

# 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)

2012-09-28 (version 2.01)

## 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>14</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>
9.0.1	<code>fcnumparser.sty</code> . . . . .	15
9.1	<code>fcprefix.sty</code> . . . . .	25
9.2	<code>fmtcount.sty</code> . . . . .	36
9.3	Multilanguage Definitions . . . . .	59
9.3.1	<code>fc-american.def</code> . . . . .	64
9.3.2	<code>fc-british.def</code> . . . . .	64
9.3.3	<code>fc-english.def</code> . . . . .	65
9.3.4	<code>fc-francais.def</code> . . . . .	75
9.3.5	<code>fc-french.def</code> . . . . .	75
9.3.6	<code>fc-frenchb.def</code> . . . . .	105
9.3.7	<code>fc-german.def</code> . . . . .	106

9.3.8	<code>fc-germanb.def</code>	115
9.3.9	<code>fc-italian</code>	115
9.3.10	<code>fc-ngerman.def</code>	116
9.3.11	<code>fc-ngermanb.def</code>	117
9.3.12	<code>fc-portuges.def</code>	117
9.3.13	<code>fc-spanish.def</code>	131
9.3.14	<code>fc-UKenglish.def</code>	147
9.3.15	<code>fc-USenglish.def</code>	147

## 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>

The package checks to see if the command `\l@<language>` is defined<sup>4</sup>, and will load the code for those languages. The commands `\ordinal`, `\ordinalstring` and `\numberstring` (and their variants) will then be formatted in the currently selected language.

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

---

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

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

<sup>4</sup>this will be true if you have loaded `babel`

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 :

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

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

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

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

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

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

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 :

- 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'alter-

nance 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 <i>⟨n⟩</i> illion et <i>⟨n⟩</i> 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 `<dash or space>` à :

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

Par défaut, l’option vaut `reformed`.

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

L’option `scale` permet de configurer l’écriture des grands nombres. Mettre `<scale>` à :

- recursive    dans ce cas  $10^{30}$  donne mille milliards de milliards de milliards, pour  $10^n$ , on écrit  $10^{n-9 \times \max\{(n \div 9) - 1, 0\}}$  suivi de la répétition  $\max\{(n \div 9) - 1, 0\}$  fois de “de milliards”
- long           $10^{6 \times n}$  donne un  $\langle n \rangle$ 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  $n > 0$ ,
- infy        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 « milles » 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  $\TeX$ 2HTML 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

### 9.0.1 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 integer and fractional part. Number to be analysed is in ‘#1’. Decimal separator may be . or , whichever first. At end of this macro, integer part goes to `\fc@integer@part` and fractional part goes to `\fc@fractional@part`.

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

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

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

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

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

No comma either, so fractional part is set empty.

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

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

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

```

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

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

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

```
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 `\fc@max@weight` in order to copy all the digits from `\fc@digit@<i + offset>` to `\fc@digit@<i>`. First we compute offset into `\@tempi`.

```
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\@tempi{##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 `\@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 \@tempi
```

```
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 }

```

```

\fc @parse@integer@digits Macro \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  $\text{\fc@digit@}\langle n \rangle$  array and form a corresponding number.

```

\fc @read@unit \fc@read@unit just reads one digit and form an integer in the
range [0..9]. First we check that the macro is not yet defined.
119 \@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  
 #2 input number: unit weight at which reach the value is to be read  
 does not need to be comprised between  $\text{\fc@min@weight}$  and  $\text{\fc@min@weight}$ ,  
 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 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@<w>`. Macro `\fc@intpart@find@last` takes one single argument which is a counter to set to the result.

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

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

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

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

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

Now propagate result after closing bracket into counter #1.

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

`\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 }%

```

## 9.1 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 $\langle xxx \rangle$  or duodec $\langle xxx \rangle$  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<sup>2</sup>’

prefix for which millia squared is noted as ‘bismillia’

```

387 \define@key{fc@prefix}{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{fc@prefix}{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\@temp#1%
408   \edef#1{millia\@temp}%
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   \ifnnum#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 output 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 \@tempu.
444 \let\@tempt\@empty
445 \def\@tempu{t}%
\@tempm will hold the millian÷3
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\@temp\@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  $d_1 d_0$  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 \count3 is  $3 \times n$ , with  $n > 0$ , i.e. this is followed by a millia<sup>n</sup>.

```

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 \count3 \count4 or \count5 is non zero.

```

532     \@tempm

```

And append previous version of \@tempa.

```

533     \@tempa
534 }%

```

“Concatenate” millia to \@tempm, so that \@tempm will expand to  $\text{millia}^{(n \div 3)+1}$  at the next iteration. Actually whether this is a concatenation or some 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>$n \div 3$</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     \@tempm
645 }%

```

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

```

646     \let\@tempm\@tempm
647     \edef\@tempm{\millies\@tempm}%
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{####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 }

```

## 9.2 fmtcount.sty

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

```

705 \NeedsTeXFormat{LaTeX2e}
706 \ProvidesPackage{fmtcount}[2012/09/28 v2.01]
707 \RequirePackage{ifthen}
708 \RequirePackage{keyval}
709 \RequirePackage{etoolbox}

```

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

```

710 \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`

```

711 \providecommand*\fmtord}[1]{\textsuperscript{#1}}

```

`\padzeroes`

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

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

```

712 \newcount\c@padzeroesN
713 \c@padzeroesN=1\relax
714 \providecommand*\padzeroes}[1][17]{\c@padzeroesN=#1}

```

`\FCloadlang` changes2.02012-06-18new

```
\FCloadlang{<language>}
```

Load `fmtcount` language file, `fc-<language>.def`, unless already loaded.

```

715 \newcommand*\FCloadlang}[1]{%
716   \@FC@iflangloaded{#1}{}%
717   {%
718     \input{fc-#1.def}%
719   }%
720 }

```

`\@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*⟩

```
721 \newcommand{\@FC@iflangloaded}[3]{%
722   \ifcsundef{ver@fc-#1.def}{#3}{#2}%
723 }
```

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

```
724 \newcommand*{\ProvidesFCLanguage}[1]{%
725   \ProvidesFile{fc-#1.def}%
726 }
```

`\@fc@loadifbabelldf` `\@fc@loadigbabelldf{⟨language⟩}`

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

```
727 \newcommand*{\@fc@loadifbabelldf}[1]{%
728   \ifcsundef{ver@#1.ldf}{}{\FCloadlang{#1}}%
729 }
```

Load appropriate language definition files:

```
730 \@fc@loadifbabelldf{english}
731 \@fc@loadifbabelldf{UKenglish}
732 \@fc@loadifbabelldf{british}
733 \@fc@loadifbabelldf{USenglish}
734 \@fc@loadifbabelldf{american}
735 \@fc@loadifbabelldf{spanish}
736 \@fc@loadifbabelldf{portuges}
737 \@fc@loadifbabelldf{french}
738 \@fc@loadifbabelldf{frenchb}
739 \@fc@loadifbabelldf{german}%
740 \@fc@loadifbabelldf{germanb}%
741 \@fc@loadifbabelldf{ngerman}%
742 \@fc@loadifbabelldf{ngermanb}%
743 \@fc@loadifbabelldf{italian}
```

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

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

french

```
745 \define@key{fmtcount}{french}[france]{%
746   \@ifundefined{datefrench}%
747   {%
748     \PackageError{fmtcount}%
749     {Language ‘french’ not defined}%
750   }
```

```

750 {You need to load babel before loading fmtcount}%
751 }%
752 {%
753 \setkeys{french}{#1}%
754 }%
755 }

```

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

```

756 \define@key{fmtcount}{fmtord}{%
757 \ifthenelse{\equal{#1}{level}
758 \or\equal{#1}{raise}
759 \or\equal{#1}{user}}%
760 {%
761 \def\fmtcount@fmtord{#1}%
762 }%
763 {%
764 \PackageError{fmtcount}%
765 {Invalid value ‘#1’ to fmtord key}%
766 {Option ‘fmtord’ can only take the values ‘level’, ‘raise’
767 or ‘user’}%
768 }%
769 }

```

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

```

770 \newif\iffmtord@abbrv
771 \fmtord@abbrvfalse
772 \define@key{fmtcount}{abbrv}[true]{%
773 \ifthenelse{\equal{#1}{true}\or\equal{#1}{false}}%
774 {%
775 \csname fmtord@abbrv#1\endcsname
776 }%
777 {%
778 \PackageError{fmtcount}%
779 {Invalid value ‘#1’ to fmtord key}%
780 {Option ‘fmtord’ can only take the values ‘true’ or
781 ‘false’}%
782 }%
783 }

```

**prefix**

```

784 \define@key{fmtcount}{prefix}[scale=long]{%
785 \RequirePackage{fmtprefix}%
786 \fmtprefixsetoption{#1}%
787 }

```

**\fmtcountsetoptions** Define command to set options.

```

788 \newcommand*{\fmtcountsetoptions}[1]{%
789 \def\fmtcount@fmtord{}%

```

```

790 \setkeys{fmtcount}{#1}%
791 \@ifundefined{datefrench}{}%
792 {%
793   \edef\@ordinalstringMfrench{\noexpand
794     \csname @ordinalstringMfrench\fmtcount@french\noexpand\endcsname}%
795   \edef\@ordinalstringFfrench{\noexpand
796     \csname @ordinalstringFfrench\fmtcount@french\noexpand\endcsname}%
797   \edef\@OrdinalstringMfrench{\noexpand
798     \csname @OrdinalstringMfrench\fmtcount@french\noexpand\endcsname}%
799   \edef\@OrdinalstringFfrench{\noexpand
800     \csname @OrdinalstringFfrench\fmtcount@french\noexpand\endcsname}%
801   \edef\@numberstringMfrench{\noexpand
802     \csname @numberstringMfrench\fmtcount@french\noexpand\endcsname}%
803   \edef\@numberstringFfrench{\noexpand
804     \csname @numberstringFfrench\fmtcount@french\noexpand\endcsname}%
805   \edef\@NumberstringMfrench{\noexpand
806     \csname @NumberstringMfrench\fmtcount@french\noexpand\endcsname}%
807   \edef\@NumberstringFfrench{\noexpand
808     \csname @NumberstringFfrench\fmtcount@french\noexpand\endcsname}%
809 }%
810 \ifthenelse{\equal{\fmtcount@fmtord}{level}}{%
811   {%
812     \renewcommand{\fmtord}[1]{##1}%
813   }%
814   {%
815     \ifthenelse{\equal{\fmtcount@fmtord}{raise}}{%
816       {%
817         \renewcommand{\fmtord}[1]{\textsuperscript{##1}}%
818       }%
819       {%
820       }%
821     }
822 }

```

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.

```

823 \InputIfFileExists{fmtcount.cfg}%
824 {%
825   \PackageInfo{fmtcount}{Using configuration file fmtcount.cfg}%
826 }%
827 {%
828 }

```

level

```

829 \DeclareOption{level}{\def\fmtcount@fmtord{level}%
830   \def\fmtord#1{#1}}

```

raise

```

831 \DeclareOption{raise}{\def\fmtcount@fmtord{raise}%

```

```
832 \def\fmtord#1{\textsuperscript{#1}}
```

Process package options

```
833 \ProcessOptions
```

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

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

```
834 \newcount\@DT@modctr
835 \def\@modulo#1#2{%
836   \@DT@modctr=#1\relax
837   \divide \@DT@modctr by #2\relax
838   \multiply \@DT@modctr by #2\relax
839   \advance #1 by -\@DT@modctr
840 }
```

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

```
841 \newcount\@ordinalctr
842 \newcount\@orgargctr
843 \newcount\@strctr
844 \newcount\@tmpstrctr
```

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

```
845 \newif\if@DT@padzeroes
846 \newcount\@DT@loopN
847 \newcount\@DT@X
```

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

```
848 \newcommand*{\@binary}[1]{%
849   \@DT@padzeroestrue
850   \@DT@loopN=17\relax
851   \@strctr=\@DT@loopN
852   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
853   \@strctr=65536\relax
854   \@DT@X=#1\relax
855   \loop
856     \@DT@modctr=\@DT@X
857     \divide\@DT@modctr by \@strctr
858     \ifthenelse{\boolean{@DT@padzeroes}
859       \and \(\@DT@modctr=0\)}
860       \and \(\@DT@loopN>\c@padzeroesN\)}%
861     {}%
862     {\the\@DT@modctr}%
863   \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
```

```

864 \multiply\@DT@modctr by \@strctr
865 \advance\@DT@X by -\@DT@modctr
866 \divide\@strctr by 2\relax
867 \advance\@DT@loopN by -1\relax
868 \ifnum\@strctr>1
869 \repeat
870 \the\@DT@X
871 }
872
873 \let\binarynum=\@binary

```

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

```

874 \newcommand*{\@octal}[1]{%
875 \ifnum#1>32768
876 \PackageError{fmtcount}%
877 {Value of counter too large for \protect\@octal}
878 {Maximum value 32768}
879 \else
880 \@DT@padzeroestrue
881 \@DT@loopN=6\relax
882 \@strctr=\@DT@loopN
883 \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
884 \@strctr=32768\relax
885 \@DT@X=#1\relax
886 \loop
887 \@DT@modctr=\@DT@X
888 \divide\@DT@modctr by \@strctr
889 \ifthenelse{\boolean{@DT@padzeroes}
890 \and \(\@DT@modctr=0\)
891 \and \(\@DT@loopN>\c@padzeroesN\)}%
892 {\the\@DT@modctr}%
893 \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
894 \multiply\@DT@modctr by \@strctr
895 \advance\@DT@X by -\@DT@modctr
896 \divide\@strctr by 8\relax
897 \advance\@DT@loopN by -1\relax
898 \ifnum\@strctr>1
899 \repeat
900 \the\@DT@X
901 \fi
902 }
903 \let\octalnum=\@octal

```

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

```

904 \newcommand*{\@@hexadecimal}[1]{%
905 \ifcase#1\or1\or2\or3\or4\or5\or
906 6\or7\or8\or9\or a\or b\or c\or d\or e\or f\fi
907 }

```

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

```

908 \newcommand*{\@hexadecimal}[1]{%
909   \DT@padzeroestru
910   \DT@loopN=5\relax
911   \@strctr=\DT@loopN
912   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
913   \@strctr=65536\relax
914   \DT@X=#1\relax
915   \loop
916     \DT@modctr=\DT@X
917     \divide\DT@modctr by \@strctr
918     \ifthenelse{\boolean{\DT@padzeroes}
919       \and \(\DT@modctr=0\}
920       \and \(\DT@loopN>\c@padzeroesN\)}
921       {\@hexadecimal\DT@modctr}%
922       \ifnum\DT@modctr=0\else\DT@padzeroesfalse\fi
923       \multiply\DT@modctr by \@strctr
924       \advance\DT@X by -\DT@modctr
925       \divide\@strctr by 16\relax
926       \advance\DT@loopN by -1\relax
927   \ifnum\@strctr>1
928     \repeat
929     \@hexadecimal\DT@X
930 }
931 \let\hexadecimalnum=\@hexadecimal

```

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

```

932 \newcommand*{\@@Hexadecimal}[1]{%
933   \ifcase#1\or 1\or 2\or 3\or 4\or 5\or 6\or
934   7\or 8\or 9\or A\or B\or C\or D\or E\or F\fi
935 }

```

`\Hexadecimalnum` Uppercase hexadecimal

```

936 \newcommand*{\@Hexadecimal}[1]{%
937   \DT@padzeroestru
938   \DT@loopN=5\relax
939   \@strctr=\DT@loopN
940   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
941   \@strctr=65536\relax
942   \DT@X=#1\relax
943   \loop
944     \DT@modctr=\DT@X
945     \divide\DT@modctr by \@strctr
946     \ifthenelse{\boolean{\DT@padzeroes}
947       \and \(\DT@modctr=0\}
948       \and \(\DT@loopN>\c@padzeroesN\)}%
949     {\@Hexadecimal\DT@modctr}%
950     \ifnum\DT@modctr=0\else\DT@padzeroesfalse\fi

```

```

951 \multiply\@DT@modctr by \@strctr
952 \advance\@DT@X by -\@DT@modctr
953 \divide\@strctr by 16\relax
954 \advance\@DT@loopN by -1\relax
955 \ifnum\@strctr>1
956 \repeat
957 \@Hexadecimal\@DT@X
958 }
959
960 \let\Hexadecimalnum=\@Hexadecimal

```

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

```

961 \newcommand*\@aaaalph}[1]{%
962 \@DT@loopN=#1\relax
963 \advance\@DT@loopN by -1\relax
964 \divide\@DT@loopN by 26\relax
965 \@DT@modctr=\@DT@loopN
966 \multiply\@DT@modctr by 26\relax
967 \@DT@X=#1\relax
968 \advance\@DT@X by -1\relax
969 \advance\@DT@X by -\@DT@modctr
970 \advance\@DT@loopN by 1\relax
971 \advance\@DT@X by 1\relax
972 \loop
973 \@alph\@DT@X
974 \advance\@DT@loopN by -1\relax
975 \ifnum\@DT@loopN>0
976 \repeat
977 }
978
979 \let\aaalphnum=\@aaaalph

```

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

```

980 \newcommand*\@AAAalph}[1]{%
981 \@DT@loopN=#1\relax
982 \advance\@DT@loopN by -1\relax
983 \divide\@DT@loopN by 26\relax
984 \@DT@modctr=\@DT@loopN
985 \multiply\@DT@modctr by 26\relax
986 \@DT@X=#1\relax
987 \advance\@DT@X by -1\relax
988 \advance\@DT@X by -\@DT@modctr
989 \advance\@DT@loopN by 1\relax
990 \advance\@DT@X by 1\relax
991 \loop
992 \@Alph\@DT@X
993 \advance\@DT@loopN by -1\relax
994 \ifnum\@DT@loopN>0
995 \repeat

```

```

996 }
997
998 \let\AAAAlphnum=\@AAAAlph

```

#### \abalphnum Lowercase alphabetical representation

```

999 \newcommand*{\@abalph}[1]{%
1000   \ifnum#1>17576\relax
1001     \PackageError{fmtcount}%
1002       {Value of counter too large for \protect\@abalph}%
1003       {Maximum value 17576}%
1004   \else
1005     \@DT@padzeroestru
1006     \@strctr=17576\relax
1007     \@DT@X=#1\relax
1008     \advance\@DT@X by -1\relax
1009     \loop
1010       \@DT@modctr=\@DT@X
1011       \divide\@DT@modctr by \@strctr
1012       \ifthenelse{\boolean{\@DT@padzeroes}
1013         \and \(\@DT@modctr=1\)}%
1014         {\@alph\@DT@modctr}%
1015       \ifnum\@DT@modctr=1\else\@DT@padzeroesfalse\fi
1016       \multiply\@DT@modctr by \@strctr
1017       \advance\@DT@X by -\@DT@modctr
1018       \divide\@strctr by 26\relax
1019     \ifnum\@strctr>1
1020     \repeat
1021     \advance\@DT@X by 1\relax
1022     \@alph\@DT@X
1023   \fi
1024 }
1025
1026 \let\abalphnum=\@abalph

```

#### \ABAlphnum Uppercase alphabetical representation

```

1027 \newcommand*{\@ABAlph}[1]{%
1028   \ifnum#1>17576\relax
1029     \PackageError{fmtcount}%
1030       {Value of counter too large for \protect\@ABAlph}%
1031       {Maximum value 17576}%
1032   \else
1033     \@DT@padzeroestru
1034     \@strctr=17576\relax
1035     \@DT@X=#1\relax
1036     \advance\@DT@X by -1\relax
1037     \loop
1038       \@DT@modctr=\@DT@X
1039       \divide\@DT@modctr by \@strctr
1040       \ifthenelse{\boolean{\@DT@padzeroes}\and

```

```

1041      \(\@DT@modctr=1\)}-{\@Alph\@DT@modctr}%
1042      \ifnum\@DT@modctr=1\else\@DT@padzeroesfalse\fi
1043      \multiply\@DT@modctr by \@strctr
1044      \advance\@DT@X by -\@DT@modctr
1045      \divide\@strctr by 26\relax
1046      \ifnum\@strctr>1
1047      \repeat
1048      \advance\@DT@X by 1\relax
1049      \@Alph\@DT@X
1050 \fi
1051 }
1052
1053 \let\ABAlphnum=\@ABAlph

```

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

```

1054 \def\@fmtc@count#1#2\relax{%
1055   \if\relax#1%
1056   \else
1057     \advance\@strctr by 1\relax
1058     \@fmtc@count#2\relax
1059   \fi
1060 }

```

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

```

1061 \newcommand{\@decimal}[1]{%
1062   \@strctr=0\relax
1063   \expandafter\@fmtc@count\number#1\relax
1064   \@DT@loopN=\c@padzeroesN
1065   \advance\@DT@loopN by -\@strctr
1066   \ifnum\@DT@loopN>0\relax
1067     \@strctr=0\relax
1068     \whiledo{\@strctr < \@DT@loopN}{0\advance\@strctr by 1\relax}%
1069   \fi
1070   \number#1\relax
1071 }
1072
1073 \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 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.

```
1074 \newcommand{\FCordinal}[1]{%
1075   \expandafter\protect\expandafter\ordinalnum{%
1076     \expandafter\the\csname c@#1\endcsname}%
1077 }
```

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

```
1078 \@ifundefined{ordinal}
1079 {\let\ordinal\FCordinal}%
1080 {%
1081   \PackageWarning{fmtcount}%
1082   {\string\ordinal \space already defined use
1083    \string\FCordinal \space instead.}
1084 }
```

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

```
1085 \newcommand*{\ordinalnum}[1]{%
1086   \new@ifnextchar[%
1087     {\@ordinalnum{#1}}%
1088     {\@ordinalnum{#1}[m]}%
1089 }
```

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

```
1090 \def\@ordinalnum#1[#2]{%
1091   {%
1092     \ifthenelse{\equal{#2}{f}}%
1093     {%
1094       \protect\@ordinalF{#1}{\@fc@ordstr}%
1095     }%
1096     {%
1097       \ifthenelse{\equal{#2}{n}}%
1098       {%
1099         \protect\@ordinalN{#1}{\@fc@ordstr}%
1100       }%
1101       {%
1102         \ifthenelse{\equal{#2}{m}}%
1103         {}%
1104         {%
1105           \PackageError{fmtcount}%
1106             {Invalid gender option '#2'}%

```

---

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

```

1107         {Available options are m, f or n}%
1108     }%
1109     \protect\@ordinalM{#1}{\@fc@ordstr}%
1110 }%
1111 }%
1112 \@fc@ordstr
1113 }%
1114 }

```

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

```

1115 \newcommand*{\storeordinal}[2]{%
1116   \expandafter\protect\expandafter\storeordinalnum{#1}{%
1117     \expandafter\the\csname c@#2\endcsname}%
1118 }

```

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

```

1119 \newcommand*{\storeordinalnum}[2]{%
1120   \@ifnextchar[%
1121     {\@storeordinalnum{#1}{#2}}%
1122     {\@storeordinalnum{#1}{#2}[m]}%
1123 }

```

`\@storeordinalnum` Store ordinal according to gender:

```

1124 \def\@storeordinalnum#1#2[#3]{%
1125   \ifthenelse{\equal{#3}{f}}{%
1126     {%
1127       \protect\@ordinalF{#2}{\@fc@ord}%
1128     }%
1129     {%
1130       \ifthenelse{\equal{#3}{n}}{%
1131         {%
1132           \protect\@ordinalN{#2}{\@fc@ord}%
1133         }%
1134         {%
1135           \ifthenelse{\equal{#3}{m}}{%
1136             {}%
1137             {%
1138               \PackageError{fmtcount}%
1139                 {Invalid gender option ‘#3’}%
1140                 {Available options are m or f}%
1141             }%
1142           \protect\@ordinalM{#2}{\@fc@ord}%
1143         }%
1144       }%
1145       \expandafter\let\csname @fcs@#1\endcsname\@fc@ord
1146 }

```

\FMCuse Get stored information:

```
1147 \newcommand*{\FMCuse}[1]{\csname @fcs@#1\endcsname}
```

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

```
1148 \newcommand*{\ordinalstring}[1]{%
1149   \expandafter\protect\expandafter\ordinalstringnum{%
1150     \expandafter\the\csname c@#1\endcsname}%
1151 }
```

