimalengine}[2]{%
4800   \@DT@padzeroestrue
4801   \@DT@loopN=\@vpt
4802   \@strctr=\@DT@loopN
4803   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
4804   \@strctr=65536\relax
4805   \@DT@X=#2\relax
4806   \loop
4807     \@DT@modctr=\@DT@X
4808     \divide\@DT@modctr by \@strctr
4809     \ifthenelse{\boolean{\@DT@padzeroes}
4810       \and \!(\@DT@modctr=0)
4811       \and \!(\@DT@loopN>\c@padzeroesN)}
4812     {#\@DT@modctr}%
4813     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
4814     \multiply\@DT@modctr by \@strctr
4815     \advance\@DT@X by -\@DT@modctr
4816     \divide\@strctr by 16\relax
4817     \advance\@DT@loopN by \m@ne
4818     \ifnum\@strctr>\@ne
4819     \repeat
4820     #1\@DT@X
4821 }
4822 \def\Hexadecimalnum{%
4823   \PackageWarning{fmtcount}{\string\Hexadecimalnum\space is deprecated, use \string\HEXADecimal
4824     instead. The \string\Hexadecimalnum\space control sequence name is confusing as it can misl
4825     that only the 1st letter is upper-cased.}%
4826   \HEXADecimalnum}
```

`\aaalphnum` Lowercase alphabetical representation (a... z aa... zz)

```
4827 \newrobustcmd*{\aaalph}{\fc\aaalph\@alph}
4828 \newcommand*\fc\aaalph[2]{%
4829   \@DT@loopN=#2\relax
4830   \@DT@X\@DT@loopN
4831   \advance\@DT@loopN by \m@ne
4832   \divide\@DT@loopN by 26\relax
4833   \@DT@modctr=\@DT@loopN
4834   \multiply\@DT@modctr by 26\relax
4835   \advance\@DT@X by \m@ne
```

```

4836 \advance\DT@X by -\DT@modctr
4837 \advance\DT@loopN by \@ne
4838 \advance\DT@X by \@ne
4839 \edef\@tempa{#1\DT@X}%
4840 \loop
4841   \@tempa
4842   \advance\DT@loopN by \m@ne
4843   \ifnum\DT@loopN>0
4844   \repeat
4845 }
4846
4847 \let\aaalphnum=\@aaalph

```

**\AAAalphnum** Uppercase alphabetical representation (a ... z aa ... zz)

```

4848 \newrobustcmd*{\@AAAAlph}{\fc@aaalph\@Alph}%
4849
4850 \let\AAAalphnum=\@AAAAlph

```

**\abalphnum** Lowercase alphabetical representation

```

4851 \newrobustcmd*{\@abalph}{\fc@abalph\@alph}%
4852 \newcommand*\fc@abalph[2]{%
4853   \DT@X=#2\relax
4854   \ifnum\DT@X>17576\relax
4855     \ifx#1\@alph\def\@tempa{\@abalph}%
4856     \else\def\@tempa{\@ABAlph}\fi
4857     \PackageError{fmtcount}%
4858     {Value of counter too large for \expandafter\protect\@tempa}%
4859     {Maximum value 17576}%
4860   \else
4861     \@DT@padzeroestrue
4862     \@strctr=17576\relax
4863     \advance\DT@X by \m@ne
4864     \loop
4865       \@DT@modctr=\DT@X
4866       \divide\DT@modctr by \@strctr
4867       \ifthenelse{\boolean{\@DT@padzeroes}
4868         \and \(\@DT@modctr=1\)}%
4869       {\#1\@DT@modctr}%
4870       \ifnum\@DT@modctr=\@ne\else\@DT@padzeroesfalse\fi
4871       \multiply\@DT@modctr by \@strctr
4872       \advance\DT@X by -\@DT@modctr
4873       \divide\@strctr by 26\relax
4874     \ifnum\@strctr>\@ne
4875     \repeat
4876     \advance\DT@X by \@ne
4877     #1\@DT@X
4878   \fi
4879 }
4880

```

```
4881 \let\abalphnum=\@abalph
```

`\ABAlphnum` Uppercase alphabetical representation

```
4882 \newrobustcmd*{\@ABAlph}{\fc@abalph\@Alph}%
```

```
4883 \let\ABAlphnum=\@ABAlph
```

`\@fmtc@count` Recursive command to count number of characters in argument. `\@strctr` should be set to zero before calling it.

```
4884 \def\@fmtc@count#1#2\relax{%
```

```
4885   \if\relax#1%
```

```
4886   \else
```

```
4887     \advance\@strctr by 1\relax
```

```
4888     \@fmtc@count#2\relax
```

```
4889   \fi
```

```
4890 }
```

`\@decimal` Format number as a decimal, possibly padded with zeroes in front.

