

# 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)

2014-06-18 (version 2.04)

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Available Commands</b>	<b>2</b>
<b>3</b>	<b>Package Options</b>	<b>8</b>
<b>4</b>	<b>Multilingual Support</b>	<b>8</b>
4.1	Options for French . . . . .	9
4.2	Prefixes . . . . .	14
<b>5</b>	<b>Configuration File <code>fmtcount.cfg</code></b>	<b>14</b>
<b>6</b>	<b>LaTeX2HTML style</b>	<b>15</b>
<b>7</b>	<b>Acknowledgements</b>	<b>15</b>
<b>8</b>	<b>Troubleshooting</b>	<b>15</b>
<b>9</b>	<b>The Code</b>	<b>16</b>
9.1	<code>fcnumparser.sty</code> . . . . .	16
9.2	<code>fcprefix.sty</code> . . . . .	26
9.3	<code>fmtcount.sty</code> . . . . .	36
9.4	Multilanguage Definitions . . . . .	60
9.4.1	<code>fc-american.def</code> . . . . .	63
9.4.2	<code>fc-british.def</code> . . . . .	64
9.4.3	<code>fc-english.def</code> . . . . .	64
9.4.4	<code>fc-francais.def</code> . . . . .	75
9.4.5	<code>fc-french.def</code> . . . . .	75
9.4.6	<code>fc-frenchb.def</code> . . . . .	105
9.4.7	<code>fc-german.def</code> . . . . .	105

9.4.8	<code>fc-germanb.def</code>	115
9.4.9	<code>fc-italian</code>	116
9.4.10	<code>fc-ngerman.def</code>	117
9.4.11	<code>fc-ngermanb.def</code>	117
9.4.12	<code>fc-portuges.def</code>	118
9.4.13	<code>fc-portuguese.def</code>	133
9.4.14	<code>fc-spanish.def</code>	133
9.4.15	<code>fc-UKenglish.def</code>	151
9.4.16	<code>fc-USenglish.def</code>	151

## 1 Introduction

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

## 2 Available Commands

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

`\ordinal`

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

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

`\fmtord`

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

is used to format the st, nd, rd, th bit. By default the ordinal is formatted as a superscript, if the package option `level` is used, it is level with the text. For example, if the current section is 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>]`

---

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

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.

`\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}`.

<code>\storeordinal</code>	<code>\storeordinal{&lt;label&gt;}{&lt;counter&gt;}[&lt;gender&gt;]</code>
<code>\storeordinalstring</code>	<code>\storeordinalstring{&lt;label&gt;}{&lt;counter&gt;}[&lt;gender&gt;]</code>
<code>\storeOrdinalstring</code>	<code>\storeOrdinalstring{&lt;label&gt;}{&lt;counter&gt;}[&lt;gender&gt;]</code>
<code>\storeORDINALstring</code>	<code>\storeORDINALstring{&lt;label&gt;}{&lt;counter&gt;}[&lt;gender&gt;]</code>
<code>\storenumberstring</code>	<code>\storenumberstring{&lt;label&gt;}{&lt;counter&gt;}[&lt;gender&gt;]</code>
<code>\storeNumberstring</code>	<code>\storeNumberstring{&lt;label&gt;}{&lt;counter&gt;}[&lt;gender&gt;]</code>
<code>\storeNUMBERstring</code>	<code>\storeNUMBERstring{&lt;label&gt;}{&lt;counter&gt;}[&lt;gender&gt;]</code>
<code>\storeordinalnum</code>	<code>\storeordinalnum{&lt;label&gt;}{&lt;number&gt;}[&lt;gender&gt;]</code>
<code>\storeordinalstringnum</code>	<code>\storeordinalstring{&lt;label&gt;}{&lt;number&gt;}[&lt;gender&gt;]</code>
<code>\storeOrdinalstringnum</code>	<code>\storeOrdinalstringnum{&lt;label&gt;}{&lt;number&gt;}[&lt;gender&gt;]</code>
<code>\storeORDINALstringnum</code>	<code>\storeORDINALstringnum{&lt;label&gt;}{&lt;number&gt;}[&lt;gender&gt;]</code>
<code>\storenumberstringnum</code>	<code>\storenumberstring{&lt;label&gt;}{&lt;number&gt;}[&lt;gender&gt;]</code>

<code>\storeNumberstringnum</code>	<code>\storeNumberstring{&lt;label&gt;}{&lt;number&gt;}[&lt;gender&gt;]</code>
<code>\storeNUMBERstringnum</code>	<code>\storeNUMBERstring{&lt;label&gt;}{&lt;number&gt;}[&lt;gender&gt;]</code>
<code>\binary</code>	<code>\binary{&lt;counter&gt;}</code> <p>This will print the value of <code>&lt;counter&gt;</code> as a binary number. E.g. <code>\binary{section}</code> will produce: 11. The declaration</p>
<code>\padzeroes</code>	<code>\padzeroes[&lt;n&gt;]</code> <p>will ensure numbers are written to <code>&lt;n&gt;</code> digits, padding with zeroes if necessary. E.g. <code>\padzeroes[8]\binary{section}</code> will produce: 00000011. The default value for <code>&lt;n&gt;</code> is 17.</p>
<code>\binarynum</code>	<code>\binary{&lt;n&gt;}</code> <p>This is like <code>\binary</code> but takes an actual number rather than a counter as the argument. For example: <code>\binarynum{5}</code> will produce: 101. The octal commands only work for values in the range 0 to 32768.</p>
<code>\octal</code>	<code>\octal{&lt;counter&gt;}</code> <p>This will print the value of <code>&lt;counter&gt;</code> as an octal number. For example, if you have a counter called, say <code>mycounter</code>, and you set the value to 125, then <code>\octal{mycounter}</code> will produce: 177. Again, the number will be padded with zeroes if necessary, depending on whether <code>\padzeroes</code> has been used.</p>
<code>\octalnum</code>	<code>\octalnum{&lt;n&gt;}</code> <p>This is like <code>\octal</code> but takes an actual number rather than a counter as the argument. For example: <code>\octalnum{125}</code> will produce: 177.</p>
<code>\hexadecimal</code>	<code>\hexadecimal{&lt;counter&gt;}</code> <p>This will print the value of <code>&lt;counter&gt;</code> as a hexadecimal number. Going back to the counter used in the previous example, <code>\hexadecimal{mycounter}</code> will produce: 7d. Again, the number will be padded with zeroes if necessary, depending on whether <code>\padzeroes</code> has been used.</p>
<code>\Hexadecimal</code>	<code>\Hexadecimal{&lt;counter&gt;}</code>

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.