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

```
1152 \newcommand{\ordinalstringnum}[1]{%
1153   \new@ifnextchar[%
1154     {\@ordinal@string{#1}}%
1155     {\@ordinal@string{#1}[m]}%
1156 }
```

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

```
1157 \def\@ordinal@string#1[#2]{%
1158   {%
1159     \ifthenelse{\equal{#2}{f}}%
1160     {%
1161       \protect\@ordinalstringF{#1}{\@fc@ordstr}%
1162     }%
1163     {%
1164       \ifthenelse{\equal{#2}{n}}%
1165       {%
1166         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
1167       }%
1168       {%
1169         \ifthenelse{\equal{#2}{m}}%
1170         {}%
1171         {%
1172           \PackageError{fmtcount}%
1173             {Invalid gender option ‘#2’ to \string\ordinalstring}%
1174             {Available options are m, f or f}%
1175         }%
1176         \protect\@ordinalstringM{#1}{\@fc@ordstr}%
1177       }%
1178     }%
1179     \@fc@ordstr
1180   }%
1181 }
```

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

```
1182 \newcommand*{\storeordinalstring}[2]{%
1183   \expandafter\protect\expandafter\storeordinalstringnum{#1}{%
1184     \expandafter\the\csname c@#2\endcsname}%
1185 }
```

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

```
1186 \newcommand*{\storeordinalstringnum}[2]{%
1187   \@ifnextchar[%
1188     {\@store@ordinal@string{#1}{#2}}%
1189     {\@store@ordinal@string{#1}{#2}[m]}}%
1190 }
```

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

```
1191 \def\@store@ordinal@string#1#2[#3]{%
1192   \ifthenelse{\equal{#3}{f}}{%
1193     {%
1194       \protect\@ordinalstringF{#2}{\@fc@ordstr}%
1195     }%
1196     {%
1197       \ifthenelse{\equal{#3}{n}}{%
1198         {%
1199           \protect\@ordinalstringN{#2}{\@fc@ordstr}%
1200         }%
1201         {%
1202           \ifthenelse{\equal{#3}{m}}{%
1203             {}%
1204             {%
1205               \PackageError{fmtcount}%
1206                 {Invalid gender option ‘#3’ to \string\ordinalstring}%
1207                 {Available options are m, f or n}%
1208             }%
1209           \protect\@ordinalstringM{#2}{\@fc@ordstr}%
1210         }%
1211       }%
1212       \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
1213 }
```

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

```
1214 \newcommand*{\Ordinalstring}[1]{%
1215   \expandafter\protect\expandafter\Ordinalstringnum{%
1216     \expandafter\the\csname c@#1\endcsname}%
1217 }
```

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

```
1218 \newcommand*{\Ordinalstringnum}[1]{%
1219   \new@ifnextchar[%
1220     {\@Ordinal@string{#1}}%
1221     {\@Ordinal@string{#1}[m]}}%
1222 }
```

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

```

1223 \def\@Ordinal@string#1[#2]{%
1224   {%
1225     \ifthenelse{\equal{#2}{f}}{%
1226       {%
1227         \protect\@OrdinalstringF{#1}{\@fc@ordstr}%
1228       }%
1229     {%
1230       \ifthenelse{\equal{#2}{n}}{%
1231         {%
1232           \protect\@OrdinalstringN{#1}{\@fc@ordstr}%
1233         }%
1234       {%
1235         \ifthenelse{\equal{#2}{m}}{%
1236           {}%
1237         {%
1238           \PackageError{fmtcount}%
1239             {Invalid gender option ‘#2’}%
1240             {Available options are m, f or n}%
1241         }%
1242         \protect\@OrdinalstringM{#1}{\@fc@ordstr}%
1243       }%
1244     }%
1245     \@fc@ordstr
1246   }%
1247 }

```

`\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.

```

1248 \newcommand*\storeOrdinalstring}[2]{%
1249   \expandafter\protect\expandafter\storeOrdinalstringnum{#1}{%
1250     \expandafter\the\csname c@#2\endcsname}%
1251 }

```

`\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.

```

1252 \newcommand*\storeOrdinalstringnum}[2]{%
1253   \@ifnextchar[%
1254     {\@store@Ordinal@string{#1}{#2}}%
1255     {\@store@Ordinal@string{#1}{#2}[m]}%
1256 }

```

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

```

1257 \def\@store@Ordinal@string#1#2[#3]{%
1258   \ifthenelse{\equal{#3}{f}}{%
1259     {%

```

```

1260 \protect\@OrdinalstringF{#2}{\@fc@ordstr}%
1261 }%
1262 {%
1263 \ifthenelse{\equal{#3}{n}}%
1264 {%
1265 \protect\@OrdinalstringN{#2}{\@fc@ordstr}%
1266 }%
1267 {%
1268 \ifthenelse{\equal{#3}{m}}%
1269 {}%
1270 {%
1271 \PackageError{fmtcount}%
1272 {Invalid gender option ‘#3’}%
1273 {Available options are m or f}%
1274 }%
1275 \protect\@OrdinalstringM{#2}{\@fc@ordstr}%
1276 }%
1277 }%
1278 \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
1279 }

```

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

```

1280 \newcommand*{\storeORDINALstring}[2]{%
1281 \expandafter\protect\expandafter\storeORDINALstringnum{#1}{%
1282 \expandafter\the\csname c@#2\endcsname}%
1283 }

```

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

```

1284 \newcommand*{\storeORDINALstringnum}[2]{%
1285 \@ifnextchar[%
1286 {\@store@ORDINAL@string{#1}{#2}}%
1287 {\@store@ORDINAL@string{#1}{#2}[m]}%
1288 }

```

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

```

1289 \def\@store@ORDINAL@string#1#2[#3]{%
1290 \ifthenelse{\equal{#3}{f}}%
1291 {%
1292 \protect\@ordinalstringF{#2}{\@fc@ordstr}%
1293 }%
1294 {%
1295 \ifthenelse{\equal{#3}{n}}%
1296 {%
1297 \protect\@ordinalstringN{#2}{\@fc@ordstr}%
1298 }%
1299 {%
1300 \ifthenelse{\equal{#3}{m}}%
1301 {}%

```

```

1302      {%
1303      \PackageError{fmtcount}%
1304      {Invalid gender option ‘#3’}%
1305      {Available options are m or f}%
1306      }%
1307      \protect\@ordinalstringM{#2}{\@fc@ordstr}%
1308  }%
1309 }%
1310 \expandafter\edef\csname @fcs@#1\endcsname{%
1311 \noexpand\MakeUppercase{\@fc@ordstr}%
1312 }%
1313 }

```

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

```

1314 \newcommand*{\ORDINALstring}[1]{%
1315 \expandafter\protect\expandafter\ORDINALstringnum{%
1316 \expandafter\the\csname c@#1\endcsname
1317 }%
1318 }

```

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

```

1319 \newcommand*{\ORDINALstringnum}[1]{%
1320 \new@ifnextchar[%
1321 {\@ORDINAL@string{#1}}%
1322 {\@ORDINAL@string{#1}[m]}%
1323 }

```

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

```

1324 \def\@ORDINAL@string#1[#2]{%
1325 {%
1326 \ifthenelse{\equal{#2}{f}}%
1327 {%
1328 \protect\@ordinalstringF{#1}{\@fc@ordstr}%
1329 }%
1330 {%
1331 \ifthenelse{\equal{#2}{n}}%
1332 {%
1333 \protect\@ordinalstringN{#1}{\@fc@ordstr}%
1334 }%
1335 {%
1336 \ifthenelse{\equal{#2}{m}}%
1337 {}%
1338 {%
1339 \PackageError{fmtcount}%
1340 {Invalid gender option ‘#2’}%
1341 {Available options are m, f or n}%
1342 }%
1343 \protect\@ordinalstringM{#1}{\@fc@ordstr}%

```

```

1344     }%
1345 }%
1346 \MakeUppercase{\@fc@ordstr}%
1347 }%
1348 }

```

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

```

1349 \newcommand*\storenumberstring[2]{%
1350   \expandafter\protect\expandafter\storenumberstringnum{#1}{%
1351     \expandafter\the\csname c@#2\endcsname}%
1352 }

```

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

```

1353 \newcommand\storenumberstringnum[2]{%
1354   \@ifnextchar[%
1355     {\@store@number@string{#1}{#2}}%
1356     {\@store@number@string{#1}{#2}[m]}%
1357 }

```

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

```

1358 \def\@store@number@string#1#2[#3]{%
1359   \ifthenelse{\equal{#3}{f}}{%
1360     {%
1361       \protect\@numberstringF{#2}{\@fc@numstr}%
1362     }%
1363     {%
1364       \ifthenelse{\equal{#3}{n}}{%
1365         {%
1366           \protect\@numberstringN{#2}{\@fc@numstr}%
1367         }%
1368         {%
1369           \ifthenelse{\equal{#3}{m}}{%
1370             {}%
1371             {%
1372               \PackageError{fmtcount}
1373               {Invalid gender option ‘#3’}%
1374               {Available options are m, f or n}%
1375             }%
1376           \protect\@numberstringM{#2}{\@fc@numstr}%
1377         }%
1378       }%
1379       \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
1380 }

```

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

```

1381 \newcommand*\numberstring[1]{%
1382   \expandafter\protect\expandafter\numberstringnum{%
1383     \expandafter\the\csname c@#1\endcsname}%

```

1384 }

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

```
1385 \newcommand*{\numberstringnum}[1]{%
1386   \new@ifnextchar[%
1387     {\@number@string{#1}}}%
1388     {\@number@string{#1}[m]}}%
1389 }
```

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

```
1390 \def\@number@string#1[#2]{%
1391   {%
1392     \ifthenelse{\equal{#2}{f}}{%
1393       {%
1394         \protect\@numberstringF{#1}{\@fc@numstr}%
1395       }%
1396       {%
1397         \ifthenelse{\equal{#2}{n}}{%
1398           {%
1399             \protect\@numberstringN{#1}{\@fc@numstr}%
1400           }%
1401           {%
1402             \ifthenelse{\equal{#2}{m}}{%
1403               {}%
1404               {%
1405                 \PackageError{fmtcount}%
1406                   {Invalid gender option ‘#2’}%
1407                   {Available options are m, f or n}%
1408               }%
1409               \protect\@numberstringM{#1}{\@fc@numstr}%
1410             }%
1411           }%
1412         \@fc@numstr
1413       }%
1414     }
```

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

```
1415 \newcommand*{\storeNumberstring}[2]{%
1416   \expandafter\protect\expandafter\storeNumberstringnum{#1}{%
1417     \expandafter\the\csname c@#2\endcsname}%
1418 }
```

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

```
1419 \newcommand{\storeNumberstringnum}[2]{%
1420   \@ifnextchar[%
1421     {\@store@Number@string{#1}{#2}}%
1422     {\@store@Number@string{#1}{#2}[m]}}%
1423 }
```

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

```

1424 \def\@store@Number@string#1#2[#3]{%
1425   \ifthenelse{\equal{#3}{f}}{%
1426     {%
1427       \protect\@NumberstringF{#2}{\@fc@numstr}%
1428     }%
1429     {%
1430       \ifthenelse{\equal{#3}{n}}{%
1431         {%
1432           \protect\@NumberstringN{#2}{\@fc@numstr}%
1433         }%
1434         {%
1435           \ifthenelse{\equal{#3}{m}}{%
1436             {}%
1437             {%
1438               \PackageError{fmtcount}%
1439                 {Invalid gender option ‘#3’}%
1440                 {Available options are m, f or n}%
1441             }%
1442           \protect\@NumberstringM{#2}{\@fc@numstr}%
1443         }%
1444       }%
1445     \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
1446 }

```

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

```

1447 \newcommand*\@Numberstring}[1]{%
1448   \expandafter\protect\expandafter\Numberstringnum{%
1449     \expandafter\the\csname c@#1\endcsname}%
1450 }

```

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

```

1451 \newcommand*\@Numberstringnum}[1]{%
1452   \new@ifnextchar[%
1453     {\@Number@string{#1}}%
1454     {\@Number@string{#1}[m]}%
1455 }

```

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

```

1456 \def\@Number@string#1[#2]{%
1457   {%
1458     \ifthenelse{\equal{#2}{f}}{%
1459       {%
1460         \protect\@NumberstringF{#1}{\@fc@numstr}%
1461       }%
1462       {%
1463         \ifthenelse{\equal{#2}{n}}{%
1464           {%
1465             \protect\@NumberstringN{#1}{\@fc@numstr}%

```

```

1466     }%
1467     {%
1468     \ifthenelse{\equal{#2}{m}}%
1469     {}%
1470     {%
1471     \PackageError{fmtcount}%
1472     {Invalid gender option ‘#2’}%
1473     {Available options are m, f or n}%
1474     }%
1475     \protect\@NumberstringM{#1}{\@fc@numstr}%
1476     }%
1477     }%
1478     \@fc@numstr
1479 }%
1480 }

```

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

```

1481 \newcommand{\storeNUMBERstring}[2]{%
1482   \expandafter\protect\expandafter\storeNUMBERstringnum{#1}{%
1483     \expandafter\the\csname c@#2\endcsname}%
1484 }

```

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

```

1485 \newcommand{\storeNUMBERstringnum}[2]{%
1486   \@ifnextchar[%
1487   {\@store@NUMBER@string{#1}{#2}}%
1488   {\@store@NUMBER@string{#1}{#2}[m]}%
1489 }

```

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

```

1490 \def\@store@NUMBER@string#1#2[#3]{%
1491   \ifthenelse{\equal{#3}{f}}%
1492   {%
1493     \protect\@numberstringF{#2}{\@fc@numstr}%
1494   }%
1495   {%
1496     \ifthenelse{\equal{#3}{n}}%
1497     {%
1498       \protect\@numberstringN{#2}{\@fc@numstr}%
1499     }%
1500     {%
1501       \ifthenelse{\equal{#3}{m}}%
1502       {}%
1503       {%
1504         \PackageError{fmtcount}%
1505         {Invalid gender option ‘#3’}%
1506         {Available options are m or f}%
1507       }%

```

```

1508     \protect\@numberstringM{#2}{\@fc@numstr}%
1509   }%
1510 }%
1511 \expandafter\edef\csname @fcs@#1\endcsname{%
1512   \noexpand\MakeUppercase{\@fc@numstr}%
1513 }%
1514 }

```

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

```

1515 \newcommand*{\NUMBERstring}[1]{%
1516   \expandafter\protect\expandafter\NUMBERstringnum{%
1517     \expandafter\the\csname c@#1\endcsname}%
1518 }

```

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

```

1519 \newcommand*{\NUMBERstringnum}[1]{%
1520   \new@ifnextchar[%
1521     {\@NUMBER@string{#1}}%
1522     {\@NUMBER@string{#1}[m]}%
1523 }

```

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

```

1524 \def\@NUMBER@string#1[#2]{%
1525   {%
1526     \ifthenelse{\equal{#2}{f}}%
1527     {%
1528       \protect\@numberstringF{#1}{\@fc@numstr}%
1529     }%
1530     {%
1531       \ifthenelse{\equal{#2}{n}}%
1532       {%
1533         \protect\@numberstringN{#1}{\@fc@numstr}%
1534       }%
1535       {%
1536         \ifthenelse{\equal{#2}{m}}%
1537         {}%
1538         {%
1539           \PackageError{fmtcount}%
1540             {Invalid gender option ‘#2’}%
1541             {Available options are m, f or n}%
1542         }%
1543         \protect\@numberstringM{#1}{\@fc@numstr}%
1544       }%
1545     }%
1546     \MakeUppercase{\@fc@numstr}%
1547   }%
1548 }

```

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

```
1549 \providecommand*\binary}[1]{%
1550   \expandafter\protect\expandafter\@binary{%
1551     \expandafter\the\csname c@#1\endcsname}%
1552 }
```

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

```
1553 \providecommand*\aaalph}[1]{%
1554   \expandafter\protect\expandafter\@aaalph{%
1555     \expandafter\the\csname c@#1\endcsname}%
1556 }
```

`\AAAAlph` As before, but upper case.

```
1557 \providecommand*\AAAAlph}[1]{%
1558   \expandafter\protect\expandafter\@AAAAlph{%
1559     \expandafter\the\csname c@#1\endcsname}%
1560 }
```

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

```
1561 \providecommand*\abalph}[1]{%
1562   \expandafter\protect\expandafter\@abalph{%
1563     \expandafter\the\csname c@#1\endcsname}%
1564 }
```

`\ABAlph` As above, but upper case.

```
1565 \providecommand*\ABAlph}[1]{%
1566   \expandafter\protect\expandafter\@ABAlph{%
1567     \expandafter\the\csname c@#1\endcsname}%
1568 }
```

`\hexadecimal` Hexadecimal:

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

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

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

`\octal` Octal:

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

`\decimal` Decimal:

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

### 9.3 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.

```
1585 \def\@setdef@ultfmtcount{%
1586   \@ifundefined{@ordinalMenglish}{\FCloadlang{english}}{}%
1587   \def\@ordinalstringM{\@ordinalstringMenglish}%
1588   \let\@ordinalstringF=\@ordinalstringMenglish
1589   \let\@ordinalstringN=\@ordinalstringMenglish
1590   \def\@OrdinalstringM{\@OrdinalstringMenglish}%
1591   \let\@OrdinalstringF=\@OrdinalstringMenglish
1592   \let\@OrdinalstringN=\@OrdinalstringMenglish
1593   \def\@numberstringM{\@numberstringMenglish}%
1594   \let\@numberstringF=\@numberstringMenglish
1595   \let\@numberstringN=\@numberstringMenglish
1596   \def\@NumberstringM{\@NumberstringMenglish}%
1597   \let\@NumberstringF=\@NumberstringMenglish
1598   \let\@NumberstringN=\@NumberstringMenglish
1599   \def\@ordinalM{\@ordinalMenglish}%
1600   \let\@ordinalF=\@ordinalM
1601   \let\@ordinalN=\@ordinalM
1602 }
```

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

```
1603 \def\@set@multiling@fmtcount{%
  The masculine version of \@numberstring:
1604   \def\@numberstringM{%
1605     \@ifundefined{@numberstringM\language}%
1606     {%
1607       \PackageWarning{fmtcount}%
1608       {No support for \@string\@numberstring\space for
1609       language '\language'}%
1610       \@numberstringMenglish
1611     }%
1612     {%
1613       \csname @numberstringM\language\endcsname
1614     }%
1615   }%
```

The feminine version of `\@numberstring`:

```
1616   \def\@numberstringF{%
```

```

1617 \@ifundefined{@numberstringF\language}%
1618 {%
1619     \PackageWarning{fmtcount}%
1620     {No support for \string\numberstring\space for
1621     language '\language'}%
1622     \@numberstringMenglish
1623 }%
1624 {%
1625     \csname @numberstringF\language\endcsname
1626 }%
1627 }%

```

The neuter version of \numberstring:

```

1628 \def\@numberstringN{%
1629     \@ifundefined{@numberstringN\language}%
1630     {%
1631         \PackageWarning{fmtcount}%
1632         {No support for \string\numberstring\space for
1633         language '\language'}%
1634         \@numberstringMenglish
1635     }%
1636     {%
1637         \csname @numberstringN\language\endcsname
1638     }%
1639 }%

```

The masculine version of \Numberstring:

```

1640 \def\@NumberstringM{%
1641     \@ifundefined{@NumberstringM\language}%
1642     {%
1643         \PackageWarning{fmtcount}%
1644         {No support for \string\Numberstring\space
1645         for language '\language'}%
1646         \@NumberstringMenglish
1647     }%
1648     {%
1649         \csname @NumberstringM\language\endcsname
1650     }%
1651 }%

```

The feminine version of \Numberstring:

```

1652 \def\@NumberstringF{%
1653     \@ifundefined{@NumberstringF\language}%
1654     {%
1655         \PackageWarning{fmtcount}%
1656         {No support for \string\Numberstring\space
1657         for language '\language'}%
1658         \@NumberstringMenglish
1659     }%
1660     {%
1661         \csname @NumberstringF\language\endcsname

```

1662 }%

1663 }%

The neuter version of \Numberstring:

```
1664 \def\@NumberstringN{%
1665   \@ifundefined{\@NumberstringN\language}%
1666   {%
1667     \PackageWarning{fmtcount}%
1668     {No support for \string\Numberstring\space
1669     for language '\language'}%
1670   \@NumberstringMenglish
1671   }%
1672   {%
1673     \csname \@NumberstringN\language\endcsname
1674   }%
1675   }%
```

The masculine version of \ordinal:

```
1676 \def\@OrdinalM{%
1677   \@ifundefined{\@OrdinalM\language}%
1678   {%
1679     \PackageWarning{fmtcount}%
1680     {No support for \string\ordinal\space
1681     for language '\language'}%
1682   \@OrdinalMenglish
1683   }%
1684   {%
1685     \csname \@OrdinalM\language\endcsname
1686   }%
1687   }%
```

The feminine version of \ordinal:

```
1688 \def\@OrdinalF{%
1689   \@ifundefined{\@OrdinalF\language}%
1690   {%
1691     \PackageWarning{fmtcount}%
1692     {No support for \string\ordinal\space
1693     for language '\language'}%
1694   \@OrdinalMenglish
1695   }%
1696   {%
1697     \csname \@OrdinalF\language\endcsname
1698   }%
1699   }%
```

The neuter version of \ordinal:

```
1700 \def\@OrdinalN{%
1701   \@ifundefined{\@OrdinalN\language}%
1702   {%
1703     \PackageWarning{fmtcount}%
1704     {No support for \string\ordinal\space
```

```

1705     for language '\language'}%
1706     \@ordinalMenglish
1707 }%
1708 {%
1709     \csname @ordinalN\language\endcsname
1710 }%
1711 }%

```

The masculine version of \ordinalstring:

```

1712 \def\@ordinalstringM{%
1713     \@ifundefined{@ordinalstringM\language}%
1714     {%
1715         \PackageWarning{fmtcount}%
1716         {No support for \string\ordinalstring\space
1717         for language '\language'}%
1718         \@ordinalstringMenglish
1719     }%
1720     {%
1721         \csname @ordinalstringM\language\endcsname
1722     }%
1723 }%

```

The feminine version of \ordinalstring:

```

1724 \def\@ordinalstringF{%
1725     \@ifundefined{@ordinalstringF\language}%
1726     {%
1727         \PackageWarning{fmtcount}%
1728         {No support for \string\ordinalstring\space
1729         for language '\language'}%
1730         \@ordinalstringFenglish
1731     }%
1732     {%
1733         \csname @ordinalstringF\language\endcsname
1734     }%
1735 }%

```

The neuter version of \ordinalstring:

```

1736 \def\@ordinalstringN{%
1737     \@ifundefined{@ordinalstringN\language}%
1738     {%
1739         \PackageWarning{fmtcount}%
1740         {No support for \string\ordinalstring\space
1741         for language '\language'}%
1742         \@ordinalstringNenglish
1743     }%
1744     {%
1745         \csname @ordinalstringN\language\endcsname
1746     }%
1747 }%

```

The masculine version of \Ordinalstring:

```

1748 \def\@OrdinalstringM{%
1749   \@ifundefined{\@OrdinalstringM\language}%
1750   {%
1751     \PackageWarning{fmtcount}%
1752     {No support for \string\Ordinalstring\space
1753     for language '\language'}%
1754     \@OrdinalstringMenglish
1755   }%
1756   {%
1757     \csname \@OrdinalstringM\language\endcsname
1758   }%
1759 }%

```

The feminine version of \Ordinalstring:

```

1760 \def\@OrdinalstringF{%
1761   \@ifundefined{\@OrdinalstringF\language}%
1762   {%
1763     \PackageWarning{fmtcount}%
1764     {No support for \string\Ordinalstring\space
1765     for language '\language'}%
1766     \@OrdinalstringMenglish
1767   }%
1768   {%
1769     \csname \@OrdinalstringF\language\endcsname
1770   }%
1771 }%

```

The neuter version of \Ordinalstring:

```

1772 \def\@OrdinalstringN{%
1773   \@ifundefined{\@OrdinalstringN\language}%
1774   {%
1775     \PackageWarning{fmtcount}%
1776     {No support for \string\Ordinalstring\space
1777     for language '\language'}%
1778     \@OrdinalstringMenglish
1779   }%
1780   {%
1781     \csname \@OrdinalstringN\language\endcsname
1782   }%
1783 }%
1784 }