```
4891 \newrobustcmd*{\@decimal}[1]{%
```

```
4892   \@strctr=0\relax
```

```
4893   \expandafter\@fmtc@count\number#1\relax
```

```
4894   \@DT@loopN=\c@padzeroesN
```

```
4895   \advance\@DT@loopN by -\@strctr
```

```
4896   \ifnum\@DT@loopN>0\relax
```

```
4897     \@strctr=0\relax
```

```
4898     \whiledo{\@strctr < \@DT@loopN}{0\advance\@strctr by 1\relax}%
```

```
4899   \fi
```

```
4900   \number#1\relax
```

```
4901 }
```

```
4902
```

```
4903 \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.

```
4904 \newcommand{\FCordinal}[1]{%
```

```
4905   \ordinalnum{%
```

```
4906     \the\value{#1}}%
```

```
4907 }
```

`\ordinal` If `\ordinal` isn't defined make `\ordinal` a synonym for `\FCordinal` to maintain compatibility with previous versions.

```
4908 \ifcsundef{ordinal}
4909 {\let\ordinal\FCordinal}%
4910 {%
4911   \PackageWarning{fmtcount}%
4912   {\protect\ordinal \space already defined use
4913    \protect\FCordinal \space instead.}
4914 }
```

`\ordinalnum` Display ordinal where value is given as a number or count register instead of a counter:

```
4915 \newrobustcmd*{\ordinalnum}[1]{%
4916   \new@ifnextchar[%
4917     {\@ordinalnum{#1}}%
4918     {\@ordinalnum{#1}[m]}%
4919 }
```

`\@ordinalnum` Display ordinal according to gender (neuter added in v1.1, `\xspace` added in v1.2, and removed in v1.3<sup>7</sup>):

```
4920 \def\@ordinalnum#1[#2]{%
4921   {%
4922     \ifthenelse{\equal{#2}{f}}%
4923     {%
4924       \protect\@ordinalF{#1}{\@fc@ordstr}%
4925     }%
4926     {%
4927       \ifthenelse{\equal{#2}{n}}%
4928       {%
4929         \protect\@ordinalN{#1}{\@fc@ordstr}%
4930       }%
4931       {%
4932         \ifthenelse{\equal{#2}{m}}%
4933         {}%
4934         {%
4935           \PackageError{fmtcount}%
4936             {Invalid gender option ‘#2’}%
4937             {Available options are m, f or n}%
4938         }%
4939         \protect\@ordinalM{#1}{\@fc@ordstr}%
4940       }%
4941     }%
4942     \@fc@ordstr
4943   }%
4944 }
```

`\storeordinal` Store the ordinal (first argument is identifying name, second argument is a counter.)

```
4945 \newcommand*{\storeordinal}[2]{%
```

<sup>7</sup>I couldn't get it to work consistently both with and without the optional argument

```

4946 {%
4947   \toks0{\storeordinalnum{#1}}%
4948   \expandafter
4949   }\the\toks0\expandafter{%
4950   \the\value{#2}}%
4951 }

```

`\storeordinalnum` Store ordinal (first argument is identifying name, second argument is a number or count register.)

```

4952 \newrobustcmd*{\storeordinalnum}[2]{%
4953   \ifnextchar[%
4954   {\@storeordinalnum{#1}{#2}}%
4955   {\@storeordinalnum{#1}{#2}[m]}}%
4956 }

```

`\storeordinalnum` Store ordinal according to gender:

```

4957 \def\@storeordinalnum#1#2[#3]{%
4958   \ifthenelse{\equal{#3}{f}}%
4959   {%
4960     \protect\@ordinalF{#2}{\@fc@ord}
4961   }%
4962   {%
4963     \ifthenelse{\equal{#3}{n}}%
4964     {%
4965       \protect\@ordinalN{#2}{\@fc@ord}%
4966     }%
4967     {%
4968       \ifthenelse{\equal{#3}{m}}%
4969       {}%
4970     }%
4971     \PackageError{fmtcount}%
4972     {Invalid gender option ‘#3’}%
4973     {Available options are m or f}%
4974   }%
4975   \protect\@ordinalM{#2}{\@fc@ord}%
4976   }%
4977 }%
4978 \expandafter\let\csname @fcs@#1\endcsname\@fc@ord
4979 }

```

`\FMCuse` Get stored information:

```

4980 \newcommand*\FMCuse}[1]{\csname @fcs@#1\endcsname}

```

`\ordinalstring` Display ordinal as a string (argument is a counter)

```

4981 \newcommand*\ordinalstring}[1]{%
4982   \ordinalstringnum{\expandafter\expandafter\expandafter
4983     \the\value{#1}}%
4984 }

```

`\ordinalstringnum` Display ordinal as a string (argument is a count register or number.)

```
4985 \newrobustcmd*{\ordinalstringnum}[1]{%
4986   \new@ifnextchar[%
4987     {\@ordinal@string{#1}}%
4988     {\@ordinal@string{#1}[m]}%
4989 }
```

