

# 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)

2013-08-17 (version 2.03)

## 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 French . . . . .	9
4.2	Prefixes . . . . .	14
<b>5</b>	<b>Configuration File <code>fmtcount.cfg</code></b>	<b>14</b>
<b>6</b>	<b>LaTeX2HTML style</b>	<b>15</b>
<b>7</b>	<b>Acknowledgements</b>	<b>15</b>
<b>8</b>	<b>Troubleshooting</b>	<b>15</b>
<b>9</b>	<b>The Code</b>	<b>15</b>
9.1	<code>fcnumparser.sty</code> . . . . .	15
9.2	<code>fcprefix.sty</code> . . . . .	26
9.3	<code>fmtcount.sty</code> . . . . .	36
9.4	Multilanguage Definitions . . . . .	60
9.4.1	<code>fc-american.def</code> . . . . .	63
9.4.2	<code>fc-british.def</code> . . . . .	63
9.4.3	<code>fc-english.def</code> . . . . .	64
9.4.4	<code>fc-francais.def</code> . . . . .	74
9.4.5	<code>fc-french.def</code> . . . . .	74
9.4.6	<code>fc-frenchb.def</code> . . . . .	104
9.4.7	<code>fc-german.def</code> . . . . .	105

9.4.8	<code>fc-germanb.def</code>	114
9.4.9	<code>fc-italian</code>	115
9.4.10	<code>fc-ngerman.def</code>	116
9.4.11	<code>fc-ngermanb.def</code>	117
9.4.12	<code>fc-portuges.def</code>	117
9.4.13	<code>fc-spanish.def</code>	131
9.4.14	<code>fc-UKenglish.def</code>	148
9.4.15	<code>fc-USenglish.def</code>	148

## 1 Introduction

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

## 2 Available Commands

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

`\ordinal`

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

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

`\fmtord`

```
\fmtord{<text>}
```

is used to format the st, nd, rd, th bit. By default the ordinal is formatted as a superscript, if the package option `level` is used, it is level with the text. For example, if the current section is 3, then `\ordinal{section}` will produce the output: 3<sup>rd</sup>. Note that the optional argument `<gender>` occurs *at the end*. This argument may only take one of the following values: `m` (masculine), `f` (feminine) or `n` (neuter.) If `<gender>` 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 `mem-oir`'s version of that command.

2. As with all commands which have an optional argument as the last argument, if the optional argument is omitted, any spaces following the final argument will be ignored. Whereas, if the optional argument is present, any spaces following the optional argument won't be ignored. so `\ordinal{section} !` will produce: 3<sup>rd</sup>! whereas `\ordinal{section}[m] !` will produce: 3<sup>rd</sup> !

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{3}` will produce: 3<sup>rd</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: Three.

`\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>]`

Theses 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.

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

`\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: Third.

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

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

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

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

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

These macros work like `\ordinalstring`, `\Ordinalstring` and `\ORDINALstring`, respectively, but take an actual number rather than a counter as the argument. For example, `\ordinalstringnum{3}` will produce: third.

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>]
\storeordinalstring	\storeordinalstring{<label>}{<counter>}[<gender>]
\storeOrdinalstring	\storeOrdinalstring{<label>}{<counter>}[<gender>]
\storeORDINALstring	\storeORDINALstring{<label>}{<counter>}[<gender>]
\storenumberstring	\storenumberstring{<label>}{<counter>}[<gender>]
\storeNumberstring	\storeNumberstring{<label>}{<counter>}[<gender>]
\storeNUMBERstring	\storeNUMBERstring{<label>}{<counter>}[<gender>]
\storeordinalnum	\storeordinalnum{<label>}{<number>}[<gender>]
\storeordinalstringnum	\storeordinalstring{<label>}{<number>}[<gender>]
\storeOrdinalstringnum	\storeOrdinalstringnum{<label>}{<number>}[<gender>]
\storeORDINALstringnum	\storeORDINALstringnum{<label>}{<number>}[<gender>]
\storenumberstringnum	\storenumberstring{<label>}{<number>}[<gender>]
\storeNumberstringnum	\storeNumberstring{<label>}{<number>}[<gender>]
\storeNUMBERstringnum	\storeNUMBERstring{<label>}{<number>}[<gender>]

`\binary`      `\binary{<counter>}`

This will print the value of `<counter>` as a binary number. E.g. `\binary{section}` will produce: 11. 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: 00000011. 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.

`\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.

`\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: 00000005.

`\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.

`\AAAlph`

`\AAAlph{<counter>}`

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

`\aaalphnum`

`\aaalphnum{<n>}`

`\AAAlphnum`

`\AAAlphnum{<n>}`

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

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

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

These can also be set using the command:

`\fmtcountsetoptions`

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

where `<type>` is either `level` or `raise`.

### 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>

To ensure the language definitions are loaded correctly for document dialects, use

`\FCloadlang`

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

in the preamble. The `<dialect>` should match the options passed to `babel` or `polyglossia`. If you don't use this, `fmtcount` will attempt to detect the required dialects, but this isn't guaranteed to work.

The commands `\ordinal`, `\ordinalstring` and `\numberstring` (and their variants) will be formatted in the currently selected language. If the current language hasn't been loaded (via `\FCloadlang` above) and `fmtcount` detects a definition file for that language it will attempt to load it, but this isn't robust and may cause problems, so it's best to use `\FCloadlang`.

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

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



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 § 4.1.

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 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.

`\fmtcountsetoptions`

```
\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 :

<code>france</code>	soixante-dix pour 70, quatre-vingts pour 80, et quate-vingts-dix pour 90,
<code>belgian</code>	septante pour 70, quatre-vingts pour 80, et nonante pour 90,
<code>swiss</code>	septante pour 70, huitante <sup>4</sup> pour 80, et nonante pour 90

<sup>4</sup>voir [Octante et huitante](#) sur le site d'Alain Lassine

Il est à noter que la variante *belgian* est parfaitement correcte pour les francophones français<sup>5</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>}`

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

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 *<n>illion* et *<n>illiard*, où *<n>* 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 :

<sup>5</sup>Je précise que l'auteur de ces lignes est français

- 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 :

traditional	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,
reformed	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,
traditional o	pareil que <code>traditional</code> mais dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur ordinale,
reformed o	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,
always	pour marquer toujours le pluriel, ceci n'est correct que pour « mil » vis à vis des règles en vigueur,
never	pour ne jamais marquer le pluriel, ceci est incorrect vis à vis des règles d'orthographe en vigueur,

multiple	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,
multiple g-last	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,
multiple l-last	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 <i>localement</i> mais <i>non globalement</i> en dernière position, où “localement” et <i>globalement</i> on la même signification que pour les options 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 <i>n</i> ’est pas <i>globalement</i> en dernière position, où “globalement” a la même signification que pour l’option multiple g-last ; ceci est la règle que j’infère être en vigueur pour les nombres de la forme $\langle n \rangle$ 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 :

⟨x⟩ dans “⟨x⟩ 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, actuellement l'effet est le même que 1990, ou à  
 always pour mettre systématiquement des traits d'union de partout.

Par défaut, l'option vaut `reformed`.

scale

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

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

recursive dans ce cas  $10^{30}$  donne mille milliards de milliards de milliards, pour  $10^n$ , on écrit  $10^{n-9 \times \max\{(n \div 9) - 1, 0\}}$  suivi de la répétition  $\max\{(n \div 9) - 1, 0\}$  fois de “de milliards”

long  $10^{6 \times n}$  donne un  $\langle n \rangle$ illion où  $\langle n \rangle$  est remplacé par “bi” pour 2, “tri” pour 3, etc. et  $10^{6 \times n + 3}$  donne un  $\langle n \rangle$ illiard avec la même convention pour  $\langle n \rangle$ . L’option long est correcte en Europe, par contre j’ignore l’usage au Québec.

short  $10^{6 \times n}$  donne un  $\langle n \rangle$ illion où  $\langle n \rangle$  est remplacé par “bi” pour 2, “tri” pour 3, etc. L’option short est incorrecte en Europe.

Par défaut, l’option vaut recursive.

n-illiard upto `\fmtcountsetoptions{french={n-illiard upto=<n-illiard upto>}}`

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

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

mil plural mark `\fmtcountsetoptions{french={mil plural mark=<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 « mille » 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).

## 4.2 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 Acknowledgements

I would like to thank all the people who have provided translations.

## 8 Troubleshooting

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

## 9 The Code

### 9.1 fcnumparser.sty

```
1 \NeedsTeXFormat{LaTeX2e}  
2 \ProvidesPackage{fcnumparser}[2012/09/28]
```

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

```

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

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

First check for the presence of a decimal point in the number.
10  \def\@tempb##1.##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%
11  \@tempb#1.\fc@end\fc@nil
12  \ifx\@tempa\fc@end@

Here \@tempa is \ifx-equal to \fc@end, which means that the number does
not contain any decimal point. So we do the same trick to search for a comma.
13    \def\@tempb##1,##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%
14    \@tempb#1,\fc@end\fc@nil
15    \ifx\@tempa\fc@end@

No comma either, so fractional part is set empty.
16      \def\fc@fractional@part{}%
17    \else

Comma has been found, so we just need to drop ',\fc@end' from the end of
\@tempa to get the fractional part.
18      \def\@tempb##1,\fc@end{\def\fc@fractional@part{##1}}%
19      \expandafter\@tempb\@tempa
20    \fi
21  \else

Decimal point has been found, so we just need to drop '.\fc@end' from the
end \@tempa to get the fractional part.
22      \def\@tempb##1.\fc@end{\def\fc@fractional@part{##1}}%
23      \expandafter\@tempb\@tempa
24    \fi
25 }

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

First remove all the spaces in #1, and place the result into \@tempa.
27  \let\@tempa\@empty
28  \def\@tempb##1##2\fc@nil{%
29    \def\@tempc{##1}%
30    \ifx\@tempc\space

```



```

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

```

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

```

43 \def\@tempb##1##2\fc@nil{\def\fc@sign{##1}\def\fc@number{##2}}%
44 \expandafter\@tempb\@tempa\fc@nil
45 \expandafter\if\fc@sign+%
46 \def\fc@sign@case{1}%
47 \else
48 \expandafter\if\fc@sign-%
49 \def\fc@sign@case{2}%
50 \else
51 \def\fc@sign{}%
52 \def\fc@sign@case{0}%
53 \let\fc@number\@tempa
54 \fi
55 \fi
56 \ifx\fc@number\@empty
57 \PackageError{fcnumparser}{Invalid number}{Number must contain at least one non blank
58 character after sign}%
59 \fi

```

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

```

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

```

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

```

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

```

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

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

```

63 \fc@digit@counter -\fc@digit@counter
64 \advance\fc@digit@counter by \fc@unit@weight

```

```

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

```

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

```

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

```

Now we loop to copy the digits. To do that we define a macro  $\text{\@templ}$  for terminal recursion.

```

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

```

Here is the loop body:

```

82 {%
83 \count0 \@temp i
84 \advance\count0 by \fc@digit@counter
85 \expandafter\def\expandafter\@tempd\expandafter{\csname fc@digit@\the\count0\endcsname}
86 \expandafter\def\expandafter\@tempe\expandafter{\csname fc@digit@\the\fc@digit@counter\endcsname}
87 \def\@tempa####1####2{\def\@tempb{\let####1####2}}%
88 \expandafter\expandafter\expandafter\@tempa\expandafter\@tempe\@tempd
89 \expandafter
90 }\@tempb
91 \advance\fc@digit@counter by 1 %
92 \fi
93 \next
94}%
95 \let\next\@templ
96 \@templ

```

Split  $\text{\fc@fractional@part}$  into a sequence of  $\text{\fc@digit@}\langle n \rangle$  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.

```

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

```

$\text{\fc@}$   $\text{@parse@integer@digits}$  Macro  $\text{\fc@parse@integer@digits}$  is used to

```

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

```

Now we have macros to read a few digits from the `\fc@digit@<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.

```

119 \ifundefined{fc@read@unit}{-}{%
120   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro 'fc@read@unit'}}

```

Arguments as follows:

#1 output counter: into which the read value is placed  
 #2 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.

```

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

```

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

```
139 \ifundefined{fc@read@hundred}{%
140   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro 'fc@read@hundred'}
```

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

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

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

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

```
155 \ifundefined{fc@read@thousand}{%
156   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
157     'fc@read@thousand'}}
```

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

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

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

`\fc` Note: one myriad is ten thousand. `@read@thousand` Macro `\fc@read@myriad` is used to read a quatuor of digits and form an integer in the range [0..9999].

First we check that the macro is not yet defined.

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

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

#1 output counter: into which the read value is placed

#2 input number: unit weight at which reach the value is to be read

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

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

```
189 \ifundefined{fc@check@nonzeros}{-}{%
190   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
191     '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 search the non-zeros

#3 output macro: let  $n$  be the number represented by digits the weight of which span from #1 to #2, then #3 is set to the number  $\min(n, 9)$ .

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

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

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

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

Then we do the real job

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

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

```

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

\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.
204 \def\fc@@check@nonzeros@inner{%
205   \ifnum\count0<\fc@min@weight
206     \count0=\fc@min@weight\relax
207   \fi
208   \ifnum\count1>\fc@max@weight\relax
209     \count1=\fc@max@weight
210   \fi
211   \count2\count0 %
212   \advance\count2 by 1 %
213   \ifnum\count0>\count1 %
214     \PackageError{fcnumparser}{Unexpected arguments}{Number in argument 2 of macro
215       'fc@check@nonzeros' must be at least equal to number in argument 1}%
216   \else
217     \fc@@check@nonzeros@inner@loopbody
218     \ifnum\@tempc>0 %
219       \ifnum\@tempc<9 %
220         \ifnum\count0>\count1 %
221           \else
222             \let\@tempd\@tempc
223             \fc@@check@nonzeros@inner@loopbody
224             \ifnum\@tempc=0 %
225               \let\@tempc\@tempd
226             \else
227               \def\@tempc{9}%
228             \fi
229           \fi
230         \fi

```

```

231     \fi
232   \fi
233 }
234 \def\fc@@check@nonzeros@inner@loopbody{%
235   % \@tempc <- digit of weight \count0
236   \expandafter\let\expandafter\@tempc\csname fc@digit@\the\count0\endcsname
237   \advance\count0 by 1 %
238   \ifnum\@tempc=0 %
239     \ifnum\count0>\count1 %
240       \let\next\relax
241     \else
242       \let\next\fc@@check@nonzeros@inner@loopbody
243     \fi
244   \else
245     \ifnum\count0>\count2 %
246       \def\@tempc{9}%
247     \fi
248     \let\next\relax
249   \fi
250   \next
251 }

```

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

```

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

```

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

```

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

```

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

```

257   \count0=\fc@unit@weight\space
258   \ifnum\count0<\fc@min@weight\space
259     \count0=\fc@min@weight\space
260   \fi

```

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

```

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

```

```

269     \fi
270     \next
271 }%
272 \let\next\@templ
273 \@templ

```

Now propagate result after closing bracket into counter #1.

```

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

```

\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

```

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

```

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

```

283     \def\@tempa##1\fc@case##2\@nil\fc@end{%
284       \toks0{##1}%

```

Actually a dummy \fc@case is appended to \toks1, because that makes easier further checking that it does not contains any other \fc@case.

```

285       \toks1{##2\fc@case}%
286     }%
287     \@tempa#1\fc@end

```

Now leading part upto last word should be in \toks0, and last word should be in \toks1. However we need to check that this is really the last word, i.e. we need to check that there is no \fc@case inside \toks1 other than the tailing dummy one. To that purpose we will loop while we find that \toks1 contains some \fc@case. First we define \@tempa to split \the\toks1 between parts before and after some potential \fc@case.

```

288     \def\@tempa##1\fc@case##2\fc@end{%
289       \toks2{##1}%
290       \def\@tempb{##2}%
291       \toks3{##2}%
292     }%

```

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

```

293     \toksdef\@tempt0 %

```

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

```

294     \def\@templ{%
295       \expandafter\@tempa\the\toks1 \fc@end

```



```

296      \ifx\@tempb\@empty
\@tempb empty means that the only \fc@case found in \the\toks1 is the
dummy one. So we end the loop here, \toks2 contains the last word.
297      \let\next\relax
298      \else
\@tempb is not empty, first we use
299      \expandafter\expandafter\expandafter\@tempt
300      \expandafter\expandafter\expandafter{%
301      \expandafter\the\expandafter\@tempt
302      \expandafter\fc@case\the\toks2}%
303      \toks1\toks3 %
304      \fi
305      \next
306  }%
307  \let\next\@templ
308  \@templ
309  \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks2}}%
310  \expandafter
311  }\@tempa
312 }

```

\fc @get@last@word Getting last letter. Arguments as follows:

```

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

```

```

313 \ifundefined{fc@get@last@letter}{\PackageError{fcnumparser}{Duplicate definition}{Redefi
314   of macro 'fc@get@last@letter'}}%
315 \def\fc@get@last@letter#1#2#3{%
316   {%

```

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

```

317   \toks1{#1}%
318   \toks0{}%
319   \toksdef\@tempt0 %

```

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

```

320   \def\@tempa##1##2\fc@nil{%
321     \toks2{##1}%
322     \toks3{##2}%
323     \def\@tempb{##2}%
324   }%

```

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

```

325   \def\@templ{%
326     \expandafter\@tempa\the\toks1 \fc@nil
327     \ifx\@tempb\@empty

```

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

```

328     \let\next\relax
329     \else

```

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

```

330     \expandafter\expandafter\expandafter\@tempt
331     \expandafter\expandafter\expandafter{%
332     \expandafter\the\expandafter\@tempt
333     \the\toks2}%

```

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

```

334     \toks1\toks3 %
335     \fi
336     \next
337 }%

```

Here run the loop.

```

338     \let\next\@templ
339     \next

```

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

```

340     \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks1}}%
341     \expandafter
342     }\@tempa
343 }%

```

## 9.2 fcprefix.sty

Pseudo-latin prefixes.

```

344 \NeedsTeXFormat{LaTeX2e}
345 \ProvidesPackage{fcprefix}[2012/09/28]
346 \RequirePackage{ifthen}
347 \RequirePackage{keyval}
348 \RequirePackage{fcnumparser}

```

Option ‘use duode and unde’ is to select whether 18 and suchlikes ( $\langle x \rangle_8$ ,  $\langle x \rangle_9$ ) writes like duodevicies, or like octodecies. For French it should be ‘below 20’. Possible values are ‘below 20’ and ‘never’.

```

349 \define@key{fcprefix}{use duode and unde}[below20]{%
350   \ifthenelse{\equal{#1}{below20}}{%
351     \def\fc@duodeandunde{2}%
352   }{%
353     \ifthenelse{\equal{#1}{never}}{%
354       \def\fc@duodeandunde{0}%
355     }{%
356       \PackageError{fcprefix}{Unexpected option}{%
357         Option ‘use duode and unde’ expects ‘below 20’ or ‘never’ }%
358     }%
359   }%
360 }

```

Default is ‘below 20’ like in French.

```

361 \def\fc@duodeandunde{2}

```

Option ‘numeral u in duo’, this can be ‘true’ or ‘false’ and is used to select whether 12 and suchlikes write like dodec<xxx> or duodec<xxx> for numerals.

```

362 \define@key{fcprefix}{numeral u in duo}[false]{%
363   \ifthenelse{\equal{#1}{false}}{%
364     \let\fc@u@in@duo\@empty
365   }{%
366     \ifthenelse{\equal{#1}{true}}{%
367       \def\fc@u@in@duo{u}%
368     }{%
369       \PackageError{fcprefix}{Unexpected option}{%
370         Option ‘numeral u in duo’ expects ‘true’ or ‘false’ }%
371     }%
372   }%
373 }

```

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.

```

374 \define@key{fcprefix}{e accute}[false]{%
375   \ifthenelse{\equal{#1}{false}}{%
376     \let\fc@prefix@eaccute\@firstofone
377   }{%
378     \ifthenelse{\equal{#1}{true}}{%
379       \let\fc@prefix@eaccute\'%
380     }{%
381       \PackageError{fcprefix}{Unexpected option}{%
382         Option ‘e accute’ expects ‘true’ or ‘false’ }%
383     }%
384   }%
385 }

```

Default is to set accute accent like in French.

```

386 \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^2’

prefix for which millia squared is noted as ‘bismillia’

```

387 \define@key{fcprefix}{power of millia}[prefix]{%
388   \ifthenelse{\equal{#1}{prefix}}{%
389     \let\fc@power@of@millia@init\@gobbletwo
390     \let\fc@power@of@millia\fc@@prefix@millia
391   }{%
392     \ifthenelse{\equal{#1}{arabic}}{%
393       \let\fc@power@of@millia@init\@gobbletwo
394       \let\fc@power@of@millia\fc@@arabic@millia
395     }{%
396       \ifthenelse{\equal{#1}{recursive}}{%
397         \let\fc@power@of@millia@init\fc@@recurse@millia@init
398         \let\fc@power@of@millia\fc@@recurse@millia
399       }{%

```

```

400     \PackageError{fcprefix}{Unexpected option}{%
401         Option 'power of millia' expects 'recursive', 'arabic', or 'prefix' }%
402     }%
403 }%
404 }%
405 }

```

Arguments as follows:

- #1 output macro
- #2 number with current weight  $w$

```

406 \def\fc@@recurse@millia#1#2{%
407     \let\@tempp#1%
408     \edef#1{millia\@tempp}%
409 }

```

Arguments as follows — same interface as `\fc@@recurse@millia`:

- #1 output macro
- #2 number with current weight  $w$

```

410 \def\fc@@recurse@millia@init#1#2{%
411     {%

```

Save input argument current weight  $w$  into local macro `\@tempb`.

```

412     \edef\@tempb{\number#2}%

```

Now main loop from 0 to  $w$ . Final value of `\@tempa` will be the result.

```

413     \count0=0 %
414     \let\@tempa\@empty
415     \loop
416         \ifnum\count0<\@tempb
417             \advance\count0 by 1 %
418             \expandafter\def
419                 \expandafter\@tempa\expandafter{\@tempa millia}%
420     \repeat

```

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

```

421     \edef\@tempb{\def\noexpand#1{\@tempa}}%
422     \expandafter
423 }\@tempb
424 }

```

Arguments as follows — same interface as `\fc@@recurse@millia`:

- #1 output macro
- #2 number with current weight  $w$

```

425 \def\fc@@arabic@millia#1#2{%
426     \ifnum#2=0 %
427         \let#1\@empty
428     \else
429         \edef#1{millia\^{}\the#2}%
430     \fi
431 }

```

Arguments as follows — same interface as `\fc@@recurse@millia`:

```
#1  output macro
#2  number with current weight  $w$ 
432 \def\fc@prefix@millia#1#2{%
433   \fc@@latin@numeral@pexif{#2}{#1}%
434 }
```

Default value of option ‘power of millia’ is ‘prefix’:

```
435 \let\fc@power@of@millia@init\@gobbletwo
436 \let\fc@power@of@millia\fc@prefix@millia
```

`\fc @@@latin@cardinal@pexif` Compute a cardinal prefix for n-illion, like  $1 \Rightarrow$  ‘m’,  $2 \Rightarrow$  ‘bi’,  $3 \Rightarrow$  ‘tri’. The algorithm to derive this prefix is that of Russ Rowlett I founds its documentation on Alain Lassine’s site: [http://www.alain.be/Boece/grands\\_nombres.html](http://www.alain.be/Boece/grands_nombres.html). First check that macro is not yet defined.

```
437 \@ifundefined{fc@@latin@cardinal@pexif}{%
438   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro ‘fc@@latin@cardinal@p
```

Arguments as follows:

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

```
439 \def\fc@@latin@cardinal@pexif#1#2{%
440   {%
```

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

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

Now parse number from expanded input.

```
442   \expandafter\fc@number@parser\expandafter{\@tempa}%
443   \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 `\@tempi`.

```
444   \let\@tempt\@empty
445   \def\@tempu{t}%
```

`\@tempm` will hold the millia <sup>$n \div 3$</sup>

```
446   \let\@tempm\@empty
```

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

```
447   \def\@templ{%
448     \ifnum\count2>\fc@max@weight
449       \let\next\relax
450     \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`.

```
451       \fc@read@unit{\count3}{\count2}%
452       \advance\count2 by 1 %
453       \fc@read@unit{\count4}{\count2}%
```

```

454      \advance\count2 by 1 %
455      \fc@read@unit{\count5}{\count2}%
456      \advance\count2 by 1 %

```

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

```

457      \edef\@tempn{%
458          \ifnum\count3=0\else 1\fi
459          \ifnum\count4=0\else 1\fi
460          \ifnum\count5=0\else 1\fi
461      }%
462      \ifx\@tempn\@empty\else
463          \let\@tempt\@tempu
464          \let\@tempu\@empty
465      \fi

```

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

```

466      \let\@tempp\@tempa
467      \edef\@tempa{%

```

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

```

468          \ifcase\count5 %
469          \or cen%
470          \or ducen%
471          \or trecen%
472          \or quadringen%
473          \or quingen%
474          \or sescen%
475          \or septigen%
476          \or octingen%
477          \or nongen%
478          \fi

```

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

```

479          \ifnum\count4=0 %
480          % x0(0..9)
481          \ifnum\count2=3 %
482          % Absolute weight zero
483          \ifcase\count3 \@tempt
484          \or m%
485          \or b%
486          \or tr%
487          \or quadr%
488          \or quin\@tempt
489          \or sex\@tempt
490          \or sep\@tempt
491          \or oc\@tempt
492          \or non%
493          \fi
494          \else

```

Here the weight of  $\backslash\text{count3}$  is  $3 \times n$ , with  $n > 0$ , i.e. this is followed by a  $\text{millia}^n$ .

```

495         \ifcase\count3 %
496         \or \ifnum\count2>\fc@max@weight\else un\fi
497         \or d\fc@u@in@duo o%
498         \or tre%
499         \or quattuor%
500         \or quin%
501         \or sex%
502         \or septen%
503         \or octo%
504         \or novem%
505         \fi
506     \fi
507 \else
508     % x(10..99)
509     \ifcase\count3 %
510     \or un%
511     \or d\fc@u@in@duo o%
512     \or tre%
513     \or quattuor%
514     \or quin%
515     \or sex%
516     \or septen%
517     \or octo%
518     \or novem%
519     \fi
520     \ifcase\count4 %
521     \or dec%
522     \or vigin\@tempt
523     \or trigin\@tempt
524     \or quadragin\@tempt
525     \or quinquagin\@tempt
526     \or sexagin\@tempt
527     \or septuagin\@tempt
528     \or octogin\@tempt
529     \or nonagin\@tempt
530     \fi
531 \fi

```

Insert the  $\text{millia}^{(n \div 3)}$  only if  $d_2 d_1 d_0 \neq 0$ , i.e. if one of  $\backslash\text{count3}$   $\backslash\text{count4}$  or  $\backslash\text{count5}$  is non zero.

```

532     \@tempm

```

And append previous version of  $\backslash\text{tempa}$ .

```

533     \@tempp
534 }%

```

“Concatenate”  $\text{millia}$  to  $\backslash\text{tempm}$ , so that  $\backslash\text{tempm}$  will expand to  $\text{millia}^{(n \div 3)+1}$  at the next iteration. Actually whether this is a concatenation or some  $\text{millia}$

prefixing depends of option ‘power of millia’.

```

535      \fc@power@of@millia \@tempm{\count2}%
536      \fi
537      \next
538    }%
539    \let \@tempa \@empty
540    \let \next \@templ
541    \@templ

```

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

```

542    \def \@tempb##1{\def \@tempa{\def #2{##1}}}%
543    \expandafter \@tempb \expandafter {\@tempa}%
544    \expandafter
545  }\@tempa
546 }

```

\fc @@latin@numeral@pexif 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.

```

547 \ifundefined{fc@@latin@numeral@pexif}{%
548   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
549     ‘fc@@latin@numeral@pexif’}%

```

Arguments as follows:

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

```

550 \def \fc@@latin@numeral@pexif#1#2{%
551   {%
552     \edef \@tempa {\number #1}%
553     \def \fc@unit@weight{0}%
554     \expandafter \fc@number@parser \expandafter {\@tempa}%
555     \count2=0 %

```

Macro \@tempm will hold the millies<sup>n÷3</sup>.