`\AAAAlph`

`\AAAAlph{<counter>}`

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

`\aaalphnum`

`\aaalphnum{<n>}`

`\AAAAlphnum`

`\AAAAlphnum{<n>}`

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

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

`\abalph`      `\abalph{<counter>}`

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

`\ABAlph`      `\ABAlph{<counter>}`

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

`\abalphnum`      `\abalphnum{<n>}`

`\ABAlphnum`      `\ABAlphnum{<n>}`

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

### 3 Package Options

The following options can be passed to this package:

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

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

These can also be set using the command:

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

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

### 4 Multilingual Support

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

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

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

---

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

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



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

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

If the French language is selected, the `french` option let you configure the dialect and other aspects. The `abbr` also has some influence with French. Please refer to § 4.1.

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

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

## 4.1 Options for French

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

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

`\fmtcountsetoptions`

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

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

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

`dialect`

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

`french`

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

Pour alléger la notation et par souci de rétro-compatibilité `france`, `belgian`

ou `swiss` sont également des *⟨clef⟩*s pour *⟨french options⟩* à utiliser sans *⟨valeur⟩*.

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

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

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

`swiss`    septante pour 70, huitante<sup>4</sup> pour 80, et nonante pour 90

Il est à noter que la variante `belgian` est parfaitement correcte pour les franco-phones français<sup>5</sup>, et qu'elle est également utilisée en Suisse Romande hormis dans les cantons de Vaud, du Valais et de Fribourg. En ce qui concerne le mot "octante", il n'est actuellement pas pris en charge et n'est guère plus utilisé, ce qui est sans doute dommage car il est sans doute plus acceptable que le "huitante" de certains de nos amis suisses.

`abbr`    `\fmtcountsetoptions{abbr=⟨boolean⟩}`

L'option générale `abbr` permet de changer l'effet de `\ordinal`. Selon *⟨boolean⟩* on a :

`true`    pour produire des ordinaux de la forme 2<sup>e</sup>, ou

`false`    pour produire des ordinaux de la forme 2<sup>eme</sup> (par défaut)

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

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

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

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

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

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

Les options `vingt plural`, `cent plural`, `mil plural`, `n-illion plural`, et `n-illiard plural`, permettent de contrôler très finement l'accord en nombre

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

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

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

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

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

Le paramètre  *$\langle french plural control \rangle$*  peut prendre les valeurs suivantes :

<code>traditional</code>	pour sélectionner la règle en usage chez les adultes à la date de parution de ce document, et dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur cardinale,
<code>reformed</code>	pour suivre toute nouvelle recommandation à la date de parution de ce document, , et dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur cardinale, l'idée des options <code>traditional</code> et <code>reformed</code> est donc de pouvoir contenter à la fois les anciens et les modernes, mais à dire vrai à la date où ce document est écrit elles ont exactement le même effet,

traditional o	pareil que traditional mais dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur ordinale,
reformed o	pareil que reformed mais dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur ordinale, de même que précédemment reformed o et traditional o ont exactement le même effet,
always	pour marquer toujours le pluriel, ceci n'est correct que pour « mil » vis à vis des règles en vigueur,
never	pour ne jamais marquer le pluriel, ceci est incorrect vis à vis des règles d'orthographe en vigueur,
multiple	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2, ceci est la règle en vigueur pour les nombres de la forme $\langle n \rangle$ illion et $\langle n \rangle$ illiard lorsque le nombre a une valeur cardinale,
multiple g-last	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 est est <i>globalement</i> en dernière position, où “globalement” signifie qu'on considère le nombre formaté en entier, ceci est incorrect vis à vis des règles d'orthographe en vigueur,
multiple l-last	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 et est <i>localement</i> en dernière position, où “localement” signifie qu'on considère seulement la portion du nombre qui multiplie soit l'unité, soit un $\langle n \rangle$ illion ou un $\langle n \rangle$ illiard ; ceci est la convention en vigueur pour le pluriel de “vingt” et de “cent” lorsque le nombre formaté a une valeur cardinale,
multiple lng-last	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 et est <i>localement</i> mais <i>non globalement</i> en dernière position, où “localement” et <i>globalement</i> ont 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'est pas 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=\langle n-illiard upto \rangle}}`

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

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

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

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

## 4.2 Prefixes

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

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

## 5 Configuration File `fmtcount.cfg`

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

package.

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

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

and if `fmtcount.cfg` has the line:

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

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

## 6 LaTeX2HTML style

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

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

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

This, I agree, is an unpleasant cludge.

## 7 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/>.

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

## 9 The Code

### 9.1 fcnumparser.sty

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

\fc@counter@parser is just a shorthand to parse a number held in a counter.
3 \def\fc@counter@parser#1{%
4   \expandafter\fc@number@parser\expandafter{\the#1.}%
5 }
6 \newcount\fc@digit@counter
7
8 \def\fc@end@{\fc@end}

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

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

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

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

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

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

\fc @number@parser Macro \fc@number@parser is the main engine to parse a
number. Argument '#1' is input and contains the number to be parsed. At end
```



of this macro, each digit is stored separately in a `\fc@digit@<n>`, and macros `\fc@min@weight` and `\fc@max@weight` are set to the bounds for `<n>`.

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

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

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

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

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

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

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

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

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

First we compute the weight of the most significant digit: after `\fc@parse@integer@digits`, `\fc@digit@counter` is equal to `\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  $\text{\fc@max@weight}$  in order to copy all the digits from `\fc@digit@<i + offset>` to `\fc@digit@<i>`. First we compute offset into `\@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 `\fc@fractional@part` into a sequence of `\fc@digit@<n>` with  $\langle n \rangle$  ranging from `\fc@unit@weight - 1` to `\fc@min@weight` by step of  $-1$ . This is much

more simpler because we get the digits with the final range of index, so no post-processing loop is needed.

```

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

```

```

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

```

Now we have macros to read a few digits from the `\fc@digit@<n>` array and form a corresponding number.