`\@ordinal@string` Display ordinal as a string according to gender.

```
4990 \def\@ordinal@string#1[#2]{%
4991   {%
4992     \ifthenelse{\equal{#2}{f}}%
4993     {%
4994       \protect\@ordinalstringF{#1}{\@fc@ordstr}%
4995     }%
4996     {%
4997       \ifthenelse{\equal{#2}{n}}%
4998       {%
4999         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
5000       }%
5001       {%
5002         \ifthenelse{\equal{#2}{m}}%
5003         {}%
5004         {%
5005           \PackageError{fmtcount}%
5006             {Invalid gender option ‘#2’ to \protect\ordinalstring}%
5007             {Available options are m, f or n}%
5008         }%
5009         \protect\@ordinalstringM{#1}{\@fc@ordstr}%
5010       }%
5011     }%
5012     \@fc@ordstr
5013   }%
5014 }
```

`\storeordinalstring` Store textual representation of number. First argument is identifying name, second argument is the counter set to the required number.

```
5015 \newcommand*{\storeordinalstring}[2]{%
5016   {%
5017     \toks0{\storeordinalstringnum{#1}}%
5018     \expandafter
5019   }\the\toks0\expandafter{\the\value{#2}}%
5020 }
```

`\storeordinalstringnum` Store textual representation of number. First argument is identifying name, second argument is a count register or number.

```
5021 \newrobustcmd*{\storeordinalstringnum}[2]{%
5022   \@ifnextchar[%
5023     {\@store@ordinal@string{#1}{#2}}%
```

```

5024 {\@store@ordinal@string{#1}{#2}[m]}%
5025 }

```

`@ordinal@string` Store textual representation of number according to gender.

```

5026 \def\@store@ordinal@string#1#2[#3]{%
5027 \ifthenelse{\equal{#3}{f}}%
5028 {%
5029 \protect\@ordinalstringF{#2}{\@fc@ordstr}%
5030 }%
5031 {%
5032 \ifthenelse{\equal{#3}{n}}%
5033 {%
5034 \protect\@ordinalstringN{#2}{\@fc@ordstr}%
5035 }%
5036 {%
5037 \ifthenelse{\equal{#3}{m}}%
5038 {}%
5039 {%
5040 \PackageError{fmtcount}%
5041 {Invalid gender option ‘#3’ to \protect\ordinalstring}%
5042 {Available options are m, f or n}%
5043 }%
5044 \protect\@ordinalstringM{#2}{\@fc@ordstr}%
5045 }%
5046 }%
5047 \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
5048 }

```

`\Ordinalstring` Display ordinal as a string with initial letters in upper case (argument is a counter)

```

5049 \newcommand*\Ordinalstring[1]{%
5050 \Ordinalstringnum{\expandafter\expandafter\expandafter\the\value{#1}}%
5051 }

```

`rdinalstringnum` Display ordinal as a string with initial letters in upper case (argument is a number or count register)

```

5052 \newrobustcmd*\Ordinalstringnum[1]{%
5053 \new@ifnextchar[%
5054 {\@Ordinal@string{#1}}%
5055 {\@Ordinal@string{#1}[m]}%
5056 }

```

`@Ordinal@string` Display ordinal as a string with initial letters in upper case according to gender

```

5057 \def\@Ordinal@string#1[#2]{%
5058 {%
5059 \ifthenelse{\equal{#2}{f}}%
5060 {%
5061 \protect\@OrdinalstringF{#1}{\@fc@ordstr}%
5062 }%
5063 {%

```



```

5064     \ifthenelse{\equal{#2}{n}}%
5065     {%
5066       \protect\@OrdinalstringN{#1}{\@fc@ordstr}%
5067     }%
5068     {%
5069       \ifthenelse{\equal{#2}{m}}%
5070       {}%
5071       {%
5072         \PackageError{fmtcount}%
5073         {Invalid gender option ‘#2’}%
5074         {Available options are m, f or n}%
5075       }%
5076       \protect\@OrdinalstringM{#1}{\@fc@ordstr}%
5077     }%
5078   }%
5079   \@fc@ordstr
5080 }%
5081 }

```

`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.

```

5082 \newcommand*{\storeOrdinalstring}[2]{%
5083   {%
5084     \toks0{\storeOrdinalstringnum{#1}}%
5085     \expandafter
5086   }\the\toks0\expandafter{\the\value{#2}}%
5087 }

```

`rdinalstringnum` Store textual representation of number, with initial letters in upper case. First argument is identifying name, second argument is a count register or number.

```

5088 \newrobustcmd*{\storeOrdinalstringnum}[2]{%
5089   \@ifnextchar[%
5090   {\@store@Ordinal@string{#1}{#2}}%
5091   {\@store@Ordinal@string{#1}{#2}[m]}%
5092 }