```

Check to see if babel or ngerman packages have been loaded.

```

1785 \@ifpackageloaded{babel}%
1786 {%
1787   \@set@multiling@fmtcount
1788 }%
1789 {%
1790   \@ifpackageloaded{ngerman}%
1791   {%

```

```

1792 \FCloadlang{ngerman}%
1793 \@set@multiling@fmtcount
1794 }%
1795 {%
1796 \@setdef@ultfmtcount
1797 }%
1798 }

```

Backwards compatibility:

```

1799 \let\@ordinal=\@ordinalM
1800 \let\@ordinalstring=\@ordinalstringM
1801 \let\@Ordinalstring=\@OrdinalstringM
1802 \let\@numberstring=\@numberstringM
1803 \let\@Numberstring=\@NumberstringM

```

### 9.3.1 fc-american.def

American English definitions

```
1804 \ProvidesFCLanguage{american}[2012/06/18]
```

Loaded fc-USenglish.def if not already loaded

```
1805 \FCloadlang{USenglish}
```

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

```

1806 \let\@ordinalMamerican\@ordinalMUSenglish
1807 \let\@ordinalFamerican\@ordinalMUSenglish
1808 \let\@ordinalNamerican\@ordinalMUSenglish
1809 \let\@numberstringMamerican\@numberstringMUSenglish
1810 \let\@numberstringFamerican\@numberstringMUSenglish
1811 \let\@numberstringNamerican\@numberstringMUSenglish
1812 \let\@NumberstringMamerican\@NumberstringMUSenglish
1813 \let\@NumberstringFamerican\@NumberstringMUSenglish
1814 \let\@NumberstringNamerican\@NumberstringMUSenglish
1815 \let\@ordinalstringMamerican\@ordinalstringMUSenglish
1816 \let\@ordinalstringFamerican\@ordinalstringMUSenglish
1817 \let\@ordinalstringNamerican\@ordinalstringMUSenglish
1818 \let\@OrdinalstringMamerican\@OrdinalstringMUSenglish
1819 \let\@OrdinalstringFamerican\@OrdinalstringMUSenglish
1820 \let\@OrdinalstringNamerican\@OrdinalstringMUSenglish

```

### 9.3.2 fc-british.def

British definitions

```
1821 \ProvidesFCLanguage{british}[2012/06/18]
```

Load fc-english.def, if not already loaded

```
1822 \FCloadlang{english}
```

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

```

1823 \let\@ordinalMbritish\@ordinalMenglish
1824 \let\@ordinalFbritish\@ordinalMenglish

```

```

1825 \let\@ordinalNbritish\@ordinalMenglish
1826 \let\@numberstringMbritish\@numberstringMenglish
1827 \let\@numberstringFbritish\@numberstringMenglish
1828 \let\@numberstringNbritish\@numberstringMenglish
1829 \let\@NumberstringMbritish\@NumberstringMenglish
1830 \let\@NumberstringFbritish\@NumberstringMenglish
1831 \let\@NumberstringNbritish\@NumberstringMenglish
1832 \let\@ordinalstringMbritish\@ordinalstringMenglish
1833 \let\@ordinalstringFbritish\@ordinalstringMenglish
1834 \let\@ordinalstringNbritish\@ordinalstringMenglish
1835 \let\@OrdinalstringMbritish\@OrdinalstringMenglish
1836 \let\@OrdinalstringFbritish\@OrdinalstringMenglish
1837 \let\@OrdinalstringNbritish\@OrdinalstringMenglish

```

### 9.3.3 fc-english.def

English definitions

```
1838 \ProvidesFCLanguage{english}[2012/06/18]
```

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.

```

1839 \newcommand*{\@ordinalMenglish}[2]{%
1840 \def\@fc@ord{}%
1841 \@orgargctr=#1\relax
1842 \@ordinalctr=#1%
1843 \@modulo{\@ordinalctr}{100}%
1844 \ifnum\@ordinalctr=11\relax
1845   \def\@fc@ord{th}%
1846 \else
1847   \ifnum\@ordinalctr=12\relax
1848     \def\@fc@ord{th}%
1849   \else
1850     \ifnum\@ordinalctr=13\relax
1851       \def\@fc@ord{th}%
1852     \else
1853       \@modulo{\@ordinalctr}{10}%
1854       \ifcase\@ordinalctr
1855         \def\@fc@ord{th}%           case 0
1856         \or \def\@fc@ord{st}%       case 1
1857         \or \def\@fc@ord{nd}%       case 2
1858         \or \def\@fc@ord{rd}%       case 3
1859       \else
1860         \def\@fc@ord{th}%           default case
1861       \fi
1862     \fi
1863   \fi
1864 \fi
1865 \edef#2{\number#1\relax\noexpand\fmtord{\@fc@ord}}%

```

1866 }

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

1867 \let\@ordinalFenglish=\@ordinalMenglish

1868 \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.

1869 \newcommand\*\@@unitstringenglish}[1]{%

1870 \ifcase#1\relax

1871 zero%

1872 \or one%

1873 \or two%

1874 \or three%

1875 \or four%

1876 \or five%

1877 \or six%

1878 \or seven%

1879 \or eight%

1880 \or nine%

1881 \fi

1882 }

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

1883 \newcommand\*\@@tenstringenglish}[1]{%

1884 \ifcase#1\relax

1885 \or ten%

1886 \or twenty%

1887 \or thirty%

1888 \or forty%

1889 \or fifty%

1890 \or sixty%

1891 \or seventy%

1892 \or eighty%

1893 \or ninety%

1894 \fi

1895 }

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

1896 \newcommand\*\@@teenstringenglish}[1]{%

1897 \ifcase#1\relax

1898 ten%

1899 \or eleven%

1900 \or twelve%

1901 \or thirteen%

1902 \or fourteen%

1903 \or fifteen%

1904 \or sixteen%

1905 \or seventeen%

```

1906 \or eighteen%
1907 \or nineteen%
1908 \fi
1909 }

```

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

```

1910 \newcommand*\@@Unitstringenglish}[1]{%
1911 \ifcase#1\relax
1912 Zero%
1913 \or One%
1914 \or Two%
1915 \or Three%
1916 \or Four%
1917 \or Five%
1918 \or Six%
1919 \or Seven%
1920 \or Eight%
1921 \or Nine%
1922 \fi
1923 }

```

The tens:

```

1924 \newcommand*\@@Tenstringenglish}[1]{%
1925 \ifcase#1\relax
1926 \or Ten%
1927 \or Twenty%
1928 \or Thirty%
1929 \or Forty%
1930 \or Fifty%
1931 \or Sixty%
1932 \or Seventy%
1933 \or Eighty%
1934 \or Ninety%
1935 \fi
1936 }

```

The teens:

```

1937 \newcommand*\@@Teenstringenglish}[1]{%
1938 \ifcase#1\relax
1939 Ten%
1940 \or Eleven%
1941 \or Twelve%
1942 \or Thirteen%
1943 \or Fourteen%
1944 \or Fifteen%
1945 \or Sixteen%
1946 \or Seventeen%
1947 \or Eighteen%
1948 \or Nineteen%
1949 \fi
1950 }

```

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.)

```

1951 \newcommand*{\@@numberstringenglish}[2]{%
1952 \ifnum#1>99999
1953 \PackageError{fmtcount}{Out of range}%
1954 {This macro only works for values less than 100000}%
1955 \else
1956 \ifnum#1<0
1957 \PackageError{fmtcount}{Negative numbers not permitted}%
1958 {This macro does not work for negative numbers, however
1959 you can try typing "minus" first, and then pass the modulus of
1960 this number}%
1961 \fi
1962 \fi
1963 \def#2{}%
1964 \@strctr=#1\relax \divide\@strctr by 1000\relax
1965 \ifnum\@strctr>9
1966   \divide\@strctr by 10
1967   \ifnum\@strctr>1\relax
1968     \let\@@fc@numstr#2\relax
1969     \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
1970     \@strctr=#1 \divide\@strctr by 1000\relax
1971     \@modulo{\@strctr}{10}%
1972     \ifnum\@strctr>0\relax
1973       \let\@@fc@numstr#2\relax
1974       \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
1975     \fi
1976   \else
1977     \@strctr=#1\relax
1978     \divide\@strctr by 1000\relax
1979     \@modulo{\@strctr}{10}%
1980     \let\@@fc@numstr#2\relax
1981     \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
1982   \fi
1983   \let\@@fc@numstr#2\relax
1984   \edef#2{\@@fc@numstr\@thousand}%
1985 \else
1986   \ifnum\@strctr>0\relax
1987     \let\@@fc@numstr#2\relax
1988     \edef#2{\@@fc@numstr\@unitstring{\@strctr}\@thousand}%
1989   \fi
1990 \fi
1991 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
1992 \divide\@strctr by 100
1993 \ifnum\@strctr>0\relax
1994   \ifnum#1>1000\relax
1995     \let\@@fc@numstr#2\relax

```

```

1996      \edef#2{\@@fc@numstr\ }%
1997  \fi
1998  \let\@@fc@numstr#2\relax
1999  \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@hundred}%
2000 \fi
2001 \@strctr=#1\relax \@modulo{\@strctr}{100}%
2002 \ifnum#1>100\relax
2003   \ifnum\@strctr>0\relax
2004     \let\@@fc@numstr#2\relax
2005     \edef#2{\@@fc@numstr\ \@andname\ }%
2006   \fi
2007 \fi
2008 \ifnum\@strctr>19\relax
2009   \divide\@strctr by 10\relax
2010   \let\@@fc@numstr#2\relax
2011   \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
2012   \@strctr=#1\relax \@modulo{\@strctr}{10}%
2013   \ifnum\@strctr>0\relax
2014     \let\@@fc@numstr#2\relax
2015     \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
2016   \fi
2017 \else
2018   \ifnum\@strctr<10\relax
2019     \ifnum\@strctr=0\relax
2020       \ifnum#1<100\relax
2021         \let\@@fc@numstr#2\relax
2022         \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
2023       \fi
2024     \else
2025       \let\@@fc@numstr#2\relax
2026       \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
2027     \fi
2028   \else
2029     \@modulo{\@strctr}{10}%
2030     \let\@@fc@numstr#2\relax
2031     \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
2032   \fi
2033 \fi
2034 }

```

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

```

2035 \DeclareRobustCommand{\@numberstringMenglish}[2]{%
2036 \let\@unitstring=\@unitstringenglish
2037 \let\@teenstring=\@teenstringenglish
2038 \let\@tenstring=\@tenstringenglish
2039 \def\@hundred{hundred}\def\@thousand{thousand}%
2040 \def\@andname{and}%
2041 \@numberstringenglish{#1}{#2}%
2042 }

```

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

```
2043 \let\@numberstringFenglish=\@numberstringMenglish
2044 \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.

```
2045 \newcommand*{\@NumberstringMenglish}[2]{%
2046 \let\@unitstring=\@Unitstringenglish
2047 \let\@teenstring=\@Teenstringenglish
2048 \let\@tenstring=\@Tenstringenglish
2049 \def\@hundred{Hundred}\def\@thousand{Thousand}%
2050 \def\@andname{and}%
2051 \@@numberstringenglish{#1}{#2}}
```

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

```
2052 \let\@NumberstringFenglish=\@NumberstringMenglish
2053 \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:

```
2054 \newcommand*{\@@unitthstringenglish}[1]{%
2055 \ifcase#1\relax
2056 zeroth%
2057 \or first%
2058 \or second%
2059 \or third%
2060 \or fourth%
2061 \or fifth%
2062 \or sixth%
2063 \or seventh%
2064 \or eighth%
2065 \or ninth%
2066 \fi
2067 }
```

Next the tens:

```
2068 \newcommand*{\@@tenthstringenglish}[1]{%
2069 \ifcase#1\relax
2070 \or tenth%
2071 \or twentieth%
2072 \or thirtieth%
2073 \or fortieth%
2074 \or fiftieth%
2075 \or sixtieth%
2076 \or seventieth%
2077 \or eightieth%
2078 \or ninetieth%
2079 \fi
2080 }
```

The teens:

```
2081 \newcommand*{\@@teenthstringenglish}[1]{%
2082 \ifcase#1\relax
2083 tenth%
2084 \or eleventh%
2085 \or twelfth%
2086 \or thirteenth%
2087 \or fourteenth%
2088 \or fifteenth%
2089 \or sixteenth%
2090 \or seventeenth%
2091 \or eighteenth%
2092 \or nineteenth%
2093 \fi
2094 }
```

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

```
2095 \newcommand*{\@@Unitthstringenglish}[1]{%
2096 \ifcase#1\relax
2097 Zeroth%
2098 \or First%
2099 \or Second%
2100 \or Third%
2101 \or Fourth%
2102 \or Fifth%
2103 \or Sixth%
2104 \or Seventh%
2105 \or Eighth%
2106 \or Ninth%
2107 \fi
2108 }
```

The tens:

```
2109 \newcommand*{\@@Tenthstringenglish}[1]{%
2110 \ifcase#1\relax
2111 \or Tenth%
2112 \or Twentieth%
2113 \or Thirtieth%
2114 \or Fortieth%
2115 \or Fiftieth%
2116 \or Sixtieth%
2117 \or Seventieth%
2118 \or Eightieth%
2119 \or Ninetieth%
2120 \fi
2121 }
```

The teens:

```
2122 \newcommand*{\@@Teenthstringenglish}[1]{%
2123 \ifcase#1\relax
```

```

2124 Tenth%
2125 \or Eleventh%
2126 \or Twelfth%
2127 \or Thirteenth%
2128 \or Fourteenth%
2129 \or Fifteenth%
2130 \or Sixteenth%
2131 \or Seventeenth%
2132 \or Eighteenth%
2133 \or Nineteenth%
2134 \fi
2135 }

```

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.

```

2136 \newcommand*{\@@ordinalstringenglish}[2]{%
2137 \@strctr=#1\relax
2138 \ifnum#1>99999
2139 \PackageError{fmtcount}{Out of range}%
2140 {This macro only works for values less than 100000 (value given: \number\@strctr)}%
2141 \else
2142 \ifnum#1<0
2143 \PackageError{fmtcount}{Negative numbers not permitted}%
2144 {This macro does not work for negative numbers, however
2145 you can try typing "minus" first, and then pass the modulus of
2146 this number}%
2147 \fi
2148 \def#2{}%
2149 \fi
2150 \@strctr=#1\relax \divide\@strctr by 1000\relax
2151 \ifnum\@strctr>9\relax
    #1 is greater or equal to 10000
2152 \divide\@strctr by 10
2153 \ifnum\@strctr>1\relax
2154 \let\@@fc@ordstr#2\relax
2155 \edef#2{\@@fc@ordstr\@teenstring{\@strctr}}%
2156 \@strctr=#1\relax
2157 \divide\@strctr by 1000\relax
2158 \@modulo{\@strctr}{10}%
2159 \ifnum\@strctr>0\relax
2160 \let\@@fc@ordstr#2\relax
2161 \edef#2{\@@fc@ordstr-\@unitstring{\@strctr}}%
2162 \fi
2163 \else
2164 \@strctr=#1\relax \divide\@strctr by 1000\relax
2165 \@modulo{\@strctr}{10}%
2166 \let\@@fc@ordstr#2\relax
2167 \edef#2{\@@fc@ordstr\@teenstring{\@strctr}}%

```

```

2168 \fi
2169 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
2170 \ifnum\@strctr=0\relax
2171   \let\@@fc@ordstr#2\relax
2172   \edef#2{\@@fc@ordstr\ \@thousandth}%
2173 \else
2174   \let\@@fc@ordstr#2\relax
2175   \edef#2{\@@fc@ordstr\ \@thousand}%
2176 \fi
2177 \else
2178   \ifnum\@strctr>0\relax
2179     \let\@@fc@ordstr#2\relax
2180     \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
2181     \@strctr=#1\relax \@modulo{\@strctr}{1000}%
2182     \let\@@fc@ordstr#2\relax
2183     \ifnum\@strctr=0\relax
2184       \edef#2{\@@fc@ordstr\ \@thousandth}%
2185     \else
2186       \edef#2{\@@fc@ordstr\ \@thousand}%
2187     \fi
2188   \fi
2189 \fi
2190 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
2191 \divide\@strctr by 100
2192 \ifnum\@strctr>0\relax
2193   \ifnum#1>1000\relax
2194     \let\@@fc@ordstr#2\relax
2195     \edef#2{\@@fc@ordstr\ }%
2196   \fi
2197   \let\@@fc@ordstr#2\relax
2198   \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
2199   \@strctr=#1\relax \@modulo{\@strctr}{100}%
2200   \let\@@fc@ordstr#2\relax
2201   \ifnum\@strctr=0\relax
2202     \edef#2{\@@fc@ordstr\ \@hundredth}%
2203   \else
2204     \edef#2{\@@fc@ordstr\ \@hundred}%
2205   \fi
2206 \fi
2207 \@strctr=#1\relax \@modulo{\@strctr}{100}%
2208 \ifnum#1>100\relax
2209   \ifnum\@strctr>0\relax
2210     \let\@@fc@ordstr#2\relax
2211     \edef#2{\@@fc@ordstr\ \@andname\ }%
2212   \fi
2213 \fi
2214 \ifnum\@strctr>19\relax
2215   \@tmpstrctr=\@strctr
2216   \divide\@strctr by 10\relax

```

```

2217 \@modulo{\@tmpstrctr}{10}%
2218 \let\@@fc@ordstr#2\relax
2219 \ifnum\@tmpstrctr=0\relax
2220   \edef#2{\@@fc@ordstr\@tenthstring{\@strctr}}%
2221 \else
2222   \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
2223 \fi
2224 \@strctr=#1\relax \@modulo{\@strctr}{10}%
2225 \ifnum\@strctr>0\relax
2226   \let\@@fc@ordstr#2\relax
2227   \edef#2{\@@fc@ordstr-\@unitthstring{\@strctr}}%
2228 \fi
2229 \else
2230   \ifnum\@strctr<10\relax
2231     \ifnum\@strctr=0\relax
2232       \ifnum#1<100\relax
2233         \let\@@fc@ordstr#2\relax
2234         \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
2235       \fi
2236     \else
2237       \let\@@fc@ordstr#2\relax
2238       \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
2239     \fi
2240   \else
2241     \@modulo{\@strctr}{10}%
2242     \let\@@fc@ordstr#2\relax
2243     \edef#2{\@@fc@ordstr\@teenthstring{\@strctr}}%
2244   \fi
2245 \fi
2246 }

```

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

```

2247 \DeclareRobustCommand{\@ordinalstringMenglish}[2]{%
2248 \let\@unitthstring=\@unitthstringenglish
2249 \let\@teenthstring=\@teenthstringenglish
2250 \let\@tenthstring=\@tenthstringenglish
2251 \let\@unitstring=\@unitstringenglish
2252 \let\@teenstring=\@teenstringenglish
2253 \let\@tenstring=\@tenstringenglish
2254 \def\@andname{and}%
2255 \def\@hundred{hundred}\def\@thousand{thousand}%
2256 \def\@hundredth{hundredth}\def\@thousandth{thousandth}%
2257 \@@ordinalstringenglish{#1}{#2}}

```

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

```

2258 \let\@ordinalstringFenglish=\@ordinalstringMenglish
2259 \let\@ordinalstringNenglish=\@ordinalstringMenglish

```

First letter of each word in upper case:

```

2260 \DeclareRobustCommand{\@OrdinalstringMenglish}[2]{%
2261 \let\@unitthstring=\@Unitthstringenglish
2262 \let\@teenthstring=\@Teenthstringenglish
2263 \let\@tenthstring=\@Tenthstringenglish
2264 \let\@unitstring=\@Unitstringenglish
2265 \let\@teenstring=\@Teenstringenglish
2266 \let\@tenstring=\@Tenstringenglish
2267 \def\@andname{and}%
2268 \def\@hundred{Hundred}\def\@thousand{Thousand}%
2269 \def\@hundredth{Hundredth}\def\@thousandth{Thousandth}%
2270 \@@ordinalstringenglish{#1}{#2}}

```

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

```

2271 \let\@OrdinalstringFenglish=\@OrdinalstringMenglish
2272 \let\@OrdinalstringNenglish=\@OrdinalstringMenglish

```

### 9.3.4 fc-francais.def

```

2273 \ProvidesFCLanguage{francais}[2012/06/18]
2274 \FCLoadlang{french}

```

Set francais to be equivalent to french.

```

2275 \let\@ordinalMfrancais=\@ordinalMfrench
2276 \let\@ordinalFfrancais=\@ordinalFfrench
2277 \let\@ordinalNfrancais=\@ordinalNfrench
2278 \let\@numberstringMfrancais=\@numberstringMfrench
2279 \let\@numberstringFfrancais=\@numberstringFfrench
2280 \let\@numberstringNfrancais=\@numberstringNfrench
2281 \let\@NumberstringMfrancais=\@NumberstringMfrench
2282 \let\@NumberstringFfrancais=\@NumberstringFfrench
2283 \let\@NumberstringNfrancais=\@NumberstringNfrench
2284 \let\@ordinalstringMfrancais=\@ordinalstringMfrench
2285 \let\@ordinalstringFfrancais=\@ordinalstringFfrench
2286 \let\@ordinalstringNfrancais=\@ordinalstringNfrench
2287 \let\@OrdinalstringMfrancais=\@OrdinalstringMfrench
2288 \let\@OrdinalstringFfrancais=\@OrdinalstringFfrench
2289 \let\@OrdinalstringNfrancais=\@OrdinalstringNfrench

```

### 9.3.5 fc-french.def

Definitions for French.

```

2290 \ProvidesFCLanguage{french}[2012/06/18]

```

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

```

2291 \RequirePackage{fcprefix}

```

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'.
2292 \def\fc@french@set@plural#1#2#3#4#5#6{%
2293   \ifthenelse{\equal{#2}{reformed}}{%
2294     \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#3}%
2295   }{%
2296     \ifthenelse{\equal{#2}{traditional}}{%
2297       \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#4}%
2298     }{%
2299       \ifthenelse{\equal{#2}{reformed o}}{%
2300         \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#5}%
2301       }{%
2302         \ifthenelse{\equal{#2}{traditional o}}{%
2303           \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#6}%
2304         }{%
2305           \ifthenelse{\equal{#2}{always}}{%
2306             \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{0}%
2307           }{%
2308             \ifthenelse{\equal{#2}{never}}{%
2309               \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{1}%
2310             }{%
2311               \ifthenelse{\equal{#2}{multiple}}{%
2312                 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{2}%
2313               }{%
2314                 \ifthenelse{\equal{#2}{multiple g-last}}{%
2315                   \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{3}%
2316                 }{%
2317                   \ifthenelse{\equal{#2}{multiple l-last}}{%
2318                     \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{4}%
2319                   }{%
2320                     \ifthenelse{\equal{#2}{multiple lng-last}}{%
2321                       \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{5}%
2322                     }{%
2323                       \ifthenelse{\equal{#2}{multiple ng-last}}{%
2324                         \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{6}%
2325                       }{%
2326                         \PackageError{fmtcount}{Unexpected argument}{%
2327                           '#2' was unexpected: french option '#1 plural' expects 'reformed'
2328                           'reformed o', 'traditional o', 'always', 'never', 'multiple', 'mu
2329                           'multiple l-last', 'multiple lng-last', or 'multiple ng-last'.%
2330                         }

```

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

```

o'.
2331 \def\@tempa#1#2#3{%
2332   \define@key{fcfrench}{#1 plural}[reformed]{%
2333     \fc@french@set@plural{#1}{##1}{#2}{#2}{#3}{#3}%
2334   }%
2335 }
2336 \@tempa{vingt}{4}{5}
2337 \@tempa{cent}{4}{5}
2338 \@tempa{mil}{0}{0}
2339 \@tempa{n-illion}{2}{6}
2340 \@tempa{n-illiard}{2}{6}

```

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

```

2341 \define@key{fcfrench}{all plural}[reformed]{%
2342   \csname KV@fcfrench@vingt plural\endcsname{#1}%
2343   \csname KV@fcfrench@cent plural\endcsname{#1}%
2344   \csname KV@fcfrench@mil plural\endcsname{#1}%
2345   \csname KV@fcfrench@n-illion plural\endcsname{#1}%
2346   \csname KV@fcfrench@n-illiard plural\endcsname{#1}%
2347 }

```

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

traditional	use dash for numbers below 100, except when ‘et’ is used, and space otherwise
reformed	reform of 1990, use dash except with million & milliard, and suchlikes, i.e. $\langle n \rangle$ illion and $\langle n \rangle$ illiard,
always	always use dashes to separate all words

```

2348 \define@key{fcfrench}{dash or space}[reformed]{%
2349   \ifthenelse{\equal{#1}{traditional}}{%
2350     \let\fc@frenchoptions@supermillion@dos\space%
2351     \let\fc@frenchoptions@submillion@dos\space
2352   }{%
2353     \ifthenelse{\equal{#1}{reformed}\or\equal{#1}{1990}}{%
2354       \let\fc@frenchoptions@supermillion@dos\space
2355       \def\fc@frenchoptions@submillion@dos{-}%
2356     }{%
2357       \ifthenelse{\equal{#1}{always}}{%
2358         \def\fc@frenchoptions@supermillion@dos{-}%
2359         \def\fc@frenchoptions@submillion@dos{-}%
2360       }{%
2361         \PackageError{fmtcount}{Unexpected argument}{%
2362           French option ‘dash or space’ expects ‘always’, ‘reformed’ or ‘traditional’
2363         }
2364       }%
2365     }%
2366   }%
2367 }

```

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

```

2368 \define@key{fcfrench}{scale}[recursive]{%
2369   \ifthenelse{\equal{#1}{long}}{%
2370     \let\fc@poweroften\fc@pot@longscalefrench
2371   }{%
2372     \ifthenelse{\equal{#1}{recursive}}{%
2373       \let\fc@poweroften\fc@pot@recursivefrench
2374     }{%
2375       \ifthenelse{\equal{#1}{short}}{%
2376         \let\fc@poweroften\fc@pot@shortscalefrench
2377       }{%
2378         \PackageError{fmtcount}{Unexpected argument}{%
2379           French option ‘scale’ expects ‘long’, ‘recursive’ or ‘short’
2380         }
2381       }%
2382     }%
2383   }%
2384 }
```

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”

```

2385 \define@key{fcfrench}{n-illiard upto}[infinity]{%
2386   \ifthenelse{\equal{#1}{infinity}}{%
2387     \def\fc@longscale@nilliard@upto{0}%
2388   }{%
2389     \ifthenelse{\equal{#1}{infty}}{%
2390       \def\fc@longscale@nilliard@upto{0}%
2391     }{%
2392       \if Q\ifnum9<1#1Q\fi\else
2393         \PackageError{fmtcount}{Unexpected argument}{%
2394           French option ‘milliard threshold’ expects ‘infinity’, or equivalently ‘infty’, or a
2395           integer.}%
2396       \fi
2397       \def\fc@longscale@nilliard@upto{#1}%
2398     }%
2399 }
```

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.