```

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

```

Arguments as follows:

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

```

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

```

```

132         \def\@tempb##1{\def\@tempa{#1==#1\relax}}%
133         \expandafter\@tempb\expandafter{\@tempa}%
134         \expandafter
135     }\@tempa
136 \fi
137 \fi
138 }

\fc @read@hundred Macro \fc@read@hundred is used to read a pair of digits and
form an integer in the range [0..99]. First we check that the macro is not yet
defined.
139 \ifcsundef{fc@read@hundred}{-}{%
140   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro ‘fc@read@hundred’}}

Arguments as follows — same interface as \fc@read@unit:
#1  output counter: into which the read value is placed
#2  input number: unit weight at which reach the value is to be read
141 \def\fc@read@hundred#1#2{%
142   {%
143     \fc@read@unit{\count0}{#2}%
144     \def\@tempa##1{\fc@read@unit{\count1}{##1}}%
145     \count2=#2%
146     \advance\count2 by 1 %
147     \expandafter\@tempa{\the\count2}%
148     \multiply\count1 by 10 %
149     \advance\count1 by \count0 %
150     \def\@tempa##1{\def\@tempb{#1==#1\relax}}
151     \expandafter\@tempa\expandafter{\the\count1}%
152     \expandafter
153   }\@tempb
154 }

\fc @read@thousand Macro \fc@read@thousand is used to read a trio of digits
and form an integer in the range [0..999]. First we check that the macro is not
yet defined.
155 \ifcsundef{fc@read@thousand}{-}{%
156   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
157     ‘fc@read@thousand’}}

Arguments as follows — same interface as \fc@read@unit:
#1  output counter: into which the read value is placed
#2  input number: unit weight at which reach the value is to be read
158 \def\fc@read@thousand#1#2{%
159   {%
160     \fc@read@unit{\count0}{#2}%
161     \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%
162     \count2=#2%
163     \advance\count2 by 1 %
164     \expandafter\@tempa{\the\count2}%
165     \multiply\count1 by 10 %
166     \advance\count1 by \count0 %

```

```

167 \def\@tempa##1{\def\@tempb{#1=##1\relax}}
168 \expandafter\@tempa\expandafter{\the\count1}%
169 \expandafter
170 }\@tempb
171 }

```

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

```

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

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

```

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

```

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

```

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

```

Arguments as follows:

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

Now propagate result after closing bracket into counter #1.

```

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

```

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

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

```

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

```

First we split #1 into two parts: everything that is upto \fc@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 \ifcsundef\fc@get@last@letter\{}{\PackageError{fcnumparser}{Duplicate definition}{Redefinit.
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 is to bubble one by one letters from \toks1 which initial contains the whole word, into \toks0. At the end of the macro \toks0 will therefore contain the whole word but the last letter, and the last letter will be in \toks1.

```

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

```

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

```

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

```

```

323     \def\@tempb{##2}%
324 }%

Now we define \@templ to do the loop by terminal recursion.
325 \def\@templ{%
326     \expandafter\@tempa\the\toks1 \fc@nil
327     \ifx\@tempb\@empty

Stop loop, as \toks1 has been detected to be one single letter.
328     \let\next\relax
329     \else

Here we append to \toks0 the content of \toks2, i.e. the next letter.
330     \expandafter\expandafter\expandafter\@tempt
331     \expandafter\expandafter\expandafter{%
332         \expandafter\the\expandafter\@tempt
333         \the\toks2}%

And the remaining letters go to \toks1 for the next iteration.
334     \toks1\toks3 %
335     \fi
336     \next
337 }%

Here run the loop.
338 \let\next\@templ
339 \next

Now propagate the results into macros #2 and #3 after closing brace.
340 \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks1}}%
341 \expandafter
342 }\@tempa
343 }%

```

## 9.2 fcprefix.sty

Pseudo-latin prefixes.

```

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

Option ‘use duode and unde’ is to select whether 18 and suchlikes (<x>8, <x>9)
writes like duodevicies, or like octodecies. For French it should be ‘below 20’.
Possible values are ‘below 20’ and ‘never’.
349 \define@key{fcprefix}{use duode and unde}[below20]{%
350     \ifthenelse{\equal{#1}{below20}}{%
351         \def\fc@duodeandunde{2}%
352     }{%
353         \ifthenelse{\equal{#1}{never}}{%
354             \def\fc@duodeandunde{0}%
355         }{%
356             \PackageError{fcprefix}{Unexpected option}{%

```

```

357         Option 'use duode and unde' expects 'below 20' or 'never' }%
358     }%
359 }%
360 }

```

Default is 'below 20' like in French.

```

361 \def\fc@duodeandunde{2}

```

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

```

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

```

Option 'e accute', this can be 'true' or 'false' and is used to select whether letter 'e' has an accute accent when it pronounce [e] in French.

```

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

```

Default is to set accute accent like in French.

```

386 \let\fc@prefix@eaccute\'%

```

Option 'power of millia' tells how millia is raise to power n. It expects value: recursive for which millia squared is noted as 'milliamillia'

arabic for which millia squared is noted as 'millia^2'

prefix for which millia squared is noted as 'bismillia'

```

387 \define@key{fcprefix}{power of millia}[prefix]{%
388     \ifthenelse{\equal{#1}{prefix}}{%
389         \let\fc@power@of@millia@init\@gobbletwo
390         \let\fc@power@of@millia\fc@@prefix@millia
391     }{%
392         \ifthenelse{\equal{#1}{arabic}}{%

```

```

393     \let\fc@power@of@millia@init@gobbletwo
394     \let\fc@power@of@millia\fc@@arabic@millia
395   }{%
396     \ifthenelse{\equal{#1}{recursive}}{%
397       \let\fc@power@of@millia@init\fc@@recurse@millia@init
398       \let\fc@power@of@millia\fc@@recurse@millia
399     }{%
400       \PackageError{fcprefix}{Unexpected option}{%
401         Option ‘power of millia’ expects ‘recursive’, ‘arabic’, or ‘prefix’ }%
402     }%
403   }%
404 }%
405 }

```

Arguments as follows:

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

```

406 \def\fc@@recurse@millia#1#2{%
407   \let\@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 \ifcsundef\fc@@latin@cardinal@pexif\{}{%
438 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro ‘fc@@latin@cardinal@p

```

Arguments as follows:

```

#1 input number to be formatted
#2 outut macro name into which to place the formatted result
439 \def\fc@@latin@cardinal@pexif#1#2{%
440 {%

```

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

```

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

```

Now parse number from expanded input.

```

442 \expandafter\fc@number@parser\expandafter{\@tempa}%
443 \count2=0 %

```

`\@tempt` will hold the optional final t, `\@tempu` is used to initialize `\@tempt` to ‘t’ when the first non-zero 3digit group is met, which is the job made by `\@tempti`.

```

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

```

`\@tempm` will hold the  $\text{millia}^{n \div 3}$

```

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

```

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

```

```

491          \or oc\@tempt
492          \or non%
493          \fi
494      \else

```

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

```

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

```

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

```

532      \@tempm

```

And append previous version of \@tempa.

```

533      \@temp
534    }%

```

“Concatenate” millia to \@temp, so that \@temp will expand to  $\text{millia}^{(n+3)+1}$  at the next iteration. Actually whether this is a concatenation or some millia prefixing depends of option ‘power of millia’.