```

556   \let \@tempm \@empty

```

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

```

557   \def \@templ{%
558     \ifnum \count2 > \fc@max@weight
559       \let \next \relax
560     \else

```

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

```

561       \fc@read@unit{\count3}{\count2}%
562       \advance \count2 by 1 %
563       \fc@read@unit{\count4}{\count2}%
564       \advance \count2 by 1 %
565       \fc@read@unit{\count5}{\count2}%

```



```

566      \advance\count2 by 1 %
Check the use of duodevicies instead of octodecies.
567      \let\@tempn\@secondoftwo
568      \ifnum\count3>7 %
569          \ifnum\count4<\fc@duodeandunde
570              \ifnum\count4>0 %
571                  \let\@tempn\@firstoftwo
572              \fi
573          \fi
574      \fi
575      \@tempn
576      {% use duodevicies for eighteen
577          \advance\count4 by 1 %
578          \let\@temps\@secondoftwo
579      }{% do not use duodevicies for eighteen
580          \let\@temps\@firstoftwo
581      }%
582      \let\@tempp\@tempa
583      \edef\@tempa{%
584          % hundreds
585          \ifcase\count5 %
586              \expandafter\@gobble
587              \or c%
588              \or duc%
589              \or trec%
590              \or quadring%
591              \or quing%
592              \or sesc%
593              \or septing%
594              \or octing%
595              \or nong%
596          \fi
597          {enties}%
598          \ifnum\count4=0 %

```

Here  $d_2 d_1 d_0$  is such that  $d_1 = 0$ .

```

599      \ifcase\count3 %
600      \or
601          \ifnum\count2=3 %
602              s\fc@prefix@eacute emel%
603          \else
604              \ifnum\count2>\fc@max@weight\else un\fi
605          \fi
606      \or bis%
607      \or ter%
608      \or quater%
609      \or quinquies%
610      \or sexies%
611      \or septies%

```

```

612          \or octies%
613          \or novies%
614          \fi
615      \else
    Here  $d_2 d_1 d_0$  is such that  $d_1 \geq 1$ .
616          \ifcase\count3 %
617          \or un%
618          \or d\fc@u@in@duo o%
619          \or ter%
620          \or quater%
621          \or quin%
622          \or sex%
623          \or septen%
624          \or \@temps{octo}{duod\fc@prefix@eacute e}% x8 = two before next (x+1)0
625          \or \@temps{novem}{und\fc@prefix@eacute e}% x9 = one before next (x+1)0
626          \fi
627          \ifcase\count4 %
628          % can't get here
629          \or d\fc@prefix@eacute ec%
630          \or vic%
631          \or tric%
632          \or quadrag%
633          \or quinquag%
634          \or sexag%
635          \or septuag%
636          \or octog%
637          \or nonag%
638          \fi
639          ies%
640      \fi
641      % Insert the millies(n/3) only if one of \count3 \count4 \count5 is non zero
642      \@tempm
643      % add up previous version of \@tempa
644      \@tempp
645  }%

```

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

```

646      \let\@tempm\@tempp
647      \edef\@tempm{millies\@tempp}%
648      \fi
649      \next
650  }%
651  \let\@tempa\@empty
652  \let\@next\@templ
653  \@templ

```

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

```

654 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
655 \expandafter\@tempb\expandafter{\@tempa}%
656 \expandafter
657 }\@tempa
658 }

```

Stuff for calling macros. Construct `\fc@call⟨some macro⟩` can be used to pass two arguments to `⟨some macro⟩` with a configurable calling convention:

- the calling convention is such that there is one mandatory argument `⟨marg⟩` and an optional argument `⟨oarg⟩`
- either `\fc@call` is `\let` to be equal to `\fc@call@opt@arg@second`, and then calling convention is that the `⟨marg⟩` is first and `⟨oarg⟩` is second,
- or `\fc@call` is `\let` to be equal to `\fc@call@opt@arg@first`, and then calling convention is that the `⟨oarg⟩` is first and `⟨aarg⟩` is second,
- if `⟨oarg⟩` is absent, then it is by convention set empty,
- `⟨some macro⟩` is supposed to have two mandatory arguments of which `⟨oarg⟩` is passed to the first, and `⟨marg⟩` is passed to the second, and
- `⟨some macro⟩` is called within a group.

```

659 \def\fc@call@opt@arg@second#1#2{%
660 \def\@tempb{%
661 \ifx[\@tempa
662 \def\@tempc[####1]{%
663 {#1{####1}{#2}}%
664 }%
665 \else
666 \def\@tempc{{#1}{#2}}}%
667 \fi
668 \@tempc
669 }%
670 \futurelet\@tempa
671 \@tempb
672 }

673 \def\fc@call@opt@arg@first#1{%
674 \def\@tempb{%
675 \ifx[\@tempa
676 \def\@tempc[####1]####2{{#1{####1}{####2}}}%
677 \else
678 \def\@tempc####1{{#1}{####1}}}%
679 \fi
680 \@tempc
681 }%
682 \futurelet\@tempa
683 \@tempb

```

```

684 }
685
686 \let\fc@call\fc@call@opt@arg@first

```

User API.

\@ latinnumeralstringnum Macro \@latinnumeralstringnum. Arguments as follows:

- #1 local options
- #2 input number

```

687 \newcommand*{\@latinnumeralstringnum}[2]{%
688   \setkeys{fcprefix}{#1}%
689   \fc@@latin@numeral@pexif{#2}\@tempa
690   \@tempa
691 }

```

Arguments as follows:

- #1 local options
- #2 input counter

```

692 \newcommand*{\@latinnumeralstring}[2]{%
693   \setkeys{fcprefix}{#1}%
694   \expandafter\let\expandafter
695     \@tempa\expandafter\csname c@#2\endcsname
696   \expandafter\fc@@latin@numeral@pexif\expandafter{\the\@tempa}\@tempa
697   \@tempa
698 }

699 \newcommand*{\latinnumeralstring}{%
700   \fc@call\@latinnumeralstring
701 }

702 \newcommand*{\latinnumeralstringnum}{%
703   \fc@call\@latinnumeralstringnum
704 }

```

### 9.3 fmtcount.sty

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

```

705 \NeedsTeXFormat{LaTeX2e}
706 \ProvidesPackage{fmtcount}[2013/08/17 v2.03]
707 \RequirePackage{ifthen}
708 \RequirePackage{keyval}
709 \RequirePackage{etoolbox}
710 \RequirePackage{fcprefix}

```

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

```

711 \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.

`\fmtord`

```
712 \providecommand*{\fmtord}[1]{\textsuperscript{#1}}
```

`\padzeroes`

`\padzeroes[n]`

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

```
713 \newcount\c@padzeroesN
```

```
714 \c@padzeroesN=1\relax
```

```
715 \providecommand*{\padzeroes}[1][17]{\c@padzeroesN=#1}
```

`\FCloadlang` changes2.02012-06-18new changes2.022012-10-24ensured catcode for @ set to 'letter' before loading file

`\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.

```
716 \newcount\fc@tmpcatcode
```

```
717 \def\fc@languages{}%
```

```
718 \def\fc@mainlang{}%
```

```
719 \newcommand*{\FCloadlang}[1]{%
```

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

```
721   {%
```

```
722     \fc@tmpcatcode=\catcode'\@ \relax
```

```
723     \catcode '\@ 11\relax
```

```
724     \InputIfFileExists{fc-#1.def}%
```

```
725     {%
```

```
726       \ifdefempty{\fc@languages}%
```

```
727       {%
```

```
728         \gdef\fc@languages{#1}%
```

```
729       }%
```

```
730       {%
```

```
731         \gappto\fc@languages{, #1}%
```

```
732       }%
```

```
733       \gdef\fc@mainlang{#1}%
```

```
734     }%
```

```
735   }%
```

```
736   \catcode '\@ \fc@tmpcatcode \relax
```

```
737 }%
```

```
738 }
```

`\@FC@iflangloaded` changes2.02012-06-18new

```
\@FC@iflangloaded{<language>}{<true>}{<false>}
```

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

```
739 \newcommand{\@FC@iflangloaded}[3]{%
740   \ifcsundef{ver@fc-#1.def}{#3}{#2}%
741 }
```

`\ProvidesFCLanguage` changes2.02012-06-18new Declare fmtcount language definition file. Adapted from `\ProvidesFile`.

```
742 \newcommand*{\ProvidesFCLanguage}[1]{%
743   \ProvidesFile{fc-#1.def}%
744 }
```

`babelorpolyglossialdf` `\@fc@loadigbabelldf{<language>}`

Loads fmtcount language file, `fc-<language>.def`, if babel language definition file `<language>.ldf` has been loaded.

```
745 \newcommand*{\@fc@loadifbabelorpolyglossialdf}[1]{%
746   \ifcsundef{ver@#1.ldf}{\FCloadlang{#1}}%
747   \IfFileExists{gloss-#1.ldf}{\ifcsundef{#1@loaded}{\FCloadlang{#1}}}%
748 }
```

Load appropriate language definition files:

```
749 \@fc@loadifbabelorpolyglossialdf{english}
750 \@fc@loadifbabelorpolyglossialdf{UKenglish}
751 \@fc@loadifbabelorpolyglossialdf{british}
752 \@fc@loadifbabelorpolyglossialdf{USenglish}
753 \@fc@loadifbabelorpolyglossialdf{american}
754 \@fc@loadifbabelorpolyglossialdf{spanish}
755 \@fc@loadifbabelorpolyglossialdf{portuges}
756 \@fc@loadifbabelorpolyglossialdf{french}
757 \@fc@loadifbabelorpolyglossialdf{frenchb}
758 \@fc@loadifbabelorpolyglossialdf{francais}
759 \@fc@loadifbabelorpolyglossialdf{german}%
760 \@fc@loadifbabelorpolyglossialdf{germanb}%
761 \@fc@loadifbabelorpolyglossialdf{ngerman}%
762 \@fc@loadifbabelorpolyglossialdf{ngermanb}%
763 \@fc@loadifbabelorpolyglossialdf{italian}
```

`\fmtcount@french` Define keys for use with `\fmtcountsetoptions`. Key to switch French dialects (Does babel store this kind of information?)

```
764 \def\fmtcount@french{france}
```

`french`

```
765 \define@key{fmtcount}{french}[france]{%
```

```

766 \@ifundefined{datefrench}%
767 {%
768   \PackageError{fmtcount}%
769   {Language ‘french’ not defined}%
770   {You need to load babel before loading fmtcount}%
771 }%
772 {%
773   \setkeys{french}{#1}%
774 }%
775 }

```

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

```

776 \define@key{fmtcount}{fmtord}{%
777   \ifthenelse{\equal{#1}{level}
778             \or\equal{#1}{raise}
779             \or\equal{#1}{user}}%
780   {%
781     \def\fmtcount@fmtord{#1}%
782   }%
783   {%
784     \PackageError{fmtcount}%
785     {Invalid value ‘#1’ to fmtord key}%
786     {Option ‘fmtord’ can only take the values ‘level’, ‘raise’
787      or ‘user’}%
788   }%
789 }

```

**\iffmtord@abbrv** Key to determine whether the ordinal should be abbreviated (language dependent, currently only affects French ordinals.)

```

790 \newif\iffmtord@abbrv
791 \fmtord@abbrvfalse
792 \define@key{fmtcount}{abbrv}[true]{%
793   \ifthenelse{\equal{#1}{true}\or\equal{#1}{false}}%
794   {%
795     \csname fmtord@abbrv#1\endcsname
796   }%
797   {%
798     \PackageError{fmtcount}%
799     {Invalid value ‘#1’ to fmtord key}%
800     {Option ‘fmtord’ can only take the values ‘true’ or
801      ‘false’}%
802   }%
803 }

```

**prefix**

```

804 \define@key{fmtcount}{prefix}[scale=long]{%
805   \RequirePackage{fmtprefix}%
806   \fmtprefixsetoption{#1}%
807 }

```

`\fmtcountsetoptions` Define command to set options.

```
808 \newcommand*{\fmtcountsetoptions}[1]{%
809   \def\fmtcount@fmtord{}%
810   \setkeys{fmtcount}{#1}%
811   \ifundefined{datefrench}{}%
812   {%
813     \edef\@OrdinalstringMfrench{\noexpand
814       \csname @OrdinalstringMfrench\fmtcount@french\noexpand\endcsname}%
815     \edef\@OrdinalstringFfrench{\noexpand
816       \csname @OrdinalstringFfrench\fmtcount@french\noexpand\endcsname}%
817     \edef\@OrdinalstringMfrench{\noexpand
818       \csname @OrdinalstringMfrench\fmtcount@french\noexpand\endcsname}%
819     \edef\@OrdinalstringFfrench{\noexpand
820       \csname @OrdinalstringFfrench\fmtcount@french\noexpand\endcsname}%
821     \edef\@NumberstringMfrench{\noexpand
822       \csname @NumberstringMfrench\fmtcount@french\noexpand\endcsname}%
823     \edef\@NumberstringFfrench{\noexpand
824       \csname @NumberstringFfrench\fmtcount@french\noexpand\endcsname}%
825     \edef\@NumberstringMfrench{\noexpand
826       \csname @NumberstringMfrench\fmtcount@french\noexpand\endcsname}%
827     \edef\@NumberstringFfrench{\noexpand
828       \csname @NumberstringFfrench\fmtcount@french\noexpand\endcsname}%
829   }%
830   \ifthenelse{\equal{\fmtcount@fmtord}{level}}{%
831     {%
832       \renewcommand{\fmtord}[1]{##1}%
833     }%
834     {%
835       \ifthenelse{\equal{\fmtcount@fmtord}{raise}}{%
836         {%
837           \renewcommand{\fmtord}[1]{\textsuperscript{##1}}%
838         }%
839         {%
840         }%
841       }
842 }
```

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.

```
843 \InputIfFileExists{fmtcount.cfg}%
844 {%
845   \PackageInfo{fmtcount}{Using configuration file fmtcount.cfg}%
846 }%
847 {%
848 }
```

level

```
849 \DeclareOption{level}{\def\fmtcount@fmtord{level}%
850   \def\fmtord#1{##1}}
```



raise

```
851 \DeclareOption{raise}{\def\fmtcount@fmtord{raise}%
852 \def\fmtord#1{\textsuperscript{#1}}}
```

Process package options

```
853 \ProcessOptions
```

\@modulo     \@modulo{<count reg>}{<n>}

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

```
854 \newcount\@DT@modctr
855 \def\@modulo#1#2{%
856 \@DT@modctr=#1\relax
857 \divide \@DT@modctr by #2\relax
858 \multiply \@DT@modctr by #2\relax
859 \advance #1 by -\@DT@modctr
860 }
```

The following registers are needed by \@ordinal etc

```
861 \newcount\@ordinalctr
862 \newcount\@orgargctr
863 \newcount\@strctr
864 \newcount\@tmpstrctr
```

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

```
865 \newif\if@DT@padzeroes
866 \newcount\@DT@loopN
867 \newcount\@DT@X
```

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

```
868 \newcommand*{\@binary}[1]{%
869 \@DT@padzeroestrue
870 \@DT@loopN=17\relax
871 \@strctr=\@DT@loopN
872 \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
873 \@strctr=65536\relax
874 \@DT@X=#1\relax
875 \loop
876 \@DT@modctr=\@DT@X
877 \divide\@DT@modctr by \@strctr
878 \ifthenelse{\boolean{@DT@padzeroes}
879 \and \(\@DT@modctr=0\)}
880 \and \(\@DT@loopN>\c@padzeroesN\)}%
881 {}%
```

```

882     {\the\@DT@modctr}%
883     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
884     \multiply\@DT@modctr by \@strctr
885     \advance\@DT@X by -\@DT@modctr
886     \divide\@strctr by 2\relax
887     \advance\@DT@loopN by -1\relax
888     \ifnum\@strctr>1
889     \repeat
890     \the\@DT@X
891 }
892
893 \let\binarynum=\@binary

```

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

```

894 \newcommand*{\@octal}[1]{%
895   \ifnum#1>32768
896     \PackageError{fmtcount}%
897     {Value of counter too large for \protect\@octal}
898     {Maximum value 32768}
899   \else
900     \@DT@padzeroestrue
901     \@DT@loopN=6\relax
902     \@strctr=\@DT@loopN
903     \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
904     \@strctr=32768\relax
905     \@DT@X=#1\relax
906     \loop
907       \@DT@modctr=\@DT@X
908       \divide\@DT@modctr by \@strctr
909       \ifthenelse{\boolean{\@DT@padzeroes}
910         \and \(\@DT@modctr=0\)}
911         \and \(\@DT@loopN>\c@padzeroesN\)}%
912       {\the\@DT@modctr}%
913       \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
914       \multiply\@DT@modctr by \@strctr
915       \advance\@DT@X by -\@DT@modctr
916       \divide\@strctr by 8\relax
917       \advance\@DT@loopN by -1\relax
918     \ifnum\@strctr>1
919     \repeat
920     \the\@DT@X
921   \fi
922 }
923 \let\octalnum=\@octal

```

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

```

924 \newcommand*{\@@hexadecimal}[1]{%
925   \ifcase#10\or1\or2\or3\or4\or5\or
926   6\or7\or8\or9\or a\or b\or c\or d\or e\or f\fi

```

927 }

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

```
928 \newcommand*{\@hexadecimal}[1]{%
929   \@DT@padzeroestru
930   \@DT@loopN=5\relax
931   \@strctr=\@DT@loopN
932   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
933   \@strctr=65536\relax
934   \@DT@X=#1\relax
935   \loop
936     \@DT@modctr=\@DT@X
937     \divide\@DT@modctr by \@strctr
938     \ifthenelse{\boolean{\@DT@padzeroes}
939       \and \(\@DT@modctr=0\)}
940       \and \(\@DT@loopN>\c@padzeroesN\)}
941     {\@hexadecimal\@DT@modctr}%
942     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
943     \multiply\@DT@modctr by \@strctr
944     \advance\@DT@X by -\@DT@modctr
945     \divide\@strctr by 16\relax
946     \advance\@DT@loopN by -1\relax
947   \ifnum\@strctr>1
948     \repeat
949   \@hexadecimal\@DT@X
950 }
951 \let\hexadecimalnum=\@hexadecimal
```

**\@@Hexadecimalnum** Converts number from 0 to 15 into uppercase hexadecimal notation.

```
952 \newcommand*{\@@Hexadecimal}[1]{%
953   \ifcase#1\or 1\or 2\or 3\or 4\or 5\or 6\or
954   7\or 8\or 9\or A\or B\or C\or D\or E\or F\fi
955 }
```

**\Hexadecimalnum** Uppercase hexadecimal

```
956 \newcommand*{\@Hexadecimal}[1]{%
957   \@DT@padzeroestru
958   \@DT@loopN=5\relax
959   \@strctr=\@DT@loopN
960   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
961   \@strctr=65536\relax
962   \@DT@X=#1\relax
963   \loop
964     \@DT@modctr=\@DT@X
965     \divide\@DT@modctr by \@strctr
966     \ifthenelse{\boolean{\@DT@padzeroes}
967       \and \(\@DT@modctr=0\)}
968     \and \(\@DT@loopN>\c@padzeroesN\)}%
```

```

969    {}{\@@Hexadecimal\@DT@modctr}%
970    \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
971    \multiply\@DT@modctr by \@strctr
972    \advance\@DT@X by -\@DT@modctr
973    \divide\@strctr by 16\relax
974    \advance\@DT@loopN by -1\relax
975    \ifnum\@strctr>1
976    \repeat
977    \@@Hexadecimal\@DT@X
978 }
979
980 \let\Hexadecimalnum=\@Hexadecimal

```

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

```

981 \newcommand*\@aaalph}[1]{%
982   \@DT@loopN=#1\relax
983   \advance\@DT@loopN by -1\relax
984   \divide\@DT@loopN by 26\relax
985   \@DT@modctr=\@DT@loopN
986   \multiply\@DT@modctr by 26\relax
987   \@DT@X=#1\relax
988   \advance\@DT@X by -1\relax
989   \advance\@DT@X by -\@DT@modctr
990   \advance\@DT@loopN by 1\relax
991   \advance\@DT@X by 1\relax
992   \loop
993     \@alph\@DT@X
994     \advance\@DT@loopN by -1\relax
995   \ifnum\@DT@loopN>0
996   \repeat
997 }
998
999 \let\aaalphnum=\@aaalph

```

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

```

1000 \newcommand*\@AAAalph}[1]{%
1001   \@DT@loopN=#1\relax
1002   \advance\@DT@loopN by -1\relax
1003   \divide\@DT@loopN by 26\relax
1004   \@DT@modctr=\@DT@loopN
1005   \multiply\@DT@modctr by 26\relax
1006   \@DT@X=#1\relax
1007   \advance\@DT@X by -1\relax
1008   \advance\@DT@X by -\@DT@modctr
1009   \advance\@DT@loopN by 1\relax
1010   \advance\@DT@X by 1\relax
1011   \loop
1012     \@Alph\@DT@X
1013   \advance\@DT@loopN by -1\relax

```

```

1014 \ifnum\@DT@loopN>0
1015 \repeat
1016 }
1017
1018 \let\AAAAlphnum=\@AAAAlph

```

#### \abalphnum Lowercase alphabetical representation

```

1019 \newcommand*\@abalph}[1]{%
1020 \ifnum#1>17576\relax
1021 \PackageError{fmtcount}%
1022 {Value of counter too large for \protect\@abalph}%
1023 {Maximum value 17576}%
1024 \else
1025 \@DT@padzeroestru
1026 \@strctr=17576\relax
1027 \@DT@X=#1\relax
1028 \advance\@DT@X by -1\relax
1029 \loop
1030 \@DT@modctr=\@DT@X
1031 \divide\@DT@modctr by \@strctr
1032 \ifthenelse{\boolean{\@DT@padzeroes}
1033 \and \(\@DT@modctr=1\)}%
1034 {\@alph\@DT@modctr}%
1035 \ifnum\@DT@modctr=1\else\@DT@padzeroesfalse\fi
1036 \multiply\@DT@modctr by \@strctr
1037 \advance\@DT@X by -\@DT@modctr
1038 \divide\@strctr by 26\relax
1039 \ifnum\@strctr>1
1040 \repeat
1041 \advance\@DT@X by 1\relax
1042 \@alph\@DT@X
1043 \fi
1044 }
1045
1046 \let\abalphnum=\@abalph

```

#### \ABAlphnum Uppercase alphabetical representation

```

1047 \newcommand*\@ABAlph}[1]{%
1048 \ifnum#1>17576\relax
1049 \PackageError{fmtcount}%
1050 {Value of counter too large for \protect\@ABAlph}%
1051 {Maximum value 17576}%
1052 \else
1053 \@DT@padzeroestru
1054 \@strctr=17576\relax
1055 \@DT@X=#1\relax
1056 \advance\@DT@X by -1\relax
1057 \loop
1058 \@DT@modctr=\@DT@X

```

```

1059      \divide\@DT@modctr by \@strctr
1060      \ifthenelse{\boolean{@DT@padzeroes}}\and
1061      \(\@DT@modctr=1\)}{\@Alph\@DT@modctr}%
1062      \ifnum\@DT@modctr=1\else\@DT@padzeroesfalse\fi
1063      \multiply\@DT@modctr by \@strctr
1064      \advance\@DT@X by -\@DT@modctr
1065      \divide\@strctr by 26\relax
1066      \ifnum\@strctr>1
1067      \repeat
1068      \advance\@DT@X by 1\relax
1069      \@Alph\@DT@X
1070    \fi
1071  }
1072
1073 \let\ABAlphnum=\@ABAlph

```

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

```

1074 \def\@fmtc@count#1#2\relax{%
1075   \if\relax#1%
1076   \else
1077     \advance\@strctr by 1\relax
1078     \@fmtc@count#2\relax
1079   \fi
1080 }

```

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

```

1081 \newcommand{\@decimal}[1]{%
1082   \@strctr=0\relax
1083   \expandafter\@fmtc@count\number#1\relax
1084   \@DT@loopN=\c@padzeroesN
1085   \advance\@DT@loopN by -\@strctr
1086   \ifnum\@DT@loopN>0\relax
1087     \@strctr=0\relax
1088     \whiledo{\@strctr < \@DT@loopN}{\advance\@strctr by 1\relax}%
1089   \fi
1090   \number#1\relax
1091 }
1092
1093 \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.

```
1094 \newcommand{\FCordinal}[1]{%
1095   \expandafter\protect\expandafter\ordinalnum{%
1096     \expandafter\the\csname c@#1\endcsname}%
1097 }
```

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

```
1098 \@ifundefined{ordinal}
1099 {\let\ordinal\FCordinal}%
1100 {%
1101   \PackageWarning{fmtcount}%
1102   {\string\ordinal \space already defined use
1103    \string\FCordinal \space instead.}
1104 }
```

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

```
1105 \newcommand*{\ordinalnum}[1]{%
1106   \new@ifnextchar[%
1107     {\@ordinalnum{#1}}%
1108     {\@ordinalnum{#1}[m]}}%
1109 }
```

`\@ordinalnum` Display ordinal according to gender (neuter added in v1.1, `\xspace` added in v1.2, and removed in v1.3<sup>6</sup>):

```
1110 \def\@ordinalnum#1[#2]{%
1111   {%
1112     \ifthenelse{\equal{#2}{f}}%
1113     {%
1114       \protect\@ordinalF{#1}{\@fc@ordstr}%
1115     }%
1116     {%
1117       \ifthenelse{\equal{#2}{n}}%
1118       {%
1119         \protect\@ordinalN{#1}{\@fc@ordstr}%
1120       }%
1121       {%
1122         \ifthenelse{\equal{#2}{m}}%
1123         {}%
1124         {%
```

---

<sup>6</sup>I couldn't get it to work consistently both with and without the optional argument

```

1125         \PackageError{fmtcount}%
1126         {Invalid gender option ‘#2’}%
1127         {Available options are m, f or n}%
1128     }%
1129     \protect\@ordinalM{#1}{\@fc@ordstr}%
1130 }%
1131 }%
1132 \@fc@ordstr
1133 }%
1134 }

```

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

```

1135 \newcommand*{\storeordinal}[2]{%
1136   \expandafter\protect\expandafter\storeordinalnum{#1}{%
1137     \expandafter\the\csname c@#2\endcsname}%
1138 }

```

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

```

1139 \newcommand*{\storeordinalnum}[2]{%
1140   \ifnextchar[%
1141     {\@storeordinalnum{#1}{#2}}%
1142     {\@storeordinalnum{#1}{#2}[m]}%
1143 }

```

`\@storeordinalnum` Store ordinal according to gender:

```

1144 \def\@storeordinalnum#1#2[#3]{%
1145   \ifthenelse{\equal{#3}{f}}{%
1146     {%
1147       \protect\@ordinalF{#2}{\@fc@ord}
1148     }%
1149     {%
1150       \ifthenelse{\equal{#3}{n}}{%
1151         {%
1152           \protect\@ordinalN{#2}{\@fc@ord}%
1153         }%
1154         {%
1155           \ifthenelse{\equal{#3}{m}}{%
1156             {}%
1157             {%
1158               \PackageError{fmtcount}%
1159               {Invalid gender option ‘#3’}%
1160               {Available options are m or f}%
1161             }%
1162             \protect\@ordinalM{#2}{\@fc@ord}%
1163           }%
1164         }%
1165       \expandafter\let\csname @fcs@#1\endcsname\@fc@ord

```



1166 }

\FMCuse Get stored information:

1167 \newcommand\*{\FMCuse}[1]{\csname @fcs@#1\endcsname}

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

```
1168 \newcommand*{\ordinalstring}[1]{%
1169   \expandafter\protect\expandafter\ordinalstringnum{%
1170     \expandafter\the\csname c@#1\endcsname}%
1171 }
```

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

```
1172 \newcommand{\ordinalstringnum}[1]{%
1173   \new@ifnextchar[%
1174     {\@ordinal@string{#1}}%
1175     {\@ordinal@string{#1}[m]}%
1176 }
```

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

```
1177 \def\@ordinal@string#1[#2]{%
1178   {%
1179     \ifthenelse{\equal{#2}{f}}%
1180     {%
1181       \protect\@ordinalstringF{#1}{\@fc@ordstr}%
1182     }%
1183     {%
1184       \ifthenelse{\equal{#2}{n}}%
1185       {%
1186         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
1187       }%
1188       {%
1189         \ifthenelse{\equal{#2}{m}}%
1190         {}%
1191         {%
1192           \PackageError{fmtcount}%
1193             {Invalid gender option ‘#2’ to \string\ordinalstring}%
1194             {Available options are m, f or f}%
1195         }%
1196         \protect\@ordinalstringM{#1}{\@fc@ordstr}%
1197       }%
1198     }%
1199     \@fc@ordstr
1200   }%
1201 }
```

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

1202 \newcommand\*{\storeordinalstring}[2]{%

```

1203 \expandafter\protect\expandafter\storeordinalstringnum{#1}{%
1204 \expandafter\the\csname c@#2\endcsname}%
1205 }

```

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

```

1206 \newcommand*{\storeordinalstringnum}[2]{%
1207 \@ifnextchar[%
1208 {\@store@ordinal@string{#1}{#2}}%
1209 {\@store@ordinal@string{#1}{#2}[m]}%
1210 }

```

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

```

1211 \def\@store@ordinal@string#1#2[#3]{%
1212 \ifthenelse{\equal{#3}{f}}{%
1213 {%
1214 \protect\@ordinalstringF{#2}{\@fc@ordstr}%
1215 }%
1216 {%
1217 \ifthenelse{\equal{#3}{n}}{%
1218 {%
1219 \protect\@ordinalstringN{#2}{\@fc@ordstr}%
1220 }%
1221 {%
1222 \ifthenelse{\equal{#3}{m}}{%
1223 {}%
1224 {%
1225 \PackageError{fmtcount}%
1226 {Invalid gender option ‘#3’ to \string\ordinalstring}%
1227 {Available options are m, f or n}%
1228 }%
1229 \protect\@ordinalstringM{#2}{\@fc@ordstr}%
1230 }%
1231 }%
1232 \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
1233 }

```

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

```

1234 \newcommand*{\Ordinalstring}[1]{%
1235 \expandafter\protect\expandafter\Ordinalstringnum{%
1236 \expandafter\the\csname c@#1\endcsname}%
1237 }

```

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

```

1238 \newcommand*{\Ordinalstringnum}[1]{%
1239 \new@ifnextchar[%

```

```

1240 {\@Ordinal@string{#1}}%
1241 {\@Ordinal@string{#1}[m]}%
1242 }

```

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

```

1243 \def\@Ordinal@string#1[#2]{%
1244   {%
1245     \ifthenelse{\equal{#2}{f}}{%
1246       {%
1247         \protect\@OrdinalstringF{#1}{\@fc@ordstr}%
1248       }%
1249     }%
1250     \ifthenelse{\equal{#2}{n}}{%
1251       {%
1252         \protect\@OrdinalstringN{#1}{\@fc@ordstr}%
1253       }%
1254     }%
1255     \ifthenelse{\equal{#2}{m}}{%
1256       {}%
1257     }%
1258     \PackageError{fmtcount}%
1259       {Invalid gender option ‘#2’}%
1260       {Available options are m, f or n}%
1261   }%
1262   \protect\@OrdinalstringM{#1}{\@fc@ordstr}%
1263 }%
1264 }%
1265 \@fc@ordstr
1266 }%
1267 }

```

`\storeOrdinalstring` 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.