```

`@Ordinal@string` Store textual representation of number according to gender, with initial letters in upper case.

```

5093 \def\@store@Ordinal@string#1#2[#3]{%
5094   \ifthenelse{\equal{#3}{f}}%
5095   {%
5096     \protect\@OrdinalstringF{#2}{\@fc@ordstr}%
5097   }%
5098   {%
5099     \ifthenelse{\equal{#3}{n}}%
5100     {}%
5101     \protect\@OrdinalstringN{#2}{\@fc@ordstr}%
5102   }%
5103   {%
5104     \ifthenelse{\equal{#3}{m}}%

```

```

5105     {}%
5106     {%
5107         \PackageError{fmtcount}%
5108         {Invalid gender option ‘#3’}%
5109         {Available options are m or f}%
5110     }%
5111     \protect\@OrdinalstringM{#2}{\@fc@ordstr}%
5112 }%
5113 }%
5114 \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
5115 }

```

`\reORDINALstring` Store upper case textual representation of ordinal. The first argument is identifying name, the second argument is a counter.

```

5116 \newcommand*\storeORDINALstring}[2]{%
5117     {%
5118         \toks0{\storeORDINALstringnum{#1}}%
5119         \expandafter
5120     }\the\toks0\expandafter{\the\value{#2}}%
5121 }

```

`\RDINALstringnum` As above, but the second argument is a count register or a number.

```

5122 \newrobustcmd*\storeORDINALstringnum}[2]{%
5123     \@ifnextchar[%
5124     {\@store@ORDINAL@string{#1}{#2}}%
5125     {\@store@ORDINAL@string{#1}{#2}[m]}%
5126 }

```

`\@ORDINAL@string` Gender is specified as an optional argument at the end.

```

5127 \def\@store@ORDINAL@string#1#2[#3]{%
5128     \ifthenelse{\equal{#3}{f}}%
5129     {%
5130         \protect\@ordinalstringF{#2}{\@fc@ordstr}%
5131     }%
5132     {%
5133         \ifthenelse{\equal{#3}{n}}%
5134         {%
5135             \protect\@ordinalstringN{#2}{\@fc@ordstr}%
5136         }%
5137         {%
5138             \ifthenelse{\equal{#3}{m}}%
5139             {}%
5140             {%
5141                 \PackageError{fmtcount}%
5142                 {Invalid gender option ‘#3’}%
5143                 {Available options are m or f}%
5144             }%
5145             \protect\@ordinalstringM{#2}{\@fc@ordstr}%
5146         }%

```

```

5147 }%
5148 \expandafter\protected@edef\csname @fcs@#1\endcsname{%
5149   \noexpand\MakeUppercase{\@fc@ordstr}%
5150 }%
5151 }

```

`\ORDINALstring` Display upper case textual representation of an ordinal. The argument must be a counter.

```

5152 \newcommand*{\ORDINALstring}[1]{%
5153   \ORDINALstringnum{\expandafter\expandafter\expandafter
5154     \the\value{#1}}%
5155 }%
5156 }

```

`ORDINALstringnum` As above, but the argument is a count register or a number.

```

5157 \newrobustcmd*{\ORDINALstringnum}[1]{%
5158   \new@ifnextchar[%
5159     {\@ORDINAL@string{#1}}%
5160     {\@ORDINAL@string{#1}[m]}%
5161 }

```

`@ORDINAL@string` Gender is specified as an optional argument at the end.

```

5162 \def\@ORDINAL@string#1[#2]{%
5163   {%
5164     \ifthenelse{\equal{#2}{f}}%
5165     {%
5166       \protect\@ordinalstringF{#1}{\@fc@ordstr}%
5167     }%
5168     {%
5169       \ifthenelse{\equal{#2}{n}}%
5170       {%
5171         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
5172       }%
5173       {%
5174         \ifthenelse{\equal{#2}{m}}%
5175         {}%
5176         {%
5177           \PackageError{fmtcount}%
5178             {Invalid gender option ‘#2’}%
5179             {Available options are m, f or n}%
5180         }%
5181         \protect\@ordinalstringM{#1}{\@fc@ordstr}%
5182       }%
5183     }%
5184     \MakeUppercase{\@fc@ordstr}%
5185   }%
5186 }

```

`orennumberstring` Convert number to textual representation, and store. First argument is the identifying name, second argument is a counter containing the number.

```

5187 \newcommand*\storenumberstring}[2]{%
5188   \expandafter\protect\expandafter\storenumberstringnum{#1}{%
5189     \expandafter\the\value{#2}}}%
5190 }

```

`numberstringnum` As above, but second argument is a number or count register.

```

5191 \newcommand*\storenumberstringnum}[2]{%
5192   \@ifnextchar[%
5193     {\@store@number@string{#1}{#2}}%
5194     {\@store@number@string{#1}{#2}[m]}}%
5195 }