```

2400 \def\@tempa#1{%
```

```

2401 \define@key{fcfrench}{#1}[]{%
2402   \PackageError{fmtcount}{Unexpected argument}{French option with key ‘#1’ does not take
2403     any value}}%
2404 \expandafter\def\csname KV@fcfrench@#1@default\endcsname{%
2405   \def\fmtcount@french{#1}}%
2406}%
2407 \@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.

```

2408 \define@key{fcfrench}{dialect}[france]{%
2409   \ifthenelse{\equal{#1}{france}}
2410     \or\equal{#1}{swiss}
2411     \or\equal{#1}{belgian}}{%
2412   \def\fmtcount@french{#1}}{%
2413   \PackageError{fmtcount}{Invalid value ‘#1’ to french option dialect key}
2414     {Option ‘french’ can only take the values ‘france’,
2415       ‘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’.

```

2416 \define@key{fcfrench}{mil plural mark}[le]{%
2417   \def\fc@frenchoptions@mil@plural@mark{#1}}

```

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

```

2418 \def\fc@UpperCaseFirstLetter#1#2\@nil{%
2419   \uppercase{#1}#2}
2420
2421 \def\fc@CaseIden#1\@nil{%
2422   #1%
2423 }
2424 \def\fc@UpperCaseAll#1\@nil{%
2425   \uppercase{#1}%
2426 }
2427
2428 \let\fc@case\fc@CaseIden
2429

```

\@ ordinalMfrench

```

2430 \newcommand*{\@ordinalMfrench}[2]{%
2431   \iffmtord@abbrv
2432   \edef#2{\number#1\relax\noexpand\fmtord{e}}%
2433   \else
2434     \ifnum#1=1\relax
2435       \edef#2{\number#1\relax\noexpand\fmtord{er}}%
2436     \else
2437       \edef#2{\number#1\relax\noexpand\fmtord{eme}}%
2438     \fi
2439   \fi}

```

\@ ordinalFfrench

```
2440 \newcommand*{\@ordinalFfrench}[2]{%
2441 \iffmtord@abbrv
2442 \edef#2{\number#1\relax\noexpand\fmtord{e}}%
2443 \else
2444 \ifnum#1=1 %
2445 \edef#2{\number#1\relax\noexpand\fmtord{i\'ere}}%
2446 \else
2447 \edef#2{\number#1\relax\noexpand\fmtord{i\'eme}}%
2448 \fi
2449 \fi}
```

In French neutral gender and masculine gender are formally identical.

```
2450 \let\@ordinalNfrench\@ordinalMfrench
```

\@ @unitstringfrench

```
2451 \newcommand*{\@unitstringfrench}[1]{%
2452 \noexpand\fc@case
2453 \ifcase#1 %
2454 z\'ero%
2455 \or un%
2456 \or deux%
2457 \or trois%
2458 \or quatre%
2459 \or cinq%
2460 \or six%
2461 \or sept%
2462 \or huit%
2463 \or neuf%
2464 \fi
2465 \noexpand\@nil
2466 }
```

\@ @tenstringfrench

```
2467 \newcommand*{\@tenstringfrench}[1]{%
2468 \noexpand\fc@case
2469 \ifcase#1 %
2470 \or dix%
2471 \or vingt%
2472 \or trente%
2473 \or quarante%
2474 \or cinquante%
2475 \or soixante%
2476 \or septante%
2477 \or huitante%
2478 \or nonante%
2479 \or cent%
2480 \fi
2481 \noexpand\@nil
2482 }
```

```

\@ @teenstringfrench
2483 \newcommand*{\@teenstringfrench}[1]{%
2484 \noexpand\fc@case
2485 \ifcase#1 %
2486     dix%
2487 \or onze%
2488 \or douze%
2489 \or treize%
2490 \or quatorze%
2491 \or quinze%
2492 \or seize%
2493 \or dix\noexpand\@nil-\noexpand\fc@case sept%
2494 \or dix\noexpand\@nil-\noexpand\fc@case huit%
2495 \or dix\noexpand\@nil-\noexpand\fc@case neuf%
2496 \fi
2497 \noexpand\@nil
2498 }

\@ @seventiesfrench
2499 \newcommand*{\@seventiesfrench}[1]{%
2500 \@tenstring{6}%
2501 \ifnum#1=1 %
2502 \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
2503 \else
2504 -%
2505 \fi
2506 \@teenstring{#1}%
2507 }

\@ @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
          “ $\langle n \rangle$ illion/ $\langle n \rangle$ illiard”.

2508 \newcommand*\@eightiesfrench[1]{%
2509 \fc@case quatre\@nil-\noexpand\fc@case vingt%
2510 \ifnum#1>0 %
2511     \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
2512     s%
2513     \fi
2514     \noexpand\@nil
2515     -\@unitstring{#1}%

```

```

2516 \else
2517   \ifcase\fc@frenchoptions@vingt@plural\space
2518     s% 0: always
2519   \or
2520     % 1: never
2521   \or
2522     s% 2: multiple
2523   \or
2524     % 3: multiple g-last
2525     \ifnum\count0=\count6\ifnum\count9=0 s\fi\fi
2526   \or
2527     % 4: multiple l-last
2528     \ifnum\count9=1 %
2529     \else
2530       s%
2531     \fi
2532   \or
2533     % 5: multiple lng-last
2534     \ifnum\count9=1 %
2535     \else
2536       \ifnum\count0>0 %
2537         s%
2538       \fi
2539     \fi
2540   \or
2541     % or 6: multiple ng-last
2542     \ifnum\count0>0 %
2543       s%
2544     \fi
2545   \fi
2546   \noexpand\@nil
2547 \fi
2548 }
2549 \newcommand*\@@@ninetiesfrench}[1]{%
2550 \fc@case quatre\@nil-\noexpand\fc@case vingt%
2551 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
2552   s%
2553 \fi
2554 \noexpand\@nil
2555 -\@teenstring{#1}%
2556 }
2557 \newcommand*\@@@seventiesfrenchswiss}[1]{%
2558 \@tenstring{7}%
2559 \ifnum#1=1\ \@andname\ \fi
2560 \ifnum#1>1-\fi
2561 \ifnum#1>0 \@unitstring{#1}\fi
2562 }
2563 \newcommand*\@@@eightiesfrenchswiss}[1]{%
2564 \@tenstring{8}%

```

```

2565 \ifnum#1=1\ \@andname\ \fi
2566 \ifnum#1>1-\fi
2567 \ifnum#1>0 \@unitstring{#1}\fi
2568 }
2569 \newcommand*{\@ninetiesfrenchswiss}[1]{%
2570 \@tenstring{9}%
2571 \ifnum#1=1\ \@andname\ \fi
2572 \ifnum#1>1-\fi
2573 \ifnum#1>0 \@unitstring{#1}\fi
2574 }

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

2575 \newcommand*\fc@french@common{%
2576 \let\@unitstring=\@unitstringfrench
2577 \let\@teenstring=\@teenstringfrench
2578 \let\@tenstring=\@tenstringfrench
2579 \def\@hundred{cent}%
2580 \def\@andname{et}%
2581 }

2582 \DeclareRobustCommand{\@numberstringMfrenchswiss}[2]{%
2583 \let\fc@case\fc@CaseIden
2584 \fc@french@common
2585 \let\@seventies=\@seventiesfrenchswiss
2586 \let\@eighties=\@eightiesfrenchswiss
2587 \let\@nineties=\@ninetiesfrenchswiss
2588 \let\fc@nbrstr@preamble\@empty
2589 \let\fc@nbrstr@postamble\@empty
2590 \@@numberstringfrench{#1}{#2}}
2591 \DeclareRobustCommand{\@numberstringMfrenchfrance}[2]{%
2592 \let\fc@case\fc@CaseIden
2593 \fc@french@common
2594 \let\@seventies=\@seventiesfrench
2595 \let\@eighties=\@eightiesfrench
2596 \let\@nineties=\@ninetiesfrench
2597 \let\fc@nbrstr@preamble\@empty
2598 \let\fc@nbrstr@postamble\@empty
2599 \@@numberstringfrench{#1}{#2}}
2600 \DeclareRobustCommand{\@numberstringMfrenchbelgian}[2]{%
2601 \let\fc@case\fc@CaseIden
2602 \fc@french@common
2603 \let\@seventies=\@seventiesfrenchswiss
2604 \let\@eighties=\@eightiesfrench
2605 \let\@nineties=\@ninetiesfrench
2606 \let\fc@nbrstr@preamble\@empty
2607 \let\fc@nbrstr@postamble\@empty
2608 \@@numberstringfrench{#1}{#2}}
2609 \let\@numberstringMfrench=\@numberstringMfrenchfrance
2610 \DeclareRobustCommand{\@numberstringFfrenchswiss}[2]{%

```

```

2611 \let\fc@case\fc@CaseIden
2612 \fc@french@common
2613 \let\@seventies=\@seventiesfrenchswiss
2614 \let\@eighties=\@eightiesfrenchswiss
2615 \let\@nineties=\@ninetiesfrenchswiss
2616 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2617 \let\fc@nbrstr@postamble\@empty
2618 \@@numberstringfrench{#1}{#2}}
2619 \DeclareRobustCommand{\@numberstringFfrenchfrance}[2]{%
2620 \let\fc@case\fc@CaseIden
2621 \fc@french@common
2622 \let\@seventies=\@seventiesfrench
2623 \let\@eighties=\@eightiesfrench
2624 \let\@nineties=\@ninetiesfrench
2625 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2626 \let\fc@nbrstr@postamble\@empty
2627 \@@numberstringfrench{#1}{#2}}
2628 \DeclareRobustCommand{\@numberstringFfrenchbelgian}[2]{%
2629 \let\fc@case\fc@CaseIden
2630 \fc@french@common
2631 \let\@seventies=\@seventiesfrenchswiss
2632 \let\@eighties=\@eightiesfrench
2633 \let\@nineties=\@ninetiesfrench
2634 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2635 \let\fc@nbrstr@postamble\@empty
2636 \@@numberstringfrench{#1}{#2}}
2637 \let\@numberstringFfrench=\@numberstringFfrenchfrance
2638 \let\@ordinalstringNfrench\@ordinalstringMfrench
2639 \DeclareRobustCommand{\@NumberstringMfrenchswiss}[2]{%
2640 \let\fc@case\fc@UpperCaseFirstLetter
2641 \fc@french@common
2642 \let\@seventies=\@seventiesfrenchswiss
2643 \let\@eighties=\@eightiesfrenchswiss
2644 \let\@nineties=\@ninetiesfrenchswiss
2645 \let\fc@nbrstr@preamble\@empty
2646 \let\fc@nbrstr@postamble\@empty
2647 \@@numberstringfrench{#1}{#2}}
2648 \DeclareRobustCommand{\@NumberstringMfrenchfrance}[2]{%
2649 \let\fc@case\fc@UpperCaseFirstLetter
2650 \fc@french@common
2651 \let\@seventies=\@seventiesfrench
2652 \let\@eighties=\@eightiesfrench
2653 \let\@nineties=\@ninetiesfrench
2654 \let\fc@nbrstr@preamble\@empty
2655 \let\fc@nbrstr@postamble\@empty
2656 \@@numberstringfrench{#1}{#2}}
2657 \DeclareRobustCommand{\@NumberstringMfrenchbelgian}[2]{%
2658 \let\fc@case\fc@UpperCaseFirstLetter
2659 \fc@french@common

```

```

2660 \let\@seventies=\@seventiesfrenchswiss
2661 \let\@eighties=\@eightiesfrench
2662 \let\@nineties=\@ninetiesfrench
2663 \let\fc@nbrstr@preamble\@empty
2664 \let\fc@nbrstr@postamble\@empty
2665 \@@numberstringfrench{#1}{#2}}
2666 \let\@NumberstringMfrench=\@NumberstringMfrenchfrance
2667 \DeclareRobustCommand{\@NumberstringFfrenchswiss}[2]{%
2668 \let\fc@case\fc@UpperCaseFirstLetter
2669 \fc@french@common
2670 \let\@seventies=\@seventiesfrenchswiss
2671 \let\@eighties=\@eightiesfrenchswiss
2672 \let\@nineties=\@ninetiesfrenchswiss
2673 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2674 \let\fc@nbrstr@postamble\@empty
2675 \@@numberstringfrench{#1}{#2}}
2676 \DeclareRobustCommand{\@NumberstringFfrenchfrance}[2]{%
2677 \let\fc@case\fc@UpperCaseFirstLetter
2678 \fc@french@common
2679 \let\@seventies=\@seventiesfrench
2680 \let\@eighties=\@eightiesfrench
2681 \let\@nineties=\@ninetiesfrench
2682 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2683 \let\fc@nbrstr@postamble\@empty
2684 \@@numberstringfrench{#1}{#2}}
2685 \DeclareRobustCommand{\@NumberstringFfrenchbelgian}[2]{%
2686 \let\fc@case\fc@UpperCaseFirstLetter
2687 \fc@french@common
2688 \let\@seventies=\@seventiesfrenchswiss
2689 \let\@eighties=\@eightiesfrench
2690 \let\@nineties=\@ninetiesfrench
2691 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2692 \let\fc@nbrstr@postamble\@empty
2693 \@@numberstringfrench{#1}{#2}}
2694 \let\@NumberstringFfrench=\@NumberstringFfrenchfrance
2695 \let\@NumberstringNfrench\@NumberstringMfrench
2696 \DeclareRobustCommand{\@ordinalstringMfrenchswiss}[2]{%
2697 \let\fc@case\fc@CaseIden
2698 \let\fc@first=\fc@@firstfrench
2699 \fc@french@common
2700 \let\@seventies=\@seventiesfrenchswiss
2701 \let\@eighties=\@eightiesfrenchswiss
2702 \let\@nineties=\@ninetiesfrenchswiss
2703 \@@ordinalstringfrench{#1}{#2}%
2704 }
2705 \newcommand*\fc@@firstfrench{premier}
2706 \newcommand*\fc@@firstFfrench{premi\`ere}
2707 \DeclareRobustCommand{\@ordinalstringMfrenchfrance}[2]{%
2708 \let\fc@case\fc@CaseIden

```

```

2709 \let\fc@first=\fc@@firstfrench
2710 \fc@french@common
2711 \let\@seventies=\@@seventiesfrench
2712 \let\@eighties=\@@eightiesfrench
2713 \let\@nineties=\@@ninetiesfrench
2714 \@@ordinalstringfrench{#1}{#2}}
2715 \DeclareRobustCommand{\@ordinalstringMfrenchbelgian}[2]{%
2716 \let\fc@case\fc@CaseIden
2717 \let\fc@first=\fc@@firstfrench
2718 \fc@french@common
2719 \let\@seventies=\@@seventiesfrench
2720 \let\@eighties=\@@eightiesfrench
2721 \let\@nineties=\@@ninetiesfrench
2722 \@@ordinalstringfrench{#1}{#2}%
2723 }
2724 \let\@ordinalstringMfrench=\@ordinalstringMfrenchfrance
2725 \DeclareRobustCommand{\@ordinalstringFfrenchswiss}[2]{%
2726 \let\fc@case\fc@CaseIden
2727 \let\fc@first=\fc@@firstFfrench
2728 \fc@french@common
2729 \let\@seventies=\@@seventiesfrenchswiss
2730 \let\@eighties=\@@eightiesfrenchswiss
2731 \let\@nineties=\@@ninetiesfrenchswiss
2732 \@@ordinalstringfrench{#1}{#2}%
2733 }
2734 \DeclareRobustCommand{\@ordinalstringFfrenchfrance}[2]{%
2735 \let\fc@case\fc@CaseIden
2736 \let\fc@first=\fc@@firstFfrench
2737 \fc@french@common
2738 \let\@seventies=\@@seventiesfrench
2739 \let\@eighties=\@@eightiesfrench
2740 \let\@nineties=\@@ninetiesfrench
2741 \@@ordinalstringfrench{#1}{#2}%
2742 }
2743 \DeclareRobustCommand{\@ordinalstringFfrenchbelgian}[2]{%
2744 \let\fc@case\fc@CaseIden
2745 \let\fc@first=\fc@@firstFfrench
2746 \fc@french@common
2747 \let\@seventies=\@@seventiesfrench
2748 \let\@eighties=\@@eightiesfrench
2749 \let\@nineties=\@@ninetiesfrench
2750 \@@ordinalstringfrench{#1}{#2}%
2751 }
2752 \let\@ordinalstringFfrench=\@ordinalstringFfrenchfrance
2753 \let\@ordinalstringNfrench\@ordinalstringMfrench
2754 \DeclareRobustCommand{\@OrdinalstringMfrenchswiss}[2]{%
2755 \let\fc@case\fc@UpperCaseFirstLetter
2756 \let\fc@first=\fc@@firstfrench
2757 \fc@french@common

```

```

2758 \let\@seventies=\@seventiesfrenchswiss
2759 \let\@eighties=\@eightiesfrenchswiss
2760 \let\@nineties=\@ninetiesfrenchswiss
2761 \@@ordinalstringfrench{#1}{#2}%
2762 }
2763 \DeclareRobustCommand{\@OrdinalstringMfrenchfrance}[2]{%
2764 \let\fc@case\fc@UpperCaseFirstLetter
2765 \let\fc@first=\fc@@firstfrench
2766 \fc@french@common
2767 \let\@seventies=\@seventiesfrench
2768 \let\@eighties=\@eightiesfrench
2769 \let\@nineties=\@ninetiesfrench
2770 \@@ordinalstringfrench{#1}{#2}%
2771 }
2772 \DeclareRobustCommand{\@OrdinalstringMfrenchbelgian}[2]{%
2773 \let\fc@case\fc@UpperCaseFirstLetter
2774 \let\fc@first=\fc@@firstfrench
2775 \fc@french@common
2776 \let\@seventies=\@seventiesfrench
2777 \let\@eighties=\@eightiesfrench
2778 \let\@nineties=\@ninetiesfrench
2779 \@@ordinalstringfrench{#1}{#2}%
2780 }
2781 \let\@OrdinalstringMfrench=\@OrdinalstringMfrenchfrance
2782 \DeclareRobustCommand{\@OrdinalstringFfrenchswiss}[2]{%
2783 \let\fc@case\fc@UpperCaseFirstLetter
2784 \let\fc@first=\fc@@firstfrench
2785 \fc@french@common
2786 \let\@seventies=\@seventiesfrenchswiss
2787 \let\@eighties=\@eightiesfrenchswiss
2788 \let\@nineties=\@ninetiesfrenchswiss
2789 \@@ordinalstringfrench{#1}{#2}%
2790 }
2791 \DeclareRobustCommand{\@OrdinalstringFfrenchfrance}[2]{%
2792 \let\fc@case\fc@UpperCaseFirstLetter
2793 \let\fc@first=\fc@@firstFfrench
2794 \fc@french@common
2795 \let\@seventies=\@seventiesfrench
2796 \let\@eighties=\@eightiesfrench
2797 \let\@nineties=\@ninetiesfrench
2798 \@@ordinalstringfrench{#1}{#2}%
2799 }
2800 \DeclareRobustCommand{\@OrdinalstringFfrenchbelgian}[2]{%
2801 \let\fc@case\fc@UpperCaseFirstLetter
2802 \let\fc@first=\fc@@firstFfrench
2803 \fc@french@common
2804 \let\@seventies=\@seventiesfrench
2805 \let\@eighties=\@eightiesfrench
2806 \let\@nineties=\@ninetiesfrench

```

2807 \@@ordinalstringfrench{#1}{#2}%

2808 }

2809 \let\@OrdinalstringFfrench=\@OrdinalstringFfrenchfrance

2810 \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.

2811 \ifundefined{fc@@do@plural@mark}{\PackageError{fmtcount}{Duplicate definition}{Redefinit.

2812 'fc@@do@plural@mark'}}}

Arguments as follows:

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

Implicit arguments as follows:

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

$\text{\count1}$  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,

$\text{\count6}$  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,

$\text{\count10}$  input, counter giving the plural mark control option.

2813 \def\fc@@do@plural@mark#1{%

2814 \ifcase\count10 %

2815 #1% 0=always

2816 \or% 1=never

2817 \or% 2=multiple

2818 \ifnum\count1>1 %

2819 #1%

2820 \fi

2821 \or% 3= multiple g-last

2822 \ifnum\count1>1 %

2823 \ifnum\count0=\count6 %

2824 #1%

2825 \fi

2826 \fi

2827 \or% 4= multiple l-last

2828 \ifnum\count1>1 %

2829 \ifnum\count9=1 %

2830 \else

2831 #1%

2832 \fi

2833 \fi

2834 \or% 5= multiple lng-last

2835 \ifnum\count1>1 %

```

2836         \ifnum\count9=1 %
2837         \else
2838             \if\count0>\count6 %
2839                 #1%
2840             \fi
2841         \fi
2842     \fi
2843 \or% 6= multiple ng-last
2844     \ifnum\count1>1 %
2845         \ifnum\count0>\count6 %
2846             #1%
2847         \fi
2848     \fi
2849 \fi
2850 }

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

2851 \ifundefined{fc@@nbrstr@Fpreamble}{-}{%
2852     \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2853         'fc@@nbrstr@Fpreamble'}}}

\fc  @@nbrstr@Fpreamble
2854 \def\fc@@nbrstr@Fpreamble{%
2855     \fc@read@unit{\count1}{0}%
2856     \ifnum\count1=1 %
2857         \let\fc@case@save\fc@case
2858         \def\fc@case{\noexpand\fc@case}%
2859         \def\@nil{\noexpand\@nil}%
2860         \let\fc@nbrstr@postamble\fc@@nbrstr@Fpostamble
2861     \fi
2862 }

\fc  @@nbrstr@Fpostamble
2863 \def\fc@@nbrstr@Fpostamble{%
2864     \let\fc@case\fc@case@save
2865     \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
2866     \def\@tempd{un}%
2867     \ifx\@tempc\@tempd
2868         \let\@tempc\@tempa
2869         \edef\@tempa{\@tempb\fc@case une\@nil}%
2870     \fi
2871 }

\fc  @@pot@longscalefrench Macro \fc@@pot@longscalefrench is used to pro-
duce 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.