```

535      \fc@power@of@millia\@temp{\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 \ifcsundef{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 \@temp will hold the millies<sup>n+3</sup>.

```

556   \let\@tempm\@empty

```

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

```

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

```

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



```

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

```

Check the use of duodevicies instead of octodecies.

```

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

```

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

```

599      \ifcase\count3 %
600      \or
601          \ifnum\count2=3 %
602              s\fc@prefix@eacute emel%
603          \else
604              \ifnum\count2>\fc@max@weight\else un\fi
605          \fi
606      \or bis%

```

```

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

```

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

```

646      \let\@temp\@temp
647      \edef\@tempm{millies\@temp}%
648      \fi
649      \next
650  }%

```

```

651 \let\@tempa\@empty
652 \let\next\@templ
653 \@templ

```

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

```

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

```

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

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

```

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

673 \def\fc@call@opt@arg@first#1{%
674 \def\@tempb{%
675 \ifx[\@tempa
676 \def\@tempc[####1]####2{#{#1{####1}{####2}}}%
677 \else
678 \def\@tempc####1{#{#1}{####1}}}%

```

```

679 \fi
680 \@tempc
681 }%
682 \futurelet\@tempa
683 \@tempb
684 }
685
686 \let\fc@call\fc@call@opt@arg@first

```

User API.

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

- #1 local options
- #2 input number

```

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

```

Arguments as follows:

- #1 local options
- #2 input counter

```

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

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

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

```

### 9.3 fmtcount.sty

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

```

705 \NeedsTeXFormat{LaTeX2e}
706 \ProvidesPackage{fmtcount}[2014/06/18 v2.04]
707 \RequirePackage{ifthen}
708 \RequirePackage{keyval}
709 \RequirePackage{etoolbox}
710 \RequirePackage{fcprefix}

```

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

711 \RequirePackage{amsgen}

These commands need to be defined before the configuration file is loaded.  
Define the macro to format the st, nd, rd or th of an ordinal.

\fmtord

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

\padzeroes

\padzeroes[ $\langle n \rangle$ ]

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

713 \newcount\c@padzeroesN

714 \c@padzeroesN=1\relax

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

\FCloadlang

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

\FCloadlang{ $\langle language \rangle$ }

Load fmtcount language file, fc- $\langle language \rangle$ .def, unless already loaded. Unfortunately neither babel nor polyglossia keep a list of loaded dialects, so we can't load all the necessary def files in the preamble as we don't know which dialects the user requires. Therefore the dialect definitions get loaded when a command such as \ordinalnum is used, if they haven't already been loaded.

716 \newcount\fc@tmpcatcode

717 \def\fc@languages{}

718 \def\fc@mainlang{}

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

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

721 {%

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

723 \catcode '\@ 11\relax

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

725 {%

726 \ifdefempty{\fc@languages}%

727 {%

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

729 }%

730 {%

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

732 }%

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

734 }%

735 }%

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

```
737 }%
738 }
```

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

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

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

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

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

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

babelorpolyglossialdf \@fc@loadigbabe1ldf{<language>}

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

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

Load appropriate language definition files:

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

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

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

french

```
765 \define@key{fmtcount}{french}[france]{%
766   \@FC@iflangloaded{french}%
767   {%
768     \setkeys{french}{#1}%
769   }%
770   {%
771     \PackageError{fmtcount}%
772     {Language ‘french’ not defined}%
773     {You need to load babel before loading fmtcount}%
774   }%
775 }
```

fmtord Key to determine how to display the ordinal

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

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

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

prefix

```

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

```

\fmtcountsetoptions    Define command to set options.

```

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

```

Load configuration file if it exists. This needs to be done before the package options, to allow the user to override the settings in the configuration file.

```

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

```



```
848 }
```

level

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

raise

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

Process package options

```
853 \ProcessOptions
```

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

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

```
854 \newcount\@DT@modctr
855 \newcommand*{\@FCmodulo}[2]{%
856   \@DT@modctr=#1\relax
857   \divide \@DT@modctr by #2\relax
858   \multiply \@DT@modctr by #2\relax
859   \advance #1 by -\@DT@modctr
860 }
```

The following registers are needed by \@ordinal etc

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

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

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

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

```
868 \newcommand*{\@binary}[1]{%
869   \@DT@padzeroes=true
870   \@DT@loopN=17\relax
871   \@strctr=\@DT@loopN
872   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
873   \@strctr=65536\relax
874   \@DT@X=#1\relax
875   \loop
```

```

876 \@DT@modctr=\@DT@X
877 \divide\@DT@modctr by \@strctr
878 \ifthenelse{\boolean{@DT@padzeroes}
879 \and \(\@DT@modctr=0\}
880 \and \(\@DT@loopN>\c@padzeroesN\)}%
881 {}%
882 {\the\@DT@modctr}%
883 \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
884 \multiply\@DT@modctr by \@strctr
885 \advance\@DT@X by -\@DT@modctr
886 \divide\@strctr by 2\relax
887 \advance\@DT@loopN by -1\relax
888 \ifnum\@strctr>1
889 \repeat
890 \the\@DT@X
891 }
892
893 \let\binarynum=\@binary

```

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

```

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

```

```
923 \let\octalnum=\@octal
```

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

```
924 \newcommand*\@hexadecimal}[1]{%
925   \ifcase#1\or1\or2\or3\or4\or5\or
926   6\or7\or8\or9\or a\or b\or c\or d\or e\or f\fi
927 }
```

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

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

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

```
952 \newcommand*\@Hexadecimal}[1]{%
953   \ifcase#1\or1\or2\or3\or4\or5\or6\or
954   7\or8\or9\or A\or B\or C\or D\or E\or F\fi
955 }
```

**\Hexadecimalnum** Uppercase hexadecimal

```
956 \newcommand*\@Hexadecimal}[1]{%
957   \@DT@padzeroestrue
958   \@DT@loopN=5\relax
959   \@strctr=\@DT@loopN
960   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
961   \@strctr=65536\relax
962   \@DT@X=#1\relax
```

```

963 \loop
964 \@DT@modctr=\@DT@X
965 \divide\@DT@modctr by \@strctr
966 \ifthenelse{\boolean{\@DT@padzeroes}
967 \and \(\@DT@modctr=0\}
968 \and \(\@DT@loopN>\c@padzeroesN\)}%
969 {\{\@Hexadecimal\@DT@modctr}%
970 \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
971 \multiply\@DT@modctr by \@strctr
972 \advance\@DT@X by -\@DT@modctr
973 \divide\@strctr by 16\relax
974 \advance\@DT@loopN by -1\relax
975 \ifnum\@strctr>1
976 \repeat
977 \@Hexadecimal\@DT@X
978 }
979
980 \let\Hexadecimalnum=\@Hexadecimal