```

1268 \newcommand*\storeOrdinalstring}[2]{%
1269   \expandafter\protect\expandafter\storeOrdinalstringnum{#1}{%
1270     \expandafter\the\csname c@#2\endcsname}%
1271 }

```

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

```

1272 \newcommand*\storeOrdinalstringnum}[2]{%
1273   \@ifnextchar[%
1274     {\@store@Ordinal@string{#1}{#2}}%
1275     {\@store@Ordinal@string{#1}{#2}[m]}%
1276 }

```

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

```

1277 \def\@store@Ordinal@string#1#2[#3]{%
1278   \ifthenelse{\equal{#3}{f}}{%
1279     {%
1280       \protect\@OrdinalstringF{#2}{\@fc@ordstr}%
1281     }%
1282     {%
1283       \ifthenelse{\equal{#3}{n}}{%
1284         {%
1285           \protect\@OrdinalstringN{#2}{\@fc@ordstr}%
1286         }%
1287         {%
1288           \ifthenelse{\equal{#3}{m}}{%
1289             {}%
1290             {%
1291               \PackageError{fmtcount}%
1292                 {Invalid gender option ‘#3’}%
1293                 {Available options are m or f}%
1294             }%
1295           \protect\@OrdinalstringM{#2}{\@fc@ordstr}%
1296         }%
1297       }%
1298     \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
1299 }

```

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

```

1300 \newcommand*\@storeORDINALstring}[2]{%
1301   \expandafter\protect\expandafter\storeORDINALstringnum{#1}{%
1302     \expandafter\the\csname c@#2\endcsname}%
1303 }

```

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

```

1304 \newcommand*\@storeORDINALstringnum}[2]{%
1305   \@ifnextchar[%
1306     {\@store@ORDINAL@string{#1}{#2}}%
1307     {\@store@ORDINAL@string{#1}{#2}[m]}%
1308 }

```

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

```

1309 \def\@store@ORDINAL@string#1#2[#3]{%
1310   \ifthenelse{\equal{#3}{f}}{%
1311     {%
1312       \protect\@ordinalstringF{#2}{\@fc@ordstr}%
1313     }%
1314     {%
1315       \ifthenelse{\equal{#3}{n}}{%
1316         {%
1317           \protect\@ordinalstringN{#2}{\@fc@ordstr}%
1318         }%

```

```

1319   {%
1320     \ifthenelse{\equal{#3}{m}}{%
1321       {}%
1322     {%
1323       \PackageError{fmtcount}%
1324         {Invalid gender option ‘#3’}%
1325         {Available options are m or f}%
1326     }%
1327     \protect\@ordinalstringM{#2}{\@fc@ordstr}%
1328   }%
1329 }%
1330 \expandafter\edef\csname @fcs@#1\endcsname{%
1331   \noexpand\MakeUppercase{\@fc@ordstr}%
1332 }%
1333 }

```

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

```

1334 \newcommand*{\ORDINALstring}[1]{%
1335   \expandafter\protect\expandafter\ORDINALstringnum{%
1336     \expandafter\the\csname c@#1\endcsname
1337   }%
1338 }

```

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

```

1339 \newcommand*{\ORDINALstringnum}[1]{%
1340   \new@ifnextchar[%
1341     {\@ORDINAL@string{#1}}%
1342     {\@ORDINAL@string{#1}[m]}%
1343 }

```

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

```

1344 \def\@ORDINAL@string#1[#2]{%
1345   {%
1346     \ifthenelse{\equal{#2}{f}}{%
1347       {%
1348         \protect\@ordinalstringF{#1}{\@fc@ordstr}%
1349       }%
1350     {%
1351       \ifthenelse{\equal{#2}{n}}{%
1352         {%
1353           \protect\@ordinalstringN{#1}{\@fc@ordstr}%
1354         }%
1355       {%
1356         \ifthenelse{\equal{#2}{m}}{%
1357           {}%
1358         {%
1359           \PackageError{fmtcount}%
1360             {Invalid gender option ‘#2’}%

```

```

1361         {Available options are m, f or n}%
1362     }%
1363     \protect\@ordinalstringM{#1}{\@fc@ordstr}%
1364 }%
1365 }%
1366     \MakeUppercase{\@fc@ordstr}%
1367 }%
1368 }

```

`\storenumberstring` Convert number to textual representation, and store. First argument is the identifying name, second argument is a counter containing the number.

```

1369 \newcommand*\storenumberstring[2]{%
1370     \expandafter\protect\expandafter\storenumberstringnum{#1}{%
1371         \expandafter\the\csname c@#2\endcsname}%
1372 }

```

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

```

1373 \newcommand*\storenumberstringnum[2]{%
1374     \@ifnextchar[%
1375         {\@store@number@string{#1}{#2}}%
1376         {\@store@number@string{#1}{#2}[m]}%
1377 }

```

`\store@number@string` Gender is given as optional argument, *at the end*.

```

1378 \def\@store@number@string#1#2[#3]{%
1379     \ifthenelse{\equal{#3}{f}}%
1380     {%
1381         \protect\@numberstringF{#2}{\@fc@numstr}%
1382     }%
1383     {%
1384         \ifthenelse{\equal{#3}{n}}%
1385         {%
1386             \protect\@numberstringN{#2}{\@fc@numstr}%
1387         }%
1388         {%
1389             \ifthenelse{\equal{#3}{m}}%
1390             {}%
1391             {%
1392                 \PackageError{fmtcount}
1393                 {Invalid gender option ‘#3’}%
1394                 {Available options are m, f or n}%
1395             }%
1396             \protect\@numberstringM{#2}{\@fc@numstr}%
1397         }%
1398     }%
1399     \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
1400 }

```

`\numberstring` Display textual representation of a number. The argument must be a counter.

```

1401 \newcommand*{\numberstring}[1]{%
1402   \expandafter\protect\expandafter\numberstringnum{%
1403     \expandafter\the\csname c@#1\endcsname}%
1404 }

```

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

```

1405 \newcommand*{\numberstringnum}[1]{%
1406   \new@ifnextchar[%
1407     {\@number@string{#1}}%
1408     {\@number@string{#1}[m]}%
1409 }

```

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

```

1410 \def\@number@string#1[#2]{%
1411   {%
1412     \ifthenelse{\equal{#2}{f}}%
1413     {%
1414       \protect\@numberstringF{#1}{\@fc@numstr}%
1415     }%
1416     {%
1417       \ifthenelse{\equal{#2}{n}}%
1418       {%
1419         \protect\@numberstringN{#1}{\@fc@numstr}%
1420       }%
1421       {%
1422         \ifthenelse{\equal{#2}{m}}%
1423         {}%
1424         {%
1425           \PackageError{fmtcount}%
1426             {Invalid gender option ‘#2’}%
1427             {Available options are m, f or n}%
1428         }%
1429         \protect\@numberstringM{#1}{\@fc@numstr}%
1430       }%
1431     }%
1432     \@fc@numstr
1433   }%
1434 }

```

`\storeNumberstring` Store textual representation of number. First argument is identifying name, second argument is a counter.

```

1435 \newcommand*{\storeNumberstring}[2]{%
1436   \expandafter\protect\expandafter\storeNumberstringnum{#1}{%
1437     \expandafter\the\csname c@#2\endcsname}%
1438 }

```

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

```

1439 \newcommand{\storeNumberstringnum}[2]{%
1440   \@ifnextchar[%

```

```

1441 {\@store@Number@string{#1}{#2}}%
1442 {\@store@Number@string{#1}{#2}[m]}%
1443 }

```

`\store@Number@string` Gender is specified as an optional argument *at the end*:

```

1444 \def\@store@Number@string#1#2[#3]{%
1445   \ifthenelse{\equal{#3}{f}}{%
1446     {%
1447       \protect\@NumberstringF{#2}{\@fc@numstr}%
1448     }%
1449     {%
1450       \ifthenelse{\equal{#3}{n}}{%
1451         {%
1452           \protect\@NumberstringN{#2}{\@fc@numstr}%
1453         }%
1454         {%
1455           \ifthenelse{\equal{#3}{m}}{%
1456             {}%
1457             {%
1458               \PackageError{fmtcount}%
1459               {Invalid gender option ‘#3’}%
1460               {Available options are m, f or n}%
1461             }%
1462             \protect\@NumberstringM{#2}{\@fc@numstr}%
1463           }%
1464         }%
1465       \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
1466     }

```

`\Numberstring` Display textual representation of number. The argument must be a counter.

```

1467 \newcommand*\@Numberstring}[1]{%
1468   \expandafter\protect\expandafter\Numberstringnum{%
1469     \expandafter\the\csname c@#1\endcsname}%
1470 }

```

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

```

1471 \newcommand*\@Numberstringnum}[1]{%
1472   \new@ifnextchar[%
1473     {\@Number@string{#1}}%
1474     {\@Number@string{#1}[m]}%
1475 }

```

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

```

1476 \def\@Number@string#1[#2]{%
1477   {%
1478     \ifthenelse{\equal{#2}{f}}{%
1479       {%
1480         \protect\@NumberstringF{#1}{\@fc@numstr}%
1481       }%

```



```

1482   {%
1483     \ifthenelse{\equal{#2}{n}}{%
1484       {%
1485         \protect\@NumberstringN{#1}{\@fc@numstr}%
1486       }%
1487     {%
1488       \ifthenelse{\equal{#2}{m}}{%
1489         {}%
1490       }%
1491       \PackageError{fmtcount}%
1492         {Invalid gender option ‘#2’}%
1493         {Available options are m, f or n}%
1494     }%
1495     \protect\@NumberstringM{#1}{\@fc@numstr}%
1496   }%
1497 }%
1498 \@fc@numstr
1499 }%
1500 }

```

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

```

1501 \newcommand{\storeNUMBERstring}[2]{%
1502   \expandafter\protect\expandafter\storeNUMBERstringnum{#1}{%
1503     \expandafter\the\csname c@#2\endcsname}%
1504 }

```

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

```

1505 \newcommand{\storeNUMBERstringnum}[2]{%
1506   \@ifnextchar[%
1507     {\@store@NUMBER@string{#1}{#2}}%
1508     {\@store@NUMBER@string{#1}{#2}[m]}%
1509 }

```

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

```

1510 \def\@store@NUMBER@string#1#2[#3]{%
1511   \ifthenelse{\equal{#3}{f}}{%
1512     {%
1513       \protect\@numberstringF{#2}{\@fc@numstr}%
1514     }%
1515   {%
1516     \ifthenelse{\equal{#3}{n}}{%
1517       {%
1518         \protect\@numberstringN{#2}{\@fc@numstr}%
1519       }%
1520     {%
1521       \ifthenelse{\equal{#3}{m}}{%
1522         {}%
1523       }%

```

```

1524         \PackageError{fmtcount}%
1525         {Invalid gender option ‘#3’}%
1526         {Available options are m or f}%
1527     }%
1528     \protect\@numberstringM{#2}{\@fc@numstr}%
1529 }%
1530 }%
1531 \expandafter\edef\csname @fcs@#1\endcsname{%
1532     \noexpand\MakeUppercase{\@fc@numstr}%
1533 }%
1534 }

```

`\NUMBERstring` Display upper case textual representation of a number. The argument must be a counter.

```

1535 \newcommand*{\NUMBERstring}[1]{%
1536     \expandafter\protect\expandafter\NUMBERstringnum{%
1537         \expandafter\the\csname c@#1\endcsname}%
1538 }

```

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

```

1539 \newcommand*{\NUMBERstringnum}[1]{%
1540     \new@ifnextchar[%
1541     {\@NUMBER@string{#1}}%
1542     {\@NUMBER@string{#1}[m]}%
1543 }

```

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

```

1544 \def\@NUMBER@string#1[#2]{%
1545     {%
1546         \ifthenelse{\equal{#2}{f}}%
1547         {%
1548             \protect\@numberstringF{#1}{\@fc@numstr}%
1549         }%
1550         {%
1551             \ifthenelse{\equal{#2}{n}}%
1552             {%
1553                 \protect\@numberstringN{#1}{\@fc@numstr}%
1554             }%
1555             {%
1556                 \ifthenelse{\equal{#2}{m}}%
1557                 {}%
1558                 {%
1559                     \PackageError{fmtcount}%
1560                     {Invalid gender option ‘#2’}%
1561                     {Available options are m, f or n}%
1562                 }%
1563                 \protect\@numberstringM{#1}{\@fc@numstr}%
1564             }%
1565         }%
1566     }%

```

```

1566 \MakeUppercase{\@fc@numstr}%
1567 }%
1568 }

```

`\binary` Number representations in other bases. Binary:

```

1569 \providecommand*\binary}[1]{%
1570 \expandafter\protect\expandafter\@binary{%
1571 \expandafter\the\csname c@#1\endcsname}%
1572 }

```

`\aaalph` Like `\alph`, but goes beyond 26. (a ... z aa ... zz ...)

```

1573 \providecommand*\aaalph}[1]{%
1574 \expandafter\protect\expandafter\@aaalph{%
1575 \expandafter\the\csname c@#1\endcsname}%
1576 }

```

`\AAAAlph` As before, but upper case.

```

1577 \providecommand*\AAAAlph}[1]{%
1578 \expandafter\protect\expandafter\@AAAAlph{%
1579 \expandafter\the\csname c@#1\endcsname}%
1580 }

```

`\abalph` Like `\alph`, but goes beyond 26. (a ... z ab ... az ...)

```

1581 \providecommand*\abalph}[1]{%
1582 \expandafter\protect\expandafter\@abalph{%
1583 \expandafter\the\csname c@#1\endcsname}%
1584 }

```

`\ABAlph` As above, but upper case.

```

1585 \providecommand*\ABAlph}[1]{%
1586 \expandafter\protect\expandafter\@ABAlph{%
1587 \expandafter\the\csname c@#1\endcsname}%
1588 }

```

`\hexadecimal` Hexadecimal:

```

1589 \providecommand*\hexadecimal}[1]{%
1590 \expandafter\protect\expandafter\@hexadecimal{%
1591 \expandafter\the\csname c@#1\endcsname}%
1592 }

```

`\Hexadecimal` As above, but in upper case.

```

1593 \providecommand*\Hexadecimal}[1]{%
1594 \expandafter\protect\expandafter\@Hexadecimal{%
1595 \expandafter\the\csname c@#1\endcsname}%
1596 }

```

`\octal` Octal:

```

1597 \providecommand*\octal}[1]{%

```

```

1598 \expandafter\protect\expandafter\@octal{%
1599 \expandafter\the\csname c@#1\endcsname}%
1600 }

```

\decimal Decimal:

```

1601 \providecommand*{\decimal}[1]{%
1602 \expandafter\protect\expandafter\@decimal{%
1603 \expandafter\the\csname c@#1\endcsname}%
1604 }

```

## 9.4 Multilingual Definitions

@setdef@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.

```

1605 \def\@setdef@ultfmtcount{%
1606 \ifundefined{@OrdinalMenglish}{\FCloadlang{english}}{}%
1607 \def\@OrdinalstringM{\@OrdinalstringMenglish}%
1608 \let\@OrdinalstringF=\@OrdinalstringMenglish
1609 \let\@OrdinalstringN=\@OrdinalstringMenglish
1610 \def\@OrdinalstringM{\@OrdinalstringMenglish}%
1611 \let\@OrdinalstringF=\@OrdinalstringMenglish
1612 \let\@OrdinalstringN=\@OrdinalstringMenglish
1613 \def\@NumberstringM{\@NumberstringMenglish}%
1614 \let\@NumberstringF=\@NumberstringMenglish
1615 \let\@NumberstringN=\@NumberstringMenglish
1616 \def\@NumberstringM{\@NumberstringMenglish}%
1617 \let\@NumberstringF=\@NumberstringMenglish
1618 \let\@NumberstringN=\@NumberstringMenglish
1619 \def\@OrdinalM{\@OrdinalMenglish}%
1620 \let\@OrdinalF=\@OrdinalM
1621 \let\@OrdinalN=\@OrdinalM
1622 }

```

\fc@multiling changes2.022012-10-24new \fc@multiling{<name>}{<gender>}

```

1623 \newcommand*{\fc@multiling}[2]{%
1624 \ifcsundef{@#1#2\language}%
1625 {% try loading it
1626 \FCloadlang{\language}%
1627 }%
1628 {%
1629 }%
1630 \ifcsundef{@#1#2\language}%
1631 {%
1632 \PackageWarning{fmtcount}%
1633 {No support for \expandafter\string\csname#1\endcsname\space for
1634 language '\language'}%
1635 \ifthenelse{\equal{\language}{\fc@mainlang}}{}%

```

```

1636    {%
1637        \FCloadlang{english}%
1638    }%
1639    {%
1640    }%
1641    \ifcsdef{@#1#2\fc@mainlang}%
1642    {%
1643        \csuse{@#1#2\fc@mainlang}%
1644    }%
1645    {%
1646        \PackageWarningNoLine{fmtcount}%
1647        {No languages loaded at all! Loading english definitions}%
1648        \FCloadlang{english}%
1649        \def\fc@mainlang{english}%
1650        \csuse{@#1#2english}%
1651    }%
1652    }%
1653    {%
1654        \csuse{@#1#2\language}%
1655    }%
1656 }

```

`@multiling@fmtcount` This defines the number and ordinal string macros to use `\language`:

```

1657 \def\@set@multiling@fmtcount{%
    The masculine version of \numberstring:
1658     \def\@numberstringM{%
1659         \fc@multiling{numberstring}{M}%
1660     }%
    The feminine version of \numberstring:
1661     \def\@numberstringF{%
1662         \fc@multiling{numberstring}{F}%
1663     }%
    The neuter version of \numberstring:
1664     \def\@numberstringN{%
1665         \fc@multiling{numberstring}{N}%
1666     }%
    The masculine version of \Numberstring:
1667     \def\@NumberstringM{%
1668         \fc@multiling{Numberstring}{M}%
1669     }%
    The feminine version of \Numberstring:
1670     \def\@NumberstringF{%
1671         \fc@multiling{Numberstring}{F}%
1672     }%
    The neuter version of \Numberstring:
1673     \def\@NumberstringN{%

```

```

1674 \fc@multiling{Numberstring}{N}%
1675 }%

The masculine version of \ordinal:
1676 \def\@ordinalM{%
1677 \fc@multiling{ordinal}{M}%
1678 }%

The feminine version of \ordinal:
1679 \def\@ordinalF{%
1680 \fc@multiling{ordinal}{F}%
1681 }%

The neuter version of \ordinal:
1682 \def\@ordinalN{%
1683 \fc@multiling{ordinal}{N}%
1684 }%

The masculine version of \ordinalstring:
1685 \def\@ordinalstringM{%
1686 \fc@multiling{ordinalstring}{M}%
1687 }%

The feminine version of \ordinalstring:
1688 \def\@ordinalstringF{%
1689 \fc@multiling{ordinalstring}{F}%
1690 }%

The neuter version of \ordinalstring:
1691 \def\@ordinalstringN{%
1692 \fc@multiling{ordinalstring}{N}%
1693 }%

The masculine version of \Ordinalstring:
1694 \def\@OrdinalstringM{%
1695 \fc@multiling{Ordinalstring}{M}%
1696 }%

The feminine version of \Ordinalstring:
1697 \def\@OrdinalstringF{%
1698 \fc@multiling{Ordinalstring}{F}%
1699 }%

The neuter version of \Ordinalstring:
1700 \def\@OrdinalstringN{%
1701 \fc@multiling{Ordinalstring}{N}%
1702 }%
1703 }

Check to see if babel or ngerman packages have been loaded.
1704 \@ifpackageloaded{babel}%
1705 {%
1706 \set@multiling@fmtcount

```

```

1707 }%
1708 {%
1709   \@ifpackageloaded{ngerman}%
1710   {%
1711     \FCloadlang{ngerman}%
1712     \@set@multiling@fmtcount
1713   }%
1714   {%
1715     \@setdef@ultfmtcount
1716   }%
1717 }

```

Backwards compatibility:

```

1718 \let\@ordinal=\@ordinalM
1719 \let\@ordinalstring=\@ordinalstringM
1720 \let\@Ordinalstring=\@OrdinalstringM
1721 \let\@numberstring=\@numberstringM
1722 \let\@Numberstring=\@NumberstringM

```

#### 9.4.1 fc-american.def

American English definitions

```

1723 \ProvidesFCLanguage{american}[2013/08/17]%

```

Loaded fc-USenglish.def if not already loaded

```

1724 \FCloadlang{USenglish}%

```

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

```

1725 \global\let\@ordinalMamerican\@ordinalMUSenglish
1726 \global\let\@ordinalFamerican\@ordinalMUSenglish
1727 \global\let\@ordinalNamerican\@ordinalMUSenglish
1728 \global\let\@numberstringMamerican\@numberstringMUSenglish
1729 \global\let\@numberstringFamerican\@numberstringMUSenglish
1730 \global\let\@numberstringNamerican\@numberstringMUSenglish
1731 \global\let\@NumberstringMamerican\@NumberstringMUSenglish
1732 \global\let\@NumberstringFamerican\@NumberstringMUSenglish
1733 \global\let\@NumberstringNamerican\@NumberstringMUSenglish
1734 \global\let\@ordinalstringMamerican\@ordinalstringMUSenglish
1735 \global\let\@ordinalstringFamerican\@ordinalstringMUSenglish
1736 \global\let\@ordinalstringNamerican\@ordinalstringMUSenglish
1737 \global\let\@OrdinalstringMamerican\@OrdinalstringMUSenglish
1738 \global\let\@OrdinalstringFamerican\@OrdinalstringMUSenglish
1739 \global\let\@OrdinalstringNamerican\@OrdinalstringMUSenglish

```

#### 9.4.2 fc-british.def

British definitions

```

1740 \ProvidesFCLanguage{british}[2013/08/17]%

```

Load fc-english.def, if not already loaded

```

1741 \FCloadlang{english}%

```

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

```

1742 \global\let\@ordinalMbritish\@ordinalMenglish
1743 \global\let\@ordinalFbritish\@ordinalMenglish
1744 \global\let\@ordinalNbritish\@ordinalMenglish
1745 \global\let\@numberstringMbritish\@numberstringMenglish
1746 \global\let\@numberstringFbritish\@numberstringMenglish
1747 \global\let\@numberstringNbritish\@numberstringMenglish
1748 \global\let\@NumberstringMbritish\@NumberstringMenglish
1749 \global\let\@NumberstringFbritish\@NumberstringMenglish
1750 \global\let\@NumberstringNbritish\@NumberstringMenglish
1751 \global\let\@ordinalstringMbritish\@ordinalstringMenglish
1752 \global\let\@ordinalstringFbritish\@ordinalstringMenglish
1753 \global\let\@ordinalstringNbritish\@ordinalstringMenglish
1754 \global\let\@OrdinalstringMbritish\@OrdinalstringMenglish
1755 \global\let\@OrdinalstringFbritish\@OrdinalstringMenglish
1756 \global\let\@OrdinalstringNbritish\@OrdinalstringMenglish

```

### 9.4.3 fc-english.def

English definitions

```

1757 \ProvidesFCLanguage{english}[2013/08/17]%

```

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.