2872 \ifundefined{fc@@pot@longscalefrench}{-}{%
2873     \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2874         '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

```
2875 \def\fc@pot@longscalefrench#1#2#3{%
```

```
2876   {%
```

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

```
2877   \edef\@tempb{\number#1}%
```

Let `\count1` be the plural value.

```
2878   \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  $r$ .

```
2879   \count2\count0 %
```

```
2880   \divide\count2 by 6 %
```

```
2881   \count3\count2 %
```

```
2882   \multiply\count3 by 6 %
```

```
2883   \count3-\count3 %
```

```
2884   \advance\count3 by \count0 %
```

```
2885   \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)”.

```
2886   \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.

```
2887   \edef\@temp{%
```

```
2888   \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.

```
2889   1%
```

```
2890   \else
```

```
2891   \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`.

2892                   \ifnum\fc@longscale@nilliard@upto=0 %

Here option ‘n-illiard upto’ is ‘infinity’, so we always use “⟨n⟩illiard(s)”.

2893                   2%

2894                   \else

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

2895                   \ifnum\count2>\fc@longscale@nilliard@upto

2896                   1%

2897                   \else

2898                   2%

2899                   \fi

2900                   \fi

2901                   \else

2902                   2%

2903                   \fi

2904                   \fi

2905                   }%

2906                   \ifnum\@temph=1 %

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

2907                   \count10=\fc@frenchoptions@mil@plural\space

2908                   \edef\@tempe{%

2909                   \noexpand\fc@case

2910                   mil%

2911                   \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark

2912                   \noexpand\@nil

2913                   }%

2914                   \else

2915                   % weight >= 6

2916                   \expandafter\fc@@latin@cardinal@pexif\expandafter{\the\count2}\@tempg

2917                   % now form the xxx-illion(s) or xxx-illiard(s) word

2918                   \ifnum\count3>2 %

2919                   \toks10{illiard}%

2920                   \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space

2921                   \else

2922                   \toks10{illion}%

2923                   \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space

2924                   \fi

2925                   \edef\@tempe{%

2926                   \noexpand\fc@case

2927                   \@tempg

2928                   \the\toks10 %

2929                   \fc@@do@plural@mark s%

2930                   \noexpand\@nil

2931                   }%

2932                   \fi

2933                   \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.

```
2934      \let\@tempe\@empty
2935      \def\@temph{0}%
2936      \fi
2937      \else
```

Case of  $w = 0$ .

```
2938      \let\@tempe\@empty
2939      \def\@temph{0}%
2940      \fi
```

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

```
2941      \expandafter\toks\expandafter1\expandafter{\@tempe}%
2942      \toks0{#2}%
2943      \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
2944      \expandafter
2945      }\@tempa
2946 }
```

\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.

```
2947 \ifundefined{fc@@pot@shortscalefrench}{\%
2948   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2949     '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

```
2950 \def\fc@@pot@shortscalefrench#1#2#3{%
2951   {%
```

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

```
2952   \edef\@tempb{\number#1}%
```

And let \count1 be the plural value.

```
2953   \count1=\@tempb
```

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

```
2954 \count2\count0 %
2955 \divide\count2 by 3 %
2956 \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)”.

```
2957 \ifnum\count0>0 % If weight>=3, i.e we do have to append thousand or n-illion(s)/n-illiard
2958 \ifnum\count1>0 % we have at least one thousand/n-illion/n-illiard
2959 \ifnum\count2=0 %
2960 \def\@temph{1}%
2961 \count1=\fc@frenchoptions@mil@plural\space
2962 \edef\@tempe{%
2963 mil%
2964 \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
2965 }%
2966 \else
2967 \def\@temph{2}%
2968 % weight >= 6
2969 \expandafter\fc@@latin@cardinal@prefix\expandafter{\the\count2}\@tempg
2970 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2971 \edef\@tempe{%
2972 \noexpand\fc@case
2973 \@tempg
2974 illion%
2975 \fc@@do@plural@mark s%
2976 \noexpand\@nil
2977 }%
2978 \fi
2979 \else
```

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

```
2980 \def\@temph{0}%
2981 \let\@tempe\@empty
2982 \fi
2983 \else
```

Here  $w = 0$ .

```
2984 \def\@temph{0}%
2985 \let\@tempe\@empty
2986 \fi
2987 % now place into \@tempa the assignment of results \cs{@temph} and \cs{@tempe} to to \
2988 % \texttt{#3} for further propagation after closing brace.
2989 % \begin{macrocode}
2990 \expandafter\toks\expandafter1\expandafter{\@tempe}%
2991 \toks0{#2}%
2992 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
```

```

2993   \expandafter
2994 } \@tempa
2995 }

\fc  @@pot@recursivefrench Macro \fc@@pot@recursivefrench is used to pro-
      duce power of tens that are of the form “million de milliards de milliards” for
      1024. First we check that the macro is not yet defined.
2996 \ifundefined{fc@@pot@recursivefrench}{%
2997   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2998     ‘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
2999 \def\fc@@pot@recursivefrench#1#2#3{%
3000   {%

      First the input arguments are saved into local objects: #1 and #1 are respec-
      tively saved into \@tempa and \@tempb.
3001   \edef\@tempb{\number#1}%
3002   \let\@tempa\@tempa

      New get the inputs #1 and #1 into counters \count0 and \count1 as this is
      more practical.
3003   \count1=\@tempb\space

      Now compute into \count2 how many times “de milliards” has to be repeated.
3004   \ifnum\count1>0 %
3005     \count2\count0 %
3006     \divide\count2 by 9 %
3007     \advance\count2 by -1 %
3008     \let\@tempe\@empty
3009     \edef\@tempf{\fc@frenchoptions@supermillion@dos
3010       de\fc@frenchoptions@supermillion@dos\fc@case milliards\@nil}%
3011     \count11\count0 %
3012     \ifnum\count2>0 %
3013       \count3\count2 %
3014       \count3-\count3 %
3015       \multiply\count3 by 9 %
3016       \advance\count11 by \count3 %
3017     \loop
3018       % (\count2, \count3) <- (\count2 div 2, \count2 mod 2)
3019       \count3\count2 %

```

```

3020         \divide\count3 by 2 %
3021         \multiply\count3 by 2 %
3022         \count3-\count3 %
3023         \advance\count3 by \count2 %
3024         \divide\count2 by 2 %
3025         \ifnum\count3=1 %
3026             \let\@tempg\@tempe
3027             \edef\@tempe{\@tempg\@tempf}%
3028         \fi
3029         \let\@tempg\@tempf
3030         \edef\@tempf{\@tempg\@tempf}%
3031         \ifnum\count2>0 %
3032             \repeat
3033         \fi
3034         \divide\count11 by 3 %
3035         \ifcase\count11 % 0 .. 5
3036             % 0 => d milliard(s) (de milliards)*
3037             \def\@temph{2}%
3038             \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
3039         \or % 1 => d mille milliard(s) (de milliards)*
3040             \def\@temph{1}%
3041             \count10=\fc@frenchoptions@mil@plural\space
3042         \or % 2 => d million(s) (de milliards)*
3043             \def\@temph{2}%
3044             \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
3045         \or % 3 => d milliard(s) (de milliards)*
3046             \def\@temph{2}%
3047             \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
3048         \or % 4 => d mille milliards (de milliards)*
3049             \def\@temph{1}%
3050             \count10=\fc@frenchoptions@mil@plural\space
3051         \else % 5 => d million(s) (de milliards)*
3052             \def\@temph{2}%
3053             \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
3054         \fi
3055         \let\@tempg\@tempe
3056         \edef\@tempf{%
3057             \ifcase\count11 % 0 .. 5
3058             \or
3059                 mil\fc@@do@plural@mark \fc@frenchoptions@mil@plural@mark
3060             \or
3061                 million\fc@@do@plural@mark s%
3062             \or
3063                 milliard\fc@@do@plural@mark s%
3064             \or
3065                 mil\fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
3066                 \noexpand\@nil\fc@frenchoptions@supermillion@dos
3067                 \noexpand\fc@case milliards% 4
3068             \or

```

```

3069      million\fc@@do@plural@mark s%
3070      \noexpand\@nil\fc@frenchoptions@supermillion@dos
3071      de\fc@frenchoptions@supermillion@dos\noexpand\fc@case  milliards% 5
3072  \fi
3073 }%
3074  \edef\@tempe{%
3075    \ifx\@tempf\@empty\else
3076    \expandafter\fc@case\@tempf\@nil
3077    \fi
3078    \@tempg
3079  }%
3080  \else
3081    \def\@temph{0}%
3082    \let\@tempe\@empty
3083  \fi

```

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

```

3084  \expandafter\toks\expandafter1\expandafter{\@tempe}%
3085  \toks0{#2}%
3086  \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
3087  \expandafter
3088  }\@tempa
3089 }

```

\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.

```

3090 \ifundefined{fc@muladdfrench}{\%
3091   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3092   '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 “< $n$ >illion(s)|< $n$ >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 “< $n$ >illion(s)|< $n$ >illiard(s)” — for index = 2

```

3093 \def\fc@muladdfrench#1#2#3{%
3094   {%

```

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

```

3095   \edef\@tempc{#1}%
3096   \edef\@tempd{#2}%
3097   \edef\@tempf{#3}%
3098   \let\@tempc\@tempc
3099   \let\@tempd\@tempd

```

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

```

3100   \ifnum\@tempc=1 %
3101     \ifnum\count9=1 %
3102       \let\@tempd\@empty
3103     \fi
3104   \fi

```

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

```

3105   \edef\@tempg{%
3106     \@tempd
3107     \ifx\@tempd\@empty\else
3108       \ifx\@tempf\@empty\else
3109         \ifcase\count9 %
3110           \or
3111             \fc@frenchoptions@submillion@dos
3112           \or
3113             \fc@frenchoptions@supermillion@dos
3114           \fi
3115         \fi
3116       \fi
3117     \@tempf
3118   }%

```

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

```

3119   \edef\@temph{%
3120     \@tempa
3121     \ifx\@tempa\@empty\else
3122       \ifx\@tempg\@empty\else
3123         \ifcase\count8 %
3124           \or
3125             \fc@frenchoptions@submillion@dos
3126           \or
3127             \fc@frenchoptions@supermillion@dos
3128           \fi

```

```

3129      \fi
3130      \fi
3131      \@tempg
3132  }%

```

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

```

3133      \def\@tempb##1{\def\@tempa{\def\@tempa{##1}}}%
3134      \expandafter\@tempb\expandafter{\@temph}%
3135      \expandafter
3136  }\@tempa
3137}%

```

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

```

3138 \ifundefined{fc@lthundredstringfrench}{}{%
3139   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3140     '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@<math>w</math> macro after this number has been parsed. So the only thing that \fc@lthundredstringfrench needs is to know <math>w</math> 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.

```

3141 \def\fc@lthundredstringfrench#1{%
3142   {%

```

First save arguments into local temporary macro.

```

3143     \let\@tempc#1%

```

Read units  $d_w$  to \count1.

```

3144     \fc@read@unit{\count1}{\count0}%

```

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

```

3145     \count3\count0 %
3146     \advance\count3 1 %
3147     \fc@read@unit{\count2}{\count3}%

```

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

```

3148     \edef\@tempa{%
3149       \@tempc
3150       \ifnum\count2>1 %
3151         % 20 .. 99
3152       \ifnum\count2>6 %
3153         % 70 .. 99
3154       \ifnum\count2<8 %
3155         % 70 .. 79

```

```

3156         \@seventies{\count1}%
3157     \else
3158         % 80..99
3159         \ifnum\count2<9 %
3160             % 80 .. 89
3161             \@eighties{\count1}%
3162         \else
3163             % 90 .. 99
3164             \@nineties{\count1}%
3165         \fi
3166     \fi
3167 \else
3168     % 20..69
3169     \@tenstring{\count2}%
3170     \ifnum\count1>0 %
3171         % x1 .. x0
3172         \ifnum\count1=1 %
3173             % x1
3174             \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
3175         \else
3176             % x2 .. x9
3177             -%
3178         \fi
3179         \@unitstring{\count1}%
3180     \fi
3181 \fi
3182 \else
3183     % 0 .. 19
3184     \ifnum\count2=0 % when tens = 0
3185         % 0 .. 9
3186         \ifnum\count1=0 % when units = 0
3187             % \count3=1 when #1 = 0, i.e. only for the unit of the top level number
3188             \ifnum\count3=1 %
3189                 \ifnum\fc@max@weight=0 %
3190                     \@unitstring{0}%
3191                 \fi
3192             \fi
3193         \else
3194             % 1 .. 9
3195             \@unitstring{\count1}%
3196         \fi
3197     \else
3198         % 10 .. 19
3199         \@teenstring{\count1}%
3200     \fi
3201 \fi
3202 }%

```

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

```

3203     \def\@tempb##1{\def\@tempa{\def#1{##1}}}%

```

```

3204     \expandafter\@tempb\expandafter{\@tempa}%
3205     \expandafter
3206   }\@tempa
3207 }

\fc @ltthousandstringfrench Macro \fc@ltthousandstringfrench is used to for-
mat a number in interval [0..999]. First we check that it is not already defined.
3208 \ifundefined{fc@ltthousandstringfrench}{\}%
3209   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3210     '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)",  $2 \Rightarrow$ 
<n>illion(s)|<n>illiard(s)
3211 \def\fc@ltthousandstringfrench#1{%
3212   {%
Set counter \count2 to digit  $d_{w+2}$ , i.e. hundreds.
3213   \count4\count0 %
3214   \advance\count4 by 2 %
3215   \fc@read@unit{\count2 }\count4 }%

Check that the two subsequent digits  $d_{w+1}d_w$  are non zero, place check-result
into \@tempa.
3216   \advance\count4 by -1 %
3217   \count3\count4 %
3218   \advance\count3 by -1 %
3219   \fc@check@nonzeros{\count3 }\count4 }\@tempa

Compute plural mark of 'cent' into \@temps.
3220   \edef\@temps{%
3221     \ifcase\fc@frenchoptions@cent@plural\space
3222     % 0 => always
3223     s%
3224     \or
3225     % 1 => never
3226     \or
3227     % 2 => multiple
3228     \ifnum\count2>1s\fi
3229     \or
3230     % 3 => multiple g-last
3231     \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count0=\count6s\fi\fi\fi
3232     \or
3233     % 4 => multiple l-last
3234     \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count9=0s\else\ifnum\count9=2s\fi\fi\fi\fi
3235     \fi

```

```

3236 }%
3237 % compute spacing after cent(s?) into \@tempb
3238 \expandafter\let\expandafter\@tempb
3239 \ifnum\@tempa>0 \fc@frenchoptions@submillion@dos\else\@empty\fi
3240 % now place into \@tempa the hundreds
3241 \edef\@tempa{%
3242 \ifnum\count2=0 %
3243 \else
3244 \ifnum\count2=1 %
3245 \expandafter\fc@case\@hundred\@nil
3246 \else
3247 \@unitstring{\count2}\fc@frenchoptions@submillion@dos
3248 \noexpand\fc@case\@hundred\@temps\noexpand\@nil
3249 \fi
3250 \@tempb
3251 \fi
3252 }%
3253 % now append to \@tempa the ten and unit
3254 \fc@lthundredstringfrench\@tempa

```

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

```

3255 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
3256 \expandafter\@tempb\expandafter{\@tempa}%
3257 \expandafter
3258 }\@tempa
3259 }

```

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

```

3260 \@ifundefined{@@numberstringfrench}{-}{%
3261 \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

```

3262 \def\@@numberstringfrench#1#2{%
3263 {%

```

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

```

3264 \edef\@tempa{#1}%
3265 \expandafter\fc@number@parser\expandafter{\@tempa}%
3266 \ifnum\fc@min@weight<0 %
3267 \PackageError{fmtcount}{Out of range}%
3268 {This macro does not work with fractional numbers}%
3269 \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'.

```

3270 \edef\@tempa{\ifcase\fc@sign@case\space\or\fc@case plus\@nil\or\fc@case moins\@nil\fi}%
3271 \fc@nbrstr@preamble
3272 \fc@@nbrstrfrench@inner
3273 \fc@nbrstr@postamble

```

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

```

3274 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
3275 \expandafter\@tempb\expandafter{\@tempa}%
3276 \expandafter
3277 }\@tempa
3278 }

```

\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.

```

3279 \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.

```

3280 \count0=\fc@max@weight
3281 \divide\count0 by 3 %
3282 \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.

```

3283 \fc@intpart@find@last{\count5 }%
3284 \count6\count5 %
3285 \divide\count6 3 %
3286 \multiply\count6 3 %
3287 \count8=0 %
3288 \loop

```

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

```

3289 \count1\count0 %
3290 \advance\count1 by 2 %
3291 \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.

```

3292 \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

```

3293 \fc@ltthousandstringfrench\@tempd

```

Finally do the multiplication-addition. Implicit arguments are \@tempa for input/output growing formatted number, \count8 for input previous power

type, i.e. power type of  $10^{w+3}$ , \count9 for input current power type, i.e. power type of  $10^w$ .

```
3294      \fc@muladdfrench\@tempt\@tempd\@tempb
```

Then iterate.

```
3295      \count8\count9 %
3296      \advance\count0 by -3 %
3297      \ifnum\count6>\count0 \else
3298      \repeat
3299 }
```

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

```
3300 \ifundefined{@@ordinalstringfrench}{\{%
3301   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3302   '@@ordinalstringfrench'}}}
```

Arguments are as follows:

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

```
3303 \def\@@ordinalstringfrench#1#2{%
3304   {%
```

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

```
3305   \edef\@tempa{#1}%
3306   \expandafter\fc@number@parser\expandafter{\@tempa}%
3307   \ifnum\fc@min@weight<0 %
3308     \PackageError{fmtcount}{Out of range}%
3309     {This macro does not work with fractional numbers}%
3310   \fi
3311   \ifnum\fc@sign@case>0 %
3312     \PackageError{fmtcount}{Out of range}%
3313     {This macro does with negative or explicitly marked as positive numbers}%
3314   \fi
```

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

```
3315   \ifnum\fc@max@weight=0 %
3316     \ifnum\csname fc@digit0\endcsname=1 %
3317       \count0=1 %
3318     \else
3319       \count0=0 %
3320     \fi
3321   \else
3322     \count0=0 %
3323   \fi
3324   \ifnum\count0=1 %
3325     \edef\@tempa{\expandafter\fc@case\fc@first\@nil}%
3326   \else
```

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

```

3327 \def\@tempa##1{%
3328 \expandafter\edef\csname fc@frenchoptions@##1@plural\endcsname{%
3329 \ifcase\csname fc@frenchoptions@##1@plural\endcsname\space
3330 0% 0: always => always
3331 \or
3332 1% 1: never => never
3333 \or
3334 6% 2: multiple => multiple ng-last
3335 \or
3336 1% 3: multiple g-last => never
3337 \or
3338 5% 4: multiple l-last => multiple lng-last
3339 \or
3340 5% 5: multiple lng-last => multiple lng-last
3341 \or
3342 6% 6: multiple ng-last => multiple ng-last
3343 \fi
3344 }%
3345 }%
3346 \@tempa{vingt}%
3347 \@tempa{cent}%
3348 \@tempa{mil}%
3349 \@tempa{n-illion}%
3350 \@tempa{n-illiard}%

```

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

```

3351 \let\fc@case@save\fc@case
3352 \def\fc@case{\noexpand\fc@case}%
3353 \def\@nil{\noexpand\@nil}%

```

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

```

3354 \let\@tempa\@empty
3355 \fc@@nbrstrfrench@inner

```

Now restore \fc@case

```

3356 \let\fc@case\fc@case@save

```

Now we add the “ième” ending

```

3357 \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
3358 \expandafter\fc@get@last@letter\expandafter{\@tempc}\@tempd\@tempe
3359 \def\@tempf{e}%
3360 \ifx\@tempe\@tempf
3361 \edef\@tempa{\@tempb\expandafter\fc@case\@tempd i\'eme\@nil}%
3362 \else
3363 \def\@tempf{q}%
3364 \ifx\@tempe\@tempf
3365 \edef\@tempa{\@tempb\expandafter\fc@case\@tempd qui\'eme\@nil}%
3366 \else
3367 \def\@tempf{f}%

```

```

3368         \ifx\@tempe\@tempf
3369         \edef\@tempa{\@tempb\expandafter\fc@case\@tempd vi\‘eme\@nil}%
3370     \else
3371         \edef\@tempa{\@tempb\expandafter\fc@case\@tempc i\‘eme\@nil}%
3372     \fi
3373 \fi
3374 \fi
3375 \fi

```

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

```

3376     \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
3377     \expandafter\@tempb\expandafter{\@tempa}%
3378     \expandafter
3379 } \@tempa
3380 }

```

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

```

3381 \newcommand*\fc@frenchoptions@setdefaults{%
3382   \csname KV@fcfrench@all plural\endcsname{reformed}%
3383   \def\fc@frenchoptions@submillion@dos{-}%
3384   \let\fc@frenchoptions@supermillion@dos\space
3385   \let\fc@u@in@duo\@empty% Could be ‘u’
3386   % \let\fc@poweroften\fc@pot@longscalefrench
3387   \let\fc@poweroften\fc@pot@recursivefrench
3388   \def\fc@longscale@illiard@upto{0}% infinity
3389   \def\fc@frenchoptions@mil@plural@mark{le}%
3390 }
3391 \fc@frenchoptions@setdefaults

```

### 9.3.6 fc-frenchb.def

```

3392 \ProvidesFCLanguage{frenchb}[2012/06/18]
3393 \FCloadlang{french}

```

Set frenchb to be equivalent to french.

```

3394 \let\@ordinalMfrenchb=\@ordinalMfrench
3395 \let\@ordinalFfrenchb=\@ordinalFfrench
3396 \let\@ordinalNfrenchb=\@ordinalNfrench
3397 \let\@numberstringMfrenchb=\@numberstringMfrench
3398 \let\@numberstringFfrenchb=\@numberstringFfrench
3399 \let\@numberstringNfrenchb=\@numberstringNfrench
3400 \let\@NumberstringMfrenchb=\@NumberstringMfrench
3401 \let\@NumberstringFfrenchb=\@NumberstringFfrench
3402 \let\@NumberstringNfrenchb=\@NumberstringNfrench
3403 \let\@ordinalstringMfrenchb=\@ordinalstringMfrench
3404 \let\@ordinalstringFfrenchb=\@ordinalstringFfrench
3405 \let\@ordinalstringNfrenchb=\@ordinalstringNfrench
3406 \let\@OrdinalstringMfrenchb=\@OrdinalstringMfrench

```

```

3407 \let\@OrdinalstringFfrenchb=\@OrdinalstringFfrench
3408 \let\@OrdinalstringNfrenchb=\@OrdinalstringNfrench

```

### 9.3.7 fc-german.def

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

```

3409 \ProvidesFCLanguage{german}[2012/06/18]

```

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:

```

3410 \newcommand{\@OrdinalMgerman}[2]{%
3411 \edef#2{\number#1\relax.}}

```

Feminine:

```

3412 \newcommand{\@OrdinalFgerman}[2]{%
3413 \edef#2{\number#1\relax.}}

```

Neuter:

```

3414 \newcommand{\@OrdinalNgerman}[2]{%
3415 \edef#2{\number#1\relax.}}

```

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):

```

3416 \newcommand{\@@unitstringgerman}[1]{%
3417 \ifcase#1%
3418 null%
3419 \or eins%
3420 \or zwei%
3421 \or drei%
3422 \or vier%
3423 \or f"unf%
3424 \or sechs%
3425 \or sieben%
3426 \or acht%
3427 \or neun%
3428 \fi
3429 }

```

Tens (argument must go from 1 to 10):

```

3430 \newcommand{\@@tenstringgerman}[1]{%
3431 \ifcase#1%
3432 \or zehn%
3433 \or zwanzig%
3434 \or drei{\ss}ig%
3435 \or vierzig%
3436 \or f"unfzig%
3437 \or sechzig%
3438 \or siebzig%
3439 \or achtzig%

```

```

3440 \or neunzig%
3441 \or einhundert%
3442 \fi
3443 }

```

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

```

3444 \providecommand*\einhundert{\einhundert}
3445 \providecommand*\eintausend{\eintausend}

```

Teens:

```

3446 \newcommand{\@@teenstringgerman}[1]{%
3447 \ifcase#1%
3448 zehn%
3449 \or elf%
3450 \or zw\ "olf%
3451 \or dreizehn%
3452 \or vierzehn%
3453 \or f\ "unfzehn%
3454 \or sechzehn%
3455 \or siebzehn%
3456 \or achtzehn%
3457 \or neunzehn%
3458 \fi
3459 }

```

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

```

3460 \DeclareRobustCommand{\@numberstringMgerman}[2]{%
3461 \let\@unitstring=\@unitstringgerman
3462 \let\@teenstring=\@teenstringgerman
3463 \let\@tenstring=\@tenstringgerman
3464 \@numberstringgerman{#1}{#2}}

```

Feminine and neuter forms:

```

3465 \let\@numberstringFgerman=\@numberstringMgerman
3466 \let\@numberstringNgerman=\@numberstringMgerman

```

As above, but initial letters in upper case:

```

3467 \DeclareRobustCommand{\@NumberstringMgerman}[2]{%
3468 \@numberstringMgerman{#1}{\@num@str}%
3469 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3470 }

```

Feminine and neuter form:

```

3471 \let\@NumberstringFgerman=\@NumberstringMgerman
3472 \let\@NumberstringNgerman=\@NumberstringMgerman

```

As above, but for ordinals.