```

`e@number@string` Gender is given as optional argument, *at the end*.

```

5196 \def\@store@number@string#1#2[#3]{%
5197   \ifthenelse{\equal{#3}{f}}%
5198     {%
5199       \protect\@numberstringF{#2}{\@fc@numstr}%
5200     }%
5201     {%
5202       \ifthenelse{\equal{#3}{n}}%
5203         {%
5204           \protect\@numberstringN{#2}{\@fc@numstr}%
5205         }%
5206         {%
5207           \ifthenelse{\equal{#3}{m}}%
5208             {}%
5209             {%
5210               \PackageError{fmtcount}
5211                 {Invalid gender option ‘#3’}%
5212                 {Available options are m, f or n}%
5213             }%
5214           \protect\@numberstringM{#2}{\@fc@numstr}%
5215         }%
5216     }%
5217   \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
5218 }

```

`\numberstring` Display textual representation of a number. The argument must be a counter.

```

5219 \newcommand*\numberstring}[1]{%
5220   \numberstringnum{\expandafter\expandafter\expandafter
5221     \the\value{#1}}}%
5222 }

```

`numberstringnum` As above, but the argument is a count register or a number.

```

5223 \newrobustcmd*\numberstringnum}[1]{%
5224   \new@ifnextchar[%
5225     {\@number@string{#1}}%
5226     {\@number@string{#1}[m]}}%
5227 }

```

`\@number@string` Gender is specified as an optional argument *at the end*.

```
5228 \def\@number@string#1[#2]{%
5229   {%
5230     \ifthenelse{\equal{#2}{f}}%
5231     {%
5232       \protect\@numberstringF{#1}{\@fc@numstr}%
5233     }%
5234     {%
5235       \ifthenelse{\equal{#2}{n}}%
5236       {%
5237         \protect\@numberstringN{#1}{\@fc@numstr}%
5238       }%
5239       {%
5240         \ifthenelse{\equal{#2}{m}}%
5241         {}%
5242         {%
5243           \PackageError{fmtcount}%
5244             {Invalid gender option ‘#2’}%
5245             {Available options are m, f or n}%
5246         }%
5247         \protect\@numberstringM{#1}{\@fc@numstr}%
5248       }%
5249     }%
5250   \@fc@numstr
5251 }%
5252 }
```

`\storeNumberstring` Store textual representation of number. First argument is identifying name, second argument is a counter.

```
5253 \newcommand*\@storeNumberstring[2]{%
5254   {%
5255     \toks0{\@storeNumberstringnum{#1}}%
5256     \expandafter
5257     }\the\toks0\expandafter{\the\value{#2}}%
5258 }
```

`\Numberstringnum` As above, but second argument is a count register or number.

```
5259 \newcommand{\@storeNumberstringnum}[2]{%
5260   \@ifnextchar[%
5261   {\@store@Number@string{#1}{#2}}%
5262   {\@store@Number@string{#1}{#2}[m]}%
5263 }
```

`\e@Number@string` Gender is specified as an optional argument *at the end*:

```
5264 \def\@store@Number@string#1#2[#3]{%
5265   \ifthenelse{\equal{#3}{f}}%
5266   {%
5267     \protect\@NumberstringF{#2}{\@fc@numstr}%
5268   }%
```

```

5269  {%
5270    \ifthenelse{\equal{#3}{n}}%
5271    {%
5272      \protect\@NumberstringN{#2}{\@fc@numstr}%
5273    }%
5274    {%
5275      \ifthenelse{\equal{#3}{m}}%
5276      {}%
5277      {%
5278        \PackageError{fmtcount}%
5279        {Invalid gender option ‘#3’}%
5280        {Available options are m, f or n}%
5281      }%
5282      \protect\@NumberstringM{#2}{\@fc@numstr}%
5283    }%
5284  }%
5285  \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
5286 }

```

`\Numberstring` Display textual representation of number. The argument must be a counter.

```

5287 \newcommand*{\Numberstring}[1]{%
5288   \Numberstringnum{\expandafter\expandafter\expandafter
5289     \the\value{#1}}%
5290 }

```

`Numberstringnum` As above, but the argument is a count register or number.

```

5291 \newrobustcmd*{\Numberstringnum}[1]{%
5292   \new@ifnextchar[%
5293     {\@Number@string{#1}}%
5294     {\@Number@string{#1}[m]}%
5295 }

```

`\@Number@string` Gender is specified as an optional argument at the end.

```

5296 \def\@Number@string#1[#2]{%
5297   {%
5298     \ifthenelse{\equal{#2}{f}}%
5299     {%
5300       \protect\@NumberstringF{#1}{\@fc@numstr}%
5301     }%
5302     {%
5303       \ifthenelse{\equal{#2}{n}}%
5304       {%
5305         \protect\@NumberstringN{#1}{\@fc@numstr}%
5306       }%
5307       {%
5308         \ifthenelse{\equal{#2}{m}}%
5309         {}%
5310         {%
5311           \PackageError{fmtcount}%