```

1758 \gdef\@ordinalMenglish#1#2{%
1759 \def\@fc@ord{%
1760 \@orgargctr=#1\relax
1761 \@ordinalctr=#1%
1762 \@modulo{\@ordinalctr}{100}%
1763 \ifnum\@ordinalctr=11\relax
1764   \def\@fc@ord{th}%
1765 \else
1766   \ifnum\@ordinalctr=12\relax
1767     \def\@fc@ord{th}%
1768   \else
1769     \ifnum\@ordinalctr=13\relax
1770       \def\@fc@ord{th}%
1771     \else
1772       \@modulo{\@ordinalctr}{10}%
1773       \ifcase\@ordinalctr
1774         \def\@fc@ord{th}%           case 0
1775         \or \def\@fc@ord{st}%       case 1
1776         \or \def\@fc@ord{nd}%       case 2
1777         \or \def\@fc@ord{rd}%       case 3
1778       \else
1779         \def\@fc@ord{th}%           default case
1780       \fi
1781 \fi

```



```

1782 \fi
1783 \fi
1784 \edef#2{\number#1\relax\noexpand\fmtord{\@fc@ord}}}%
1785 }%

```

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

```

1786 \global\let\@ordinalFenglish=\@ordinalMenglish
1787 \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.

```

1788 \gdef\@@unitstringenglish#1{%
1789   \ifcase#1\relax
1790     zero%
1791     \or one%
1792     \or two%
1793     \or three%
1794     \or four%
1795     \or five%
1796     \or six%
1797     \or seven%
1798     \or eight%
1799     \or nine%
1800 \fi
1801 }%

```

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

```

1802 \gdef\@@tenstringenglish#1{%
1803   \ifcase#1\relax
1804     \or ten%
1805     \or twenty%
1806     \or thirty%
1807     \or forty%
1808     \or fifty%
1809     \or sixty%
1810     \or seventy%
1811     \or eighty%
1812     \or ninety%
1813 \fi
1814 }%

```

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

```

1815 \gdef\@@teenstringenglish#1{%
1816   \ifcase#1\relax
1817     ten%
1818     \or eleven%
1819     \or twelve%
1820     \or thirteen%
1821     \or fourteen%

```

```

1822 \or fifteen%
1823 \or sixteen%
1824 \or seventeen%
1825 \or eighteen%
1826 \or nineteen%
1827 \fi
1828 }%

```

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

```

1829 \gdef\@@Unitstringenglish#1{%
1830 \ifcase#1\relax
1831 Zero%
1832 \or One%
1833 \or Two%
1834 \or Three%
1835 \or Four%
1836 \or Five%
1837 \or Six%
1838 \or Seven%
1839 \or Eight%
1840 \or Nine%
1841 \fi
1842 }%

```

The tens:

```

1843 \gdef\@@Tenstringenglish#1{%
1844 \ifcase#1\relax
1845 \or Ten%
1846 \or Twenty%
1847 \or Thirty%
1848 \or Forty%
1849 \or Fifty%
1850 \or Sixty%
1851 \or Seventy%
1852 \or Eighty%
1853 \or Ninety%
1854 \fi
1855 }%

```

The teens:

```

1856 \gdef\@@Teenstringenglish#1{%
1857 \ifcase#1\relax
1858 Ten%
1859 \or Eleven%
1860 \or Twelve%
1861 \or Thirteen%
1862 \or Fourteen%
1863 \or Fifteen%
1864 \or Sixteen%
1865 \or Seventeen%
1866 \or Eighteen%

```

```

1867 \or Nineteen%
1868 \fi
1869 }%

```

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.)

```

1870 \gdef\@@numberstringenglish#1#2{%
1871 \ifnum#1>99999
1872 \PackageError{fmtcount}{Out of range}%
1873 {This macro only works for values less than 100000}%
1874 \else
1875 \ifnum#1<0
1876 \PackageError{fmtcount}{Negative numbers not permitted}%
1877 {This macro does not work for negative numbers, however
1878 you can try typing "minus" first, and then pass the modulus of
1879 this number}%
1880 \fi
1881 \fi
1882 \def#2{%
1883 \@strctr=#1\relax \divide\@strctr by 1000\relax
1884 \ifnum\@strctr>9
1885 \divide\@strctr by 10
1886 \ifnum\@strctr>1\relax
1887 \let\@@fc@numstr#2\relax
1888 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
1889 \@strctr=#1 \divide\@strctr by 1000\relax
1890 \@modulo{\@strctr}{10}%
1891 \ifnum\@strctr>0\relax
1892 \let\@@fc@numstr#2\relax
1893 \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
1894 \fi
1895 \else
1896 \@strctr=#1\relax
1897 \divide\@strctr by 1000\relax
1898 \@modulo{\@strctr}{10}%
1899 \let\@@fc@numstr#2\relax
1900 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
1901 \fi
1902 \let\@@fc@numstr#2\relax
1903 \edef#2{\@@fc@numstr\@thousand}%
1904 \else
1905 \ifnum\@strctr>0\relax
1906 \let\@@fc@numstr#2\relax
1907 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\@thousand}%
1908 \fi
1909 \fi
1910 \@strctr=#1\relax \@modulo{\@strctr}{1000}%

```

```

1911 \divide \@strctr by 100
1912 \ifnum \@strctr>0\relax
1913   \ifnum#1>1000\relax
1914     \let\@@fc@numstr#2\relax
1915     \edef#2{\@@fc@numstr\ }%
1916   \fi
1917   \let\@@fc@numstr#2\relax
1918   \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@hundred}%
1919 \fi
1920 \@strctr=#1\relax \@modulo{\@strctr}{100}%
1921 \ifnum#1>100\relax
1922   \ifnum \@strctr>0\relax
1923     \let\@@fc@numstr#2\relax
1924     \edef#2{\@@fc@numstr\ \@andname\ }%
1925   \fi
1926 \fi
1927 \ifnum \@strctr>19\relax
1928   \divide \@strctr by 10\relax
1929   \let\@@fc@numstr#2\relax
1930   \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
1931   \@strctr=#1\relax \@modulo{\@strctr}{10}%
1932   \ifnum \@strctr>0\relax
1933     \let\@@fc@numstr#2\relax
1934     \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
1935   \fi
1936 \else
1937   \ifnum \@strctr<10\relax
1938     \ifnum \@strctr=0\relax
1939       \ifnum#1<100\relax
1940         \let\@@fc@numstr#2\relax
1941         \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
1942       \fi
1943     \else
1944       \let\@@fc@numstr#2\relax
1945       \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
1946     \fi
1947   \else
1948     \@modulo{\@strctr}{10}%
1949     \let\@@fc@numstr#2\relax
1950     \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
1951   \fi
1952 \fi
1953 }%

```

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

```

1954 \DeclareRobustCommand{\@numberstringMenglish}[2]{%
1955   \let\@unitstring=\@unitstringenglish
1956   \let\@teenstring=\@teenstringenglish
1957   \let\@tenstring=\@tenstringenglish
1958   \def\@hundred{hundred}\def\@thousand{thousand}%

```

```

1959 \def\@andname{and}%
1960 \@numberstringenglish{#1}{#2}%
1961 }%
1962 \global\let\@numberstringMenglish\@numberstringMenglish

```

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

```

1963 \global\let\@numberstringFenglish=\@numberstringMenglish
1964 \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.

```

1965 \gdef\@NumberstringMenglish#1#2{%
1966 \let\@unitstring=\@Unitstringenglish
1967 \let\@teenstring=\@Teenstringenglish
1968 \let\@tenstring=\@Tenstringenglish
1969 \def\@hundred{Hundred}\def\@thousand{Thousand}%
1970 \def\@andname{and}%
1971 \@numberstringenglish{#1}{#2}%
1972 }%

```

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

```

1973 \global\let\@NumberstringFenglish=\@NumberstringMenglish
1974 \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:

```

1975 \gdef\@unitthstringenglish#1{%
1976 \ifcase#1\relax
1977 zeroth%
1978 \or first%
1979 \or second%
1980 \or third%
1981 \or fourth%
1982 \or fifth%
1983 \or sixth%
1984 \or seventh%
1985 \or eighth%
1986 \or ninth%
1987 \fi
1988 }%

```

Next the tens:

```

1989 \gdef\@tenthstringenglish#1{%
1990 \ifcase#1\relax
1991 \or tenth%
1992 \or twentieth%
1993 \or thirtieth%
1994 \or fortieth%
1995 \or fiftieth%

```

```

1996      \or sixtieth%
1997      \or seventieth%
1998      \or eightieth%
1999      \or ninetieth%
2000      \fi
2001 }%

```

The teens:

```

2002 \gdef\@@teenthstringenglish#1{%
2003      \ifcase#1\relax
2004          tenth%
2005      \or eleventh%
2006      \or twelfth%
2007      \or thirteenth%
2008      \or fourteenth%
2009      \or fifteenth%
2010      \or sixteenth%
2011      \or seventeenth%
2012      \or eighteenth%
2013      \or nineteenth%
2014      \fi
2015 }%

```

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

```

2016 \gdef\@@Unitthstringenglish#1{%
2017      \ifcase#1\relax
2018          Zeroth%
2019      \or First%
2020      \or Second%
2021      \or Third%
2022      \or Fourth%
2023      \or Fifth%
2024      \or Sixth%
2025      \or Seventh%
2026      \or Eighth%
2027      \or Ninth%
2028      \fi
2029 }%

```

The tens:

```

2030 \gdef\@@Tenthstringenglish#1{%
2031      \ifcase#1\relax
2032      \or Tenth%
2033      \or Twentieth%
2034      \or Thirtieth%
2035      \or Fortieth%
2036      \or Fiftieth%
2037      \or Sixtieth%
2038      \or Seventieth%
2039      \or Eightieth%
2040      \or Ninetieth%

```

```

2041 \fi
2042 }%

The teens:

2043 \gdef\@@Teenthstringenglish#1{%
2044 \ifcase#1\relax
2045 Tenth%
2046 \or Eleventh%
2047 \or Twelfth%
2048 \or Thirteenth%
2049 \or Fourteenth%
2050 \or Fifteenth%
2051 \or Sixteenth%
2052 \or Seventeenth%
2053 \or Eighteenth%
2054 \or Nineteenth%
2055 \fi
2056 }%

```

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.

```

2057 \gdef\@@ordinalstringenglish#1#2{%
2058 \@strctr=#1\relax
2059 \ifnum#1>99999
2060 \PackageError{fmtcount}{Out of range}%
2061 {This macro only works for values less than 100000 (value given: \number\@strctr)}%
2062 \else
2063 \ifnum#1<0
2064 \PackageError{fmtcount}{Negative numbers not permitted}%
2065 {This macro does not work for negative numbers, however
2066 you can try typing "minus" first, and then pass the modulus of
2067 this number}%
2068 \fi
2069 \def#2{}%
2070 \fi
2071 \@strctr=#1\relax \divide\@strctr by 1000\relax
2072 \ifnum\@strctr>9\relax

#1 is greater or equal to 10000

2073 \divide\@strctr by 10
2074 \ifnum\@strctr>1\relax
2075 \let\@@fc@ordstr#2\relax
2076 \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
2077 \@strctr=#1\relax
2078 \divide\@strctr by 1000\relax
2079 \@modulo{\@strctr}{10}%
2080 \ifnum\@strctr>0\relax
2081 \let\@@fc@ordstr#2\relax
2082 \edef#2{\@@fc@ordstr-\@unitstring{\@strctr}}%

```

```

2083 \fi
2084 \else
2085 \@strctr=#1\relax \divide\@strctr by 1000\relax
2086 \@modulo{\@strctr}{10}%
2087 \let\@@fc@ordstr#2\relax
2088 \edef#2{\@@fc@ordstr\@teenstring{\@strctr}}%
2089 \fi
2090 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
2091 \ifnum\@strctr=0\relax
2092 \let\@@fc@ordstr#2\relax
2093 \edef#2{\@@fc@ordstr\ \@thousandth}%
2094 \else
2095 \let\@@fc@ordstr#2\relax
2096 \edef#2{\@@fc@ordstr\ \@thousand}%
2097 \fi
2098 \else
2099 \ifnum\@strctr>0\relax
2100 \let\@@fc@ordstr#2\relax
2101 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
2102 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
2103 \let\@@fc@ordstr#2\relax
2104 \ifnum\@strctr=0\relax
2105 \edef#2{\@@fc@ordstr\ \@thousandth}%
2106 \else
2107 \edef#2{\@@fc@ordstr\ \@thousand}%
2108 \fi
2109 \fi
2110 \fi
2111 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
2112 \divide\@strctr by 100
2113 \ifnum\@strctr>0\relax
2114 \ifnum#1>1000\relax
2115 \let\@@fc@ordstr#2\relax
2116 \edef#2{\@@fc@ordstr\ }%
2117 \fi
2118 \let\@@fc@ordstr#2\relax
2119 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
2120 \@strctr=#1\relax \@modulo{\@strctr}{100}%
2121 \let\@@fc@ordstr#2\relax
2122 \ifnum\@strctr=0\relax
2123 \edef#2{\@@fc@ordstr\ \@hundredth}%
2124 \else
2125 \edef#2{\@@fc@ordstr\ \@hundred}%
2126 \fi
2127 \fi
2128 \@strctr=#1\relax \@modulo{\@strctr}{100}%
2129 \ifnum#1>100\relax
2130 \ifnum\@strctr>0\relax
2131 \let\@@fc@ordstr#2\relax

```



```

2132 \edef#2{\@@fc@ordstr\ \@andname\ }%
2133 \fi
2134 \fi
2135 \ifnum\@strctr>19\relax
2136 \@tmpstrctr=\@strctr
2137 \divide\@strctr by 10\relax
2138 \@modulo{\@tmpstrctr}{10}%
2139 \let\@@fc@ordstr#2\relax
2140 \ifnum\@tmpstrctr=0\relax
2141 \edef#2{\@@fc@ordstr\@tenthstring{\@strctr}}%
2142 \else
2143 \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
2144 \fi
2145 \@strctr=#1\relax \@modulo{\@strctr}{10}%
2146 \ifnum\@strctr>0\relax
2147 \let\@@fc@ordstr#2\relax
2148 \edef#2{\@@fc@ordstr-\@unitthstring{\@strctr}}%
2149 \fi
2150 \else
2151 \ifnum\@strctr<10\relax
2152 \ifnum\@strctr=0\relax
2153 \ifnum#1<100\relax
2154 \let\@@fc@ordstr#2\relax
2155 \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
2156 \fi
2157 \else
2158 \let\@@fc@ordstr#2\relax
2159 \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
2160 \fi
2161 \else
2162 \@modulo{\@strctr}{10}%
2163 \let\@@fc@ordstr#2\relax
2164 \edef#2{\@@fc@ordstr\@teenthstring{\@strctr}}%
2165 \fi
2166 \fi
2167 }%

```

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

```

2168 \DeclareRobustCommand{\@ordinalstringMenglish}[2]{%
2169 \let\@unitthstring=\@unitthstringenglish
2170 \let\@teenthstring=\@teenthstringenglish
2171 \let\@tenthstring=\@tenthstringenglish
2172 \let\@unitstring=\@unitstringenglish
2173 \let\@teenstring=\@teenstringenglish
2174 \let\@tenstring=\@tenstringenglish
2175 \def\@andname{and}%
2176 \def\@hundred{hundred}\def\@thousand{thousand}%
2177 \def\@hundredth{hundredth}\def\@thousandth{thousandth}%
2178 \@ordinalstringenglish{#1}{#2}%

```

```

2179 }%
2180 \global\let\@ordinalstringMenglish\@ordinalstringMenglish

  No gender in English, so make feminine and neuter same as masculine:
2181 \global\let\@ordinalstringFenglish=\@ordinalstringMenglish
2182 \global\let\@ordinalstringNenglish=\@ordinalstringMenglish

  First letter of each word in upper case:
2183 \DeclareRobustCommand{\@OrdinalstringMenglish}[2]{%
2184   \let\@unitthstring=\@Unitthstringenglish
2185   \let\@teenthstring=\@Teenthstringenglish
2186   \let\@tenthstring=\@Tenthstringenglish
2187   \let\@unitstring=\@Unitstringenglish
2188   \let\@teenstring=\@Teenstringenglish
2189   \let\@tenstring=\@Tenstringenglish
2190   \def\@andname{and}%
2191   \def\@hundred{Hundred}\def\@thousand{Thousand}%
2192   \def\@hundredth{Hundredth}\def\@thousandth{Thousandth}%
2193   \@ordinalstringenglish{#1}{#2}%
2194 }%
2195 \global\let\@OrdinalstringMenglish\@OrdinalstringMenglish

  No gender in English, so make feminine and neuter same as masculine:
2196 \global\let\@OrdinalstringFenglish=\@OrdinalstringMenglish
2197 \global\let\@OrdinalstringNenglish=\@OrdinalstringMenglish

```

#### 9.4.4 fc-francais.def

```

2198 \ProvidesFCLanguage{francais}[2013/08/17]%
2199 \FCloadlang{french}%

  Set francais to be equivalent to french.
2200 \global\let\@ordinalMfrancais=\@ordinalMfrench
2201 \global\let\@ordinalFfrancais=\@ordinalFfrench
2202 \global\let\@ordinalNfrancais=\@ordinalNfrench
2203 \global\let\@numberstringMfrancais=\@numberstringMfrench
2204 \global\let\@numberstringFfrancais=\@numberstringFfrench
2205 \global\let\@numberstringNfrancais=\@numberstringNfrench
2206 \global\let\@NumberstringMfrancais=\@NumberstringMfrench
2207 \global\let\@NumberstringFfrancais=\@NumberstringFfrench
2208 \global\let\@NumberstringNfrancais=\@NumberstringNfrench
2209 \global\let\@ordinalstringMfrancais=\@ordinalstringMfrench
2210 \global\let\@ordinalstringFfrancais=\@ordinalstringFfrench
2211 \global\let\@ordinalstringNfrancais=\@ordinalstringNfrench
2212 \global\let\@OrdinalstringMfrancais=\@OrdinalstringMfrench
2213 \global\let\@OrdinalstringFfrancais=\@OrdinalstringFfrench
2214 \global\let\@OrdinalstringNfrancais=\@OrdinalstringNfrench

```

#### 9.4.5 fc-french.def

```

  Definitions for French.
2215 \ProvidesFCLanguage{french}[2012/10/24]%

```

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 reference: [http://www.alain.be/boece/noms\\_de\\_nombre.html](http://www.alain.be/boece/noms_de_nombre.html) (Package now loaded by fmtcount)

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'.
2216 \def\fc@french@set@plural#1#2#3#4#5#6{%
2217   \ifthenelse{\equal{#2}{reformed}}{%
2218     \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#3}%
2219   }{%
2220     \ifthenelse{\equal{#2}{traditional}}{%
2221       \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#4}%
2222     }{%
2223       \ifthenelse{\equal{#2}{reformed o}}{%
2224         \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#5}%
2225       }{%
2226         \ifthenelse{\equal{#2}{traditional o}}{%
2227           \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#6}%
2228         }{%
2229           \ifthenelse{\equal{#2}{always}}{%
2230             \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{0}%
2231           }{%
2232             \ifthenelse{\equal{#2}{never}}{%
2233               \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{1}%
2234             }{%
2235               \ifthenelse{\equal{#2}{multiple}}{%
2236                 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{2}%
2237               }{%
2238                 \ifthenelse{\equal{#2}{multiple g-last}}{%
2239                   \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{3}%
2240                 }{%
2241                   \ifthenelse{\equal{#2}{multiple l-last}}{%
2242                     \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{4}%
2243                   }{%
2244                     \ifthenelse{\equal{#2}{multiple lng-last}}{%
2245                       \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{5}%
2246                     }{%
2247                       \ifthenelse{\equal{#2}{multiple ng-last}}{%
2248                         \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{6}%
2249                       }{%
2250                         \PackageError{fmtcount}{Unexpected argument}{%
2251                           '#2' was unexpected: french option '#1 plural' expects 'reformed'
```



```

2286         French option ‘dash or space’ expects ‘always’, ‘reformed’ or ‘traditional’
2287     }
2288 }%
2289 }%
2290 }%
2291 }

```

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  $10^{18} = \text{un milliard de milliards}$

```

2292 \define@key{fcfrench}{scale}[recursive]{%
2293   \ifthenelse{\equal{#1}{long}}{%
2294     \let\fc@poweroften\fc@pot@longscalefrench
2295   }{%
2296     \ifthenelse{\equal{#1}{recursive}}{%
2297       \let\fc@poweroften\fc@pot@recursivefrench
2298     }{%
2299       \ifthenelse{\equal{#1}{short}}{%
2300         \let\fc@poweroften\fc@pot@shortscalefrench
2301       }{%
2302         \PackageError{fmtcount}{Unexpected argument}{%
2303           French option ‘scale’ expects ‘long’, ‘recursive’ or ‘short’
2304         }
2305       }%
2306     }%
2307   }%
2308 }

```

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”

```

2309 \define@key{fcfrench}{n-illiard upto}[infinity]{%
2310   \ifthenelse{\equal{#1}{infinity}}{%
2311     \def\fc@longscale@nilliard@upto{0}%
2312   }{%
2313     \ifthenelse{\equal{#1}{infty}}{%
2314       \def\fc@longscale@nilliard@upto{0}%
2315     }{%
2316       \if Q\ifnum9<1#1Q\fi\else
2317         \PackageError{fmtcount}{Unexpected argument}{%
2318           French option ‘milliard threshold’ expects ‘infinity’, or equivalently ‘infty’, or a
2319           integer.}%
2320       \fi
2321       \def\fc@longscale@nilliard@upto{#1}%

```

```
2322   } }%
```

```
2323 }
```

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.

```
2324 \def\@tempa#1{%
```

```
2325   \define@key{fcfrench}{#1}[] {%
```

```
2326     \PackageError{fmtcount}{Unexpected argument}{French option with key ‘#1’ does not take
2327       any value}}%
```

```
2328   \expandafter\def\csname KV@fcfrench@#1@default\endcsname{%
```

```
2329     \def\fmtcount@french{#1}}%
```

```
2330 }%
```

```
2331 \@tempa{france}\@tempa{swiss}\@tempa{belgian}%
```

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.

```
2332 \define@key{fcfrench}{dialect}[france] {%
```

```
2333   \ifthenelse{\equal{#1}{france}}
```

```
2334     \or\equal{#1}{swiss}
```

```
2335     \or\equal{#1}{belgian}} {%
```

```
2336   \def\fmtcount@french{#1}}%
```

```
2337   \PackageError{fmtcount}{Invalid value ‘#1’ to french option dialect key}
```

```
2338   {Option ‘french’ can only take the values ‘france’,
```

```
2339     ‘belgian’ or ‘swiss’}}}
```

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’.

```
2340 \define@key{fcfrench}{mil plural mark}[le] {%
```

```
2341   \def\fc@frenchoptions@mil@plural@mark{#1}}
```

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

```
2342 \def\fc@UpperCaseFirstLetter#1#2\@nil{%
```

```
2343   \uppercase{#1}#2}
```

```
2344
```

```
2345 \def\fc@CaseIden#1\@nil{%
```

```
2346   #1%
```

```
2347 }
```

```
2348 \def\fc@UpperCaseAll#1\@nil{%
```

```
2349   \uppercase{#1}%
```

```
2350 }
```

```
2351
```

```
2352 \let\fc@case\fc@CaseIden
```

```
2353
```

\@ ordinalMfrench

```
2354 \newcommand*{\@ordinalMfrench}[2] {%
```

```
2355   \iffmtord@abbrv
```

```
2356   \edef#2{\number#1\relax\noexpand\fmtord{e}}%
```

```

2357 \else
2358   \ifnum#1=1\relax
2359     \edef#2{\number#1\relax\noexpand\fmtord{er}}}%
2360   \else
2361     \edef#2{\number#1\relax\noexpand\fmtord{eme}}}%
2362   \fi
2363 \fi}

\@ ordinalFfrench
2364 \newcommand*{\@ordinalFfrench}[2]{%
2365 \iffmtord@abbrv
2366   \edef#2{\number#1\relax\noexpand\fmtord{e}}}%
2367 \else
2368   \ifnum#1=1 %
2369     \edef#2{\number#1\relax\noexpand\fmtord{i\'ere}}}%
2370   \else
2371     \edef#2{\number#1\relax\noexpand\fmtord{i\'eme}}}%
2372   \fi
2373 \fi}

  In French neutral gender and masculine gender are formally identical.

2374 \let\@ordinalNfrench\@ordinalMfrench

\@ @unitstringfrench
2375 \newcommand*{\@unitstringfrench}[1]{%
2376 \noexpand\fc@case
2377 \ifcase#1 %
2378 z\'ero%
2379 \or un%
2380 \or deux%
2381 \or trois%
2382 \or quatre%
2383 \or cinq%
2384 \or six%
2385 \or sept%
2386 \or huit%
2387 \or neuf%
2388 \fi
2389 \noexpand\@nil
2390 }

\@ @tenstringfrench
2391 \newcommand*{\@tenstringfrench}[1]{%
2392 \noexpand\fc@case
2393 \ifcase#1 %
2394 \or dix%
2395 \or vingt%
2396 \or trente%
2397 \or quarante%
2398 \or cinquante%
2399 \or soixante%

```

```

2400 \or septante%
2401 \or huitante%
2402 \or nonante%
2403 \or cent%
2404 \fi
2405 \noexpand\@nil
2406 }

\@ @teenstringfrench
2407 \newcommand*\@teenstringfrench}[1]{%
2408 \noexpand\fc@case
2409 \ifcase#1 %
2410     dix%
2411 \or onze%
2412 \or douze%
2413 \or treize%
2414 \or quatorze%
2415 \or quinze%
2416 \or seize%
2417 \or dix\noexpand\@nil-\noexpand\fc@case sept%
2418 \or dix\noexpand\@nil-\noexpand\fc@case huit%
2419 \or dix\noexpand\@nil-\noexpand\fc@case neuf%
2420 \fi
2421 \noexpand\@nil
2422 }

\@ @seventiesfrench
2423 \newcommand*\@seventiesfrench}[1]{%
2424 \@tenstring{6}%
2425 \ifnum#1=1 %
2426 \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
2427 \else
2428 -%
2429 \fi
2430 \@teenstring{#1}%
2431 }

\@ @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 follow-
ing the eighties to be formatted; that is ‘1’ for “mil” and ‘2’ for
“< $n$ >illion|< $n$ >illiard”.

```



```

2432 \newcommand*\@@eightiesfrench[1]{%
2433 \fc@case quatre\@nil-\noexpand\fc@case vingt%
2434 \ifnum#1>0 %
2435   \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
2436   s%
2437   \fi
2438   \noexpand\@nil
2439   -\@unitstring{#1}%
2440 \else
2441   \ifcase\fc@frenchoptions@vingt@plural\space
2442     s% 0: always
2443   \or
2444     % 1: never
2445   \or
2446     s% 2: multiple
2447   \or
2448     % 3: multiple g-last
2449     \ifnum\count0=\count6\ifnum\count9=0 s\fi\fi
2450   \or
2451     % 4: multiple l-last
2452     \ifnum\count9=1 %
2453     \else
2454       s%
2455     \fi
2456   \or
2457     % 5: multiple lng-last
2458     \ifnum\count9=1 %
2459     \else
2460       \ifnum\count0>0 %
2461       s%
2462     \fi
2463   \fi
2464   \or
2465     % or 6: multiple ng-last
2466     \ifnum\count0>0 %
2467     s%
2468   \fi
2469   \fi
2470   \noexpand\@nil
2471 \fi
2472 }
2473 \newcommand*\@@ninetiesfrench}[1]{%
2474 \fc@case quatre\@nil-\noexpand\fc@case vingt%
2475 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
2476 s%
2477 \fi
2478 \noexpand\@nil
2479 -\@teenstring{#1}%
2480 }

```

```

2481 \newcommand*{\@@seventiesfrenchswiss}[1]{%
2482 \@tenstring{7}%
2483 \ifnum#1=1\ \@andname\ \fi
2484 \ifnum#1>1-\fi
2485 \ifnum#1>0 \@unitstring{#1}\fi
2486 }
2487 \newcommand*{\@@eightiesfrenchswiss}[1]{%
2488 \@tenstring{8}%
2489 \ifnum#1=1\ \@andname\ \fi
2490 \ifnum#1>1-\fi
2491 \ifnum#1>0 \@unitstring{#1}\fi
2492 }
2493 \newcommand*{\@@ninetiesfrenchswiss}[1]{%
2494 \@tenstring{9}%
2495 \ifnum#1=1\ \@andname\ \fi
2496 \ifnum#1>1-\fi
2497 \ifnum#1>0 \@unitstring{#1}\fi
2498 }

\fc @french@common Macro \fc@french@common does all the preliminary set-
tings common to all French dialects & formatting options.