```

3473 \DeclareRobustCommand{\@ordinalstringMgerman}[2]{%
3474 \let\@unitthstring=\@unitthstringMgerman
3475 \let\@teenthstring=\@teenthstringMgerman
3476 \let\@tenthstring=\@tenthstringMgerman

```

```

3477 \let\@unitstring=\@unitstringgerman
3478 \let\@teenstring=\@teenstringgerman
3479 \let\@tenstring=\@tenstringgerman
3480 \def\@thousandth{tausendster}%
3481 \def\@hundredth{hundertster}%
3482 \@@ordinalstringgerman{#1}{#2}}

```

Feminine form:

```

3483 \DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
3484 \let\@unitthstring=\@unitthstringFgerman
3485 \let\@teenthstring=\@teenthstringFgerman
3486 \let\@tenthstring=\@tenthstringFgerman
3487 \let\@unitstring=\@unitstringgerman
3488 \let\@teenstring=\@teenstringgerman
3489 \let\@tenstring=\@tenstringgerman
3490 \def\@thousandth{tausendste}%
3491 \def\@hundredth{hundertste}%
3492 \@@ordinalstringgerman{#1}{#2}}

```

Neuter form:

```

3493 \DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
3494 \let\@unitthstring=\@unitthstringNgerman
3495 \let\@teenthstring=\@teenthstringNgerman
3496 \let\@tenthstring=\@tenthstringNgerman
3497 \let\@unitstring=\@unitstringgerman
3498 \let\@teenstring=\@teenstringgerman
3499 \let\@tenstring=\@tenstringgerman
3500 \def\@thousandth{tausendstes}%
3501 \def\@hundredth{hunderstes}%
3502 \@@ordinalstringgerman{#1}{#2}}

```

As above, but with initial letters in upper case.

```

3503 \DeclareRobustCommand{\@OrdinalstringMgerman}[2]{%
3504 \@OrdinalstringMgerman{#1}{\@num@str}%
3505 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3506 }

```

Feminine form:

```

3507 \DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
3508 \@OrdinalstringFgerman{#1}{\@num@str}%
3509 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3510 }

```

Neuter form:

```

3511 \DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
3512 \@OrdinalstringNgerman{#1}{\@num@str}%
3513 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3514 }

```

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

```

3515 \newcommand{\@unitthstringMgerman}[1]{%

```

```

3516 \ifcase#1%
3517 nullter%
3518 \or erster%
3519 \or zweiter%
3520 \or dritter%
3521 \or vierter%
3522 \or f\"unfter%
3523 \or sechster%
3524 \or siebter%
3525 \or achter%
3526 \or neunter%
3527 \fi
3528 }

```

Tens:

```

3529 \newcommand{\@@tenthstringMgerman}[1]{%
3530 \ifcase#1%
3531 \or zehnter%
3532 \or zwanzigster%
3533 \or drei{\ss}igster%
3534 \or vierzigster%
3535 \or f\"unzigster%
3536 \or sechzigster%
3537 \or siebzigster%
3538 \or achtzigster%
3539 \or neunzigster%
3540 \fi
3541 }

```

Teens:

```

3542 \newcommand{\@@teenthstringMgerman}[1]{%
3543 \ifcase#1%
3544 zehnter%
3545 \or elfter%
3546 \or zw\"olfter%
3547 \or dreizehnter%
3548 \or vierzehnter%
3549 \or f\"unfzehnter%
3550 \or sechzehnter%
3551 \or siebzehnter%
3552 \or achtzehnter%
3553 \or neunzehnter%
3554 \fi
3555 }

```

Units (feminine):

```

3556 \newcommand{\@@unitthstringFgerman}[1]{%
3557 \ifcase#1%
3558 nullte%
3559 \or erste%
3560 \or zweite%

```

```

3561 \or dritte%
3562 \or vierte%
3563 \or f\"unfte%
3564 \or sechste%
3565 \or siebte%
3566 \or achte%
3567 \or neunte%
3568 \fi
3569 }

```

Tens (feminine):

```

3570 \newcommand{\@@tenthstringFgerman}[1]{%
3571 \ifcase#1%
3572 \or zehnte%
3573 \or zwanzigste%
3574 \or drei{\ss}igste%
3575 \or vierzigste%
3576 \or f\"unzigste%
3577 \or sechzigste%
3578 \or siebzigste%
3579 \or achtzigste%
3580 \or neunzigste%
3581 \fi
3582 }

```

Teens (feminine)

```

3583 \newcommand{\@@teenthstringFgerman}[1]{%
3584 \ifcase#1%
3585 zehnte%
3586 \or elfte%
3587 \or zw\"olfte%
3588 \or dreizehnte%
3589 \or vierzehnte%
3590 \or f\"unfzehnte%
3591 \or sechzehnte%
3592 \or siebzehnte%
3593 \or achtzehnte%
3594 \or neunzehnte%
3595 \fi
3596 }

```

Units (neuter):

```

3597 \newcommand{\@@unitthstringNgerman}[1]{%
3598 \ifcase#1%
3599 nulltes%
3600 \or erstes%
3601 \or zweites%
3602 \or drittes%
3603 \or viertes%
3604 \or f\"unftes%
3605 \or sechstes%

```

```

3606 \or siebtes%
3607 \or achtes%
3608 \or neuntes%
3609 \fi
3610 }

```

Tens (neuter):

```

3611 \newcommand{\@@tenthstringNgerman}[1]{%
3612 \ifcase#1%
3613 \or zehntes%
3614 \or zwanzigstes%
3615 \or drei{\ss}igstes%
3616 \or vierzigstes%
3617 \or f\"unfzigstes%
3618 \or sechzigstes%
3619 \or siebzigstes%
3620 \or achtzigstes%
3621 \or neunzigstes%
3622 \fi
3623 }

```

Teens (neuter)

```

3624 \newcommand{\@@teenthstringNgerman}[1]{%
3625 \ifcase#1%
3626 zehntes%
3627 \or elftes%
3628 \or zw\"olftes%
3629 \or dreizehntes%
3630 \or vierzehntes%
3631 \or f\"unfzehntes%
3632 \or sechzehntes%
3633 \or siebzehntes%
3634 \or achtzehntes%
3635 \or neunzehntes%
3636 \fi
3637 }

```

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

```

3638 \newcommand{\@@numberunderhundredgerman}[2]{%
3639 \ifnum#1<10\relax
3640   \ifnum#1>0\relax
3641     \eappto#2{\@unitstring{#1}}%
3642   \fi
3643 \else
3644   \@tmpstrctr=#1\relax
3645   \@modulo{\@tmpstrctr}{10}%
3646   \ifnum#1<20\relax
3647     \eappto#2{\@teenstring{\@tmpstrctr}}%
3648   \else
3649     \ifnum\@tmpstrctr=0\relax

```

```

3650 \else
3651 \eappto#2{\@unitstring{\@tmpstrctr}und}%
3652 \fi
3653 \@tmpstrctr=#1\relax
3654 \divide\@tmpstrctr by 10\relax
3655 \eappto#2{\@tenstring{\@tmpstrctr}}%
3656 \fi
3657 \fi
3658 }

```

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

```

3659 \newcommand{\@@numberstringgerman}[2]{%
3660 \ifnum#1>99999\relax
3661 \PackageError{fmtcount}{Out of range}%
3662 {This macro only works for values less than 100000}%
3663 \else
3664 \ifnum#1<0\relax
3665 \PackageError{fmtcount}{Negative numbers not permitted}%
3666 {This macro does not work for negative numbers, however
3667 you can try typing "minus" first, and then pass the modulus of
3668 this number}%
3669 \fi
3670 \fi
3671 \def#2{}%
3672 \@strctr=#1\relax \divide\@strctr by 1000\relax
3673 \ifnum\@strctr>1\relax

```

#1 is  $\geq 2000$ , \@strctr now contains the number of thousands

```

3674 \@@numberunderhundredgerman{\@strctr}{#2}%
3675 \appto#2{tausend}%
3676 \else

```

#1 lies in range [1000,1999]

```

3677 \ifnum\@strctr=1\relax
3678 \eappto#2{\eintausend}%
3679 \fi
3680 \fi
3681 \@strctr=#1\relax
3682 \@modulo{\@strctr}{1000}%
3683 \divide\@strctr by 100\relax
3684 \ifnum\@strctr>1\relax

```

now dealing with number in range [200,999]

```

3685 \eappto#2{\@unitstring{\@strctr}hundert}%
3686 \else
3687 \ifnum\@strctr=1\relax

```

dealing with number in range [100,199]

```

3688 \ifnum#1>1000\relax

```

```

    if original number > 1000, use einhundert
3689         \appto#2{einhundert}%
3690     \else
        otherwise use \einhundert
3691         \eappto#2{\einhundert}%
3692     \fi
3693 \fi
3694 \fi
3695 \@strctr=#1\relax
3696 \@modulo{\@strctr}{100}%
3697 \ifnum#1=0\relax
3698     \def#2{null}%
3699 \else
3700     \ifnum\@strctr=1\relax
3701         \appto#2{eins}%
3702     \else
3703         \@@numberunderhundredgerman{\@strctr}{#2}%
3704     \fi
3705 \fi
3706 }

    As above, but for ordinals
3707 \newcommand{\@@numberunderhundredthgerman}[2]{%
3708 \ifnum#1<10\relax
3709     \eappto#2{\@unitthstring{#1}}%
3710 \else
3711     \@tmpstrctr=#1\relax
3712     \@modulo{\@tmpstrctr}{10}%
3713     \ifnum#1<20\relax
3714         \eappto#2{\@teenthstring{\@tmpstrctr}}%
3715     \else
3716         \ifnum\@tmpstrctr=0\relax
3717             \else
3718                 \eappto#2{\@unitstring{\@tmpstrctr}und}%
3719             \fi
3720             \@tmpstrctr=#1\relax
3721             \divide\@tmpstrctr by 10\relax
3722             \eappto#2{\@tenthstring{\@tmpstrctr}}%
3723         \fi
3724 \fi
3725 }

3726 \newcommand{\@@ordinalstringgerman}[2]{%
3727 \ifnum#1>99999\relax
3728     \PackageError{fmtcount}{Out of range}%
3729     {This macro only works for values less than 100000}%
3730 \else
3731     \ifnum#1<0\relax
3732         \PackageError{fmtcount}{Negative numbers not permitted}%
3733         {This macro does not work for negative numbers, however

```

```

3734     you can try typing "minus" first, and then pass the modulus of
3735     this number}%
3736 \fi
3737 \fi
3738 \def#2{}%
3739 \@strctr=#1\relax \divide\@strctr by 1000\relax
3740 \ifnum\@strctr>1\relax

    #1 is  $\geq 2000$ , \@strctr now contains the number of thousands
3741 \@numberunderhundredgerman{\@strctr}{#2}%

    is that it, or is there more?
3742 \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{1000}%
3743 \ifnum\@tmpstrctr=0\relax
3744     \eappto#2{\@thousandth}%
3745 \else
3746     \appto#2{tausend}%
3747 \fi
3748 \else

    #1 lies in range [1000,1999]
3749 \ifnum\@strctr=1\relax
3750     \ifnum#1=1000\relax
3751         \eappto#2{\@thousandth}%
3752     \else
3753         \eappto#2{\eintausend}%
3754     \fi
3755 \fi
3756 \fi
3757 \@strctr=#1\relax
3758 \@modulo{\@strctr}{1000}%
3759 \divide\@strctr by 100\relax
3760 \ifnum\@strctr>1\relax

    now dealing with number in range [200,999] is that it, or is there more?
3761 \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{100}%
3762 \ifnum\@tmpstrctr=0\relax
3763     \ifnum\@strctr=1\relax
3764         \eappto#2{\@hundredth}%
3765     \else
3766         \eappto#2{\@unitstring{\@strctr}\@hundredth}%
3767     \fi
3768 \else
3769     \eappto#2{\@unitstring{\@strctr}hundert}%
3770 \fi
3771 \else
3772     \ifnum\@strctr=1\relax

        dealing with number in range [100,199] is that it, or is there more?
3773         \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{100}%
3774         \ifnum\@tmpstrctr=0\relax

```

```

3775         \eappto#2{\@hundredth}%
3776     \else
3777         \ifnum#1>1000\relax
3778         \appto#2{einhundert}%
3779     \else
3780         \eappto#2{einhundert}%
3781     \fi
3782 \fi
3783 \fi
3784 \fi
3785 \@strctr=#1\relax
3786 \@modulo{\@strctr}{100}%
3787 \ifthenelse{\@strctr=0 \and #1>0}{-}{%
3788 \@numberunderhundredthgerman{\@strctr}{#2}%
3789 }%
3790 }

```

Load fc-germanb.def if not already loaded

```
3791 \FClloadlang{germanb}
```

### 9.3.8 fc-germanb.def

```
3792 \ProvidesFCLanguage{germanb}[2012/06/18]
```

Load fc-german.def if not already loaded

```
3793 \FClloadlang{german}
```

Set germanb to be equivalent to german.

```

3794 \let\@ordinalMgermanb=\@ordinalMgerman
3795 \let\@ordinalFgermanb=\@ordinalFgerman
3796 \let\@ordinalNgermanb=\@ordinalNgerman
3797 \let\@numberstringMgermanb=\@numberstringMgerman
3798 \let\@numberstringFgermanb=\@numberstringFgerman
3799 \let\@numberstringNgermanb=\@numberstringNgerman
3800 \let\@NumberstringMgermanb=\@NumberstringMgerman
3801 \let\@NumberstringFgermanb=\@NumberstringFgerman
3802 \let\@NumberstringNgermanb=\@NumberstringNgerman
3803 \let\@ordinalstringMgermanb=\@ordinalstringMgerman
3804 \let\@ordinalstringFgermanb=\@ordinalstringFgerman
3805 \let\@ordinalstringNgermanb=\@ordinalstringNgerman
3806 \let\@OrdinalstringMgermanb=\@OrdinalstringMgerman
3807 \let\@OrdinalstringFgermanb=\@OrdinalstringFgerman
3808 \let\@OrdinalstringNgermanb=\@OrdinalstringNgerman

```

### 9.3.9 fc-italian

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

```
3809 \ProvidesFCLanguage{italian}[2012/06/18]
```

```
3810
```

```
3811 \RequirePackage{itnumpar}
```

```

3812
3813 \newcommand{\@numberstringMitalian}[2]{%
3814   \edef#2{\noexpand\printnumeroinparole{#1}}%
3815 }
3816
3817 \newcommand{\@numberstringFitalian}[2]{%
3818   \edef#2{\noexpand\printnumeroinparole{#1}}%
3819 }
3820 \newcommand{\@NumberstringMitalian}[2]{%
3821   \edef#2{\noexpand\printNumeroinparole{#1}}%
3822 }
3823 \newcommand{\@NumberstringFitalian}[2]{%
3824   \edef#2{\noexpand\printNumeroinparole{#1}}%
3825 }
3826 \newcommand{\@ordinalstringMitalian}[2]{%
3827   \edef#2{\noexpand\printordinalem{#1}}%
3828 }
3829 \newcommand{\@ordinalstringFitalian}[2]{%
3830   \edef#2{\noexpand\printordinalef{#1}}%
3831 }
3832 \newcommand{\@OrdinalstringMitalian}[2]{%
3833   \edef#2{\noexpand\printOrdinalem{#1}}%
3834 }
3835 \newcommand{\@OrdinalstringFitalian}[2]{%
3836   \edef#2{\noexpand\printOrdinalef{#1}}%
3837 }
3838 \newcommand{\@ordinalMitalian}[2]{%
3839   \edef#2{#1\relax\noexpand\fmtord{o}}%
3840 }
3841 \newcommand{\@ordinalFitalian}[2]{%
3842   \edef#2{#1\relax\noexpand\fmtord{a}}%

```

### 9.3.10 fc-ngerman.def

```

3842 \ProvidesFCLanguage{ngerman}[2012/06/18]
3843 \FCloadlang{german}
3844 \FCloadlang{ngermanb}

```

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

```

3845 \let\@ordinalMngerman=\@ordinalMgerman
3846 \let\@ordinalFngerman=\@ordinalFgerman
3847 \let\@ordinalNngerman=\@ordinalNgerman
3848 \let\@numberstringMngerman=\@numberstringMgerman
3849 \let\@numberstringFngerman=\@numberstringFgerman
3850 \let\@numberstringNngerman=\@numberstringNgerman
3851 \let\@NumberstringMngerman=\@NumberstringMgerman
3852 \let\@NumberstringFngerman=\@NumberstringFgerman
3853 \let\@NumberstringNngerman=\@NumberstringNgerman
3854 \let\@ordinalstringMngerman=\@ordinalstringMgerman
3855 \let\@ordinalstringFngerman=\@ordinalstringFgerman

```

```

3856 \let\@ordinalstringNgerman=\@ordinalstringNgerman
3857 \let\@ordinalstringMgerman=\@ordinalstringMgerman
3858 \let\@ordinalstringFgerman=\@ordinalstringFgerman
3859 \let\@ordinalstringNgerman=\@ordinalstringNgerman

```

### 9.3.11 fc-ngermanb.def

```

3860 \ProvidesFCLanguage{ngermanb}[2012/06/18]
3861 \FCloadlang{german}

```

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

```

3862 \let\@ordinalMgermanb=\@ordinalMgerman
3863 \let\@ordinalFgermanb=\@ordinalFgerman
3864 \let\@ordinalNgermanb=\@ordinalNgerman
3865 \let\@numberstringMgermanb=\@numberstringMgerman
3866 \let\@numberstringFgermanb=\@numberstringFgerman
3867 \let\@numberstringNgermanb=\@numberstringNgerman
3868 \let\@NumberstringMgermanb=\@NumberstringMgerman
3869 \let\@NumberstringFgermanb=\@NumberstringFgerman
3870 \let\@NumberstringNgermanb=\@NumberstringNgerman
3871 \let\@ordinalstringMgermanb=\@ordinalstringMgerman
3872 \let\@ordinalstringFgermanb=\@ordinalstringFgerman
3873 \let\@ordinalstringNgermanb=\@ordinalstringNgerman
3874 \let\@OrdinalstringMgermanb=\@OrdinalstringMgerman
3875 \let\@OrdinalstringFgermanb=\@OrdinalstringFgerman
3876 \let\@OrdinalstringNgermanb=\@OrdinalstringNgerman

```

Load fc-ngerman.def if not already loaded

```

3877 \FCloadlang{ngerman}

```

### 9.3.12 fc-portuges.def

Portuguese definitions

```

3878 \ProvidesFCLanguage{portuges}[2012/06/18]

```

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:

```

3879 \newcommand*{\@ordinalMportuges}[2]{%
3880 \ifnum#1=0\relax
3881 \edef#2{\number#1}%
3882 \else
3883 \edef#2{\number#1\relax\noexpand\fmtord{o}}%
3884 \fi}

```

Feminine:

```

3885 \newcommand*{\@ordinalFportuges}[2]{%
3886 \ifnum#1=0\relax
3887 \edef#2{\number#1}%
3888 \else
3889 \edef#2{\number#1\relax\noexpand\fmtord{a}}%
3890 \fi}

```

Make neuter same as masculine:

```
3891 \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):

```
3892 \newcommand*{\@unitstringportuges}[1]{%
3893 \ifcase#1\relax
3894 zero%
3895 \or um%
3896 \or dois%
3897 \or tr\~es%
3898 \or quatro%
3899 \or cinco%
3900 \or seis%
3901 \or sete%
3902 \or oito%
3903 \or nove%
3904 \fi
3905 }
3906 % \end{macrocode}
3907 % As above, but for feminine:
3908 % \begin{macrocode}
3909 \newcommand*{\@unitstringFportuges}[1]{%
3910 \ifcase#1\relax
3911 zero%
3912 \or uma%
3913 \or duas%
3914 \or tr\~es%
3915 \or quatro%
3916 \or cinco%
3917 \or seis%
3918 \or sete%
3919 \or oito%
3920 \or nove%
3921 \fi
3922 }
```

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

```
3923 \newcommand*{\@tenstringportuges}[1]{%
3924 \ifcase#1\relax
3925 \or dez%
3926 \or vinte%
3927 \or trinta%
3928 \or quarenta%
3929 \or cinq\"uenta%
3930 \or sessenta%
3931 \or setenta%
3932 \or oitenta%
3933 \or noventa%
```

```

3934 \or cem%
3935 \fi
3936 }

```

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

```

3937 \newcommand*{\@@teenstringportuges}[1]{%
3938 \ifcase#1\relax
3939 dez%
3940 \or onze%
3941 \or doze%
3942 \or treze%
3943 \or quatorze%
3944 \or quinze%
3945 \or dezesseis%
3946 \or dezessete%
3947 \or dezoito%
3948 \or dezenove%
3949 \fi
3950 }

```

Hundreds:

```

3951 \newcommand*{\@@hundredstringportuges}[1]{%
3952 \ifcase#1\relax
3953 \or cento%
3954 \or duzentos%
3955 \or trezentos%
3956 \or quatrocentos%
3957 \or quinhentos%
3958 \or seiscentos%
3959 \or setecentos%
3960 \or oitocentos%
3961 \or novecentos%
3962 \fi}

```

Hundreds (feminine):

```

3963 \newcommand*{\@@hundredstringFportuges}[1]{%
3964 \ifcase#1\relax
3965 \or cento%
3966 \or duzentas%
3967 \or trezentas%
3968 \or quatrocentas%
3969 \or quinhentas%
3970 \or seiscentas%
3971 \or setecentas%
3972 \or oitocentas%
3973 \or novecentas%
3974 \fi}

```

Units (initial letter in upper case):

```

3975 \newcommand*{\@@Unitstringportuges}[1]{%
3976 \ifcase#1\relax

```

```

3977 Zero%
3978 \or Um%
3979 \or Dois%
3980 \or Tr\~es%
3981 \or Quatro%
3982 \or Cinco%
3983 \or Seis%
3984 \or Sete%
3985 \or Oito%
3986 \or Nove%
3987 \fi
3988 }

```

As above, but feminine:

```

3989 \newcommand*\@@UnitstringFportuges}[1]{%
3990 \ifcase#1\relax
3991 Zera%
3992 \or Uma%
3993 \or Duas%
3994 \or Tr\~es%
3995 \or Quatro%
3996 \or Cinco%
3997 \or Seis%
3998 \or Sete%
3999 \or Oito%
4000 \or Nove%
4001 \fi
4002 }