```

```

5312         {Invalid gender option ‘#2’}%
5313         {Available options are m, f or n}%
5314     }%
5315     \protect\@NumberstringM{#1}{\@fc@numstr}%
5316 }%
5317 }%
5318 \@fc@numstr
5319 }%
5320 }

```

`\storeNUMBERstring` Store upper case textual representation of number. The first argument is identifying name, the second argument is a counter.

```

5321 \newcommand{\storeNUMBERstring}[2]{%
5322   {%
5323     \toks0{\storeNUMBERstringnum{#1}}%
5324     \expandafter
5325     }\the\toks0\expandafter{\the\value{#2}}%
5326 }

```

`\NUMBERstringnum` As above, but the second argument is a count register or a number.

```

5327 \newcommand{\storeNUMBERstringnum}[2]{%
5328   \@ifnextchar[%
5329   {\@store@NUMBER@string{#1}{#2}}%
5330   {\@store@NUMBER@string{#1}{#2}[m]}%
5331 }

```

`\e@NUMBER@string` Gender is specified as an optional argument at the end.

```

5332 \def\@store@NUMBER@string#1#2[#3]{%
5333   \ifthenelse{\equal{#3}{f}}%
5334   {%
5335     \protect\@numberstringF{#2}{\@fc@numstr}%
5336   }%
5337   {%
5338     \ifthenelse{\equal{#3}{n}}%
5339     {%
5340       \protect\@numberstringN{#2}{\@fc@numstr}%
5341     }%
5342     {%
5343       \ifthenelse{\equal{#3}{m}}%
5344       {}%
5345     }%
5346     \PackageError{fmtcount}%
5347     {Invalid gender option ‘#3’}%
5348     {Available options are m or f}%
5349   }%
5350   \protect\@numberstringM{#2}{\@fc@numstr}%
5351 }%
5352 }%
5353 \expandafter\edef\csname @fcs@#1\endcsname{%

```

```

5354 \noexpand\MakeUppercase{\@fc@numstr}%
5355 }%
5356 }

```

`\NUMBERstring` Display upper case textual representation of a number. The argument must be a counter.

```

5357 \newcommand*{\NUMBERstring}[1]{%
5358 \NUMBERstringnum{\expandafter\expandafter\expandafter
5359 \the\value{#1}}%
5360 }

```

`NUMBERstringnum` As above, but the argument is a count register or a number.

```

5361 \newrobustcmd*{\NUMBERstringnum}[1]{%
5362 \new@ifnextchar[%
5363 {\@NUMBER@string{#1}}%
5364 {\@NUMBER@string{#1}[m]}%
5365 }

```

`\@NUMBER@string` Gender is specified as an optional argument at the end.

```

5366 \def\@NUMBER@string#1[#2]{%
5367 {%
5368 \ifthenelse{\equal{#2}{f}}%
5369 {%
5370 \protect\@numberstringF{#1}{\@fc@numstr}%
5371 }%
5372 {%
5373 \ifthenelse{\equal{#2}{n}}%
5374 {%
5375 \protect\@numberstringN{#1}{\@fc@numstr}%
5376 }%
5377 {%
5378 \ifthenelse{\equal{#2}{m}}%
5379 {}%
5380 {%
5381 \PackageError{fmtcount}%
5382 {Invalid gender option ‘#2’}%
5383 {Available options are m, f or n}%
5384 }%
5385 \protect\@numberstringM{#1}{\@fc@numstr}%
5386 }%
5387 }%
5388 \MakeUppercase{\@fc@numstr}%
5389 }%
5390 }

```

`\binary` Number representations in other bases. Binary:

```

5391 \providecommand*{\binary}[1]{%
5392 \@binary{\expandafter\expandafter\expandafter
5393 \the\value{#1}}%
5394 }

```



`\aaalph` Like `\alph`, but goes beyond 26. (a ... z aa ... zz ...)

```
5395 \providecommand*\aaalph[1]{%
5396   \@aaaalph{\expandafter\expandafter\expandafter
5397     \the\value{#1}}%
5398 }
```

`\AAAAlph` As before, but upper case.

```
5399 \providecommand*\AAAAlph[1]{%
5400   \@AAAAlph{\expandafter\expandafter\expandafter
5401     \the\value{#1}}%
5402 }
```

`\abalph` Like `\alph`, but goes beyond 26. (a ... z ab ... az ...)

```
5403 \providecommand*\abalph[1]{%
5404   \@abalph{\expandafter\expandafter\expandafter
5405     \the\value{#1}}%
5406 }
```

`\ABAlph` As above, but upper case.

```
5407 \providecommand*\ABAlph[1]{%
5408   \@ABAlph{\expandafter\expandafter\expandafter
5409     \the\value{#1}}%
5410 }
```