2499 \newcommand*\fc@french@common{%
2500 \let\@unitstring=\@unitstringfrench
2501 \let\@teenstring=\@teenstringfrench
2502 \let\@tenstring=\@tenstringfrench
2503 \def\@hundred{cent}%
2504 \def\@andname{et}%
2505 }

2506 \DeclareRobustCommand{\@numberstringMfrenchswiss}[2]{%
2507 \let\fc@case\fc@CaseIden
2508 \fc@french@common
2509 \let\@seventies=\@seventiesfrenchswiss
2510 \let\@eighties=\@eightiesfrenchswiss
2511 \let\@nineties=\@ninetiesfrenchswiss
2512 \let\fc@nbrstr@preamble\@empty
2513 \let\fc@nbrstr@postamble\@empty
2514 \@@numberstringfrench{#1}{#2}}
2515 \DeclareRobustCommand{\@numberstringMfrenchfrance}[2]{%
2516 \let\fc@case\fc@CaseIden
2517 \fc@french@common
2518 \let\@seventies=\@seventiesfrench
2519 \let\@eighties=\@eightiesfrench
2520 \let\@nineties=\@ninetiesfrench
2521 \let\fc@nbrstr@preamble\@empty
2522 \let\fc@nbrstr@postamble\@empty
2523 \@@numberstringfrench{#1}{#2}}
2524 \DeclareRobustCommand{\@numberstringMfrenchbelgian}[2]{%
2525 \let\fc@case\fc@CaseIden
2526 \fc@french@common

```

```

2527 \let\@seventies=\@seventiesfrenchswiss
2528 \let\@eighties=\@eightiesfrench
2529 \let\@nineties=\@ninetiesfrench
2530 \let\fc@nbrstr@preamble\@empty
2531 \let\fc@nbrstr@postamble\@empty
2532 \@@numberstringfrench{#1}{#2}}
2533 \let\@numberstringMfrench=\@numberstringMfrenchfrance
2534 \DeclareRobustCommand{\@numberstringFfrenchswiss}[2]{%
2535 \let\fc@case\fc@CaseIden
2536 \fc@french@common
2537 \let\@seventies=\@seventiesfrenchswiss
2538 \let\@eighties=\@eightiesfrenchswiss
2539 \let\@nineties=\@ninetiesfrenchswiss
2540 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2541 \let\fc@nbrstr@postamble\@empty
2542 \@@numberstringfrench{#1}{#2}}
2543 \DeclareRobustCommand{\@numberstringFfrenchfrance}[2]{%
2544 \let\fc@case\fc@CaseIden
2545 \fc@french@common
2546 \let\@seventies=\@seventiesfrench
2547 \let\@eighties=\@eightiesfrench
2548 \let\@nineties=\@ninetiesfrench
2549 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2550 \let\fc@nbrstr@postamble\@empty
2551 \@@numberstringfrench{#1}{#2}}
2552 \DeclareRobustCommand{\@numberstringFfrenchbelgian}[2]{%
2553 \let\fc@case\fc@CaseIden
2554 \fc@french@common
2555 \let\@seventies=\@seventiesfrenchswiss
2556 \let\@eighties=\@eightiesfrench
2557 \let\@nineties=\@ninetiesfrench
2558 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2559 \let\fc@nbrstr@postamble\@empty
2560 \@@numberstringfrench{#1}{#2}}
2561 \let\@numberstringFfrench=\@numberstringFfrenchfrance
2562 \let\@ordinalstringNfrench\@ordinalstringMfrench
2563 \DeclareRobustCommand{\@NumberstringMfrenchswiss}[2]{%
2564 \let\fc@case\fc@UpperCaseFirstLetter
2565 \fc@french@common
2566 \let\@seventies=\@seventiesfrenchswiss
2567 \let\@eighties=\@eightiesfrenchswiss
2568 \let\@nineties=\@ninetiesfrenchswiss
2569 \let\fc@nbrstr@preamble\@empty
2570 \let\fc@nbrstr@postamble\@empty
2571 \@@numberstringfrench{#1}{#2}}
2572 \DeclareRobustCommand{\@NumberstringMfrenchfrance}[2]{%
2573 \let\fc@case\fc@UpperCaseFirstLetter
2574 \fc@french@common
2575 \let\@seventies=\@seventiesfrench

```

```

2576 \let\@eighties=\@@eightiesfrench
2577 \let\@nineties=\@@ninetiesfrench
2578 \let\fc@nbrstr@preamble\@empty
2579 \let\fc@nbrstr@postamble\@empty
2580 @@numberstringfrench{#1}{#2}}
2581 \DeclareRobustCommand{\@NumberstringMfrenchbelgian}[2]{%
2582 \let\fc@case\fc@UpperCaseFirstLetter
2583 \fc@french@common
2584 \let\@seventies=\@@seventiesfrenchswiss
2585 \let\@eighties=\@@eightiesfrench
2586 \let\@nineties=\@@ninetiesfrench
2587 \let\fc@nbrstr@preamble\@empty
2588 \let\fc@nbrstr@postamble\@empty
2589 @@numberstringfrench{#1}{#2}}
2590 \let\@NumberstringMfrench=\@NumberstringMfrenchfrance
2591 \DeclareRobustCommand{\@NumberstringFfrenchswiss}[2]{%
2592 \let\fc@case\fc@UpperCaseFirstLetter
2593 \fc@french@common
2594 \let\@seventies=\@@seventiesfrenchswiss
2595 \let\@eighties=\@@eightiesfrenchswiss
2596 \let\@nineties=\@@ninetiesfrenchswiss
2597 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2598 \let\fc@nbrstr@postamble\@empty
2599 @@numberstringfrench{#1}{#2}}
2600 \DeclareRobustCommand{\@NumberstringFfrenchfrance}[2]{%
2601 \let\fc@case\fc@UpperCaseFirstLetter
2602 \fc@french@common
2603 \let\@seventies=\@@seventiesfrench
2604 \let\@eighties=\@@eightiesfrench
2605 \let\@nineties=\@@ninetiesfrench
2606 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2607 \let\fc@nbrstr@postamble\@empty
2608 @@numberstringfrench{#1}{#2}}
2609 \DeclareRobustCommand{\@NumberstringFfrenchbelgian}[2]{%
2610 \let\fc@case\fc@UpperCaseFirstLetter
2611 \fc@french@common
2612 \let\@seventies=\@@seventiesfrenchswiss
2613 \let\@eighties=\@@eightiesfrench
2614 \let\@nineties=\@@ninetiesfrench
2615 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2616 \let\fc@nbrstr@postamble\@empty
2617 @@numberstringfrench{#1}{#2}}
2618 \let\@NumberstringFfrench=\@NumberstringFfrenchfrance
2619 \let\@NumberstringNfrench\@NumberstringMfrench
2620 \DeclareRobustCommand{\@ordinalstringMfrenchswiss}[2]{%
2621 \let\fc@case\fc@CaseIden
2622 \let\fc@first=\fc@@firstfrench
2623 \fc@french@common
2624 \let\@seventies=\@@seventiesfrenchswiss

```

```

2625 \let\@eighties=\@eightiesfrenchswiss
2626 \let\@nineties=\@ninetiesfrenchswiss
2627 \@ordinalstringfrench{#1}{#2}%
2628 }
2629 \newcommand*\fc@@firstfrench{premier}
2630 \newcommand*\fc@@firstFfrench{premi\`ere}
2631 \DeclareRobustCommand{\@ordinalstringMfrenchfrance}[2]{%
2632 \let\fc@case\fc@CaseIden
2633 \let\fc@first=\fc@@firstfrench
2634 \fc@french@common
2635 \let\@seventies=\@seventiesfrench
2636 \let\@eighties=\@eightiesfrench
2637 \let\@nineties=\@ninetiesfrench
2638 \@ordinalstringfrench{#1}{#2}}
2639 \DeclareRobustCommand{\@ordinalstringMfrenchbelgian}[2]{%
2640 \let\fc@case\fc@CaseIden
2641 \let\fc@first=\fc@@firstfrench
2642 \fc@french@common
2643 \let\@seventies=\@seventiesfrench
2644 \let\@eighties=\@eightiesfrench
2645 \let\@nineties=\@ninetiesfrench
2646 \@ordinalstringfrench{#1}{#2}%
2647 }
2648 \let\@ordinalstringMfrench=\@ordinalstringMfrenchfrance
2649 \DeclareRobustCommand{\@ordinalstringFfrenchswiss}[2]{%
2650 \let\fc@case\fc@CaseIden
2651 \let\fc@first=\fc@@firstFfrench
2652 \fc@french@common
2653 \let\@seventies=\@seventiesfrenchswiss
2654 \let\@eighties=\@eightiesfrenchswiss
2655 \let\@nineties=\@ninetiesfrenchswiss
2656 \@ordinalstringfrench{#1}{#2}%
2657 }
2658 \DeclareRobustCommand{\@ordinalstringFfrenchfrance}[2]{%
2659 \let\fc@case\fc@CaseIden
2660 \let\fc@first=\fc@@firstFfrench
2661 \fc@french@common
2662 \let\@seventies=\@seventiesfrench
2663 \let\@eighties=\@eightiesfrench
2664 \let\@nineties=\@ninetiesfrench
2665 \@ordinalstringfrench{#1}{#2}%
2666 }
2667 \DeclareRobustCommand{\@ordinalstringFfrenchbelgian}[2]{%
2668 \let\fc@case\fc@CaseIden
2669 \let\fc@first=\fc@@firstFfrench
2670 \fc@french@common
2671 \let\@seventies=\@seventiesfrench
2672 \let\@eighties=\@eightiesfrench
2673 \let\@nineties=\@ninetiesfrench

```

```

2674 \@@ordinalstringfrench{#1}{#2}%
2675 }
2676 \let\@ordinalstringFfrench=\@ordinalstringFfrenchfrance
2677 \let\@ordinalstringNfrench\@ordinalstringMfrench
2678 \DeclareRobustCommand{\@OrdinalstringMfrenchswiss}[2]{%
2679 \let\fc@case\fc@UpperCaseFirstLetter
2680 \let\fc@first=\fc@@firstfrench
2681 \fc@french@common
2682 \let\@seventies=\@seventiesfrenchswiss
2683 \let\@eighties=\@eightiesfrenchswiss
2684 \let\@nineties=\@ninetiesfrenchswiss
2685 \@@ordinalstringfrench{#1}{#2}%
2686 }
2687 \DeclareRobustCommand{\@OrdinalstringMfrenchfrance}[2]{%
2688 \let\fc@case\fc@UpperCaseFirstLetter
2689 \let\fc@first=\fc@@firstfrench
2690 \fc@french@common
2691 \let\@seventies=\@seventiesfrench
2692 \let\@eighties=\@eightiesfrench
2693 \let\@nineties=\@ninetiesfrench
2694 \@@ordinalstringfrench{#1}{#2}%
2695 }
2696 \DeclareRobustCommand{\@OrdinalstringMfrenchbelgian}[2]{%
2697 \let\fc@case\fc@UpperCaseFirstLetter
2698 \let\fc@first=\fc@@firstfrench
2699 \fc@french@common
2700 \let\@seventies=\@seventiesfrench
2701 \let\@eighties=\@eightiesfrench
2702 \let\@nineties=\@ninetiesfrench
2703 \@@ordinalstringfrench{#1}{#2}%
2704 }
2705 \let\@OrdinalstringMfrench=\@OrdinalstringMfrenchfrance
2706 \DeclareRobustCommand{\@OrdinalstringFfrenchswiss}[2]{%
2707 \let\fc@case\fc@UpperCaseFirstLetter
2708 \let\fc@first=\fc@@firstfrench
2709 \fc@french@common
2710 \let\@seventies=\@seventiesfrenchswiss
2711 \let\@eighties=\@eightiesfrenchswiss
2712 \let\@nineties=\@ninetiesfrenchswiss
2713 \@@ordinalstringfrench{#1}{#2}%
2714 }
2715 \DeclareRobustCommand{\@OrdinalstringFfrenchfrance}[2]{%
2716 \let\fc@case\fc@UpperCaseFirstLetter
2717 \let\fc@first=\fc@@firstFfrench
2718 \fc@french@common
2719 \let\@seventies=\@seventiesfrench
2720 \let\@eighties=\@eightiesfrench
2721 \let\@nineties=\@ninetiesfrench
2722 \@@ordinalstringfrench{#1}{#2}%

```

```

2723 }
2724 \DeclareRobustCommand{\@OrdinalstringFfrenchbelgian}[2]{%
2725 \let\fc@case\fc@UpperCaseFirstLetter
2726 \let\fc@first=\fc@@firstFfrench
2727 \fc@french@common
2728 \let\@seventies=\@seventiesfrench
2729 \let\@eighties=\@eightiesfrench
2730 \let\@nineties=\@ninetiesfrench
2731 \@ordinalstringfrench{#1}{#2}%
2732 }
2733 \let\@OrdinalstringFfrench=\@OrdinalstringFfrenchfrance
2734 \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 vintg, whichever is applicable. First check that the macro is not yet defined.

```

2735 \@ifundefined{fc@@do@plural@mark}{\PackageError{fmtcount}{Duplicate definition}{Redefinit
2736 'fc@@do@plural@mark'}}

```

Arguments as follows:

#1 plural mark, 's' in general, but for mil it is  
\fc@frenchoptions@mil@plural@mark1

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 vintg, 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.

```

2737 \def\fc@@do@plural@mark#1{%
2738 \ifcase\count10 %
2739 #1% 0=always
2740 \or% 1=never
2741 \or% 2=multiple
2742 \ifnum\count1>1 %
2743 #1%
2744 \fi
2745 \or% 3= multiple g-last
2746 \ifnum\count1>1 %
2747 \ifnum\count0=\count6 %
2748 #1%
2749 \fi
2750 \fi
2751 \or% 4= multiple l-last

```

```

2752     \ifnum\count1>1 %
2753         \ifnum\count9=1 %
2754             \else
2755                 #1%
2756             \fi
2757         \fi
2758 \or% 5= multiple lng-last
2759     \ifnum\count1>1 %
2760         \ifnum\count9=1 %
2761             \else
2762                 \if\count0>\count6 %
2763                     #1%
2764                 \fi
2765             \fi
2766         \fi
2767 \or% 6= multiple ng-last
2768     \ifnum\count1>1 %
2769         \ifnum\count0>\count6 %
2770             #1%
2771         \fi
2772     \fi
2773 \fi
2774 }

\fc  @@nbrstr@Fpreamble Macro \fc@@nbrstr@Fpreamble do the necessary pre-
liminaries before formatting a cardinal with feminine gender.

2775 \ifundefined{fc@@nbrstr@Fpreamble}{%
2776     \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2777         'fc@@nbrstr@Fpreamble'}}

\fc  @@nbrstr@Fpreamble
2778 \def\fc@@nbrstr@Fpreamble{%
2779     \fc@read@unit{\count1}{0}%
2780     \ifnum\count1=1 %
2781         \let\fc@case@save\fc@case
2782         \def\fc@case{\noexpand\fc@case}%
2783         \def\@nil{\noexpand\@nil}%
2784         \let\fc@nbrstr@postamble\fc@@nbrstr@Fpostamble
2785     \fi
2786 }

\fc  @@nbrstr@Fpostamble
2787 \def\fc@@nbrstr@Fpostamble{%
2788     \let\fc@case\fc@case@save
2789     \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
2790     \def\@tempd{un}%
2791     \ifx\@tempc\@tempd
2792         \let\@tempc\@tempa
2793         \edef\@tempa{\@tempb\fc@case une\@nil}%
2794     \fi
2795 }

```



`\fc` `@@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.

```

2796 \ifundefined{fc@@pot@longscalefrench}{%
2797   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2798     '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

```

2799 \def\fc@@pot@longscalefrench#1#2#3{%
2800   {%

```

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

```

2801   \edef\@tempb{\number#1}%

```

Let `\count1` be the plural value.

```

2802   \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  $r$ .

```

2803   \count2\count0 %
2804   \divide\count2 by 6 %
2805   \count3\count2 %
2806   \multiply\count3 by 6 %
2807   \count3-\count3 %
2808   \advance\count3 by \count0 %
2809   \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)”.

```

2810   \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.

```

2811   \edef\@temp{%
2812     \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.

```
2813          1%
2814      \else
2815      \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`.

```
2816      \ifnum\fc@longscale@nilliard@upto=0 %
```

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

```
2817          2%
2818      \else
```

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

```
2819      \ifnum\count2>\fc@longscale@nilliard@upto
2820          1%
2821      \else
2822          2%
2823      \fi
2824      \fi
2825      \else
2826          2%
2827      \fi
2828      \fi
2829  }%
2830      \ifnum\@temph=1 %
```

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

```
2831      \count10=\fc@frenchoptions@mil@plural\space
2832      \edef\@tempe{%
2833          \noexpand\fc@case
2834          mil%
2835          \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
2836          \noexpand\@nil
2837      }%
2838      \else
2839          % weight >= 6
2840          \expandafter\fc@@latin@cardinal@pefix\expandafter{\the\count2}\@tempg
2841          % now form the xxx-illion(s) or xxx-illiard(s) word
2842          \ifnum\count3>2 %
2843              \toks10{illiard}%
2844              \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
2845          \else
2846              \toks10{illion}%
2847              \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2848          \fi
2849      \edef\@tempe{%
```

```

2850      \noexpand\fc@case
2851      \@tempg
2852      \the\toks10 %
2853      \fc@@do@plural@mark s%
2854      \noexpand\@nil
2855      }%
2856      \fi
2857      \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.

```

2858      \let\@tempe\@empty
2859      \def\@temph{0}%
2860      \fi
2861      \else

```

Case of  $w = 0$ .

```

2862      \let\@tempe\@empty
2863      \def\@temph{0}%
2864      \fi

```

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

```

2865      \expandafter\toks\expandafter1\expandafter{\@tempe}%
2866      \toks0{#2}%
2867      \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
2868      \expandafter
2869      }\@tempa
2870 }

```

`\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.

```

2871 \ifundefined{fc@@pot@shortscalefrench}{\}%
2872 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2873   '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

```

2874 \def\fc@@pot@shortscalefrench#1#2#3{%
2875   {%

```

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

```
2876 \edef\@tempb{\number#1}%
```

And let `\count1` be the plural value.

```
2877 \count1=\@tempb
```

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

```
2878 \count2\count0 %
```

```
2879 \divide\count2 by 3 %
```

```
2880 \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 “*<n>*illion(s)|*<n>*illiard(s)”.  
`\ifnum\count0>0` % If  $w \geq 3$ , i.e we do have to append thousand or n-illion(s)/n-illiard

```
2881 \ifnum\count0>0 % If weight>=3, i.e we do have to append thousand or n-illion(s)/n-illiard
2882 \ifnum\count1>0 % we have at least one thousand/n-illion/n-illiard
```

```
2883 \ifnum\count2=0 %
```

```
2884 \def\@temph{1}%
```

```
2885 \count1=\fc@frenchoptions@mil@plural\space
```

```
2886 \edef\@tempe{%
```

```
2887 mil%
```

```
2888 \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
```

```
2889 }%
```

```
2890 \else
```

```
2891 \def\@temph{2}%
```

```
2892 % weight >= 6
```

```
2893 \expandafter\fc@@latin@cardinal@pefix\expandafter{\the\count2}\@tempg
```

```
2894 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
```

```
2895 \edef\@tempe{%
```

```
2896 \noexpand\fc@case
```

```
2897 \@tempg
```

```
2898 illion%
```

```
2899 \fc@@do@plural@mark s%
```

```
2900 \noexpand\@nil
```

```
2901 }%
```

```
2902 \fi
```

```
2903 \else
```

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

```
2904 \def\@temph{0}%
```

```
2905 \let\@tempe\@empty
```

```
2906 \fi
```

```
2907 \else
```

Here  $w = 0$ .

```
2908 \def\@temph{0}%
```

```
2909 \let\@tempe\@empty
```

```

2910 \fi
2911 % now place into \cs{@tempa} the assignment of results \cs{@temph} and \cs{@tempe} to to \
2912 % \texttt{\#3} for further propagation after closing brace.
2913 % \begin{macrocode}
2914 \expandafter\toks\expandafter1\expandafter{\@tempe}%
2915 \toks0{\#2}%
2916 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
2917 \expandafter
2918 }\@tempa
2919 }

```

\fc @@pot@recursivefrench 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.

```

2920 \ifundefined{fc@@pot@recursivefrench}{%
2921 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2922 '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

```

2923 \def\fc@@pot@recursivefrench#1#2#3{%
2924 {%

```

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

```

2925 \edef\@tempb{\number#1}%
2926 \let\@tempa\@tempa

```

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

```

2927 \count1=\@tempb\space

```

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

```

2928 \ifnum\count1>0 %
2929 \count2\count0 %
2930 \divide\count2 by 9 %
2931 \advance\count2 by -1 %
2932 \let\@tempe\@empty
2933 \edef\@tempf{\fc@frenchoptions@supermillion@dos
2934 de\fc@frenchoptions@supermillion@dos\fc@case milliards\@nil}%
2935 \count11\count0 %
2936 \ifnum\count2>0 %

```

```

2937      \count3\count2 %
2938      \count3-\count3 %
2939      \multiply\count3 by 9 %
2940      \advance\count11 by \count3 %
2941      \loop
2942          % (\count2, \count3) <- (\count2 div 2, \count2 mod 2)
2943          \count3\count2 %
2944          \divide\count3 by 2 %
2945          \multiply\count3 by 2 %
2946          \count3-\count3 %
2947          \advance\count3 by \count2 %
2948          \divide\count2 by 2 %
2949          \ifnum\count3=1 %
2950              \let\@tempg\@tempe
2951              \edef\@tempe{\@tempg\@tempf}%
2952          \fi
2953          \let\@tempg\@tempf
2954          \edef\@tempf{\@tempg\@tempg}%
2955          \ifnum\count2>0 %
2956              \repeat
2957          \fi
2958      \divide\count11 by 3 %
2959      \ifcase\count11 % 0 .. 5
2960          % 0 => d milliard(s) (de milliards)*
2961          \def\@temph{2}%
2962          \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
2963      \or % 1 => d mille milliard(s) (de milliards)*
2964          \def\@temph{1}%
2965          \count10=\fc@frenchoptions@mil@plural\space
2966      \or % 2 => d million(s) (de milliards)*
2967          \def\@temph{2}%
2968          \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2969      \or % 3 => d milliard(s) (de milliards)*
2970          \def\@temph{2}%
2971          \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
2972      \or % 4 => d mille milliards (de milliards)*
2973          \def\@temph{1}%
2974          \count10=\fc@frenchoptions@mil@plural\space
2975      \else % 5 => d million(s) (de milliards)*
2976          \def\@temph{2}%
2977          \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2978      \fi
2979      \let\@tempg\@tempe
2980      \edef\@tempf{%
2981          \ifcase\count11 % 0 .. 5
2982          \or
2983              mil\fc@@do@plural@mark \fc@frenchoptions@mil@plural@mark
2984          \or
2985              million\fc@@do@plural@mark s%

```

```

2986      \or
2987      milliard\fc@@do@plural@mark s%
2988      \or
2989      mil\fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
2990      \noexpand\@nil\fc@frenchoptions@supermillion@dos
2991      \noexpand\fc@case milliards% 4
2992      \or
2993      million\fc@@do@plural@mark s%
2994      \noexpand\@nil\fc@frenchoptions@supermillion@dos
2995      de\fc@frenchoptions@supermillion@dos\noexpand\fc@case milliards% 5
2996      \fi
2997      }%
2998      \edef\@tempe{%
2999      \ifx\@tempf\@empty\else
3000      \expandafter\fc@case\@tempf\@nil
3001      \fi
3002      \@tempg
3003      }%
3004      \else
3005      \def\@temph{0}%
3006      \let\@tempe\@empty
3007      \fi

```

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

```

3008      \expandafter\toks\expandafter1\expandafter{\@tempe}%
3009      \toks0{#2}%
3010      \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
3011      \expandafter
3012      }\@tempa
3013 }

```

\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.

```

3014 \ifundefined{fc@muladdfrench}{\%
3015 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3016 '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 “metan-  
 othing” — for index = 0, “mil(le)” — for index = 1 — or by  
 “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)” — for index = 2

```

3017 \def\fc@muladdfrench#1#2#3{%
3018   {%

```

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

```

3019   \edef\@tempc{#1}%
3020   \edef\@tempd{#2}%
3021   \edef\@tempf{#3}%
3022   \let\@tempc\@tempc
3023   \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”.

```

3024   \ifnum\@tempc=1 %
3025     \ifnum\count9=1 %
3026       \let\@tempd\empty
3027     \fi
3028   \fi

```

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

```

3029   \edef\@tempg{%
3030     \@tempd
3031     \ifx\@tempd\empty\else
3032       \ifx\@tempf\empty\else
3033         \ifcase\count9 %
3034           \or
3035             \fc@frenchoptions@submillion@dos
3036           \or
3037             \fc@frenchoptions@supermillion@dos
3038           \fi
3039         \fi
3040       \fi
3041     \@tempf
3042   }%

```

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



```

3043 \edef\@temph{%
3044 \@tempa
3045 \ifx\@tempa\@empty\else
3046 \ifx\@tempg\@empty\else
3047 \ifcase\count8 %
3048 \or
3049 \fc@frenchoptions@submillion@dos
3050 \or
3051 \fc@frenchoptions@supermillion@dos
3052 \fi
3053 \fi
3054 \fi
3055 \@tempg
3056 }%

```

Now propagate the result — i.e. the expansion of `\@temph` — into macro `\@tempa` after closing brace.

```

3057 \def\@tempb##1{\def\@tempa{\def\@tempa{##1}}}%
3058 \expandafter\@tempb\expandafter{\@temph}%
3059 \expandafter
3060 }\@tempa
3061 }%

```

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

```

3062 \@ifundefined{fc@lthundredstringfrench}{}{%
3063 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3064 'fc@lthundredstringfrench'}}

```

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

#1 input/output macro to which append the result

Implicit input arguments as follows:

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

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

```

3065 \def\fc@lthundredstringfrench#1{%
3066 {%

```

First save arguments into local temporary macro.

```

3067 \let\@tempc#1%

```

Read units  $d_w$  to `\count1`.

```

3068 \fc@read@unit{\count1}{\count0}%

```

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

```

3069 \count3\count0 %
3070 \advance\count3 1 %
3071 \fc@read@unit{\count2}{\count3}%

```

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

```

3072 \edef\@tempa{%
3073   \@tempc
3074   \ifnum\count2>1 %
3075     % 20 .. 99
3076     \ifnum\count2>6 %
3077       % 70 .. 99
3078       \ifnum\count2<8 %
3079         % 70 .. 79
3080         \@seventies{\count1}%
3081       \else
3082         % 80..99
3083         \ifnum\count2<9 %
3084           % 80 .. 89
3085           \@eighties{\count1}%
3086         \else
3087           % 90 .. 99
3088           \@nineties{\count1}%
3089         \fi
3090       \fi
3091     \else
3092       % 20..69
3093       \@tenstring{\count2}%
3094       \ifnum\count1>0 %
3095         % x1 .. x0
3096         \ifnum\count1=1 %
3097           % x1
3098           \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
3099         \else
3100           % x2 .. x9
3101           -%
3102         \fi
3103         \@unitstring{\count1}%
3104       \fi
3105     \fi
3106   \else
3107     % 0 .. 19
3108     \ifnum\count2=0 % when tens = 0
3109       % 0 .. 9
3110       \ifnum\count1=0 % when units = 0
3111         % \count3=1 when #1 = 0, i.e. only for the unit of the top level number
3112         \ifnum\count3=1 %
3113           \ifnum\fc@max@weight=0 %
3114             \@unitstring{0}%
3115           \fi
3116         \fi
3117       \else
3118         % 1 .. 9
3119         \@unitstring{\count1}%

```

```

3120         \fi
3121     \else
3122         % 10 .. 19
3123         \@teenstring{\count1}%
3124     \fi
3125 \fi
3126 }%

```

Now propagate the expansion of \@tempa into #2 after closing brace.

```

3127     \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
3128     \expandafter\@tempb\expandafter{\@tempa}%
3129     \expandafter
3130 } \@tempa
3131 }

```

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

```

3132 \@ifundefined{fc@ltthousandstringfrench}{\}%
3133 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3134 'fc@ltthousandstringfrench'}}

```

Output is empty for 0. Arguments as follows:

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

Implicit input arguments as follows:

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

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

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

```

3135 \def\fc@ltthousandstringfrench#1{%
3136     {%

```

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

```

3137     \count4\count0 %
3138     \advance\count4 by 2 %
3139     \fc@read@unit{\count2 }{\count4 }%

```

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

```

3140     \advance\count4 by -1 %
3141     \count3\count4 %
3142     \advance\count3 by -1 %
3143     \fc@check@nonzeros{\count3 }{\count4 } \@tempa

```

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

```

3144     \edef\@temps{%
3145         \ifcase\fc@frenchoptions@cent@plural\space
3146         % 0 => always
3147         s%
3148         \or
3149         % 1 => never
3150         \or

```

```

3151 % 2 => multiple
3152 \ifnum\count2>1s\fi
3153 \or
3154 % 3 => multiple g-last
3155 \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count0=\count6s\fi\fi\fi
3156 \or
3157 % 4 => multiple l-last
3158 \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count9=0s\else\ifnum\count9=2s\fi\fi\fi\fi
3159 \fi
3160 }%
3161 % compute spacing after cent(s?) into \@tempb
3162 \expandafter\let\expandafter\@tempb
3163 \ifnum\@tempa>0 \fc@frenchoptions@submillion@dos\else\@empty\fi
3164 % now place into \@tempa the hundreds
3165 \edef\@tempa{%
3166 \ifnum\count2=0 %
3167 \else
3168 \ifnum\count2=1 %
3169 \expandafter\fc@case\@hundred\@nil
3170 \else
3171 \@unitstring{\count2}\fc@frenchoptions@submillion@dos
3172 \noexpand\fc@case\@hundred\@temps\noexpand\@nil
3173 \fi
3174 \@tempb
3175 \fi
3176 }%
3177 % now append to \@tempa the ten and unit
3178 \fc@lthundredstringfrench\@tempa

```

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

```

3179 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
3180 \expandafter\@tempb\expandafter{\@tempa}%
3181 \expandafter
3182 }\@tempa
3183 }

```

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

```

3184 \@ifundefined{@@numberstringfrench}{-}{%
3185 \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

```

3186 \def\@@numberstringfrench#1#2{%
3187 {%

```

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

```

3188 \edef\@tempa{#1}%
3189 \expandafter\fc@number@parser\expandafter{\@tempa}%

```

```

3190 \ifnum\fc@min@weight<0 %
3191 \PackageError{fmtcount}{Out of range}%
3192 {This macro does not work with fractional numbers}%
3193 \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’.

```

3194 \edef\@tempa{\ifcase\fc@sign@case\space\or\fc@case plus\@nil\or\fc@case moins\@nil\fi}%
3195 \fc@nbrstr@preamble
3196 \fc@@nbrstrfrench@inner
3197 \fc@nbrstr@postamble

```

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

```

3198 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
3199 \expandafter\@tempb\expandafter{\@tempa}%
3200 \expandafter
3201 }\@tempa
3202 }

```

\fc @@nbrstrfrench@inner Common part of @@numberstringfrench and @@ordinalstringfrench.