```

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

```

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

```

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

```

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

```

```

1008 \advance\@DT@X by -\@DT@modctr
1009 \advance\@DT@loopN by 1\relax
1010 \advance\@DT@X by 1\relax
1011 \loop
1012   \@Alph\@DT@X
1013   \advance\@DT@loopN by -1\relax
1014   \ifnum\@DT@loopN>0
1015   \repeat
1016 }
1017
1018 \let\AAAAlphnum=\@AAAAlph

```

#### \abalphnum Lowercase alphabetical representation

```

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

```

#### \ABAlphnum Uppercase alphabetical representation

```

1047 \newcommand*\@ABAlph}[1]{%
1048   \ifnum#1>17576\relax
1049     \PackageError{fmtcount}%
1050     {Value of counter too large for \protect\@ABAlph}%
1051     {Maximum value 17576}%
1052   \else

```

```

1053 \DT@padzeroestru
1054 \@strctr=17576\relax
1055 \DT@X=#1\relax
1056 \advance\DT@X by -1\relax
1057 \loop
1058 \DT@modctr=\DT@X
1059 \divide\DT@modctr by \@strctr
1060 \ifthenelse{\boolean{DT@padzeroes}\and
1061 \(\DT@modctr=1\)}{\@Alph\DT@modctr}%
1062 \ifnum\DT@modctr=1\else\DT@padzeroesfalse\fi
1063 \multiply\DT@modctr by \@strctr
1064 \advance\DT@X by -\DT@modctr
1065 \divide\@strctr by 26\relax
1066 \ifnum\@strctr>1
1067 \repeat
1068 \advance\DT@X by 1\relax
1069 \@Alph\DT@X
1070 \fi
1071 }
1072
1073 \let\ABAlphnum=\@ABAlph

```

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

```

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

```

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

```

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

```

`\FCordinal` `\FCordinal{<number>}`

This is a bit cumbersome. Previously `\@ordinal` was defined in a similar way to `\abalph` etc. This ensured that the actual value of the counter was written in the new label stuff in the .aux file. However adding in an optional argument to determine the gender for multilingual compatibility messed things up somewhat. This was the only work around I could get to keep the the cross-referencing stuff working, which is why the optional argument comes *after* the compulsory argument, instead of the usual manner of placing it before. Note however, that putting the optional argument means that any spaces will be ignored after the command if the optional argument is omitted. Version 1.04 changed `\ordinal` to `\FCordinal` to prevent it clashing with the memoir class.

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

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

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

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

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

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

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

```

---

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

```

1120 }%
1121 {%
1122 \ifthenelse{\equal{#2}{m}}%
1123 {}%
1124 {%
1125 \PackageError{fmtcount}%
1126 {Invalid gender option '#2'}%
1127 {Available options are m, f or n}%
1128 }%
1129 \protect\@ordinalM{#1}{\@fc@ordstr}%
1130 }%
1131 }%
1132 \@fc@ordstr
1133 }%
1134 }

```

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

```

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

```

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

```

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

```

`\@storeordinalnum` Store ordinal according to gender:

```

1144 \def\@storeordinalnum#1#2[#3]{%
1145 \ifthenelse{\equal{#3}{f}}%
1146 {%
1147 \protect\@ordinalF{#2}{\@fc@ord}
1148 }%
1149 {%
1150 \ifthenelse{\equal{#3}{n}}%
1151 {%
1152 \protect\@ordinalN{#2}{\@fc@ord}%
1153 }%
1154 {%
1155 \ifthenelse{\equal{#3}{m}}%
1156 {}%
1157 {%
1158 \PackageError{fmtcount}%
1159 {Invalid gender option '#3'}%
1160 {Available options are m or f}%

```



```

1161     }%
1162     \protect\@ordinalM{#2}{\@fc@ord}%
1163   }%
1164 }%
1165 \expandafter\let\csname @fcs@#1\endcsname\@fc@ord
1166 }

```

\FMCuse Get stored information:

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```
1202 \newcommand*{\storeordinalstring}[2]{%
1203   \expandafter\protect\expandafter\storeordinalstringnum{#1}{%
1204     \expandafter\the\csname c@#2\endcsname}%
1205 }
```

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

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

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

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

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

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

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

```

1238 \newcommand*{\Ordinalstringnum}[1]{%
1239   \new@ifnextchar[%
1240     {\@Ordinal@string{#1}}}%
1241     {\@Ordinal@string{#1}[m]}}%
1242 }

```

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

```

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

```

`\storeOrdinalstring` Store textual representation of number, with initial letters in upper case. First argument is identifying name, second argument is the counter set to the required number.

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

1316   {%
1317   \protect\@ordinalstringN{#2}{\@fc@ordstr}%
1318   }%
1319   {%
1320   \ifthenelse{\equal{#3}{m}}%
1321   {}%
1322   {%
1323   \PackageError{fmtcount}%
1324   {Invalid gender option ‘#3’}%
1325   {Available options are m or f}%
1326   }%
1327   \protect\@ordinalstringM{#2}{\@fc@ordstr}%
1328   }%
1329   }%
1330   \expandafter\edef\csname @fcs@#1\endcsname{%
1331   \noexpand\MakeUppercase{\@fc@ordstr}%
1332   }%
1333 }

```

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

```

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

```

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

```

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

```

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

```

1344 \def\@ORDINAL@string#1[#2]{%
1345   {%
1346   \ifthenelse{\equal{#2}{f}}%
1347   {%
1348   \protect\@ordinalstringF{#1}{\@fc@ordstr}%
1349   }%
1350   {%
1351   \ifthenelse{\equal{#2}{n}}%
1352   {%
1353   \protect\@ordinalstringN{#1}{\@fc@ordstr}%
1354   }%
1355   {%
1356   \ifthenelse{\equal{#2}{m}}%
1357   {}%

```

```

1358      {%
1359      \PackageError{fmtcount}%
1360      {Invalid gender option ‘#2’}%
1361      {Available options are m, f or n}%
1362      }%
1363      \protect\ordinalstringM{#1}{\@fc@ordstr}%
1364      }%
1365      }%
1366      \MakeUppercase{\@fc@ordstr}%
1367      }%
1368 }

```

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

```

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

```

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

```

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

```

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

```

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

```

1400 }

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

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

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

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

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

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

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

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

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

```

1439 \newcommand{\storeNumberstringnum}[2]{%
1440   \@ifnextchar[%
1441     {\@store@Number@string{#1}{#2}}%
1442     {\@store@Number@string{#1}{#2}[m]}}%
1443 }
```

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

```

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

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

```

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

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