`\hexadecimal` Hexadecimal:

```
5411 \providecommand*\hexadecimal[1]{%
5412   \hexadecimalnum{\expandafter\expandafter\expandafter
5413     \the\value{#1}}%
5414 }
```

`\HEXADecimal` As above, but in upper case.

```
5415 \providecommand*\HEXADecimal[1]{%
5416   \HEXADecimalnum{\expandafter\expandafter\expandafter
5417     \the\value{#1}}%
5418 }
5419 \newrobustcmd*\FC@Hexadecimal@warning{%
5420   \PackageWarning{fmtcount}{\string\Hexadecimal\space is deprecated, use \string\HEXADecimal\space
5421     instead. The \string\Hexadecimal\space control sequence name is confusing as it can mislead
5422     that only the 1st letter is upper-cased.}%
5423 }
5424 \def\Hexadecimal{%
5425   \FC@Hexadecimal@warning
5426   \HEXADecimal}
```

`\octal` Octal:

```
5427 \providecommand*\octal[1]{%
5428   \@octal{\expandafter\expandafter\expandafter
5429     \the\value{#1}}%
5430 }
```

`\decimal` Decimal:

```
5431 \providecommand*\decimal}[1]{%
5432   \@decimal{\expandafter\expandafter\expandafter
5433     \the\value{#1}}%
5434 }
```

### 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.

```
5435 \newif\iffc@languagemode@detected
5436 \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.

```
5437 \def\@setdef@ultfmtcount{%
5438   \fc@languagemode@detectedtrue
5439   \ifcsundef{@ordinalMenglish}{\FCloadlang{english}}{}%
5440   \def\@ordinalstringM{\@ordinalstringMenglish}%
5441   \let\@ordinalstringF=\@ordinalstringMenglish
5442   \let\@ordinalstringN=\@ordinalstringMenglish
5443   \def\@OrdinalstringM{\@OrdinalstringMenglish}%
5444   \let\@OrdinalstringF=\@OrdinalstringMenglish
5445   \let\@OrdinalstringN=\@OrdinalstringMenglish
5446   \def\@numberstringM{\@numberstringMenglish}%
5447   \let\@numberstringF=\@numberstringMenglish
5448   \let\@numberstringN=\@numberstringMenglish
5449   \def\@NumberstringM{\@NumberstringMenglish}%
5450   \let\@NumberstringF=\@NumberstringMenglish
5451   \let\@NumberstringN=\@NumberstringMenglish
5452   \def\@ordinalM{\@ordinalMenglish}%
5453   \let\@ordinalF=\@ordinalM
5454   \let\@ordinalN=\@ordinalM
5455   \let\fmtord\fc@orddef@ult
5456 }
```

```
\fc@multiling \fc@multiling{<name>}{<gender>}
5457 \newcommand*\fc@multiling}[2]{%
5458   \ifcsundef{@#1#2\languagename}%
5459   {% try loading it
5460     \FCloadlang{\languagename}%
5461   }%
5462   {%
5463   }%
5464   \ifcsundef{@#1#2\languagename}%
5465   {%
5466     \PackageWarning{fmtcount}%
```

```

5467 {No support for \expandafter\protect\csname #1\endcsname\space for
5468   language '\language' }%
5469 \ifthenelse{\equal{\language}{\fc@mainlang}}%
5470 {%
5471   \FCloadlang{english}%
5472 }%
5473 {%
5474 }%
5475 \ifcsdef{@#1#2\fc@mainlang}%
5476 {%
5477   \csuse{@#1#2\fc@mainlang}%
5478 }%
5479 {%
5480   \PackageWarningNoLine{fmtcount}%
5481     {No languages loaded at all! Loading english definitions}%
5482   \FCloadlang{english}%
5483   \def\fc@mainlang{english}%
5484   \csuse{@#1#2english}%
5485 }%
5486 }%
5487 {%
5488   \csuse{@#1#2\language}%
5489 }%
5490 }

```

itling@fmtcount This defines the number and ordinal string macros to use \language:

```

5491 \def\@set@multiling@fmtcount{%
5492   \fc@languagemode@detectedtrue

```

The masculine version of \numberstring:

```

5493 \def\@numberstringM{%
5494   \fc@multiling{numberstring}{M}%
5495 }%

```

The feminine version of \numberstring:

```

5496 \def\@numberstringF{%
5497   \fc@multiling{numberstring}{F}%
5498 }%

```

The neuter version of \numberstring:

```

5499 \def\@numberstringN{%
5500   \fc@multiling{numberstring}{N}%
5501 }%

```

The masculine version of \Numberstring:

```

5502 \def\@NumberstringM{%
5503   \fc@multiling{Numberstring}{M}%
5504 }%

```

The feminine version of \Numberstring:

```

5505 \def\@NumberstringF{%

```

5506 \fc@multiling{Numberstring}{F}%  
5507 }%

The neuter version of \Numberstring:

5508 \def\@NumberstringN{%  
5509 \fc@multiling{Numberstring}{N}%  
5510 }%

The masculine version of \ordinal:

5511 \def\@ordinalM{%  
5512 \fc@multiling{ordinal}{M}%  
5513 }%

The feminine version of \ordinal:

5514 \def\@ordinalF{%  
5515 \fc@multiling{ordinal}{F}%  
5516 }%

The neuter version of \ordinal:

5517 \def\@ordinalN{%  
5518 \fc@multiling{ordinal}{N}%  
5519 }%

The masculine version of \ordinalstring:

5520 \def\@ordinalstringM{%  
5521 \fc@multiling{ordinalstring}{M}%  
5522 }%

The feminine version of \ordinalstring:

5523 \def\@ordinalstringF{%  
5524 \fc@multiling{ordinalstring}{F}%  
5525 }%

The neuter version of \ordinalstring:

5526 \def\@ordinalstringN{%  
5527 \fc@multiling{ordinalstring}{N}%  
5528 }%

The masculine version of \Ordinalstring:

5529 \def\@OrdinalstringM{%  
5530 \fc@multiling{Ordinalstring}{M}%  
5531 }%

The feminine version of \Ordinalstring:

5532 \def\@OrdinalstringF{%  
5533 \fc@multiling{Ordinalstring}{F}%  
5534 }%

The neuter version of \Ordinalstring:

5535 \def\@OrdinalstringN{%  
5536 \fc@multiling{Ordinalstring}{N}%  
5537 }%

Make `\fmtord` language dependent:

```
5538 \let\fmtord\fc@ord@multiling
5539 }
```

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.

```
5540 \def\fc@check@for@multiling#1:#2\@nil{%
5541   \@ifpackageloaded{#1}{%
5542     #2\@set@multiling@fmtcount
5543   }{}%
5544 }
```

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.

```
5545 \def\fc@loop@on@multiling@pkg#1,{%
5546   \def\@tempb{#1}%
5547   \ifx\@tempb\@nnil
```

We have reached the end of the loop, so stop here.

```
5548   \let\fc@loop@on@multiling@pkg\empty
5549   \else
```

Make the `\@ifpackageloaded` test and break the loop if it was positive.

```
5550     \fc@check@for@multiling#1\@nil
5551     \iffc@languagemode@detected
5552     \def\fc@loop@on@multiling@pkg##1\@nil,{}%
5553     \fi
5554   \fi
5555   \fc@loop@on@multiling@pkg
5556 }
```

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.:

```
5557 \AtBeginDocument{%
5558   \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.

```
5559   \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 `\language` is defined, and activate the multilingual mode only then, and otherwise fall back to default legacy mode.

```
5560     \ifcsundef{language}%
5561     {%
5562       \PackageWarning{fmtcount}{%
5563         ‘\protect\language’ is undefined, you should use a language package such as bab
```

```

5564         when loading a language via package option. Reverting to default language.
5565     }%
5566     \@setdef@ultfmtcount
5567 }{%
5568     \@set@multiling@fmtcount
5569

```

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 `\language $\langle$ name` have been loaded, and otherwise fall `\language $\langle$ name` back to the latest `fmtcount` language definition loaded.

```

5570     \@FC@iflangloaded{\language $\langle$ name}{}{%

```

The current `\language $\langle$ name` is not a `fmtcount` language that has been previously loaded. The correction is to have `\language $\langle$ name` let to `\fc@mainlang`. Please note that, as `\iffmtcount@language@option` is true, we know that `fmtcount` has loaded some language.

```

5571         \PackageWarning{fmtcount}{%
5572             Setting ‘\protect\language $\langle$ name’ to ‘\fc@mainlang’.\MessageBreak
5573             Reason is that ‘\protect\language $\langle$ name’ was ‘\language $\langle$ name’,\MessageBreak
5574             but ‘\language $\langle$ name’ was not loaded by fmtcount,\MessageBreak
5575             whereas ‘\fc@mainlang’ was the last language loaded by fmtcount ;
5576         }%
5577         \let\language $\langle$ name\fc@mainlang
5578     }%
5579 }%
5580 \else
5581     \@setdef@ultfmtcount
5582 \fi\fi
5583 }

```

```

5584 \AtBeginDocument{%
5585     \ifcsundef{FBsupR}{\let\fc@textsuperscript\textsuperscript}{\let\fc@textsuperscript\fup}%
5586 }

```

Backwards compatibility:

```

5587 \let\@ordinal=\@ordinalM
5588 \let\@ordinalstring=\@ordinalstringM
5589 \let\@Ordinalstring=\@OrdinalstringM
5590 \let\@numberstring=\@numberstringM
5591 \let\@Numberstring=\@NumberstringM

```