Arguments are as follows:

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

```

3203 \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.

```

3204 \count0=\fc@max@weight
3205 \divide\count0 by 3 %
3206 \multiply\count0 by 3 %

```

Now we compute final weight into \count5, and round down too multiple of 3 into \count6. Warning: \count6 is an implicit input argument to macro \fc@ltthousandstringfrench.

```

3207 \fc@intpart@find@last{\count5 }%
3208 \count6\count5 %
3209 \divide\count6 3 %
3210 \multiply\count6 3 %
3211 \count8=0 %
3212 \loop

```

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

```

3213 \count1\count0 %
3214 \advance\count1 by 2 %
3215 \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.

```
3216 \fc@poweroften\@tempt{\count9 }\@tempb
```

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

```
3217 \fc@lthousandstringfrench\@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  $10^w$ .

```
3218 \fc@muladdfrench\@tempt\@tempd\@tempb
```

Then iterate.

```
3219 \count8\count9 %
```

```
3220 \advance\count0 by -3 %
```

```
3221 \ifnum\count6>\count0 \else
```

```
3222 \repeat
```

```
3223 }
```

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

```
3224 \ifdefined{@@ordinalstringfrench}\{ }\{ %
```

```
3225 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
```

```
3226 '@@ordinalstringfrench'}}
```

Arguments are as follows:

#1 number to convert to string

#2 macro into which to place the result

```
3227 \def\@@ordinalstringfrench#1#2{%
```

```
3228 {%
```

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

```
3229 \edef\@tempa{#1}%
```

```
3230 \expandafter\fc@number@parser\expandafter{\@tempa}%
```

```
3231 \ifnum\fc@min@weight<0 %
```

```
3232 \PackageError{fmtcount}{Out of range}%
```

```
3233 {This macro does not work with fractional numbers}%
```

```
3234 \fi
```

```
3235 \ifnum\fc@sign@case>0 %
```

```
3236 \PackageError{fmtcount}{Out of range}%
```

```
3237 {This macro does with negative or explicitly marked as positive numbers}%
```

```
3238 \fi
```

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

```
3239 \ifnum\fc@max@weight=0 %
```

```
3240 \ifnum\csname fc@digit0\endcsname=1 %
```

```
3241 \count0=1 %
```

```
3242 \else
```

```
3243 \count0=0 %
```

```

3244     \fi
3245   \else
3246     \count0=0 %
3247   \fi
3248   \ifnum\count0=1 %
3249     \edef\@tempa{\expandafter\fc@case\fc@first\@nil}%
3250   \else

```

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

```

3251     \def\@tempa##1{%
3252       \expandafter\edef\csname fc@frenchoptions@##1@plural\endcsname{%
3253         \ifcase\csname fc@frenchoptions@##1@plural\endcsname\space
3254           0% 0: always => always
3255         \or
3256           1% 1: never => never
3257         \or
3258           6% 2: multiple => multiple ng-last
3259         \or
3260           1% 3: multiple g-last => never
3261         \or
3262           5% 4: multiple l-last => multiple lng-last
3263         \or
3264           5% 5: multiple lng-last => multiple lng-last
3265         \or
3266           6% 6: multiple ng-last => multiple ng-last
3267         \fi
3268       }%
3269     }%
3270     \@tempa{vingt}%
3271     \@tempa{cent}%
3272     \@tempa{mil}%
3273     \@tempa{n-illion}%
3274     \@tempa{n-illiard}%

```

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

```

3275     \let\fc@case@save\fc@case
3276     \def\fc@case{\noexpand\fc@case}%
3277     \def\@nil{\noexpand\@nil}%

```

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

```

3278     \let\@tempa\@empty
3279     \fc@@nbrstrfrench@inner

```

Now restore \fc@case

```

3280     \let\fc@case\fc@case@save

```

Now we add the “ième” ending

```

3281     \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
3282     \expandafter\fc@get@last@letter\expandafter{\@tempc}\@tempd\@tempe
3283     \def\@tempf{e}%

```

```

3284 \ifx\@tempe\@tempf
3285 \edef\@tempa{\@tempb\expandafter\fc@case\@tempd i\'eme\@nil}%
3286 \else
3287 \def\@tempf{q}%
3288 \ifx\@tempe\@tempf
3289 \edef\@tempa{\@tempb\expandafter\fc@case\@tempd qui\'eme\@nil}%
3290 \else
3291 \def\@tempf{f}%
3292 \ifx\@tempe\@tempf
3293 \edef\@tempa{\@tempb\expandafter\fc@case\@tempd vi\'eme\@nil}%
3294 \else
3295 \edef\@tempa{\@tempb\expandafter\fc@case\@tempc i\'eme\@nil}%
3296 \fi
3297 \fi
3298 \fi
3299 \fi

```

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

```

3300 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
3301 \expandafter\@tempb\expandafter{\@tempa}%
3302 \expandafter
3303 }\@tempa
3304 }

```

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

```

3305 \newcommand*\fc@frenchoptions@setdefaults{%
3306 \csname KV@fcfrench@all plural\endcsname{reformed}%
3307 \def\fc@frenchoptions@submillion@dos{-}%
3308 \let\fc@frenchoptions@supermillion@dos\space
3309 \let\fc@u@in@duo\@empty% Could be 'u'
3310 % \let\fc@poweroften\fc@pot@longscalefrench
3311 \let\fc@poweroften\fc@pot@recursivefrench
3312 \def\fc@longscale@nilliard@upto{0}% infinity
3313 \def\fc@frenchoptions@mil@plural@mark{le}%
3314 }
3315 \fc@frenchoptions@setdefaults

```

#### 9.4.6 fc-frenchb.def

```

3316 \ProvidesFCLanguage{frenchb}[2013/08/17]%
3317 \FCloadlang{french}%

```

Set frenchb to be equivalent to french.

```

3318 \global\let\@ordinalMfrenchb=\@ordinalMfrench
3319 \global\let\@ordinalFfrenchb=\@ordinalFfrench
3320 \global\let\@ordinalNfrenchb=\@ordinalNfrench
3321 \global\let\@numberstringMfrenchb=\@numberstringMfrench
3322 \global\let\@numberstringFfrenchb=\@numberstringFfrench

```



```

3323 \global\let\@numberstringNfrenchb=\@numberstringNfrench
3324 \global\let\@NumberstringMfrenchb=\@NumberstringMfrench
3325 \global\let\@NumberstringFfrenchb=\@NumberstringFfrench
3326 \global\let\@NumberstringNfrenchb=\@NumberstringNfrench
3327 \global\let\@ordinalstringMfrenchb=\@ordinalstringMfrench
3328 \global\let\@ordinalstringFfrenchb=\@ordinalstringFfrench
3329 \global\let\@ordinalstringNfrenchb=\@ordinalstringNfrench
3330 \global\let\@OrdinalstringMfrenchb=\@OrdinalstringMfrench
3331 \global\let\@OrdinalstringFfrenchb=\@OrdinalstringFfrench
3332 \global\let\@OrdinalstringNfrenchb=\@OrdinalstringNfrench

```

#### 9.4.7 fc-german.def

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

```

3333 \ProvidesFCLanguage{german}[2013/08/17]%

```

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:

```

3334 \newcommand{\@ordinalMgerman}[2]{%
3335   \edef#2{\number#1\relax.}%
3336 }%
3337 \global\let\@ordinalMgerman\@ordinalMgerman

```

Feminine:

```

3338 \newcommand{\@ordinalFgerman}[2]{%
3339   \edef#2{\number#1\relax.}%
3340 }%
3341 \global\let\@ordinalFgerman\@ordinalFgerman

```

Neuter:

```

3342 \newcommand{\@ordinalNgerman}[2]{%
3343   \edef#2{\number#1\relax.}%
3344 }%
3345 \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):

```

3346 \gdef\@@unitstringgerman#1{%
3347   \ifcase#1%
3348     null%
3349     \or eins%
3350     \or zwei%
3351     \or drei%
3352     \or vier%
3353     \or f\"unf%
3354     \or sechs%
3355     \or sieben%
3356     \or acht%
3357     \or neun%

```

```

3358 \fi
3359 }%

```

Tens (argument must go from 1 to 10):

```

3360 \gdef\@tenstringgerman#1{%
3361   \ifcase#1%
3362     \or zehn%
3363     \or zwanzig%
3364     \or drei{\ss}ig%
3365     \or vierzig%
3366     \or f\unfzig%
3367     \or sechzig%
3368     \or siebzig%
3369     \or achtzig%
3370     \or neunzig%
3371     \or einhundert%
3372 \fi
3373 }%

```

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

```

3374 \providecommand*\einhundert{\einhundert}%
3375 \providecommand*\eintausend{\eintausend}%
3376 \global\let\einhundert\einhundert
3377 \global\let\eintausend\eintausend

```

Teens:

```

3378 \gdef\@teenstringgerman#1{%
3379   \ifcase#1%
3380     zehn%
3381     \or elf%
3382     \or zw\olf%
3383     \or dreizehn%
3384     \or vierzehn%
3385     \or f\unfzehn%
3386     \or sechzehn%
3387     \or siebzehn%
3388     \or achtzehn%
3389     \or neunzehn%
3390 \fi
3391 }%

```

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

```

3392 \DeclareRobustCommand{\@numberstringMgerman}[2]{%
3393   \let\@unitstring=\@unitstringgerman
3394   \let\@teenstring=\@teenstringgerman
3395   \let\@tenstring=\@tenstringgerman
3396   \@numberstringgerman{#1}{#2}%
3397 }%
3398 \global\let\@numberstringMgerman\@numberstringMgerman

```

Feminine and neuter forms:

```
3399 \global\let\@numberstringFgerman=\@numberstringMgerman
3400 \global\let\@numberstringNgerman=\@numberstringMgerman
```

As above, but initial letters in upper case:

```
3401 \DeclareRobustCommand{\@NumberstringMgerman}[2]{%
3402   \@numberstringMgerman{#1}{\@num@str}%
3403   \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3404 }%
3405 \global\let\@NumberstringMgerman\@NumberstringMgerman
```

Feminine and neuter form:

```
3406 \global\let\@NumberstringFgerman=\@NumberstringMgerman
3407 \global\let\@NumberstringNgerman=\@NumberstringMgerman
```

As above, but for ordinals.

```
3408 \DeclareRobustCommand{\@OrdinalstringMgerman}[2]{%
3409   \let\@unitthstring=\@unitthstringMgerman
3410   \let\@teenthstring=\@teenthstringMgerman
3411   \let\@tenthstring=\@tenthstringMgerman
3412   \let\@unitstring=\@unitstringgerman
3413   \let\@teenstring=\@teenstringgerman
3414   \let\@tenstring=\@tenstringgerman
3415   \def\@thousandth{tausendster}%
3416   \def\@hundredth{hundertster}%
3417   \@@ordinalstringgerman{#1}{#2}%
3418 }%
3419 \global\let\@OrdinalstringMgerman\@OrdinalstringMgerman
```

Feminine form:

```
3420 \DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
3421   \let\@unitthstring=\@unitthstringFgerman
3422   \let\@teenthstring=\@teenthstringFgerman
3423   \let\@tenthstring=\@tenthstringFgerman
3424   \let\@unitstring=\@unitstringgerman
3425   \let\@teenstring=\@teenstringgerman
3426   \let\@tenstring=\@tenstringgerman
3427   \def\@thousandth{tausendste}%
3428   \def\@hundredth{hundertste}%
3429   \@@ordinalstringgerman{#1}{#2}%
3430 }%
3431 \global\let\@OrdinalstringFgerman\@OrdinalstringFgerman
```

Neuter form:

```
3432 \DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
3433   \let\@unitthstring=\@unitthstringNgerman
3434   \let\@teenthstring=\@teenthstringNgerman
3435   \let\@tenthstring=\@tenthstringNgerman
3436   \let\@unitstring=\@unitstringgerman
3437   \let\@teenstring=\@teenstringgerman
3438   \let\@tenstring=\@tenstringgerman
```

```

3439 \def\@thousandth{tausendstes}%
3440 \def\@hundredth{hunderstes}%
3441 \@ordinalstringgerman{#1}{#2}%
3442 }%
3443 \global\let\@ordinalstringNgerman\@ordinalstringNgerman

```

As above, but with initial letters in upper case.

```

3444 \DeclareRobustCommand{\@OrdinalstringMgerman}[2]{%
3445 \@ordinalstringMgerman{#1}{\@num@str}%
3446 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3447 }%
3448 \global\let\@OrdinalstringMgerman\@OrdinalstringMgerman

```

Feminine form:

```

3449 \DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
3450 \@ordinalstringFgerman{#1}{\@num@str}%
3451 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3452 }%
3453 \global\let\@OrdinalstringFgerman\@OrdinalstringFgerman

```

Neuter form:

```

3454 \DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
3455 \@ordinalstringNgerman{#1}{\@num@str}%
3456 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3457 }%
3458 \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:

```

3459 \gdef\@unitthstringMgerman#1{%
3460 \ifcase#1%
3461 nullter%
3462 \or erster%
3463 \or zweiter%
3464 \or dritter%
3465 \or vierter%
3466 \or f\"unfter%
3467 \or sechster%
3468 \or siebter%
3469 \or achter%
3470 \or neunter%
3471 \fi
3472 }%

```

Tens:

```

3473 \gdef\@tenthstringMgerman#1{%
3474 \ifcase#1%
3475 \or zehnter%
3476 \or zwanzigster%
3477 \or drei{\ss}igster%
3478 \or vierzigster%

```

```

3479 \or f\"unfzigster%
3480 \or sechzigster%
3481 \or siebzigster%
3482 \or achtzigster%
3483 \or neunzigster%
3484 \fi
3485 }%

```

#### Teens:

```

3486 \gdef\@@teenthstringMgerman#1{%
3487 \ifcase#1%
3488 zehnter%
3489 \or elfter%
3490 \or zw\"olfter%
3491 \or dreizehnter%
3492 \or vierzehnter%
3493 \or f\"unfzehnter%
3494 \or sechzehnter%
3495 \or siebzehnter%
3496 \or achtzehnter%
3497 \or neunzehnter%
3498 \fi
3499 }%

```

#### Units (feminine):

```

3500 \gdef\@@unitthstringFgerman#1{%
3501 \ifcase#1%
3502 nullte%
3503 \or erste%
3504 \or zweite%
3505 \or dritte%
3506 \or vierte%
3507 \or f\"unfte%
3508 \or sechste%
3509 \or siebte%
3510 \or achte%
3511 \or neunte%
3512 \fi
3513 }%

```

#### Tens (feminine):

```

3514 \gdef\@@tenthstringFgerman#1{%
3515 \ifcase#1%
3516 \or zehnte%
3517 \or zwanzigste%
3518 \or drei{\ss}igste%
3519 \or vierzigste%
3520 \or f\"unfzigste%
3521 \or sechzigste%
3522 \or siebzigste%
3523 \or achtzigste%

```

```

3524     \or neunzigste%
3525     \fi
3526 }%

```

#### Teens (feminine)

```

3527 \gdef\@@teenthstringFgerman#1{%
3528     \ifcase#1%
3529         zehnte%
3530         \or elfte%
3531         \or zw\olfte%
3532         \or dreizehnte%
3533         \or vierzehnte%
3534         \or f\unfzehnte%
3535         \or sechzehnte%
3536         \or siebzehnte%
3537         \or achtzehnte%
3538         \or neunzehnte%
3539     \fi
3540 }%

```

#### Units (neuter):

```

3541 \gdef\@@unitthstringNgerman#1{%
3542     \ifcase#1%
3543         nulltes%
3544         \or erstes%
3545         \or zweites%
3546         \or drittes%
3547         \or viertes%
3548         \or f\unftes%
3549         \or sechstes%
3550         \or siebtes%
3551         \or achtes%
3552         \or neuntes%
3553     \fi
3554 }%

```

#### Tens (neuter):

```

3555 \gdef\@@tenthstringNgerman#1{%
3556     \ifcase#1%
3557         \or zehntes%
3558         \or zwanzigstes%
3559         \or drei{\ss}igstes%
3560         \or vierzigstes%
3561         \or f\unfzigstes%
3562         \or sechzigstes%
3563         \or siebzigstes%
3564         \or achtzigstes%
3565         \or neunzigstes%
3566     \fi
3567 }%

```

Teens (neuter)

```

3568 \gdef\@teenthstringNgerman#1{%
3569   \ifcase#1%
3570     zehntes%
3571     \or elftes%
3572     \or zw\ "olftes%
3573     \or dreizehntes%
3574     \or vierzehntes%
3575     \or f\ "unfzehntes%
3576     \or sechzehntes%
3577     \or siebzehntes%
3578     \or achtzehntes%
3579     \or neunzehntes%
3580   \fi
3581 }%
```

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

```

3582 \gdef\@numberunderhundredgerman#1#2{%
3583 \ifnum#1<10\relax
3584   \ifnum#1>0\relax
3585     \eappto#2{\@unitstring{#1}}%
3586   \fi
3587 \else
3588   \@tmpstrctr=#1\relax
3589   \@modulo{\@tmpstrctr}{10}%
3590   \ifnum#1<20\relax
3591     \eappto#2{\@teenstring{\@tmpstrctr}}%
3592   \else
3593     \ifnum\@tmpstrctr=0\relax
3594     \else
3595       \eappto#2{\@unitstring{\@tmpstrctr}und}%
3596     \fi
3597     \@tmpstrctr=#1\relax
3598     \divide\@tmpstrctr by 10\relax
3599     \eappto#2{\@tenstring{\@tmpstrctr}}%
3600   \fi
3601 \fi
3602 }%
```

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

```

3603 \gdef\@numberstringgerman#1#2{%
3604 \ifnum#1>99999\relax
3605   \PackageError{fmtcount}{Out of range}%
3606   {This macro only works for values less than 100000}%
3607 \else
3608   \ifnum#1<0\relax
3609     \PackageError{fmtcount}{Negative numbers not permitted}%
3610     {This macro does not work for negative numbers, however
```

```

3611     you can try typing "minus" first, and then pass the modulus of
3612     this number}%
3613   \fi
3614 \fi
3615 \def#2{}%
3616 \@strctr=#1\relax \divide\@strctr by 1000\relax
3617 \ifnum\@strctr>1\relax
    #1 is  $\geq 2000$ , \@strctr now contains the number of thousands
3618 \@@numberunderhundredgerman{\@strctr}{#2}%
3619 \appto#2{tausend}%
3620 \else
    #1 lies in range [1000,1999]
3621   \ifnum\@strctr=1\relax
3622     \eappto#2{eintausend}%
3623   \fi
3624 \fi
3625 \@strctr=#1\relax
3626 \@modulo{\@strctr}{1000}%
3627 \divide\@strctr by 100\relax
3628 \ifnum\@strctr>1\relax
    now dealing with number in range [200,999]
3629   \eappto#2{\@unitstring{\@strctr}hundert}%
3630 \else
3631   \ifnum\@strctr=1\relax
    dealing with number in range [100,199]
3632     \ifnum#1>1000\relax
    if original number > 1000, use einhundert
3633       \appto#2{einhundert}%
3634     \else
    otherwise use \einhundert
3635       \eappto#2{\einhundert}%
3636     \fi
3637   \fi
3638 \fi
3639 \@strctr=#1\relax
3640 \@modulo{\@strctr}{100}%
3641 \ifnum#1=0\relax
3642   \def#2{null}%
3643 \else
3644   \ifnum\@strctr=1\relax
3645     \appto#2{eins}%
3646   \else
3647     \@@numberunderhundredgerman{\@strctr}{#2}%
3648   \fi
3649 \fi
3650 }%

```



As above, but for ordinals

```

3651 \gdef\@@numberunderhundredthgerman#1#2{%
3652 \ifnum#1<10\relax
3653 \eappto#2{\@unitthstring{#1}}%
3654 \else
3655 \@tmpstrctr=#1\relax
3656 \@modulo{\@tmpstrctr}{10}%
3657 \ifnum#1<20\relax
3658 \eappto#2{\@teenthstring{\@tmpstrctr}}%
3659 \else
3660 \ifnum\@tmpstrctr=0\relax
3661 \else
3662 \eappto#2{\@unitstring{\@tmpstrctr}und}%
3663 \fi
3664 \@tmpstrctr=#1\relax
3665 \divide\@tmpstrctr by 10\relax
3666 \eappto#2{\@tenthstring{\@tmpstrctr}}%
3667 \fi
3668 \fi
3669 }%

3670 \gdef\@@ordinalstringgerman#1#2{%
3671 \ifnum#1>99999\relax
3672 \PackageError{fmtcount}{Out of range}%
3673 {This macro only works for values less than 100000}%
3674 \else
3675 \ifnum#1<0\relax
3676 \PackageError{fmtcount}{Negative numbers not permitted}%
3677 {This macro does not work for negative numbers, however
3678 you can try typing "minus" first, and then pass the modulus of
3679 this number}%
3680 \fi
3681 \fi
3682 \def#2{}%
3683 \@strctr=#1\relax \divide\@strctr by 1000\relax
3684 \ifnum\@strctr>1\relax

#1 is  $\geq 2000$ , \@strctr now contains the number of thousands
3685 \@@numberunderhundredgerman{\@strctr}{#2}%

is that it, or is there more?
3686 \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{1000}%
3687 \ifnum\@tmpstrctr=0\relax
3688 \eappto#2{\@thousandth}%
3689 \else
3690 \appto#2{tausend}%
3691 \fi
3692 \else

#1 lies in range [1000,1999]
3693 \ifnum\@strctr=1\relax

```

```

3694 \ifnum#1=1000\relax
3695 \eappto#2{\@thousandth}%
3696 \else
3697 \eappto#2{\eintausend}%
3698 \fi
3699 \fi
3700 \fi
3701 \@strctr=#1\relax
3702 \@modulo{\@strctr}{1000}%
3703 \divide\@strctr by 100\relax
3704 \ifnum\@strctr>1\relax
    now dealing with number in range [200,999] is that it, or is there more?
3705 \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{100}%
3706 \ifnum\@tmpstrctr=0\relax
3707 \ifnum\@strctr=1\relax
3708 \eappto#2{\@hundredth}%
3709 \else
3710 \eappto#2{\@unitstring{\@strctr}\@hundredth}%
3711 \fi
3712 \else
3713 \eappto#2{\@unitstring{\@strctr}hundert}%
3714 \fi
3715 \else
3716 \ifnum\@strctr=1\relax
    dealing with number in range [100,199] is that it, or is there more?
3717 \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{100}%
3718 \ifnum\@tmpstrctr=0\relax
3719 \eappto#2{\@hundredth}%
3720 \else
3721 \ifnum#1>1000\relax
3722 \appto#2{einhundert}%
3723 \else
3724 \eappto#2{\einhundert}%
3725 \fi
3726 \fi
3727 \fi
3728 \fi
3729 \@strctr=#1\relax
3730 \@modulo{\@strctr}{100}%
3731 \ifthenelse{\@strctr=0 \and #1>0}{-}{%
3732 \@numberunderhundredthgerman{\@strctr}{#2}%
3733 }%
3734 }%
    Load fc-germanb.def if not already loaded
3735 \FCloadlang{germanb}%

```

#### 9.4.8 fc-germanb.def

3736 \ProvidesFCLanguage{germanb}[2013/08/17]%

Load fc-german.def if not already loaded

3737 \FCloadlang{german}%

Set germanb to be equivalent to german.

3738 \global\let\@ordinalMgermanb=\@ordinalMgerman

3739 \global\let\@ordinalFgermanb=\@ordinalFgerman

3740 \global\let\@ordinalNgermanb=\@ordinalNgerman

3741 \global\let\@numberstringMgermanb=\@numberstringMgerman

3742 \global\let\@numberstringFgermanb=\@numberstringFgerman

3743 \global\let\@numberstringNgermanb=\@numberstringNgerman

3744 \global\let\@NumberstringMgermanb=\@NumberstringMgerman

3745 \global\let\@NumberstringFgermanb=\@NumberstringFgerman

3746 \global\let\@NumberstringNgermanb=\@NumberstringNgerman

3747 \global\let\@ordinalstringMgermanb=\@ordinalstringMgerman

3748 \global\let\@ordinalstringFgermanb=\@ordinalstringFgerman

3749 \global\let\@ordinalstringNgermanb=\@ordinalstringNgerman

3750 \global\let\@OrdinalstringMgermanb=\@OrdinalstringMgerman

3751 \global\let\@OrdinalstringFgermanb=\@OrdinalstringFgerman

3752 \global\let\@OrdinalstringNgermanb=\@OrdinalstringNgerman

#### 9.4.9 fc-italian

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

3753 \ProvidesFCLanguage{italian}[2013/08/17]

3754

3755 \RequirePackage{itnumpar}

3756

3757 \newcommand{\@numberstringMitalian}[2]{%

3758 \edef#2{\noexpand\printnumeroinparole{#1}}%

3759 }

3760 \global\let\@numberstringMitalian\@numberstringMitalian

3761

3762 \newcommand{\@numberstringFitalian}[2]{%

3763 \edef#2{\noexpand\printnumeroinparole{#1}}%

3764

3765 \global\let\@numberstringFitalian\@numberstringFitalian

3766

3767 \newcommand{\@NumberstringMitalian}[2]{%

3768 \edef#2{\noexpand\printNumeroinparole{#1}}%

3769 }

3770 \global\let\@NumberstringMitalian\@NumberstringMitalian

3771

3772 \newcommand{\@NumberstringFitalian}[2]{%

3773 \edef#2{\noexpand\printNumeroinparole{#1}}%

3774 }

3775 \global\let\@NumberstringFitalian\@NumberstringFitalian

3776

3777 \newcommand{\@ordinalstringMitalian}[2]{%

```

3778 \edef#2{\noexpand\printordinalem{#1}}%
3779 }
3780 \global\let\@ordinalstringMitalian\@ordinalstringMitalian
3781
3782 \newcommand{\@ordinalstringFitalian}[2]{%
3783 \edef#2{\noexpand\printordinalef{#1}}%
3784 }
3785 \global\let\@ordinalstringFitalian\@ordinalstringFitalian
3786
3787 \newcommand{\@OrdinalstringMitalian}[2]{%
3788 \edef#2{\noexpand\printOrdinalem{#1}}%
3789 }
3790 \global\let\@OrdinalstringMitalian\@OrdinalstringMitalian
3791
3792 \newcommand{\@OrdinalstringFitalian}[2]{%
3793 \edef#2{\noexpand\printOrdinalef{#1}}%
3794 }
3795 \global\let\@OrdinalstringFitalian\@OrdinalstringFitalian
3796
3797 \newcommand{\@ordinalMitalian}[2]{%
3798 \edef#2{#1\relax\noexpand\fmtord{o}}%
3799 }
3800 \global\let\@ordinalMitalian\@ordinalMitalian
3801
3802 \newcommand{\@ordinalFitalian}[2]{%
3803 \edef#2{#1\relax\noexpand\fmtord{a}}%
3804 \global\let\@ordinalFitalian\@ordinalFitalian

```

#### 9.4.10 fc-ngerman.def

```

3805 \ProvidesFCLanguage{ngerman}[2012/06/18]%
3806 \FCloadlang{german}%
3807 \FCloadlang{ngermanb}%

```

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