```

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

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

```

1476 \def\@Number@string#1[#2]{%
1477   {%
1478     \ifthenelse{\equal{#2}{f}}%
```



```

1479    {%
1480      \protect\@NumberstringF{#1}{\@fc@numstr}%
1481    }%
1482    {%
1483      \ifthenelse{\equal{#2}{n}}%
1484      {%
1485        \protect\@NumberstringN{#1}{\@fc@numstr}%
1486      }%
1487      {%
1488        \ifthenelse{\equal{#2}{m}}%
1489        {}%
1490        {%
1491          \PackageError{fmtcount}%
1492            {Invalid gender option ‘#2’}%
1493            {Available options are m, f or n}%
1494        }%
1495        \protect\@NumberstringM{#1}{\@fc@numstr}%
1496      }%
1497    }%
1498    \@fc@numstr
1499  }%
1500 }

```

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

```

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

```

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

```

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

```

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

```

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

```

```

1521     \ifthenelse{\equal{#3}{m}}%
1522     {}%
1523     {%
1524         \PackageError{fmtcount}%
1525         {Invalid gender option ‘#3’}%
1526         {Available options are m or f}%
1527     }%
1528     \protect\@numberstringM{#2}{\@fc@numstr}%
1529 }%
1530 }%
1531 \expandafter\edef\csname @fcs@#1\endcsname{%
1532     \noexpand\MakeUppercase{\@fc@numstr}%
1533 }%
1534 }

```

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

```

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

```

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

```

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

```

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

```

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

```

```

1563      \protect\@numberstringM{#1}{\@fc@numstr}%
1564    }%
1565  }%
1566  \MakeUppercase{\@fc@numstr}%
1567 }%
1568 }

```

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

```

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

```

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

```

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

```

`\AAAalph` As before, but upper case.

```

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

```

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

```

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

```

`\ABAlph` As above, but upper case.

```

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

```

`\hexadecimal` Hexadecimal:

```

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

```

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

```

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

```

\octal Octal:

```
1597 \providecommand*{\octal}[1]{%
1598   \expandafter\protect\expandafter\@octal{%
1599     \expandafter\the\csname c@#1\endcsname}%
1600 }
```

\decimal Decimal:

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

## 9.4 Multilingual Definitions

@setdef@ultfmtcount If multilingual support is provided, make \@numberstring etc use the correct language (if defined). Otherwise use English definitions. "setdef@ultfmtcount" sets the macros to use English.

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

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

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

```

1634     language '\language' }%
1635     \ifthenelse{\equal{\language}{\fc@mainlang}}{%
1636     {%
1637         \FCloadlang{english}%
1638     }%
1639     {%
1640     }%
1641     \ifcsdef{@#1#2\fc@mainlang}%
1642     {%
1643         \csuse{@#1#2\fc@mainlang}%
1644     }%
1645     {%
1646         \PackageWarningNoLine{fmtcount}%
1647         {No languages loaded at all! Loading english definitions}%
1648         \FCloadlang{english}%
1649         \def\fc@mainlang{english}%
1650         \csuse{@#1#2english}%
1651     }%
1652 }%
1653 {%
1654     \csuse{@#1#2\language}%
1655 }%
1656 }

```

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

```

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

```

The neuter version of \Numberstring:

```
1673 \def\@NumberstringN{%  
1674     \fc@multiling{Numberstring}{N}%  
1675 }%
```

The masculine version of \ordinal:

```
1676 \def\@ordinalM{%  
1677     \fc@multiling{ordinal}{M}%  
1678 }%
```

The feminine version of \ordinal:

```
1679 \def\@ordinalF{%  
1680     \fc@multiling{ordinal}{F}%  
1681 }%
```

The neuter version of \ordinal:

```
1682 \def\@ordinalN{%  
1683     \fc@multiling{ordinal}{N}%  
1684 }%
```

The masculine version of \ordinalstring:

```
1685 \def\@ordinalstringM{%  
1686     \fc@multiling{ordinalstring}{M}%  
1687 }%
```

The feminine version of \ordinalstring:

```
1688 \def\@ordinalstringF{%  
1689     \fc@multiling{ordinalstring}{F}%  
1690 }%
```

The neuter version of \ordinalstring:

```
1691 \def\@ordinalstringN{%  
1692     \fc@multiling{ordinalstring}{N}%  
1693 }%
```

The masculine version of \Ordinalstring:

```
1694 \def\@OrdinalstringM{%  
1695     \fc@multiling{Ordinalstring}{M}%  
1696 }%
```

The feminine version of \Ordinalstring:

```
1697 \def\@OrdinalstringF{%  
1698     \fc@multiling{Ordinalstring}{F}%  
1699 }%
```

The neuter version of \Ordinalstring:

```
1700 \def\@OrdinalstringN{%  
1701     \fc@multiling{Ordinalstring}{N}%  
1702 }%  
1703 }
```

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

```
1704 \@ifpackageloaded{babel}%  
1705 {%  
1706   \@set@multiling@fmtcount  
1707 }%  
1708 {%  
1709   \@ifpackageloaded{ngerman}%  
1710   {%  
1711     \FCloadlang{ngerman}%  
1712     \@set@multiling@fmtcount  
1713   }%  
1714   {%  
1715     \@setdef@ultfmtcount  
1716   }%  
1717 }
```

Backwards compatibility:

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

#### 9.4.1 fc-american.def

American English definitions

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

Loaded fc-USenglish.def if not already loaded

```
1724 \FCloadlang{USenglish}%
```

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

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

### 9.4.2 fc-british.def

British definitions

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

Load fc-english.def, if not already loaded

```
1741 \FCloadlang{english}%
```

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

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

### 9.4.3 fc-english.def

English definitions

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

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

```
1758 \newcommand*\@ordinalMenglish[2]{%
1759 \def\@fc@ord{}%
1760 \@orgargctr=#1\relax
1761 \@ordinalctr=#1%
1762 \@FCmodulo{\@ordinalctr}{100}%
1763 \ifnum\@ordinalctr=11\relax
1764   \def\@fc@ord{th}%
1765 \else
1766   \ifnum\@ordinalctr=12\relax
1767     \def\@fc@ord{th}%
1768   \else
1769     \ifnum\@ordinalctr=13\relax
1770       \def\@fc@ord{th}%
1771     \else
1772       \@FCmodulo{\@ordinalctr}{10}%
1773       \ifcase\@ordinalctr
1774         \def\@fc@ord{th}%           case 0
```



```

1775      \or \def\@fc@ord{st}%   case 1
1776      \or \def\@fc@ord{nd}%   case 2
1777      \or \def\@fc@ord{rd}%   case 3
1778      \else
1779      \def\@fc@ord{th}%        default case
1780      \fi
1781  \fi
1782  \fi
1783 \fi
1784 \edef#2{\number#1\relax\noexpand\fmtord{\@fc@ord}}%
1785 }%
1786 \global\let\@ordinalMenglish\@ordinalMenglish