```

Tens (with initial letter in upper case):

```

4003 \newcommand*\@@Tenstringportuges}[1]{%
4004 \ifcase#1\relax
4005 \or Dez%
4006 \or Vinte%
4007 \or Trinta%
4008 \or Quarenta%
4009 \or Cinq\"uenta%
4010 \or Sessenta%
4011 \or Setenta%
4012 \or Oitenta%
4013 \or Noventa%
4014 \or Cem%
4015 \fi
4016 }

```

Teens (with initial letter in upper case):

```

4017 \newcommand*\@@Teenstringportuges}[1]{%
4018 \ifcase#1\relax
4019 Dez%
4020 \or Onze%
4021 \or Doze%

```

```

4022 \or Treze%
4023 \or Quatorze%
4024 \or Quinze%
4025 \or Dezesseis%
4026 \or Dezesete%
4027 \or Dezoito%
4028 \or Dezenove%
4029 \fi
4030 }

```

Hundreds (with initial letter in upper case):

```

4031 \newcommand*{\@@Hundredstringportuges}[1]{%
4032 \ifcase#1\relax
4033 \or Cento%
4034 \or Duzentos%
4035 \or Trezentos%
4036 \or Quatrocentos%
4037 \or Quinhentos%
4038 \or Seiscentos%
4039 \or Setecentos%
4040 \or Oitocentos%
4041 \or Novecentos%
4042 \fi}

```

As above, but feminine:

```

4043 \newcommand*{\@@HundredstringFportuges}[1]{%
4044 \ifcase#1\relax
4045 \or Cento%
4046 \or Duzentas%
4047 \or Trezentas%
4048 \or Quatrocentas%
4049 \or Quinhentas%
4050 \or Seiscentas%
4051 \or Setecentas%
4052 \or Oitocentas%
4053 \or Novecentas%
4054 \fi}

```

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.)

```

4055 \DeclareRobustCommand{\@numberstringMportuges}[2]{%
4056 \let\@unitstring=\@unitstringportuges
4057 \let\@teenstring=\@teenstringportuges
4058 \let\@tenstring=\@tenstringportuges
4059 \let\@hundredstring=\@hundredstringportuges
4060 \def\@hundred{cem}\def\@thousand{mil}%
4061 \def\@andname{e}%
4062 \@numberstringportuges{#1}{#2}}

```

As above, but feminine form:

```
4063 \DeclareRobustCommand{\@numberstringFportuges}[2]{%
4064 \let\@unitstring=\@unitstringFportuges
4065 \let\@teenstring=\@teenstringportuges
4066 \let\@tenstring=\@tenstringportuges
4067 \let\@hundredstring=\@hundredstringFportuges
4068 \def\@hundred{cem}\def\@thousand{mil}%
4069 \def\@andname{e}%
4070 \@numberstringportuges{#1}{#2}}
```

Make neuter same as masculine:

```
4071 \let\@numberstringNportuges\@numberstringMportuges
```

As above, but initial letters in upper case:

```
4072 \DeclareRobustCommand{\@NumberstringMportuges}[2]{%
4073 \let\@unitstring=\@Unitstringportuges
4074 \let\@teenstring=\@Teenstringportuges
4075 \let\@tenstring=\@Tenstringportuges
4076 \let\@hundredstring=\@Hundredstringportuges
4077 \def\@hundred{Cem}\def\@thousand{Mil}%
4078 \def\@andname{e}%
4079 \@Numberstringportuges{#1}{#2}}
```

As above, but feminine form:

```
4080 \DeclareRobustCommand{\@NumberstringFportuges}[2]{%
4081 \let\@unitstring=\@UnitstringFportuges
4082 \let\@teenstring=\@Teenstringportuges
4083 \let\@tenstring=\@Tenstringportuges
4084 \let\@hundredstring=\@HundredstringFportuges
4085 \def\@hundred{Cem}\def\@thousand{Mil}%
4086 \def\@andname{e}%
4087 \@Numberstringportuges{#1}{#2}}
```

Make neuter same as masculine:

```
4088 \let\@NumberstringNportuges\@NumberstringMportuges
```

As above, but for ordinals.

```
4089 \DeclareRobustCommand{\@OrdinalstringMportuges}[2]{%
4090 \let\@unitthstring=\@unitthstringportuges
4091 \let\@unitstring=\@unitstringportuges
4092 \let\@teenthstring=\@teenthstringportuges
4093 \let\@tenthstring=\@tenthstringportuges
4094 \let\@hundredthstring=\@hundredthstringportuges
4095 \def\@thousandth{mil'esimo}%
4096 \@Ordinalstringportuges{#1}{#2}}
```

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}}

```

Make neuter same as masculine:

```

4105 \let\@ordinalstringNportuges\@ordinalstringMportuges

```

As above, but initial letters in upper case (masculine):

```

4106 \DeclareRobustCommand{\@OrdinalstringMportuges}[2]{%
4107 \let\@unitthstring=\@Unitthstringportuges
4108 \let\@unitstring=\@Unitstringportuges
4109 \let\@teenthstring=\@Teenthstringportuges
4110 \let\@tenthstring=\@Tenthstringportuges
4111 \let\@hundredthstring=\@Hundredthstringportuges
4112 \def\@thousandth{Mil\'esimo}%
4113 \@ordinalstringportuges{#1}{#2}}

```

Feminine form:

```

4114 \DeclareRobustCommand{\@OrdinalstringFportuges}[2]{%
4115 \let\@unitthstring=\@UnitthstringFportuges
4116 \let\@unitstring=\@UnitstringFportuges
4117 \let\@teenthstring=\@Teenthstringportuges
4118 \let\@tenthstring=\@TenthstringFportuges
4119 \let\@hundredthstring=\@HundredthstringFportuges
4120 \def\@thousandth{Mil\'esima}%
4121 \@ordinalstringportuges{#1}{#2}}

```

Make neuter same as masculine:

```

4122 \let\@OrdinalstringNportuges\@OrdinalstringMportuges

```

In order to do the ordinals, split into units, teens, tens and hundreds. Units:

```

4123 \newcommand*{\@unitthstringportuges}[1]{%
4124 \ifcase#1\relax
4125 zero%
4126 \or primeiro%
4127 \or segundo%
4128 \or terceiro%
4129 \or quarto%
4130 \or quinto%
4131 \or sexto%
4132 \or s\'etimo%
4133 \or oitavo%
4134 \or nono%
4135 \fi
4136 }

```

Tens:

```

4137 \newcommand*{\@tenthstringportuges}[1]{%
4138 \ifcase#1\relax
4139 \or d\'ecimo%
4140 \or vig\'esimo%
4141 \or trig\'esimo%

```

```

4142 \or quadrag\'esimo%
4143 \or q\'uinquag\'esimo%
4144 \or sexag\'esimo%
4145 \or setuag\'esimo%
4146 \or octog\'esimo%
4147 \or nonag\'esimo%
4148 \fi
4149 }

```

#### Teens:

```

4150 \newcommand*{\@@teenthstringportuges}[1]{%
4151 \@tenthstring{1}%
4152 \ifnum#1>0\relax
4153 -\@unitthstring{#1}%
4154 \fi}

```

#### Hundreds:

```

4155 \newcommand*{\@@hundredthstringportuges}[1]{%
4156 \ifcase#1\relax
4157 \or cent\'esimo%
4158 \or ducent\'esimo%
4159 \or trecent\'esimo%
4160 \or quadringent\'esimo%
4161 \or q\'uingent\'esimo%
4162 \or seiscient\'esimo%
4163 \or setingent\'esimo%
4164 \or octingent\'esimo%
4165 \or nongent\'esimo%
4166 \fi}

```

#### Units (feminine):

```

4167 \newcommand*{\@@unitthstringFportuges}[1]{%
4168 \ifcase#1\relax
4169 zero%
4170 \or primeira%
4171 \or segunda%
4172 \or terceira%
4173 \or quarta%
4174 \or quinta%
4175 \or sexta%
4176 \or s\'etima%
4177 \or oitava%
4178 \or nona%
4179 \fi
4180 }

```

#### Tens (feminine):

```

4181 \newcommand*{\@@tenthstringFportuges}[1]{%
4182 \ifcase#1\relax
4183 \or d\'ecima%
4184 \or vig\'esima%

```

```

4185 \or trig\'esima%
4186 \or quadrag\'esima%
4187 \or q\"uinquag\'esima%
4188 \or sexag\'esima%
4189 \or setuag\'esima%
4190 \or octog\'esima%
4191 \or nonag\'esima%
4192 \fi
4193 }

```

Hundreds (feminine):

```

4194 \newcommand*{\@@hundredthstringFportuges}[1]{%
4195 \ifcase#1\relax
4196 \or cent\'esima%
4197 \or ducent\'esima%
4198 \or trecent\'esima%
4199 \or quadringent\'esima%
4200 \or q\"uingent\'esima%
4201 \or seiscent\'esima%
4202 \or setingent\'esima%
4203 \or octingent\'esima%
4204 \or nongent\'esima%
4205 \fi}

```

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

```

4206 \newcommand*{\@@Unitthstringportuges}[1]{%
4207 \ifcase#1\relax
4208 Zero%
4209 \or Primeiro%
4210 \or Segundo%
4211 \or Terceiro%
4212 \or Quarto%
4213 \or Quinto%
4214 \or Sexto%
4215 \or S\'etimo%
4216 \or Oitavo%
4217 \or Nono%
4218 \fi
4219 }

```

Tens:

```

4220 \newcommand*{\@@Tenthstringportuges}[1]{%
4221 \ifcase#1\relax
4222 \or D\'ecimo%
4223 \or Vig\'esimo%
4224 \or Trig\'esimo%
4225 \or Quadrag\'esimo%
4226 \or Q\"uinquag\'esimo%
4227 \or Sexag\'esimo%
4228 \or Setuag\'esimo%
4229 \or Octog\'esimo%

```

4230 \or Nonag\'esimo%  
 4231 \fi  
 4232 }

Hundreds:

4233 \newcommand\*{\@@Hundredthstringportuges}[1]{%  
 4234 \ifcase#1\relax  
 4235 \or Cent\'esimo%  
 4236 \or Ducent\'esimo%  
 4237 \or Trecent\'esimo%  
 4238 \or Quadringent\'esimo%  
 4239 \or Q\'uingent\'esimo%  
 4240 \or Seiscent\'esimo%  
 4241 \or Setingent\'esimo%  
 4242 \or Octingent\'esimo%  
 4243 \or Nongent\'esimo%  
 4244 \fi}

As above, but feminine. Units:

4245 \newcommand\*{\@@UnitthstringFportuges}[1]{%  
 4246 \ifcase#1\relax  
 4247 Zera%  
 4248 \or Primeira%  
 4249 \or Segunda%  
 4250 \or Terceira%  
 4251 \or Quarta%  
 4252 \or Quinta%  
 4253 \or Sexta%  
 4254 \or S\'etima%  
 4255 \or Oitava%  
 4256 \or Nona%  
 4257 \fi  
 4258 }

Tens (feminine);

4259 \newcommand\*{\@@TenthstringFportuges}[1]{%  
 4260 \ifcase#1\relax  
 4261 \or D\'ecima%  
 4262 \or Vig\'esima%  
 4263 \or Trig\'esima%  
 4264 \or Quadrag\'esima%  
 4265 \or Q\'uinquag\'esima%  
 4266 \or Sexag\'esima%  
 4267 \or Setuag\'esima%  
 4268 \or Octog\'esima%  
 4269 \or Nonag\'esima%  
 4270 \fi  
 4271 }

Hundreds (feminine):

4272 \newcommand\*{\@@HundredthstringFportuges}[1]{%

```

4273 \ifcase#1\relax
4274 \or Cent\'esima%
4275 \or Ducent\'esima%
4276 \or Trecent\'esima%
4277 \or Quadringent\'esima%
4278 \or Q\'uingent\'esima%
4279 \or Seiscent\'esima%
4280 \or Setingent\'esima%
4281 \or Octingent\'esima%
4282 \or Nongent\'esima%
4283 \fi}

```

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.)

```

4284 \newcommand*{\@numberstringportuges}[2]{%
4285 \ifnum#1>99999
4286 \PackageError{fmtcount}{Out of range}%
4287 {This macro only works for values less than 100000}%
4288 \else
4289 \ifnum#1<0
4290 \PackageError{fmtcount}{Negative numbers not permitted}%
4291 {This macro does not work for negative numbers, however
4292 you can try typing "minus" first, and then pass the modulus of
4293 this number}%
4294 \fi
4295 \fi
4296 \def#2{}%
4297 \@strctr=#1\relax \divide\@strctr by 1000\relax
4298 \ifnum\@strctr>9
      #1 is greater or equal to 10000
4299 \divide\@strctr by 10
4300 \ifnum\@strctr>1\relax
4301 \let\@fc@numstr#2\relax
4302 \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
4303 \@strctr=#1 \divide\@strctr by 1000\relax
4304 \@modulo{\@strctr}{10}%
4305 \ifnum\@strctr>0
4306 \ifnum\@strctr=1\relax
4307 \let\@fc@numstr#2\relax
4308 \edef#2{\@fc@numstr\ \@andname}%
4309 \fi
4310 \let\@fc@numstr#2\relax
4311 \edef#2{\@fc@numstr\ \@unitstring{\@strctr}}%
4312 \fi
4313 \else
4314 \@strctr=#1\relax
4315 \divide\@strctr by 1000\relax

```

```

4316 \@modulo{\@strctr}{10}%
4317 \let\@@fc@numstr#2\relax
4318 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
4319 \fi
4320 \let\@@fc@numstr#2\relax
4321 \edef#2{\@@fc@numstr\@thousand}%
4322 \else
4323 \ifnum\@strctr>0\relax
4324 \ifnum\@strctr>1\relax
4325 \let\@@fc@numstr#2\relax
4326 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
4327 \fi
4328 \let\@@fc@numstr#2\relax
4329 \edef#2{\@@fc@numstr\@thousand}%
4330 \fi
4331 \fi
4332 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
4333 \divide\@strctr by 100\relax
4334 \ifnum\@strctr>0\relax
4335 \ifnum#1>1000 \relax
4336 \let\@@fc@numstr#2\relax
4337 \edef#2{\@@fc@numstr\ }%
4338 \fi
4339 \@tmpstrctr=#1\relax
4340 \@modulo{\@tmpstrctr}{1000}%
4341 \let\@@fc@numstr#2\relax
4342 \ifnum\@tmpstrctr=100\relax
4343 \edef#2{\@@fc@numstr\@tenstring{10}}%
4344 \else
4345 \edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
4346 \fi%
4347 \fi
4348 \@strctr=#1\relax \@modulo{\@strctr}{100}%
4349 \ifnum#1>100\relax
4350 \ifnum\@strctr>0\relax
4351 \let\@@fc@numstr#2\relax
4352 \edef#2{\@@fc@numstr\ \@andname\ }%
4353 \fi
4354 \fi
4355 \ifnum\@strctr>19\relax
4356 \divide\@strctr by 10\relax
4357 \let\@@fc@numstr#2\relax
4358 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
4359 \@strctr=#1\relax \@modulo{\@strctr}{10}%
4360 \ifnum\@strctr>0
4361 \ifnum\@strctr=1\relax
4362 \let\@@fc@numstr#2\relax
4363 \edef#2{\@@fc@numstr\ \@andname}%
4364 \else

```

```

4365     \ifnum#1>100\relax
4366         \let\@@fc@numstr#2\relax
4367         \edef#2{\@@fc@numstr\ \@andname}%
4368     \fi
4369 \fi
4370 \let\@@fc@numstr#2\relax
4371 \edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
4372 \fi
4373 \else
4374     \ifnum\@strctr<10\relax
4375         \ifnum\@strctr=0\relax
4376             \ifnum#1<100\relax
4377                 \let\@@fc@numstr#2\relax
4378                 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
4379             \fi
4380         \else%(>0,<10)
4381             \let\@@fc@numstr#2\relax
4382             \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
4383         \fi
4384     \else%>10
4385         \@modulo{\@strctr}{10}%
4386         \let\@@fc@numstr#2\relax
4387         \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
4388     \fi
4389 \fi
4390 }

```

As above, but for ordinals.

```

4391 \newcommand*{\@@ordinalstringportuges}[2]{%
4392 \@strctr=#1\relax
4393 \ifnum#1>99999
4394 \PackageError{fmtcount}{Out of range}%
4395 {This macro only works for values less than 100000}%
4396 \else
4397 \ifnum#1<0
4398 \PackageError{fmtcount}{Negative numbers not permitted}%
4399 {This macro does not work for negative numbers, however
4400 you can try typing "minus" first, and then pass the modulus of
4401 this number}%
4402 \else
4403 \def#2{}%
4404 \ifnum\@strctr>999\relax
4405     \divide\@strctr by 1000\relax
4406     \ifnum\@strctr>1\relax
4407         \ifnum\@strctr>9\relax
4408             \@tmpstrctr=\@strctr
4409             \ifnum\@strctr<20
4410                 \@modulo{\@tmpstrctr}{10}%
4411                 \let\@@fc@ordstr#2\relax
4412                 \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%

```

```

4413     \else
4414         \divide\@tmpstrctr by 10\relax
4415         \let\@@fc@ordstr#2\relax
4416         \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4417         \@tmpstrctr=\@strctr
4418         \@modulo{\@tmpstrctr}{10}%
4419         \ifnum\@tmpstrctr>0\relax
4420             \let\@@fc@ordstr#2\relax
4421             \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
4422         \fi
4423     \fi
4424     \else
4425         \let\@@fc@ordstr#2\relax
4426         \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
4427     \fi
4428 \fi
4429 \let\@@fc@ordstr#2\relax
4430 \edef#2{\@@fc@ordstr\@thousandth}%
4431 \fi
4432 \@strctr=#1\relax
4433 \@modulo{\@strctr}{1000}%
4434 \ifnum\@strctr>99\relax
4435     \@tmpstrctr=\@strctr
4436     \divide\@tmpstrctr by 100\relax
4437     \ifnum#1>1000\relax
4438         \let\@@fc@ordstr#2\relax
4439         \edef#2{\@@fc@ordstr-}%
4440     \fi
4441     \let\@@fc@ordstr#2\relax
4442     \edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
4443 \fi
4444 \@modulo{\@strctr}{100}%
4445 \ifnum#1>99\relax
4446     \ifnum\@strctr>0\relax
4447         \let\@@fc@ordstr#2\relax
4448         \edef#2{\@@fc@ordstr-}%
4449     \fi
4450 \fi
4451 \ifnum\@strctr>9\relax
4452     \@tmpstrctr=\@strctr
4453     \divide\@tmpstrctr by 10\relax
4454     \let\@@fc@ordstr#2\relax
4455     \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4456     \@tmpstrctr=\@strctr
4457     \@modulo{\@tmpstrctr}{10}%
4458     \ifnum\@tmpstrctr>0\relax
4459         \let\@@fc@ordstr#2\relax
4460         \edef#2{\@@fc@ordstr-\@unitthstring{\@tmpstrctr}}%
4461     \fi

```

```

4462 \else
4463   \ifnum \@strctr=0\relax
4464     \ifnum #1=0\relax
4465       \let \@fc@ordstr#2\relax
4466       \edef#2{\@fc@ordstr\@unitstring{0}}%
4467     \fi
4468   \else
4469     \let \@fc@ordstr#2\relax
4470     \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
4471   \fi
4472 \fi
4473 \fi
4474 \fi
4475 }

```

### 9.3.13 fc-spanish.def

Spanish definitions

```

4476 \ProvidesFCLanguage{spanish}[2012/06/18]

```

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:

```

4477 \newcommand{\@ordinalMspanish}[2]{%
4478 \edef#2{\number#1\relax\noexpand\fmtord{o}}}

```

Feminine:

```

4479 \newcommand{\@ordinalFspanish}[2]{%
4480 \edef#2{\number#1\relax\noexpand\fmtord{a}}}

```

Make neuter same as masculine:

```

4481 \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):

```

4482 \newcommand{\@@unitstringspanish}[1]{%
4483 \ifcase#1\relax
4484   cero%
4485 \or uno%
4486 \or dos%
4487 \or tres%
4488 \or cuatro%
4489 \or cinco%
4490 \or seis%
4491 \or siete%
4492 \or ocho%
4493 \or nueve%
4494 \fi
4495 }

```

Feminine:

```
4496 \newcommand{\@@unitstringFspanish}[1]{%
4497 \ifcase#1\relax
4498 cera%
4499 \or una%
4500 \or dos%
4501 \or tres%
4502 \or cuatro%
4503 \or cinco%
4504 \or seis%
4505 \or siete%
4506 \or ocho%
4507 \or nueve%
4508 \fi
4509 }
```

Tens (argument must go from 1 to 10):

```
4510 \newcommand{\@@tenstringspanish}[1]{%
4511 \ifcase#1\relax
4512 \or diez%
4513 \or veinte%
4514 \or treinta%
4515 \or cuarenta%
4516 \or cincuenta%
4517 \or sesenta%
4518 \or setenta%
4519 \or ochenta%
4520 \or noventa%
4521 \or cien%
4522 \fi
4523 }
```

Teens:

```
4524 \newcommand{\@@teenstringspanish}[1]{%
4525 \ifcase#1\relax
4526 diez%
4527 \or once%
4528 \or doce%
4529 \or trece%
4530 \or catorce%
4531 \or quince%
4532 \or diecis\`eis%
4533 \or diecisiete%
4534 \or dieciocho%
4535 \or diecinueve%
4536 \fi
4537 }
```

Twenties:

```
4538 \newcommand{\@@twentystringspanish}[1]{%
```

```

4539 \ifcase#1\relax
4540 veinte%
4541 \or veintiuno%
4542 \or veintid\'os%
4543 \or veintitr\'es%
4544 \or veinticuatro%
4545 \or veinticinco%
4546 \or veintis\'eis%
4547 \or veintisiete%
4548 \or veintiocho%
4549 \or veintinueve%
4550 \fi}

```

Feminine form:

```

4551 \newcommand{\@twentystringFspanish}[1]{%
4552 \ifcase#1\relax
4553 veinte%
4554 \or veintiuna%
4555 \or veintid\'os%
4556 \or veintitr\'es%
4557 \or veinticuatro%
4558 \or veinticinco%
4559 \or veintis\'eis%
4560 \or veintisiete%
4561 \or veintiocho%
4562 \or veintinueve%
4563 \fi}

```

Hundreds:

```

4564 \newcommand{\@hundredstringspanish}[1]{%
4565 \ifcase#1\relax
4566 \or ciento%
4567 \or doscientos%
4568 \or trescientos%
4569 \or cuatrocientos%
4570 \or quinientos%
4571 \or seiscientos%
4572 \or setecientos%
4573 \or ochocientos%
4574 \or novecientos%
4575 \fi}

```

Feminine form:

```

4576 \newcommand{\@hundredstringFspanish}[1]{%
4577 \ifcase#1\relax
4578 \or cienta%
4579 \or doscientas%
4580 \or trescientas%
4581 \or cuatrocientas%
4582 \or quinientas%
4583 \or seiscientas%

```

```

4584 \or setecientas%
4585 \or ochocientas%
4586 \or novecientas%
4587 \fi}

```

As above, but with initial letter uppercase:

```

4588 \newcommand{\@@Unitstringspanish}[1]{%
4589 \ifcase#1\relax
4590 Cero%
4591 \or Uno%
4592 \or Dos%
4593 \or Tres%
4594 \or Cuatro%
4595 \or Cinco%
4596 \or Seis%
4597 \or Siete%
4598 \or Ocho%
4599 \or Nueve%
4600 \fi
4601 }

```

Feminine form:

```

4602 \newcommand{\@@UnitstringFspanish}[1]{%
4603 \ifcase#1\relax
4604 Cera%
4605 \or Una%
4606 \or Dos%
4607 \or Tres%
4608 \or Cuatro%
4609 \or Cinco%
4610 \or Seis%
4611 \or Siete%
4612 \or Ocho%
4613 \or Nueve%
4614 \fi
4615 }

```

Tens:

```

4616 %\changes{2.0}{2012-06-18}{fixed spelling mistake (correction
4617 %provided by Fernando Maldonado)}
4618 \newcommand{\@@Tenstringspanish}[1]{%
4619 \ifcase#1\relax
4620 \or Diez%
4621 \or Veinte%
4622 \or Treinta%
4623 \or Cuarenta%
4624 \or Cincuenta%
4625 \or Sesenta%
4626 \or Setenta%
4627 \or Ochenta%
4628 \or Noventa%