```

3808 \global\let\@ordinalMngerman=\@ordinalMgerman
3809 \global\let\@ordinalFngerman=\@ordinalFgerman
3810 \global\let\@ordinalNngerman=\@ordinalNgerman
3811 \global\let\@numberstringMngerman=\@numberstringMgerman
3812 \global\let\@numberstringFngerman=\@numberstringFgerman
3813 \global\let\@numberstringNngerman=\@numberstringNgerman
3814 \global\let\@NumberstringMngerman=\@NumberstringMgerman
3815 \global\let\@NumberstringFngerman=\@NumberstringFgerman
3816 \global\let\@NumberstringNngerman=\@NumberstringNgerman
3817 \global\let\@ordinalstringMngerman=\@ordinalstringMgerman
3818 \global\let\@ordinalstringFngerman=\@ordinalstringFgerman
3819 \global\let\@ordinalstringNngerman=\@ordinalstringNgerman
3820 \global\let\@OrdinalstringMngerman=\@OrdinalstringMgerman
3821 \global\let\@OrdinalstringFngerman=\@OrdinalstringFgerman

```

```
3822 \global\let\@OrdinalstringNgerman=\@OrdinalstringNgerman
```

#### 9.4.11 fc-ngermanb.def

```
3823 \ProvidesFCLanguage{ngermanb}[2013/08/17]%
3824 \FCloadlang{ngermanb}%
```

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

```
3825 \global\let\@ordinalMngermanb=\@ordinalMgerman
3826 \global\let\@ordinalFngermanb=\@ordinalFgerman
3827 \global\let\@ordinalNgermanb=\@ordinalNgerman
3828 \global\let\@numberstringMngermanb=\@numberstringMgerman
3829 \global\let\@numberstringFngermanb=\@numberstringFgerman
3830 \global\let\@numberstringNgermanb=\@numberstringNgerman
3831 \global\let\@NumberstringMngermanb=\@NumberstringMgerman
3832 \global\let\@NumberstringFngermanb=\@NumberstringFgerman
3833 \global\let\@NumberstringNgermanb=\@NumberstringNgerman
3834 \global\let\@ordinalstringMngermanb=\@ordinalstringMgerman
3835 \global\let\@ordinalstringFngermanb=\@ordinalstringFgerman
3836 \global\let\@ordinalstringNgermanb=\@ordinalstringNgerman
3837 \global\let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
3838 \global\let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
3839 \global\let\@OrdinalstringNgermanb=\@OrdinalstringNgerman
```

Load fc-ngerman.def if not already loaded

```
3840 \FCloadlang{ngermanb}%
```

#### 9.4.12 fc-portuges.def

Portuguese definitions

```
3841 \ProvidesFCLanguage{portuges}[2013/08/17]%
```

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:

```
3842 \gdef\@ordinalMportuges#1#2{%
3843   \ifnum#1=0\relax
3844     \edef#2{\number#1}%
3845   \else
3846     \edef#2{\number#1\relax\noexpand\fmtord{o}}%
3847   \fi
3848 }%
```

Feminine:

```
3849 \gdef\@ordinalFportuges#1#2{%
3850   \ifnum#1=0\relax
3851     \edef#2{\number#1}%
3852   \else
3853     \edef#2{\number#1\relax\noexpand\fmtord{a}}%
3854   \fi
3855 }%
```

Make neuter same as masculine:

```
3856 \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):

```
3857 \gdef\@@unitstringportuges#1#2{%
3858   \ifcase#1\relax
3859     zero%
3860     \or um%
3861     \or dois%
3862     \or tr\^es%
3863     \or quatro%
3864     \or cinco%
3865     \or seis%
3866     \or sete%
3867     \or oito%
3868     \or nove%
3869   \fi
3870 }%
3871 %   \end{macrocode}
3872 % As above, but for feminine:
3873 %   \begin{macrocode}
3874 \gdef\@@unitstringFportuges#1{%
3875   \ifcase#1\relax
3876     zero%
3877     \or uma%
3878     \or duas%
3879     \or tr\^es%
3880     \or quatro%
3881     \or cinco%
3882     \or seis%
3883     \or sete%
3884     \or oito%
3885     \or nove%
3886   \fi
3887 }%
```

Tens (argument must be a number from 0 to 10):

```
3888 \gdef\@@tenstringportuges#1{%
3889   \ifcase#1\relax
3890     \or dez%
3891     \or vinte%
3892     \or trinta%
3893     \or quarenta%
3894     \or cinq\"uenta%
3895     \or sessenta%
3896     \or setenta%
3897     \or oitenta%
3898     \or noventa%
```

```

3899     \or cem%
3900     \fi
3901 }%

```

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

```

3902 \gdef\@@teenstringportuges#1{%
3903     \ifcase#1\relax
3904         dez%
3905         \or onze%
3906         \or doze%
3907         \or treze%
3908         \or quatorze%
3909         \or quinze%
3910         \or dezesesseis%
3911         \or dezessete%
3912         \or dezoito%
3913         \or dezenove%
3914     \fi
3915 }%

```

Hundreds:

```

3916 \gdef\@@hundredstringportuges#1{%
3917     \ifcase#1\relax
3918         \or cento%
3919         \or duzentos%
3920         \or trezentos%
3921         \or quatrocentos%
3922         \or quinhentos%
3923         \or seiscentos%
3924         \or setecentos%
3925         \or oitocentos%
3926         \or novecentos%
3927     \fi
3928 }%

```

Hundreds (feminine):

```

3929 \gdef\@@hundredstringFportuges#1{%
3930     \ifcase#1\relax
3931         \or cento%
3932         \or duzentas%
3933         \or trezentas%
3934         \or quatrocentas%
3935         \or quinhentas%
3936         \or seiscentas%
3937         \or setecentas%
3938         \or oitocentas%
3939         \or novecentas%
3940     \fi
3941 }%

```

Units (initial letter in upper case):

```

3942 \gdef\@@Unitstringportuges#1{%
3943   \ifcase#1\relax
3944     Zero%
3945     \or Um%
3946     \or Dois%
3947     \or Tr\^es%
3948     \or Quatro%
3949     \or Cinco%
3950     \or Seis%
3951     \or Sete%
3952     \or Oito%
3953     \or Nove%
3954   \fi
3955 }%

```

As above, but feminine:

```

3956 \gdef\@@UnitstringFportuges#1{%
3957   \ifcase#1\relax
3958     Zera%
3959     \or Uma%
3960     \or Duas%
3961     \or Tr\^es%
3962     \or Quatro%
3963     \or Cinco%
3964     \or Seis%
3965     \or Sete%
3966     \or Oito%
3967     \or Nove%
3968   \fi
3969 }%

```

Tens (with initial letter in upper case):

```

3970 \gdef\@@Tenstringportuges#1{%
3971   \ifcase#1\relax
3972     \or Dez%
3973     \or Vinte%
3974     \or Trinta%
3975     \or Quarenta%
3976     \or Cinq\"uenta%
3977     \or Sessenta%
3978     \or Setenta%
3979     \or Oitenta%
3980     \or Noventa%
3981     \or Cem%
3982   \fi
3983 }%

```

Teens (with initial letter in upper case):

```

3984 \gdef\@@Teenstringportuges#1{%
3985   \ifcase#1\relax
3986     Dez%

```



```

3987 \or Onze%
3988 \or Doze%
3989 \or Treze%
3990 \or Quatorze%
3991 \or Quinze%
3992 \or Dezesesseis%
3993 \or Dezessete%
3994 \or Dezoito%
3995 \or Dezenove%
3996 \fi
3997 }%

```

Hundreds (with initial letter in upper case):

```

3998 \gdef\@@Hundredstringportuges#1{%
3999 \ifcase#1\relax
4000 \or Cento%
4001 \or Duzentos%
4002 \or Trezentos%
4003 \or Quatrocentos%
4004 \or Quinhentos%
4005 \or Seiscentos%
4006 \or Setecentos%
4007 \or Oitocentos%
4008 \or Novecentos%
4009 \fi
4010 }%

```

As above, but feminine:

```

4011 \gdef\@@HundredstringFportuges#1{%
4012 \ifcase#1\relax
4013 \or Cento%
4014 \or Duzentas%
4015 \or Trezentas%
4016 \or Quatrocentas%
4017 \or Quinhentas%
4018 \or Seiscentas%
4019 \or Setecentas%
4020 \or Oitocentas%
4021 \or Novecentas%
4022 \fi
4023 }%

```

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.)

```

4024 \DeclareRobustCommand{\@numberstringMportuges}[2]{%
4025 \let\@unitstring=\@unitstringportuges
4026 \let\@teenstring=\@teenstringportuges
4027 \let\@tenstring=\@tenstringportuges

```

```

4028 \let\@hundredstring=\@hundredstringportuges
4029 \def\@hundred{cem}\def\@thousand{mil}%
4030 \def\@andname{e}%
4031 \@numberstringportuges{#1}{#2}%
4032 }%
4033 \global\let\@numberstringMportuges\@numberstringMportuges

```

As above, but feminine form:

```

4034 \DeclareRobustCommand{\@numberstringFportuges}[2]{%
4035 \let\@unitstring=\@unitstringFportuges
4036 \let\@teenstring=\@teenstringportuges
4037 \let\@tenstring=\@tenstringportuges
4038 \let\@hundredstring=\@hundredstringFportuges
4039 \def\@hundred{cem}\def\@thousand{mil}%
4040 \def\@andname{e}%
4041 \@numberstringportuges{#1}{#2}%
4042 }%
4043 \global\let\@numberstringFportuges\@numberstringFportuges

```

Make neuter same as masculine:

```

4044 \global\let\@numberstringNportuges\@numberstringMportuges

```

As above, but initial letters in upper case:

```

4045 \DeclareRobustCommand{\@NumberstringMportuges}[2]{%
4046 \let\@unitstring=\@Unitstringportuges
4047 \let\@teenstring=\@Teenstringportuges
4048 \let\@tenstring=\@Tenstringportuges
4049 \let\@hundredstring=\@Hundredstringportuges
4050 \def\@hundred{Cem}\def\@thousand{Mil}%
4051 \def\@andname{e}%
4052 \@numberstringportuges{#1}{#2}%
4053 }%
4054 \global\let\@NumberstringMportuges\@NumberstringMportuges

```

As above, but feminine form:

```

4055 \DeclareRobustCommand{\@NumberstringFportuges}[2]{%
4056 \let\@unitstring=\@UnitstringFportuges
4057 \let\@teenstring=\@Teenstringportuges
4058 \let\@tenstring=\@Tenstringportuges
4059 \let\@hundredstring=\@HundredstringFportuges
4060 \def\@hundred{Cem}\def\@thousand{Mil}%
4061 \def\@andname{e}%
4062 \@numberstringportuges{#1}{#2}%
4063 }%
4064 \global\let\@NumberstringFportuges\@NumberstringFportuges

```

Make neuter same as masculine:

```

4065 \global\let\@NumberstringNportuges\@NumberstringMportuges

```

As above, but for ordinals.

```

4066 \DeclareRobustCommand{\@ordinalstringMportuges}[2]{%
4067 \let\@unitthstring=\@unitthstringportuges

```

```

4068 \let\@unitstring=\@unitstringportuges
4069 \let\@teenthstring=\@teenthstringportuges
4070 \let\@tenthstring=\@tenthstringportuges
4071 \let\@hundredthstring=\@hundredthstringportuges
4072 \def\@thousandth{mil\'esimo}%
4073 \@ordinalstringportuges{#1}{#2}%
4074 }%
4075 \global\let\@ordinalstringMportuges\@ordinalstringMportuges

  Feminine form:
4076 \DeclareRobustCommand{\@ordinalstringFportuges}[2]{%
4077 \let\@unitthstring=\@unitthstringFportuges
4078 \let\@unitstring=\@unitstringFportuges
4079 \let\@teenthstring=\@teenthstringportuges
4080 \let\@tenthstring=\@tenthstringFportuges
4081 \let\@hundredthstring=\@hundredthstringFportuges
4082 \def\@thousandth{mil\'esima}%
4083 \@ordinalstringportuges{#1}{#2}%
4084 }%
4085 \global\let\@ordinalstringFportuges\@ordinalstringFportuges

  Make neuter same as masculine:
4086 \global\let\@ordinalstringNportuges\@ordinalstringMportuges

  As above, but initial letters in upper case (masculine):
4087 \DeclareRobustCommand{\@OrdinalstringMportuges}[2]{%
4088 \let\@unitthstring=\@Unitthstringportuges
4089 \let\@unitstring=\@Unitstringportuges
4090 \let\@teenthstring=\@Teenthstringportuges
4091 \let\@tenthstring=\@Tenthstringportuges
4092 \let\@hundredthstring=\@Hundredthstringportuges
4093 \def\@thousandth{Mil\'esimo}%
4094 \@ordinalstringportuges{#1}{#2}%
4095 }%
4096 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges

  Feminine form:
4097 \DeclareRobustCommand{\@OrdinalstringFportuges}[2]{%
4098 \let\@unitthstring=\@UnitthstringFportuges
4099 \let\@unitstring=\@UnitstringFportuges
4100 \let\@teenthstring=\@Teenthstringportuges
4101 \let\@tenthstring=\@TenthstringFportuges
4102 \let\@hundredthstring=\@HundredthstringFportuges
4103 \def\@thousandth{Mil\'esima}%
4104 \@ordinalstringportuges{#1}{#2}%
4105 }%
4106 \global\let\@OrdinalstringFportuges\@OrdinalstringFportuges

  Make neuter same as masculine:
4107 \global\let\@OrdinalstringNportuges\@OrdinalstringMportuges

  In order to do the ordinals, split into units, teens, tens and hundreds. Units:

```

```

4108 \gdef\@@unitthstringportuges#1{%
4109   \ifcase#1\relax
4110     zero%
4111     \or primeiro%
4112     \or segundo%
4113     \or terceiro%
4114     \or quarto%
4115     \or quinto%
4116     \or sexto%
4117     \or s\'etimo%
4118     \or oitavo%
4119     \or nono%
4120   \fi
4121 }%

```

Tens:

```

4122 \gdef\@@tenthstringportuges#1{%
4123   \ifcase#1\relax
4124     \or d\'ecimo%
4125     \or vig\'esimo%
4126     \or trig\'esimo%
4127     \or quadrag\'esimo%
4128     \or q\'uinquag\'esimo%
4129     \or sexag\'esimo%
4130     \or setuag\'esimo%
4131     \or octog\'esimo%
4132     \or nonag\'esimo%
4133   \fi
4134 }%

```

Teens:

```

4135 \gdef\@@teenthstringportuges#1{%
4136   \@tenthstring{1}%
4137   \ifnum#1>0\relax
4138     -\@unitthstring{#1}%
4139   \fi
4140 }%

```

Hundreds:

```

4141 \gdef\@@hundredthstringportuges#1{%
4142   \ifcase#1\relax
4143     \or cent\'esimo%
4144     \or ducent\'esimo%
4145     \or trecent\'esimo%
4146     \or quadringent\'esimo%
4147     \or q\'uingent\'esimo%
4148     \or seiscent\'esimo%
4149     \or setingent\'esimo%
4150     \or octingent\'esimo%
4151     \or nongent\'esimo%
4152   \fi

```

4153 }%

Units (feminine):

```
4154 \gdef\@@unitthstringFportuges#1{%  
4155   \ifcase#1\relax  
4156     zero%  
4157     \or primeira%  
4158     \or segunda%  
4159     \or terceira%  
4160     \or quarta%  
4161     \or quinta%  
4162     \or sexta%  
4163     \or s\'etima%  
4164     \or oitava%  
4165     \or nona%  
4166   \fi  
4167 }%
```

Tens (feminine):

```
4168 \gdef\@@tenthstringFportuges#1{%  
4169   \ifcase#1\relax  
4170     \or d\'ecima%  
4171     \or vig\'esima%  
4172     \or trig\'esima%  
4173     \or quadrag\'esima%  
4174     \or q\'uinquag\'esima%  
4175     \or sexag\'esima%  
4176     \or setuag\'esima%  
4177     \or octog\'esima%  
4178     \or nonag\'esima%  
4179   \fi  
4180 }%
```

Hundreds (feminine):

```
4181 \gdef\@@hundredthstringFportuges#1{%  
4182   \ifcase#1\relax  
4183     \or cent\'esima%  
4184     \or ducent\'esima%  
4185     \or trecent\'esima%  
4186     \or quadringent\'esima%  
4187     \or q\'uingent\'esima%  
4188     \or seiscent\'esima%  
4189     \or setingent\'esima%  
4190     \or octingent\'esima%  
4191     \or nongent\'esima%  
4192   \fi  
4193 }%
```

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

```
4194 \gdef\@@Unitthstringportuges#1{%  
4195   \ifcase#1\relax
```

```

4196 Zero%
4197 \or Primeiro%
4198 \or Segundo%
4199 \or Terceiro%
4200 \or Quarto%
4201 \or Quinto%
4202 \or Sexto%
4203 \or S\'etimo%
4204 \or Oitavo%
4205 \or Nono%
4206 \fi
4207 }%

```

Tens:

```

4208 \gdef\@@Tenthstringportuges#1{%
4209 \ifcase#1\relax
4210 \or D\'ecimo%
4211 \or Vig\'esimo%
4212 \or Trig\'esimo%
4213 \or Quadrag\'esimo%
4214 \or Q\'uinquag\'esimo%
4215 \or Sexag\'esimo%
4216 \or Setuag\'esimo%
4217 \or Octog\'esimo%
4218 \or Nonag\'esimo%
4219 \fi
4220 }%

```

Hundreds:

```

4221 \gdef\@@Hundredthstringportuges#1{%
4222 \ifcase#1\relax
4223 \or Cent\'esimo%
4224 \or Ducent\'esimo%
4225 \or Trecent\'esimo%
4226 \or Quadringent\'esimo%
4227 \or Q\'uingent\'esimo%
4228 \or Seiscent\'esimo%
4229 \or Setingent\'esimo%
4230 \or Octingent\'esimo%
4231 \or Nongent\'esimo%
4232 \fi
4233 }%

```

As above, but feminine. Units:

```

4234 \gdef\@@UnitthstringFportuges#1{%
4235 \ifcase#1\relax
4236 Zera%
4237 \or Primeira%
4238 \or Segunda%
4239 \or Terceira%
4240 \or Quarta%

```

```

4241 \or Quinta%
4242 \or Sexta%
4243 \or S\'etima%
4244 \or Oitava%
4245 \or Nona%
4246 \fi
4247 }%

```

Tens (feminine);

```

4248 \gdef\@@TenthstringFportuges#1{%
4249 \ifcase#1\relax
4250 \or D\'ecima%
4251 \or Vig\'esima%
4252 \or Trig\'esima%
4253 \or Quadrag\'esima%
4254 \or Q\'uinquag\'esima%
4255 \or Sexag\'esima%
4256 \or Setuag\'esima%
4257 \or Octog\'esima%
4258 \or Nonag\'esima%
4259 \fi
4260 }%

```

Hundreds (feminine):

```

4261 \gdef\@@HundredthstringFportuges#1{%
4262 \ifcase#1\relax
4263 \or Cent\'esima%
4264 \or Ducent\'esima%
4265 \or Trecent\'esima%
4266 \or Quadringent\'esima%
4267 \or Q\'uingent\'esima%
4268 \or Seiscent\'esima%
4269 \or Setingent\'esima%
4270 \or Octingent\'esima%
4271 \or Nongent\'esima%
4272 \fi
4273 }%

```

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.)

```

4274 \gdef\@@numberstringportuges#1#2{%
4275 \ifnum#1>99999
4276 \PackageError{fmtcount}{Out of range}%
4277 {This macro only works for values less than 100000}%
4278 \else
4279 \ifnum#1<0
4280 \PackageError{fmtcount}{Negative numbers not permitted}%
4281 {This macro does not work for negative numbers, however

```

```

4282 you can try typing "minus" first, and then pass the modulus of
4283 this number}%
4284 \fi
4285 \fi
4286 \def#2{}%
4287 \@strctr=#1\relax \divide\@strctr by 1000\relax
4288 \ifnum\@strctr>9
    #1 is greater or equal to 10000
4289 \divide\@strctr by 10
4290 \ifnum\@strctr>1\relax
4291 \let\@@fc@numstr#2\relax
4292 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
4293 \@strctr=#1 \divide\@strctr by 1000\relax
4294 \@modulo{\@strctr}{10}%
4295 \ifnum\@strctr>0
4296 \ifnum\@strctr=1\relax
4297 \let\@@fc@numstr#2\relax
4298 \edef#2{\@@fc@numstr\ \@andname}%
4299 \fi
4300 \let\@@fc@numstr#2\relax
4301 \edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
4302 \fi
4303 \else
4304 \@strctr=#1\relax
4305 \divide\@strctr by 1000\relax
4306 \@modulo{\@strctr}{10}%
4307 \let\@@fc@numstr#2\relax
4308 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
4309 \fi
4310 \let\@@fc@numstr#2\relax
4311 \edef#2{\@@fc@numstr\ \@thousand}%
4312 \else
4313 \ifnum\@strctr>0\relax
4314 \ifnum\@strctr>1\relax
4315 \let\@@fc@numstr#2\relax
4316 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
4317 \fi
4318 \let\@@fc@numstr#2\relax
4319 \edef#2{\@@fc@numstr\@thousand}%
4320 \fi
4321 \fi
4322 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
4323 \divide\@strctr by 100\relax
4324 \ifnum\@strctr>0\relax
4325 \ifnum#1>1000 \relax
4326 \let\@@fc@numstr#2\relax
4327 \edef#2{\@@fc@numstr\ }%
4328 \fi
4329 \@tmpstrctr=#1\relax

```



```

4330 \@modulo{\@tmpstrctr}{1000}%
4331 \let\@@fc@numstr#2\relax
4332 \ifnum\@tmpstrctr=100\relax
4333 \edef#2{\@@fc@numstr\@tenstring{10}}%
4334 \else
4335 \edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
4336 \fi%
4337 \fi
4338 \@strctr=#1\relax \@modulo{\@strctr}{100}%
4339 \ifnum#1>100\relax
4340 \ifnum\@strctr>0\relax
4341 \let\@@fc@numstr#2\relax
4342 \edef#2{\@@fc@numstr\ \@andname\ }%
4343 \fi
4344 \fi
4345 \ifnum\@strctr>19\relax
4346 \divide\@strctr by 10\relax
4347 \let\@@fc@numstr#2\relax
4348 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
4349 \@strctr=#1\relax \@modulo{\@strctr}{10}%
4350 \ifnum\@strctr>0
4351 \ifnum\@strctr=1\relax
4352 \let\@@fc@numstr#2\relax
4353 \edef#2{\@@fc@numstr\ \@andname}%
4354 \else
4355 \ifnum#1>100\relax
4356 \let\@@fc@numstr#2\relax
4357 \edef#2{\@@fc@numstr\ \@andname}%
4358 \fi
4359 \fi
4360 \let\@@fc@numstr#2\relax
4361 \edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
4362 \fi
4363 \else
4364 \ifnum\@strctr<10\relax
4365 \ifnum\@strctr=0\relax
4366 \ifnum#1<100\relax
4367 \let\@@fc@numstr#2\relax
4368 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
4369 \fi
4370 \else%(>0,<10)
4371 \let\@@fc@numstr#2\relax
4372 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
4373 \fi
4374 \else%>10
4375 \@modulo{\@strctr}{10}%
4376 \let\@@fc@numstr#2\relax
4377 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
4378 \fi

```

```

4379 \fi
4380 }%

  As above, but for ordinals.

4381 \gdef\@@ordinalstringportuges#1#2{%
4382 \@strctr=#1\relax
4383 \ifnum#1>99999
4384 \PackageError{fmtcount}{Out of range}%
4385 {This macro only works for values less than 100000}%
4386 \else
4387 \ifnum#1<0
4388 \PackageError{fmtcount}{Negative numbers not permitted}%
4389 {This macro does not work for negative numbers, however
4390 you can try typing "minus" first, and then pass the modulus of
4391 this number}%
4392 \else
4393 \def#2{}%
4394 \ifnum\@strctr>999\relax
4395 \divide\@strctr by 1000\relax
4396 \ifnum\@strctr>1\relax
4397 \ifnum\@strctr>9\relax
4398 \@tmpstrctr=\@strctr
4399 \ifnum\@strctr<20
4400 \@modulo{\@tmpstrctr}{10}%
4401 \let\@@fc@ordstr#2\relax
4402 \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
4403 \else
4404 \divide\@tmpstrctr by 10\relax
4405 \let\@@fc@ordstr#2\relax
4406 \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4407 \@tmpstrctr=\@strctr
4408 \@modulo{\@tmpstrctr}{10}%
4409 \ifnum\@tmpstrctr>0\relax
4410 \let\@@fc@ordstr#2\relax
4411 \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
4412 \fi
4413 \fi
4414 \else
4415 \let\@@fc@ordstr#2\relax
4416 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
4417 \fi
4418 \fi
4419 \let\@@fc@ordstr#2\relax
4420 \edef#2{\@@fc@ordstr\@thousandth}%
4421 \fi
4422 \@strctr=#1\relax
4423 \@modulo{\@strctr}{1000}%
4424 \ifnum\@strctr>99\relax
4425 \@tmpstrctr=\@strctr
4426 \divide\@tmpstrctr by 100\relax

```

```

4427 \ifnum#1>1000\relax
4428 \let\@@fc@ordstr#2\relax
4429 \edef#2{\@@fc@ordstr-}%
4430 \fi
4431 \let\@@fc@ordstr#2\relax
4432 \edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
4433 \fi
4434 \@modulo{\@strctr}{100}%
4435 \ifnum#1>99\relax
4436 \ifnum\@strctr>0\relax
4437 \let\@@fc@ordstr#2\relax
4438 \edef#2{\@@fc@ordstr-}%
4439 \fi
4440 \fi
4441 \ifnum\@strctr>9\relax
4442 \@tmpstrctr=\@strctr
4443 \divide\@tmpstrctr by 10\relax
4444 \let\@@fc@ordstr#2\relax
4445 \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4446 \@tmpstrctr=\@strctr
4447 \@modulo{\@tmpstrctr}{10}%
4448 \ifnum\@tmpstrctr>0\relax
4449 \let\@@fc@ordstr#2\relax
4450 \edef#2{\@@fc@ordstr-\@unitthstring{\@tmpstrctr}}%
4451 \fi
4452 \else
4453 \ifnum\@strctr=0\relax
4454 \ifnum#1=0\relax
4455 \let\@@fc@ordstr#2\relax
4456 \edef#2{\@@fc@ordstr\@unitstring{0}}%
4457 \fi
4458 \else
4459 \let\@@fc@ordstr#2\relax
4460 \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
4461 \fi
4462 \fi
4463 \fi
4464 \fi
4465 }%

```

#### 9.4.13 fc-spanish.def

Spanish definitions

```
4466 \ProvidesFCLanguage{spanish}[2013/08/17]%
```

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:

```
4467 \gdef\@ordinalMspanish#1{%
```

```

4468 \edef#2{\number#1\relax\noexpand\fmtord{o}}}%
4469 }%

```

Feminine:

```

4470 \gdef\@ordinalFspanish}[2]{%
4471 \edef#2{\number#1\relax\noexpand\fmtord{a}}}%
4472 }%

```

Make neuter same as masculine:

```

4473 \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):

```

4474 \gdef\@@unitstringspanish#1{%
4475 \ifcase#1\relax
4476     cero%
4477     \or uno%
4478     \or dos%
4479     \or tres%
4480     \or cuatro%
4481     \or cinco%
4482     \or seis%
4483     \or siete%
4484     \or ocho%
4485     \or nueve%
4486 \fi
4487 }%

```

Feminine:

```

4488 \gdef\@@unitstringFspanish#1{%
4489 \ifcase#1\relax
4490     cera%
4491     \or una%
4492     \or dos%
4493     \or tres%
4494     \or cuatro%
4495     \or cinco%
4496     \or seis%
4497     \or siete%
4498     \or ocho%
4499     \or nueve%
4500 \fi
4501 }%

```

Tens (argument must go from 1 to 10):

```

4502 \gdef\@@tenstringspanish#1{%
4503 \ifcase#1\relax
4504     \or diez%
4505     \or veinte%
4506     \or treinta%

```

```

4507 \or cuarenta%
4508 \or cincuenta%
4509 \or sesenta%
4510 \or setenta%
4511 \or ochenta%
4512 \or noventa%
4513 \or cien%
4514 \fi
4515 }%

```

#### Teens:

```

4516 \gdef\@@teenstringspanish#1{%
4517 \ifcase#1\relax
4518 diez%
4519 \or once%
4520 \or doce%
4521 \or trece%
4522 \or catorce%
4523 \or quince%
4524 \or diecis\'eis%
4525 \or diecisiete%
4526 \or dieciocho%
4527 \or diecinueve%
4528 \fi
4529 }%

```

#### Twenties:

```

4530 \gdef\@@twentystingspanish#1{%
4531 \ifcase#1\relax
4532 veinte%
4533 \or veintiuno%
4534 \or veintid\'os%
4535 \or veintitr\'es%
4536 \or veinticuatro%
4537 \or veinticinco%
4538 \or veintis\'eis%
4539 \or veintisiete%
4540 \or veintiocho%
4541 \or veintinueve%
4542 \fi
4543 }%

```

#### Feminine form:

```

4544 \gdef\@@twentystringFspanish#1{%
4545 \ifcase#1\relax
4546 veinte%
4547 \or veintiuna%
4548 \or veintid\'os%
4549 \or veintitr\'es%
4550 \or veinticuatro%
4551 \or veinticinco%

```

```

4552 \or veintís\'eis%
4553 \or veintisiete%
4554 \or veintiocho%
4555 \or veintinueve%
4556 \fi
4557 }%

```

Hundreds:

```

4558 \gdef\@@hundredstringspanish#1{%
4559 \ifcase#1\relax
4560 \or ciento%
4561 \or doscientos%
4562 \or trescientos%
4563 \or cuatrocientos%
4564 \or quinientos%
4565 \or seiscientos%
4566 \or setecientos%
4567 \or ochocientos%
4568 \or novecientos%
4569 \fi
4570 }%

```

Feminine form:

```

4571 \gdef\@@hundredstringFspanish#1{%
4572 \ifcase#1\relax
4573 \or cienta%
4574 \or doscientas%
4575 \or trescientas%
4576 \or cuatrocientas%
4577 \or quinientas%
4578 \or seiscientas%
4579 \or setecientas%
4580 \or ochocientas%
4581 \or novecientas%
4582 \fi
4583 }%

```

As above, but with initial letter uppercase:

```

4584 \gdef\@@Unitstringspanish#1{%
4585 \ifcase#1\relax
4586 Cero%
4587 \or Uno%
4588 \or Dos%
4589 \or Tres%
4590 \or Cuatro%
4591 \or Cinco%
4592 \or Seis%
4593 \or Siete%
4594 \or Ocho%
4595 \or Nueve%
4596 \fi

```

4597 }%

Feminine form:

```
4598 \gdef\@@UnitstringFspanish#1{%  
4599   \ifcase#1\relax  
4600     Cera%  
4601     \or Una%  
4602     \or Dos%  
4603     \or Tres%  
4604     \or Cuatro%  
4605     \or Cinco%  
4606     \or Seis%  
4607     \or Siete%  
4608     \or Ocho%  
4609     \or Nueve%  
4610   \fi  
4611 }%
```

Tens:

```
4612 %\changes{2.0}{2012-06-18}{fixed spelling mistake (correction  
4613 %provided by Fernando Maldonado)}  
4614 \gdef\@@Tenstringspanish#1{%  
4615   \ifcase#1\relax  
4616     \or Diez%  
4617     \or Veinte%  
4618     \or Treinta%  
4619     \or Cuarenta%  
4620     \or Cincuenta%  
4621     \or Sesenta%  
4622     \or Setenta%  
4623     \or Ochenta%  
4624     \or Noventa%  
4625     \or Cien%  
4626   \fi  
4627 }%
```

Teens:

```
4628 \gdef\@@Teenstringspanish#1{%  
4629   \ifcase#1\relax  
4630     Diez%  
4631     \or Once%  
4632     \or Doce%  
4633     \or Trece%  
4634     \or Catorce%  
4635     \or Quince%  
4636     \or Diecis\'eis%  
4637     \or Diecisiete%  
4638     \or Dieciocho%  
4639     \or Diecinueve%  
4640   \fi  
4641 }%
```

Twenties:

```
4642 \gdef\@@Twentystringspanish#1{%  
4643   \ifcase#1\relax  
4644     Veinte%  
4645     \or Veintiuno%  
4646     \or Veintid\'os%  
4647     \or Veintitr\'es%  
4648     \or Veinticuatro%  
4649     \or Veinticinco%  
4650     \or Veintis\'eis%  
4651     \or Veintisiete%  
4652     \or Veintiocho%  
4653     \or Veintinueve%  
4654   \fi  
4655 }%
```

Feminine form:

```
4656 \gdef\@@TwentystringFspanish#1{%  
4657   \ifcase#1\relax  
4658     Veinte%  
4659     \or Veintiuna%  
4660     \or Veintid\'os%  
4661     \or Veintitr\'es%  
4662     \or Veinticuatro%  
4663     \or Veinticinco%  
4664     \or Veintis\'eis%  
4665     \or Veintisiete%  
4666     \or Veintiocho%  
4667     \or Veintinueve%  
4668   \fi  
4669 }%
```

Hundreds:

```
4670 \gdef\@@Hundredstringspanish#1{%  
4671   \ifcase#1\relax  
4672     \or Ciento%  
4673     \or Doscientos%  
4674     \or Trescientos%  
4675     \or Cuatrocientos%  
4676     \or Quinientos%  
4677     \or Seiscientos%  
4678     \or Setecientos%  
4679     \or Ochocientos%  
4680     \or Novecientos%  
4681   \fi  
4682 }%
```

Feminine form:

```
4683 \gdef\@@HundredstringFspanish#1{%  
4684   \ifcase#1\relax
```



```

4685 \or Cienta%
4686 \or Doscientas%
4687 \or Trescientas%
4688 \or Cuatrocientas%
4689 \or Quinientas%
4690 \or Seiscientas%
4691 \or Setecientas%
4692 \or Ochocientas%
4693 \or Novecientas%
4694 \fi
4695 }%

```

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.)

```

4696 \DeclareRobustCommand{\@numberstringMspanish}[2]{%
4697 \let\@unitstring=\@unitstringspanish
4698 \let\@teenstring=\@teenstringspanish
4699 \let\@tenstring=\@tenstringspanish
4700 \let\@twentystring=\@twentystringspanish
4701 \let\@hundredstring=\@hundredstringspanish
4702 \def\@hundred{cien}\def\@thousand{mil}%
4703 \def\@andname{y}%
4704 \@numberstringspanish{#1}{#2}%
4705 }%
4706 \global\let\@numberstringMspanish\@numberstringMspanish

```

Feminine form:

```

4707 \DeclareRobustCommand{\@numberstringFspanish}[2]{%
4708 \let\@unitstring=\@unitstringFspanish
4709 \let\@teenstring=\@teenstringspanish
4710 \let\@tenstring=\@tenstringspanish
4711 \let\@twentystring=\@twentystringFspanish
4712 \let\@hundredstring=\@hundredstringFspanish
4713 \def\@hundred{cien}\def\@thousand{mil}%
4714 \def\@andname{b}%
4715 \@numberstringspanish{#1}{#2}%
4716 }%
4717 \global\let\@numberstringFspanish\@numberstringFspanish

```

Make neuter same as masculine:

```

4718 \global\let\@numberstringNspanish\@numberstringMspanish

```

As above, but initial letters in upper case:

```

4719 \DeclareRobustCommand{\@NumberstringMspanish}[2]{%
4720 \let\@unitstring=\@Unitstringspanish
4721 \let\@teenstring=\@Teenstringspanish
4722 \let\@tenstring=\@Tenstringspanish
4723 \let\@twentystring=\@Twentystringspanish
4724 \let\@hundredstring=\@Hundredstringspanish

```

```

4725 \def\@andname{y}%
4726 \def\@hundred{Cien}\def\@thousand{Mil}%
4727 \@@numberstringspanish{#1}{#2}%
4728 }%
4729 \global\let\@NumberstringMspanish\@NumberstringMspanish

Feminine form:
4730 \DeclareRobustCommand{\@NumberstringFspanish}[2]{%
4731 \let\@unitstring=\@UnitstringFspanish
4732 \let\@teenstring=\@Teenstringspanish
4733 \let\@tenstring=\@Tenstringspanish
4734 \let\@twentystring=\@TwentystringFspanish
4735 \let\@hundredstring=\@HundredstringFspanish
4736 \def\@andname{b}%
4737 \def\@hundred{Cien}\def\@thousand{Mil}%
4738 \@@numberstringspanish{#1}{#2}%
4739 }%
4740 \global\let\@NumberstringFspanish\@NumberstringFspanish

Make neuter same as masculine:
4741 \global\let\@NumberstringNspanish\@NumberstringMspanish

As above, but for ordinals.
4742 \DeclareRobustCommand{\@OrdinalstringMspanish}[2]{%
4743 \let\@unitthstring=\@Unitthstringspanish
4744 \let\@unitstring=\@Unitstringspanish
4745 \let\@teenthstring=\@Teenthstringspanish
4746 \let\@tenthstring=\@Tenthstringspanish
4747 \let\@hundredthstring=\@Hundredthstringspanish
4748 \def\@thousandth{mil\'esimo}%
4749 \@@ordinalstringspanish{#1}{#2}%
4750 }%
4751 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish

Feminine form:
4752 \DeclareRobustCommand{\@OrdinalstringFspanish}[2]{%
4753 \let\@unitthstring=\@UnitthstringFspanish
4754 \let\@unitstring=\@UnitstringFspanish
4755 \let\@teenthstring=\@TeenthstringFspanish
4756 \let\@tenthstring=\@TenthstringFspanish
4757 \let\@hundredthstring=\@HundredthstringFspanish
4758 \def\@thousandth{mil\'esima}%
4759 \@@ordinalstringspanish{#1}{#2}%
4760 }%
4761 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish

Make neuter same as masculine:
4762 \global\let\@OrdinalstringNspanish\@OrdinalstringMspanish

As above, but with initial letters in upper case.
4763 \DeclareRobustCommand{\@OrdinalstringMspanish}[2]{%
4764 \let\@unitthstring=\@Unitthstringspanish

```

```

4765 \let\@unitstring=\@Unitstringspanish
4766 \let\@teenthstring=\@Teenthstringspanish
4767 \let\@tenthstring=\@Tenthstringspanish
4768 \let\@hundredthstring=\@Hundredthstringspanish
4769 \def\@thousandth{Mil\'esimo}%
4770 \@ordinalstringspanish{#1}{#2}%
4771 }
4772 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish

```

Feminine form:

```

4773 \DeclareRobustCommand{\@OrdinalstringFspanish}[2]{%
4774 \let\@unitthstring=\@UnitthstringFspanish
4775 \let\@unitstring=\@UnitstringFspanish
4776 \let\@teenthstring=\@TeenthstringFspanish
4777 \let\@tenthstring=\@TenthstringFspanish
4778 \let\@hundredthstring=\@HundredthstringFspanish
4779 \def\@thousandth{Mil\'esima}%
4780 \@ordinalstringspanish{#1}{#2}%
4781 }%
4782 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish

```

Make neuter same as masculine:

```

4783 \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:

```

4784 \gdef\@unitthstringspanish#1{%
4785 \ifcase#1\relax
4786 \or cero%
4787 \or primero%
4788 \or segundo%
4789 \or tercero%
4790 \or cuarto%
4791 \or quinto%
4792 \or sexto%
4793 \or s\'eptimo%
4794 \or octavo%
4795 \or noveno%
4796 \fi
4797 }%

```

Tens:

```

4798 \gdef\@tenthstringspanish#1{%
4799 \ifcase#1\relax
4800 \or d\'ecimo%
4801 \or vig\'esimo%
4802 \or trig\'esimo%
4803 \or cuadrag\'esimo%
4804 \or quincuag\'esimo%
4805 \or sexag\'esimo%
4806 \or septuag\'esimo%

```

```

4807 \or octog\'esimo%
4808 \or nonag\'esimo%
4809 \fi
4810 }%

```

#### Teens:

```

4811 \gdef\@teenthstringspanish#1{%
4812 \ifcase#1\relax
4813 d\'ecimo%
4814 \or und\'ecimo%
4815 \or duod\'ecimo%
4816 \or decimotercero%
4817 \or decimocuarto%
4818 \or decimoquinto%
4819 \or decimosexto%
4820 \or decimos\'eptimo%
4821 \or decimoctavo%
4822 \or decimonoveno%
4823 \fi
4824 }%

```

#### Hundreds:

```

4825 \gdef\@hundredthstringspanish#1{%
4826 \ifcase#1\relax
4827 \or cent\'esimo%
4828 \or ducent\'esimo%
4829 \or tricent\'esimo%
4830 \or cuadringent\'esimo%
4831 \or quingent\'esimo%
4832 \or sexcent\'esimo%
4833 \or septing\'esimo%
4834 \or octingent\'esimo%
4835 \or noningent\'esimo%
4836 \fi
4837 }%

```

#### Units (feminine):

```

4838 \gdef\@unitthstringFspanish#1{%
4839 \ifcase#1\relax
4840 cera%
4841 \or primera%
4842 \or segunda%
4843 \or tercera%
4844 \or cuarta%
4845 \or quinta%
4846 \or sexta%
4847 \or s\'eptima%
4848 \or octava%
4849 \or novena%
4850 \fi
4851 }%

```

Tens (feminine):

```
4852 \gdef\@tenthstringFspanish#1{%  
4853   \ifcase#1\relax  
4854     \or d\'esima%  
4855     \or vig\'esima%  
4856     \or trig\'esima%  
4857     \or cuadrag\'esima%  
4858     \or quincuag\'esima%  
4859     \or sexag\'esima%  
4860     \or septuag\'esima%  
4861     \or octog\'esima%  
4862     \or nonag\'esima%  
4863   \fi  
4864 }%
```

Teens (feminine)

```
4865 \gdef\@teenthstringFspanish#1{%  
4866   \ifcase#1\relax  
4867     d\'esima%  
4868     \or und\'esima%  
4869     \or duod\'esima%  
4870     \or decimotercera%  
4871     \or decimocuarta%  
4872     \or decimoquinta%  
4873     \or decimosexta%  
4874     \or decimos\'eptima%  
4875     \or decimoctava%  
4876     \or decimonovena%  
4877   \fi  
4878 }%
```

Hundreds (feminine)

```
4879 \gdef\@hundredthstringFspanish#1{%  
4880   \ifcase#1\relax  
4881     \or cent\'esima%  
4882     \or ducent\'esima%  
4883     \or tricent\'esima%  
4884     \or cuadringent\'esima%  
4885     \or quingent\'esima%  
4886     \or sexcent\'esima%  
4887     \or septing\'esima%  
4888     \or octingent\'esima%  
4889     \or noningent\'esima%  
4890   \fi  
4891 }%
```

As above, but with initial letters in upper case

```
4892 \gdef\@Unitthstringspanish#1{%  
4893   \ifcase#1\relax  
4894     Cero%
```

```

4895 \or Primero%
4896 \or Segundo%
4897 \or Tercero%
4898 \or Cuarto%
4899 \or Quinto%
4900 \or Sexto%
4901 \or S\'eptimo%
4902 \or Octavo%
4903 \or Noveno%
4904 \fi
4905 }%

```

#### Tens:

```

4906 \gdef\@@Tenthstringspanish#1{%
4907 \ifcase#1\relax
4908 \or D\'ecimo%
4909 \or Vig\'esimo%
4910 \or Trig\'esimo%
4911 \or Cuadrag\'esimo%
4912 \or Quincuag\'esimo%
4913 \or Sexag\'esimo%
4914 \or Septuag\'esimo%
4915 \or Octog\'esimo%
4916 \or Nonag\'esimo%
4917 \fi
4918 }%

```

#### Teens:

```

4919 \gdef\@@Teenthstringspanish#1{%
4920 \ifcase#1\relax
4921 D\'ecimo%
4922 \or Und\'ecimo%
4923 \or Duod\'ecimo%
4924 \or Decimotercero%
4925 \or Decimocuarto%
4926 \or Decimoquinto%
4927 \or Decimosexto%
4928 \or Decimos\'eptimo%
4929 \or Decimoctavo%
4930 \or Decimonoveno%
4931 \fi
4932 }%

```

#### Hundreds

```

4933 \gdef\@@Hundredthstringspanish#1{%
4934 \ifcase#1\relax
4935 \or Cent\'esimo%
4936 \or Ducent\'esimo%
4937 \or Tricent\'esimo%
4938 \or Cuadringent\'esimo%
4939 \or Quingent\'esimo%

```

```

4940 \or Sexcent\'esimo%
4941 \or Septing\'esimo%
4942 \or Octingent\'esimo%
4943 \or Noningent\'esimo%
4944 \fi
4945 }%

```

As above, but feminine.

```

4946 \gdef\@@UnitthstringFspanish#1{%
4947 \ifcase#1\relax
4948 Cera%
4949 \or Primera%
4950 \or Segunda%
4951 \or Tercera%
4952 \or Cuarta%
4953 \or Quinta%
4954 \or Sexta%
4955 \or S\'eptima%
4956 \or Octava%
4957 \or Novena%
4958 \fi
4959 }%

```

Tens (feminine)

```

4960 \gdef\@@TenthstringFspanish#1{%
4961 \ifcase#1\relax
4962 \or D\'ecima%
4963 \or Vig\'esima%
4964 \or Trig\'esima%
4965 \or Cuadrag\'esima%
4966 \or Quincuag\'esima%
4967 \or Sexag\'esima%
4968 \or Septuag\'esima%
4969 \or Octog\'esima%
4970 \or Nonag\'esima%
4971 \fi
4972 }%

```

Teens (feminine):

```

4973 \gdef\@@TeenthstringFspanish#1{%
4974 \ifcase#1\relax
4975 D\'ecima%
4976 \or Und\'ecima%
4977 \or Duod\'ecima%
4978 \or Decimotercera%
4979 \or Decimocuarta%
4980 \or Decimoquinta%
4981 \or Decimosexta%
4982 \or Decimos\'eptima%
4983 \or Decimoctava%
4984 \or Decimonovena%

```

```

4985 \fi
4986 }%

Hundreds (feminine):
4987 \gdef\@@HundredthstringFspanish#1{%
4988 \ifcase#1\relax
4989 \or Cent\'esima%
4990 \or Ducent\'esima%
4991 \or Tricent\'esima%
4992 \or Cuadringent\'esima%
4993 \or Quingent\'esima%
4994 \or Sexcent\'esima%
4995 \or Septing\'esima%
4996 \or Octingent\'esima%
4997 \or Noningent\'esima%
4998 \fi
4999 }%

```

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

```

5000 \gdef\@@numberstringspanish#1#2{%
5001 \ifnum#1>99999
5002 \PackageError{fmtcount}{Out of range}%
5003 {This macro only works for values less than 100000}%
5004 \else
5005 \ifnum#1<0
5006 \PackageError{fmtcount}{Negative numbers not permitted}%
5007 {This macro does not work for negative numbers, however
5008 you can try typing "minus" first, and then pass the modulus of
5009 this number}%
5010 \fi
5011 \fi
5012 \def#2{%
5013 \@strctr=#1\relax \divide\@strctr by 1000\relax
5014 \ifnum\@strctr>9

#1 is greater or equal to 10000
5015 \divide\@strctr by 10
5016 \ifnum\@strctr>1
5017 \let\@@fc@numstr#2\relax
5018 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
5019 \@strctr=#1 \divide\@strctr by 1000\relax
5020 \@modulo{\@strctr}{10}%
5021 \ifnum\@strctr>0\relax
5022 \let\@@fc@numstr#2\relax
5023 \edef#2{\@@fc@numstr\@andname\@unitstring{\@strctr}}%
5024 \fi
5025 \else

```



```

5026 \@strctr=#1\relax
5027 \divide\@strctr by 1000\relax
5028 \@modulo{\@strctr}{10}%
5029 \let\@@fc@numstr#2\relax
5030 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
5031 \fi
5032 \let\@@fc@numstr#2\relax
5033 \edef#2{\@@fc@numstr\@thousand}%
5034 \else
5035 \ifnum\@strctr>0\relax
5036 \ifnum\@strctr>1\relax
5037 \let\@@fc@numstr#2\relax
5038 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
5039 \fi
5040 \let\@@fc@numstr#2\relax
5041 \edef#2{\@@fc@numstr\@thousand}%
5042 \fi
5043 \fi
5044 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
5045 \divide\@strctr by 100\relax
5046 \ifnum\@strctr>0\relax
5047 \ifnum#1>1000\relax
5048 \let\@@fc@numstr#2\relax
5049 \edef#2{\@@fc@numstr\ }%
5050 \fi
5051 \@tmpstrctr=#1\relax
5052 \@modulo{\@tmpstrctr}{1000}%
5053 \ifnum\@tmpstrctr=100\relax
5054 \let\@@fc@numstr#2\relax
5055 \edef#2{\@@fc@numstr\@tenstring{10}}%
5056 \else
5057 \let\@@fc@numstr#2\relax
5058 \edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
5059 \fi
5060 \fi
5061 \@strctr=#1\relax \@modulo{\@strctr}{100}%
5062 \ifnum#1>100\relax
5063 \ifnum\@strctr>0\relax
5064 \let\@@fc@numstr#2\relax
5065 \edef#2{\@@fc@numstr\ }%
5066 \fi
5067 \fi
5068 \ifnum\@strctr>29\relax
5069 \divide\@strctr by 10\relax
5070 \let\@@fc@numstr#2\relax
5071 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
5072 \@strctr=#1\relax \@modulo{\@strctr}{10}%
5073 \ifnum\@strctr>0\relax
5074 \let\@@fc@numstr#2\relax

```

```

5075 \edef#2{\@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
5076 \fi
5077 \else
5078 \ifnum\@strctr<10\relax
5079 \ifnum\@strctr=0\relax
5080 \ifnum#1<100\relax
5081 \let\@fc@numstr#2\relax
5082 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
5083 \fi
5084 \else
5085 \let\@fc@numstr#2\relax
5086 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
5087 \fi
5088 \else
5089 \ifnum\@strctr>19\relax
5090 \@modulo{\@strctr}{10}%
5091 \let\@fc@numstr#2\relax
5092 \edef#2{\@fc@numstr\@twentystring{\@strctr}}%
5093 \else
5094 \@modulo{\@strctr}{10}%
5095 \let\@fc@numstr#2\relax
5096 \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
5097 \fi
5098 \fi
5099 \fi
5100 }%

```

As above, but for ordinals

```

5101 \gdef\@ordinalstringspanish#1#2{%
5102 \@strctr=#1\relax
5103 \ifnum#1>99999
5104 \PackageError{fmtcount}{Out of range}%
5105 {This macro only works for values less than 100000}%
5106 \else
5107 \ifnum#1<0
5108 \PackageError{fmtcount}{Negative numbers not permitted}%
5109 {This macro does not work for negative numbers, however
5110 you can try typing "minus" first, and then pass the modulus of
5111 this number}%
5112 \else
5113 \def#2{}%
5114 \ifnum\@strctr>999\relax
5115 \divide\@strctr by 1000\relax
5116 \ifnum\@strctr>1\relax
5117 \ifnum\@strctr>9\relax
5118 \@tmpstrctr=\@strctr
5119 \ifnum\@strctr<20
5120 \@modulo{\@tmpstrctr}{10}%
5121 \let\@fc@ordstr#2\relax
5122 \edef#2{\@fc@ordstr\@teenthstring{\@tmpstrctr}}%

```

```

5123     \else
5124         \divide\@tmpstrctr by 10\relax
5125         \let\@@fc@ordstr#2\relax
5126         \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
5127         \@tmpstrctr=\@strctr
5128         \@modulo{\@tmpstrctr}{10}%
5129         \ifnum\@tmpstrctr>0\relax
5130             \let\@@fc@ordstr#2\relax
5131             \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
5132         \fi
5133     \fi
5134     \else
5135         \let\@@fc@ordstr#2\relax
5136         \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
5137     \fi
5138 \fi
5139 \let\@@fc@ordstr#2\relax
5140 \edef#2{\@@fc@ordstr\@thousandth}%
5141 \fi
5142 \@strctr=#1\relax
5143 \@modulo{\@strctr}{1000}%
5144 \ifnum\@strctr>99\relax
5145     \@tmpstrctr=\@strctr
5146     \divide\@tmpstrctr by 100\relax
5147     \ifnum#1>1000\relax
5148         \let\@@fc@ordstr#2\relax
5149         \edef#2{\@@fc@ordstr\ }%
5150     \fi
5151     \let\@@fc@ordstr#2\relax
5152     \edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
5153 \fi
5154 \@modulo{\@strctr}{100}%
5155 \ifnum#1>99\relax
5156     \ifnum\@strctr>0\relax
5157         \let\@@fc@ordstr#2\relax
5158         \edef#2{\@@fc@ordstr\ }%
5159     \fi
5160 \fi
5161 \ifnum\@strctr>19\relax
5162     \@tmpstrctr=\@strctr
5163     \divide\@tmpstrctr by 10\relax
5164     \let\@@fc@ordstr#2\relax
5165     \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
5166     \@tmpstrctr=\@strctr
5167     \@modulo{\@tmpstrctr}{10}%
5168     \ifnum\@tmpstrctr>0\relax
5169         \let\@@fc@ordstr#2\relax
5170         \edef#2{\@@fc@ordstr\ \@unitthstring{\@tmpstrctr}}%
5171     \fi

```

```

5172 \else
5173   \ifnum \@strctr>9\relax
5174     \@modulo{\@strctr}{10}%
5175     \let\@fc@ordstr#2\relax
5176     \edef#2{\@fc@ordstr\@teenthsstring{\@strctr}}%
5177   \else
5178     \ifnum \@strctr=0\relax
5179       \ifnum#1=0\relax
5180         \let\@fc@ordstr#2\relax
5181         \edef#2{\@fc@ordstr\@unitstring{0}}%
5182       \fi
5183     \else
5184       \let\@fc@ordstr#2\relax
5185       \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
5186     \fi
5187   \fi
5188 \fi
5189 \fi
5190 \fi
5191 }%

```

#### 9.4.14 fc-UKenglish.def

English definitions

```
5192 \ProvidesFCLanguage{UKenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
5193 \FCloadlang{english}%
```

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

```

5194 \global\let\@ordinalMUKenglish\@ordinalMenglish
5195 \global\let\@ordinalFUKenglish\@ordinalMenglish
5196 \global\let\@ordinalNUKenglish\@ordinalMenglish
5197 \global\let\@numberstringMUKenglish\@numberstringMenglish
5198 \global\let\@numberstringFUKenglish\@numberstringMenglish
5199 \global\let\@numberstringNUKenglish\@numberstringMenglish
5200 \global\let\@NumberstringMUKenglish\@NumberstringMenglish
5201 \global\let\@NumberstringFUKenglish\@NumberstringMenglish
5202 \global\let\@NumberstringNUKenglish\@NumberstringMenglish
5203 \global\let\@ordinalstringMUKenglish\@ordinalstringMenglish
5204 \global\let\@ordinalstringFUKenglish\@ordinalstringMenglish
5205 \global\let\@ordinalstringNUKenglish\@ordinalstringMenglish
5206 \global\let\@OrdinalstringMUKenglish\@OrdinalstringMenglish
5207 \global\let\@OrdinalstringFUKenglish\@OrdinalstringMenglish
5208 \global\let\@OrdinalstringNUKenglish\@OrdinalstringMenglish

```

#### 9.4.15 fc-USenglish.def

US English definitions

```
5209 \ProvidesFCLanguage{USenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
5210 \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.)

```
5211 \global\let\@ordinalMUSenglish\@ordinalMenglish
5212 \global\let\@ordinalFUSenglish\@ordinalMenglish
5213 \global\let\@ordinalNUSenglish\@ordinalMenglish
5214 \global\let\@numberstringMUSenglish\@numberstringMenglish
5215 \global\let\@numberstringFUSenglish\@numberstringMenglish
5216 \global\let\@numberstringNUSenglish\@numberstringMenglish
5217 \global\let\@NumberstringMUSenglish\@NumberstringMenglish
5218 \global\let\@NumberstringFUSenglish\@NumberstringMenglish
5219 \global\let\@NumberstringNUSenglish\@NumberstringMenglish
5220 \global\let\@ordinalstringMUSenglish\@ordinalstringMenglish
5221 \global\let\@ordinalstringFUSenglish\@ordinalstringMenglish
5222 \global\let\@ordinalstringNUSenglish\@ordinalstringMenglish
5223 \global\let\@OrdinalstringMUSenglish\@OrdinalstringMenglish
5224 \global\let\@OrdinalstringFUSenglish\@OrdinalstringMenglish
5225 \global\let\@OrdinalstringNUSenglish\@OrdinalstringMenglish
```