```

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

```

1787 \global\let\@ordinalFenglish=\@ordinalMenglish
1788 \global\let\@ordinalNenglish=\@ordinalMenglish

```

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

```

1789 \newcommand*\@@unitstringenglish[1]{%
1790   \ifcase#1\relax
1791     zero%
1792     \or one%
1793     \or two%
1794     \or three%
1795     \or four%
1796     \or five%
1797     \or six%
1798     \or seven%
1799     \or eight%
1800     \or nine%
1801 \fi
1802 }%
1803 \global\let\@@unitstringenglish\@@unitstringenglish

```

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

```

1804 \newcommand*\@@tenstringenglish[1]{%
1805   \ifcase#1\relax
1806     \or ten%
1807     \or twenty%
1808     \or thirty%
1809     \or forty%
1810     \or fifty%
1811     \or sixty%
1812     \or seventy%
1813     \or eighty%
1814     \or ninety%
1815 \fi

```

```

1816 }%
1817 \global\let\@@tenstringenglish\@@tenstringenglish

```

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

```

1818 \newcommand*\@@teenstringenglish[1]{%
1819   \ifcase#1\relax
1820     ten%
1821     \or eleven%
1822     \or twelve%
1823     \or thirteen%
1824     \or fourteen%
1825     \or fifteen%
1826     \or sixteen%
1827     \or seventeen%
1828     \or eighteen%
1829     \or nineteen%
1830   \fi
1831 }%
1832 \global\let\@@teenstringenglish\@@teenstringenglish

```

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

```

1833 \newcommand*\@@Unitstringenglish[1]{%
1834   \ifcase#1\relax
1835     Zero%
1836     \or One%
1837     \or Two%
1838     \or Three%
1839     \or Four%
1840     \or Five%
1841     \or Six%
1842     \or Seven%
1843     \or Eight%
1844     \or Nine%
1845   \fi
1846 }%
1847 \global\let\@@Unitstringenglish\@@Unitstringenglish

```

The tens:

```

1848 \newcommand*\@@Tenstringenglish[1]{%
1849   \ifcase#1\relax
1850     \or Ten%
1851     \or Twenty%
1852     \or Thirty%
1853     \or Forty%
1854     \or Fifty%
1855     \or Sixty%
1856     \or Seventy%
1857     \or Eighty%
1858     \or Ninety%
1859   \fi
1860 }%

```

```
1861 \global\let\@@Tenstringenglish\@@Tenstringenglish
```

The teens:

```
1862 \newcommand*\@@Teenstringenglish[1]{%
```

```
1863   \ifcase#1\relax
```

```
1864     Ten%
```

```
1865     \or Eleven%
```

```
1866     \or Twelve%
```

```
1867     \or Thirteen%
```

```
1868     \or Fourteen%
```

```
1869     \or Fifteen%
```

```
1870     \or Sixteen%
```

```
1871     \or Seventeen%
```

```
1872     \or Eighteen%
```

```
1873     \or Nineteen%
```

```
1874   \fi
```

```
1875 }%
```

```
1876 \global\let\@@Teenstringenglish\@@Teenstringenglish
```

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

```
1877 \newcommand*\@@numberstringenglish[2]{%
```

```
1878   \ifnum#1>99999
```

```
1879   \PackageError{fmtcount}{Out of range}%
```

```
1880 {This macro only works for values less than 100000}%
```

```
1881 \else
```

```
1882   \ifnum#1<0
```

```
1883   \PackageError{fmtcount}{Negative numbers not permitted}%
```

```
1884 {This macro does not work for negative numbers, however
```

```
1885 you can try typing "minus" first, and then pass the modulus of
```

```
1886 this number}%
```

```
1887 \fi
```

```
1888 \fi
```

```
1889 \def#2{}%
```

```
1890 \@strctr=#1\relax \divide\@strctr by 1000\relax
```

```
1891 \ifnum\@strctr>9
```

```
1892   \divide\@strctr by 10
```

```
1893   \ifnum\@strctr>1\relax
```

```
1894     \let\@@fc@numstr#2\relax
```

```
1895     \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
```

```
1896     \@strctr=#1 \divide\@strctr by 1000\relax
```

```
1897     \@FCmodulo{\@strctr}{10}%
```

```
1898     \ifnum\@strctr>0\relax
```

```
1899       \let\@@fc@numstr#2\relax
```

```
1900       \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
```

```
1901     \fi
```

```
1902   \else
```

```
1903     \@strctr=#1\relax
```

```

1904    \divide\@strctr by 1000\relax
1905    \@FCmodulo{\@strctr}{10}%
1906    \let\@@fc@numstr#2\relax
1907    \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
1908    \fi
1909    \let\@@fc@numstr#2\relax
1910    \edef#2{\@@fc@numstr\ \@thousand}%
1911 \else
1912    \ifnum\@strctr>0\relax
1913        \let\@@fc@numstr#2\relax
1914        \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@thousand}%
1915    \fi
1916 \fi
1917 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
1918 \divide\@strctr by 100
1919 \ifnum\@strctr>0\relax
1920     \ifnum#1>1000\relax
1921         \let\@@fc@numstr#2\relax
1922         \edef#2{\@@fc@numstr\ }%
1923     \fi
1924     \let\@@fc@numstr#2\relax
1925     \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@hundred}%
1926 \fi
1927 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
1928 \ifnum#1>100\relax
1929     \ifnum\@strctr>0\relax
1930         \let\@@fc@numstr#2\relax
1931         \edef#2{\@@fc@numstr\ \@andname\ }%
1932     \fi
1933 \fi
1934 \ifnum\@strctr>19\relax
1935     \divide\@strctr by 10\relax
1936     \let\@@fc@numstr#2\relax
1937     \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
1938     \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
1939     \ifnum\@strctr>0\relax
1940         \let\@@fc@numstr#2\relax
1941         \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
1942     \fi
1943 \else
1944     \ifnum\@strctr<10\relax
1945         \ifnum\@strctr=0\relax
1946             \ifnum#1<100\relax
1947                 \let\@@fc@numstr#2\relax
1948                 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
1949             \fi
1950         \else
1951             \let\@@fc@numstr#2\relax
1952             \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%

```

```

1953 \fi
1954 \else
1955 \FCmodulo{\@strctr}{10}%
1956 \let\@fc@numstr#2\relax
1957 \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
1958 \fi
1959 \fi
1960 }%
1961 \global\let\@numberstringenglish\@numberstringenglish