```

```

4629 \or Cien%
4630 \fi
4631 }

```

#### Teens:

```

4632 \newcommand{\@@Teenstringspanish}[1]{%
4633 \ifcase#1\relax
4634 Diez%
4635 \or Once%
4636 \or Doce%
4637 \or Trece%
4638 \or Catorce%
4639 \or Quince%
4640 \or Diecis\'eis%
4641 \or Diecisiete%
4642 \or Dieciocho%
4643 \or Diecinueve%
4644 \fi
4645 }

```

#### Twenties:

```

4646 \newcommand{\@@Twentystringspanish}[1]{%
4647 \ifcase#1\relax
4648 Veinte%
4649 \or Veintiuno%
4650 \or Veintid\'os%
4651 \or Veintitr\'es%
4652 \or Veinticuatro%
4653 \or Veinticinco%
4654 \or Veintis\'eis%
4655 \or Veintisiete%
4656 \or Veintiocho%
4657 \or Veintinueve%
4658 \fi}

```

#### Feminine form:

```

4659 \newcommand{\@@TwentystringFspanish}[1]{%
4660 \ifcase#1\relax
4661 Veinte%
4662 \or Veintiuna%
4663 \or Veintid\'os%
4664 \or Veintitr\'es%
4665 \or Veinticuatro%
4666 \or Veinticinco%
4667 \or Veintis\'eis%
4668 \or Veintisiete%
4669 \or Veintiocho%
4670 \or Veintinueve%
4671 \fi}

```

#### Hundreds:

```

4672 \newcommand{\@@Hundredstringspanish}[1]{%
4673 \ifcase#1\relax
4674 \or Ciento%
4675 \or Doscientos%
4676 \or Trescientos%
4677 \or Cuatrocientos%
4678 \or Quinientos%
4679 \or Seiscientos%
4680 \or Setecientos%
4681 \or Ochocientos%
4682 \or Novecientos%
4683 \fi}

```

Feminine form:

```

4684 \newcommand{\@@HundredstringFspanish}[1]{%
4685 \ifcase#1\relax
4686 \or Cienta%
4687 \or Doscientas%
4688 \or Trescientas%
4689 \or Cuatrocientas%
4690 \or Quinientas%
4691 \or Seiscientas%
4692 \or Setecientas%
4693 \or Ochocientas%
4694 \or Novecientas%
4695 \fi}

```

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}}

```

Feminine form:

```

4705 \DeclareRobustCommand{\@numberstringFspanish}[2]{%
4706 \let\@unitstring=\@unitstringFspanish
4707 \let\@teenstring=\@teenstringspanish
4708 \let\@tenstring=\@tenstringspanish
4709 \let\@twentystring=\@twentystringFspanish
4710 \let\@hundredstring=\@hundredstringFspanish
4711 \def\@hundred{cien}\def\@thousand{mil}%
4712 \def\@andname{b}%

```

4713 \@@numberstringspanish{#1}{#2}}

Make neuter same as masculine:

4714 \let\@numberstringNspanish\@numberstringMspanish

As above, but initial letters in upper case:

4715 \DeclareRobustCommand{\@NumberstringMspanish}[2]{%

4716 \let\@unitstring=\@@Unitstringspanish

4717 \let\@teenstring=\@@Teenstringspanish

4718 \let\@tenstring=\@@Tenstringspanish

4719 \let\@twentystring=\@@Twentystringspanish

4720 \let\@hundredstring=\@@Hundredstringspanish

4721 \def\@andname{y}%

4722 \def\@hundred{Cien}\def\@thousand{Mil}%

4723 \@@numberstringspanish{#1}{#2}}

Feminine form:

4724 \DeclareRobustCommand{\@NumberstringFspanish}[2]{%

4725 \let\@unitstring=\@@UnitstringFspanish

4726 \let\@teenstring=\@@Teenstringspanish

4727 \let\@tenstring=\@@Tenstringspanish

4728 \let\@twentystring=\@@TwentystringFspanish

4729 \let\@hundredstring=\@@HundredstringFspanish

4730 \def\@andname{b}%

4731 \def\@hundred{Cien}\def\@thousand{Mil}%

4732 \@@numberstringspanish{#1}{#2}}

Make neuter same as masculine:

4733 \let\@NumberstringNspanish\@NumberstringMspanish

As above, but for ordinals.

4734 \DeclareRobustCommand{\@OrdinalstringMspanish}[2]{%

4735 \let\@unitthstring=\@@unitthstringspanish

4736 \let\@unitstring=\@@unitstringspanish

4737 \let\@teenthstring=\@@teenthstringspanish

4738 \let\@tenthstring=\@@tenthstringspanish

4739 \let\@hundredthstring=\@@hundredthstringspanish

4740 \def\@thousandth{mil\'esimo}%

4741 \@@ordinalstringspanish{#1}{#2}}

Feminine form:

4742 \DeclareRobustCommand{\@OrdinalstringFspanish}[2]{%

4743 \let\@unitthstring=\@@unitthstringFspanish

4744 \let\@unitstring=\@@unitstringFspanish

4745 \let\@teenthstring=\@@teenthstringFspanish

4746 \let\@tenthstring=\@@tenthstringFspanish

4747 \let\@hundredthstring=\@@hundredthstringFspanish

4748 \def\@thousandth{mil\'esima}%

4749 \@@ordinalstringspanish{#1}{#2}}

Make neuter same as masculine:

4750 \let\@OrdinalstringNspanish\@OrdinalstringMspanish

As above, but with initial letters in upper case.

```
4751 \DeclareRobustCommand{\@OrdinalstringMspanish}[2]{%
4752 \let\@unitthstring=\@Unitthstringspanish
4753 \let\@unitstring=\@Unitstringspanish
4754 \let\@teenthstring=\@Teenthstringspanish
4755 \let\@tenthstring=\@Tenthstringspanish
4756 \let\@hundredthstring=\@Hundredthstringspanish
4757 \def\@thousandth{Mil\'esimo}%
4758 \@ordinalstringspanish{#1}{#2}}
```

Feminine form:

```
4759 \DeclareRobustCommand{\@OrdinalstringFspanish}[2]{%
4760 \let\@unitthstring=\@UnitthstringFspanish
4761 \let\@unitstring=\@UnitstringFspanish
4762 \let\@teenthstring=\@TeenthstringFspanish
4763 \let\@tenthstring=\@TenthstringFspanish
4764 \let\@hundredthstring=\@HundredthstringFspanish
4765 \def\@thousandth{Mil\'esima}%
4766 \@ordinalstringspanish{#1}{#2}}
```

Make neuter same as masculine:

```
4767 \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:

```
4768 \newcommand{\@unitthstringspanish}[1]{%
4769 \ifcase#1\relax
4770 cero%
4771 \or primero%
4772 \or segundo%
4773 \or tercero%
4774 \or cuarto%
4775 \or quinto%
4776 \or sexto%
4777 \or s\'eptimo%
4778 \or octavo%
4779 \or noveno%
4780 \fi
4781 }
```

Tens:

```
4782 \newcommand{\@tenthstringspanish}[1]{%
4783 \ifcase#1\relax
4784 \or d\'ecimo%
4785 \or vig\'esimo%
4786 \or trig\'esimo%
4787 \or cuadrag\'esimo%
4788 \or quincuag\'esimo%
4789 \or sexag\'esimo%
4790 \or septuag\'esimo%
4791 \or octog\'esimo%
```

```

4792 \or nonag\'esimo%
4793 \fi
4794 }

```

#### Teens:

```

4795 \newcommand{\@@teenthstringspanish}[1]{%
4796 \ifcase#1\relax
4797 d\'ecimo%
4798 \or und\'ecimo%
4799 \or duod\'ecimo%
4800 \or decimotercero%
4801 \or decimocuarto%
4802 \or decimoquinto%
4803 \or decimosexto%
4804 \or decimos\'eptimo%
4805 \or decimoctavo%
4806 \or decimonoveno%
4807 \fi
4808 }

```

#### Hundreds:

```

4809 \newcommand{\@@hundredthstringspanish}[1]{%
4810 \ifcase#1\relax
4811 \or cent\'esimo%
4812 \or ducent\'esimo%
4813 \or tricent\'esimo%
4814 \or cuadringent\'esimo%
4815 \or quingent\'esimo%
4816 \or sexcent\'esimo%
4817 \or septing\'esimo%
4818 \or octingent\'esimo%
4819 \or noningent\'esimo%
4820 \fi}

```

#### Units (feminine):

```

4821 \newcommand{\@@unitthstringFspanish}[1]{%
4822 \ifcase#1\relax
4823 cera%
4824 \or primera%
4825 \or segunda%
4826 \or tercera%
4827 \or cuarta%
4828 \or quinta%
4829 \or sexta%
4830 \or s\'eptima%
4831 \or octava%
4832 \or novena%
4833 \fi
4834 }

```

#### Tens (feminine):

```

4835 \newcommand{\@@tenthstringFspanish}[1]{%
4836 \ifcase#1\relax
4837 \or d\'ecima%
4838 \or vig\'esima%
4839 \or trig\'esima%
4840 \or cuadrag\'esima%
4841 \or quincuag\'esima%
4842 \or sexag\'esima%
4843 \or septuag\'esima%
4844 \or octog\'esima%
4845 \or nonag\'esima%
4846 \fi
4847 }

```

#### Teens (feminine)

```

4848 \newcommand{\@@teenthstringFspanish}[1]{%
4849 \ifcase#1\relax
4850 d\'ecima%
4851 \or und\'ecima%
4852 \or duod\'ecima%
4853 \or decimotercera%
4854 \or decimocuarta%
4855 \or decimoquinta%
4856 \or decimosexta%
4857 \or decimos\'eptima%
4858 \or decimoctava%
4859 \or decimonovena%
4860 \fi
4861 }

```

#### Hundreds (feminine)

```

4862 \newcommand{\@@hundredthstringFspanish}[1]{%
4863 \ifcase#1\relax
4864 \or cent\'esima%
4865 \or ducent\'esima%
4866 \or tricent\'esima%
4867 \or cuadringent\'esima%
4868 \or quingent\'esima%
4869 \or sexcent\'esima%
4870 \or septing\'esima%
4871 \or octingent\'esima%
4872 \or noningent\'esima%
4873 \fi}

```

#### As above, but with initial letters in upper case

```

4874 \newcommand{\@@Unitthstringspanish}[1]{%
4875 \ifcase#1\relax
4876 Cero%
4877 \or Primero%
4878 \or Segundo%
4879 \or Tercero%

```

```

4880 \or Cuarto%
4881 \or Quinto%
4882 \or Sexto%
4883 \or S\'eptimo%
4884 \or Octavo%
4885 \or Noveno%
4886 \fi
4887 }

```

#### Tens:

```

4888 \newcommand{\@@Tenthstringspanish}[1]{%
4889 \ifcase#1\relax
4890 \or D\'ecimo%
4891 \or Vig\'esimo%
4892 \or Trig\'esimo%
4893 \or Cuadrag\'esimo%
4894 \or Quincuag\'esimo%
4895 \or Sexag\'esimo%
4896 \or Septuag\'esimo%
4897 \or Octog\'esimo%
4898 \or Nonag\'esimo%
4899 \fi
4900 }

```

#### Teens:

```

4901 \newcommand{\@@Teenthstringspanish}[1]{%
4902 \ifcase#1\relax
4903 D\'ecimo%
4904 \or Und\'ecimo%
4905 \or Duod\'ecimo%
4906 \or Decimotercero%
4907 \or Decimocuarto%
4908 \or Decimoquinto%
4909 \or Decimosexto%
4910 \or Decimos\'eptimo%
4911 \or Decimooctavo%
4912 \or Decimonoveno%
4913 \fi
4914 }

```

#### Hundreds

```

4915 \newcommand{\@@Hundredthstringspanish}[1]{%
4916 \ifcase#1\relax
4917 \or Cent\'esimo%
4918 \or Ducent\'esimo%
4919 \or Tricent\'esimo%
4920 \or Cuadringent\'esimo%
4921 \or Quingent\'esimo%
4922 \or Sexcent\'esimo%
4923 \or Septing\'esimo%
4924 \or Octingent\'esimo%

```

4925 \or Noningent\'esimo%  
4926 \fi}

As above, but feminine.

4927 \newcommand{\@@UnitthstringFspanish}[1]{%  
4928 \ifcase#1\relax  
4929 Cera%  
4930 \or Primera%  
4931 \or Segunda%  
4932 \or Tercera%  
4933 \or Cuarta%  
4934 \or Quinta%  
4935 \or Sexta%  
4936 \or S\'eptima%  
4937 \or Octava%  
4938 \or Novena%  
4939 \fi  
4940 }

Tens (feminine)

4941 \newcommand{\@@TenthstringFspanish}[1]{%  
4942 \ifcase#1\relax  
4943 \or D\'ecima%  
4944 \or Vig\'esima%  
4945 \or Trig\'esima%  
4946 \or Cuadrag\'esima%  
4947 \or Quincuag\'esima%  
4948 \or Sexag\'esima%  
4949 \or Septuag\'esima%  
4950 \or Octog\'esima%  
4951 \or Nonag\'esima%  
4952 \fi  
4953 }

Teens (feminine):

4954 \newcommand{\@@TeenthstringFspanish}[1]{%  
4955 \ifcase#1\relax  
4956 D\'ecima%  
4957 \or Und\'ecima%  
4958 \or Duod\'ecima%  
4959 \or Decimotercera%  
4960 \or Decimocuarta%  
4961 \or Decimoquinta%  
4962 \or Decimosexta%  
4963 \or Decimos\'eptima%  
4964 \or Decimoctava%  
4965 \or Decimonovena%  
4966 \fi  
4967 }

Hundreds (feminine):

```

4968 \newcommand{\@@HundredthstringFspanish}[1]{%
4969 \ifcase#1\relax
4970 \or Cent\'esima%
4971 \or Ducent\'esima%
4972 \or Tricent\'esima%
4973 \or Cuadringent\'esima%
4974 \or Quingent\'esima%
4975 \or Sexcent\'esima%
4976 \or Septing\'esima%
4977 \or Octingent\'esima%
4978 \or Noningent\'esima%
4979 \fi}

```

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

```

4980 \newcommand{\@@numberstringspanish}[2]{%
4981 \ifnum#1>99999
4982 \PackageError{fmtcount}{Out of range}%
4983 {This macro only works for values less than 100000}%
4984 \else
4985 \ifnum#1<0
4986 \PackageError{fmtcount}{Negative numbers not permitted}%
4987 {This macro does not work for negative numbers, however
4988 you can try typing "minus" first, and then pass the modulus of
4989 this number}%
4990 \fi
4991 \fi
4992 \def#2{}%
4993 \@strctr=#1\relax \divide\@strctr by 1000\relax
4994 \ifnum\@strctr>9
4995     \divide\@strctr by 10
4996     \ifnum\@strctr>1
4997         \let\@@fc@numstr#2\relax
4998         \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
4999         \@strctr=#1 \divide\@strctr by 1000\relax
5000         \@modulo{\@strctr}{10}%
5001         \ifnum\@strctr>0\relax
5002             \let\@@fc@numstr#2\relax
5003             \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
5004         \fi
5005     \else
5006         \@strctr=#1\relax
5007         \divide\@strctr by 1000\relax
5008         \@modulo{\@strctr}{10}%
5009         \let\@@fc@numstr#2\relax
5010         \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%

```

```

5011 \fi
5012 \let\@@fc@numstr#2\relax
5013 \edef#2{\@@fc@numstr\ \@thousand}%
5014 \else
5015 \ifnum\@strctr>0\relax
5016 \ifnum\@strctr>1\relax
5017 \let\@@fc@numstr#2\relax
5018 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
5019 \fi
5020 \let\@@fc@numstr#2\relax
5021 \edef#2{\@@fc@numstr\@thousand}%
5022 \fi
5023 \fi
5024 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
5025 \divide\@strctr by 100\relax
5026 \ifnum\@strctr>0\relax
5027 \ifnum#1>1000\relax
5028 \let\@@fc@numstr#2\relax
5029 \edef#2{\@@fc@numstr\ }%
5030 \fi
5031 \@tmpstrctr=#1\relax
5032 \@modulo{\@tmpstrctr}{1000}%
5033 \ifnum\@tmpstrctr=100\relax
5034 \let\@@fc@numstr#2\relax
5035 \edef#2{\@@fc@numstr\@tenstring{10}}%
5036 \else
5037 \let\@@fc@numstr#2\relax
5038 \edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
5039 \fi
5040 \fi
5041 \@strctr=#1\relax \@modulo{\@strctr}{100}%
5042 \ifnum#1>100\relax
5043 \ifnum\@strctr>0\relax
5044 \let\@@fc@numstr#2\relax
5045 \edef#2{\@@fc@numstr\ }%
5046 \fi
5047 \fi
5048 \ifnum\@strctr>29\relax
5049 \divide\@strctr by 10\relax
5050 \let\@@fc@numstr#2\relax
5051 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
5052 \@strctr=#1\relax \@modulo{\@strctr}{10}%
5053 \ifnum\@strctr>0\relax
5054 \let\@@fc@numstr#2\relax
5055 \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
5056 \fi
5057 \else
5058 \ifnum\@strctr<10\relax
5059 \ifnum\@strctr=0\relax

```

```

5060 \ifnum#1<100\relax
5061 \let\@@fc@numstr#2\relax
5062 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
5063 \fi
5064 \else
5065 \let\@@fc@numstr#2\relax
5066 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
5067 \fi
5068 \else
5069 \ifnum\@strctr>19\relax
5070 \@modulo{\@strctr}{10}%
5071 \let\@@fc@numstr#2\relax
5072 \edef#2{\@@fc@numstr\@twentystring{\@strctr}}%
5073 \else
5074 \@modulo{\@strctr}{10}%
5075 \let\@@fc@numstr#2\relax
5076 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
5077 \fi
5078 \fi
5079 \fi
5080 }

```

As above, but for ordinals

```

5081 \newcommand{\@@ordinalstringspanish}[2]{%
5082 \@strctr=#1\relax
5083 \ifnum#1>99999
5084 \PackageError{fmtcount}{Out of range}%
5085 {This macro only works for values less than 100000}%
5086 \else
5087 \ifnum#1<0
5088 \PackageError{fmtcount}{Negative numbers not permitted}%
5089 {This macro does not work for negative numbers, however
5090 you can try typing "minus" first, and then pass the modulus of
5091 this number}%
5092 \else
5093 \def#2{}%
5094 \ifnum\@strctr>999\relax
5095 \divide\@strctr by 1000\relax
5096 \ifnum\@strctr>1\relax
5097 \ifnum\@strctr>9\relax
5098 \@tmpstrctr=\@strctr
5099 \ifnum\@strctr<20
5100 \@modulo{\@tmpstrctr}{10}%
5101 \let\@@fc@ordstr#2\relax
5102 \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
5103 \else
5104 \divide\@tmpstrctr by 10\relax
5105 \let\@@fc@ordstr#2\relax
5106 \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
5107 \@tmpstrctr=\@strctr

```

```

5108         \@modulo{\@tmpstrctr}{10}%
5109         \ifnum\@tmpstrctr>0\relax
5110             \let\@fc@ordstr#2\relax
5111             \edef#2{\@fc@ordstr\@unitthstring{\@tmpstrctr}}%
5112         \fi
5113     \fi
5114 \else
5115     \let\@fc@ordstr#2\relax
5116     \edef#2{\@fc@ordstr\@unitstring{\@strctr}}%
5117 \fi
5118 \fi
5119 \let\@fc@ordstr#2\relax
5120 \edef#2{\@fc@ordstr\@thousandth}%
5121 \fi
5122 \@strctr=#1\relax
5123 \@modulo{\@strctr}{1000}%
5124 \ifnum\@strctr>99\relax
5125     \@tmpstrctr=\@strctr
5126     \divide\@tmpstrctr by 100\relax
5127     \ifnum#1>1000\relax
5128         \let\@fc@ordstr#2\relax
5129         \edef#2{\@fc@ordstr\ }%
5130     \fi
5131     \let\@fc@ordstr#2\relax
5132     \edef#2{\@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
5133 \fi
5134 \@modulo{\@strctr}{100}%
5135 \ifnum#1>99\relax
5136     \ifnum\@strctr>0\relax
5137         \let\@fc@ordstr#2\relax
5138         \edef#2{\@fc@ordstr\ }%
5139     \fi
5140 \fi
5141 \ifnum\@strctr>19\relax
5142     \@tmpstrctr=\@strctr
5143     \divide\@tmpstrctr by 10\relax
5144     \let\@fc@ordstr#2\relax
5145     \edef#2{\@fc@ordstr\@tenthstring{\@tmpstrctr}}%
5146     \@tmpstrctr=\@strctr
5147     \@modulo{\@tmpstrctr}{10}%
5148     \ifnum\@tmpstrctr>0\relax
5149         \let\@fc@ordstr#2\relax
5150         \edef#2{\@fc@ordstr\ \@unitthstring{\@tmpstrctr}}%
5151     \fi
5152 \else
5153     \ifnum\@strctr>9\relax
5154         \@modulo{\@strctr}{10}%
5155         \let\@fc@ordstr#2\relax
5156         \edef#2{\@fc@ordstr\@teenthstring{\@strctr}}%

```

```

5157 \else
5158 \ifnum\@strctr=0\relax
5159 \ifnum#1=0\relax
5160 \let\@@fc@ordstr#2\relax
5161 \edef#2{\@@fc@ordstr\@unitstring{0}}%
5162 \fi
5163 \else
5164 \let\@@fc@ordstr#2\relax
5165 \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
5166 \fi
5167 \fi
5168 \fi
5169 \fi
5170 \fi
5171 }

```

### 9.3.14 fc-UKenglish.def

English definitions

```
5172 \ProvidesFCLanguage{UKenglish}[2012/06/18]
```

Loaded fc-english.def if not already loaded

```
5173 \FCloadlang{english}
```

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

```

5174 \let\@ordinalMUKenglish\@ordinalMenglish
5175 \let\@ordinalFUKenglish\@ordinalMenglish
5176 \let\@ordinalNUKenglish\@ordinalMenglish
5177 \let\@numberstringMUKenglish\@numberstringMenglish
5178 \let\@numberstringFUKenglish\@numberstringMenglish
5179 \let\@numberstringNUKenglish\@numberstringMenglish
5180 \let\@NumberstringMUKenglish\@NumberstringMenglish
5181 \let\@NumberstringFUKenglish\@NumberstringMenglish
5182 \let\@NumberstringNUKenglish\@NumberstringMenglish
5183 \let\@ordinalstringMUKenglish\@ordinalstringMenglish
5184 \let\@ordinalstringFUKenglish\@ordinalstringMenglish
5185 \let\@ordinalstringNUKenglish\@ordinalstringMenglish
5186 \let\@OrdinalstringMUKenglish\@OrdinalstringMenglish
5187 \let\@OrdinalstringFUKenglish\@OrdinalstringMenglish
5188 \let\@OrdinalstringNUKenglish\@OrdinalstringMenglish

```

### 9.3.15 fc-USenglish.def

US English definitions

```
5189 \ProvidesFCLanguage{USenglish}[2012/06/18]
```

Loaded fc-english.def if not already loaded

```
5190 \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.)

```
5191 \let\@ordinalMUSenglish\@ordinalMenglish
5192 \let\@ordinalFUSenglish\@ordinalMenglish
5193 \let\@ordinalNUSenglish\@ordinalMenglish
5194 \let\@numberstringMUSenglish\@numberstringMenglish
5195 \let\@numberstringFUSenglish\@numberstringMenglish
5196 \let\@numberstringNUSenglish\@numberstringMenglish
5197 \let\@NumberstringMUSenglish\@NumberstringMenglish
5198 \let\@NumberstringFUSenglish\@NumberstringMenglish
5199 \let\@NumberstringNUSenglish\@NumberstringMenglish
5200 \let\@ordinalstringMUSenglish\@ordinalstringMenglish
5201 \let\@ordinalstringFUSenglish\@ordinalstringMenglish
5202 \let\@ordinalstringNUSenglish\@ordinalstringMenglish
5203 \let\@OrdinalstringMUSenglish\@OrdinalstringMenglish
5204 \let\@OrdinalstringFUSenglish\@OrdinalstringMenglish
5205 \let\@OrdinalstringNUSenglish\@OrdinalstringMenglish
```