```

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

```

1962 \DeclareRobustCommand{\@numberstringMenglish}[2]{%
1963 \let\@unitstring=\@unitstringenglish
1964 \let\@teenstring=\@teenstringenglish
1965 \let\@tenstring=\@tenstringenglish
1966 \def\@hundred{hundred}\def\@thousand{thousand}%
1967 \def\@andname{and}%
1968 \@numberstringenglish{#1}{#2}%
1969 }%
1970 \global\let\@numberstringMenglish\@numberstringMenglish

```

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

```

1971 \global\let\@numberstringFenglish=\@numberstringMenglish
1972 \global\let\@numberstringNenglish=\@numberstringMenglish

```

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

```

1973 \newcommand*\@NumberstringMenglish[2]{%
1974 \let\@unitstring=\@Unitstringenglish
1975 \let\@teenstring=\@Teenstringenglish
1976 \let\@tenstring=\@Tenstringenglish
1977 \def\@hundred{Hundred}\def\@thousand{Thousand}%
1978 \def\@andname{and}%
1979 \@numberstringenglish{#1}{#2}%
1980 }%
1981 \global\let\@NumberstringMenglish\@NumberstringMenglish

```

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

```

1982 \global\let\@NumberstringFenglish=\@NumberstringMenglish
1983 \global\let\@NumberstringNenglish=\@NumberstringMenglish

```

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

```

1984 \newcommand*\@unitthstringenglish[1]{%
1985 \ifcase#1\relax
1986 zeroth%
1987 \or first%
1988 \or second%
1989 \or third%

```

```

1990    \or fourth%
1991    \or fifth%
1992    \or sixth%
1993    \or seventh%
1994    \or eighth%
1995    \or ninth%
1996    \fi
1997 }%
1998 \global\let\@@unitthstringenglish\@@unitthstringenglish

```

Next the tens:

```

1999 \newcommand*\@@tenthstringenglish[1]{%
2000   \ifcase#1\relax
2001     \or tenth%
2002     \or twentieth%
2003     \or thirtieth%
2004     \or fortieth%
2005     \or fiftieth%
2006     \or sixtieth%
2007     \or seventieth%
2008     \or eightieth%
2009     \or ninetieth%
2010   \fi
2011 }%
2012 \global\let\@@tenthstringenglish\@@tenthstringenglish

```

The teens:

```

2013 \newcommand*\@@teenthstringenglish[1]{%
2014   \ifcase#1\relax
2015     tenth%
2016     \or eleventh%
2017     \or twelfth%
2018     \or thirteenth%
2019     \or fourteenth%
2020     \or fifteenth%
2021     \or sixteenth%
2022     \or seventeenth%
2023     \or eighteenth%
2024     \or nineteenth%
2025   \fi
2026 }%
2027 \global\let\@@teenthstringenglish\@@teenthstringenglish

```

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

```

2028 \newcommand*\@@Unitthstringenglish[1]{%
2029   \ifcase#1\relax
2030     Zeroth%
2031     \or First%
2032     \or Second%
2033     \or Third%
2034     \or Fourth%

```

```

2035 \or Fifth%
2036 \or Sixth%
2037 \or Seventh%
2038 \or Eighth%
2039 \or Ninth%
2040 \fi
2041 }%
2042 \global\let\@@Unitthstringenglish\@@Unitthstringenglish

```

The tens:

```

2043 \newcommand*\@@Tenthstringenglish[1]{%
2044 \ifcase#1\relax
2045 \or Tenth%
2046 \or Twentieth%
2047 \or Thirtieth%
2048 \or Fortieth%
2049 \or Fiftieth%
2050 \or Sixtieth%
2051 \or Seventieth%
2052 \or Eightieth%
2053 \or Ninetieth%
2054 \fi
2055 }%
2056 \global\let\@@Tenthstringenglish\@@Tenthstringenglish

```

The teens:

```

2057 \newcommand*\@@Teenthstringenglish[1]{%
2058 \ifcase#1\relax
2059 Tenth%
2060 \or Eleventh%
2061 \or Twelfth%
2062 \or Thirteenth%
2063 \or Fourteenth%
2064 \or Fifteenth%
2065 \or Sixteenth%
2066 \or Seventeenth%
2067 \or Eighteenth%
2068 \or Nineteenth%
2069 \fi
2070 }%
2071 \global\let\@@Teenthstringenglish\@@Teenthstringenglish

```

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

```

2072 \newcommand*\@@ordinalstringenglish[2]{%
2073 \@strctr=#1\relax
2074 \ifnum#1>99999
2075 \PackageError{fmtcount}{Out of range}%
2076 {This macro only works for values less than 100000 (value given: \number\@strctr)}}%

```

```

2077 \else
2078 \ifnum#1<0
2079 \PackageError{fmtcount}{Negative numbers not permitted}%
2080 {This macro does not work for negative numbers, however
2081 you can try typing "minus" first, and then pass the modulus of
2082 this number}%
2083 \fi
2084 \def#2{}%
2085 \fi
2086 \@strctr=#1\relax \divide\@strctr by 1000\relax
2087 \ifnum\@strctr>9\relax
    #1 is greater or equal to 10000
2088 \divide\@strctr by 10
2089 \ifnum\@strctr>1\relax
2090 \let\@fc@ordstr#2\relax
2091 \edef#2{\@fc@ordstr\@tenstring{\@strctr}}%
2092 \@strctr=#1\relax
2093 \divide\@strctr by 1000\relax
2094 \@FCmodulo{\@strctr}{10}%
2095 \ifnum\@strctr>0\relax
2096 \let\@fc@ordstr#2\relax
2097 \edef#2{\@fc@ordstr-\@unitstring{\@strctr}}%
2098 \fi
2099 \else
2100 \@strctr=#1\relax \divide\@strctr by 1000\relax
2101 \@FCmodulo{\@strctr}{10}%
2102 \let\@fc@ordstr#2\relax
2103 \edef#2{\@fc@ordstr\@teenstring{\@strctr}}%
2104 \fi
2105 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
2106 \ifnum\@strctr=0\relax
2107 \let\@fc@ordstr#2\relax
2108 \edef#2{\@fc@ordstr\ \@thousandth}%
2109 \else
2110 \let\@fc@ordstr#2\relax
2111 \edef#2{\@fc@ordstr\ \@thousand}%
2112 \fi
2113 \else
2114 \ifnum\@strctr>0\relax
2115 \let\@fc@ordstr#2\relax
2116 \edef#2{\@fc@ordstr\@unitstring{\@strctr}}%
2117 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
2118 \let\@fc@ordstr#2\relax
2119 \ifnum\@strctr=0\relax
2120 \edef#2{\@fc@ordstr\ \@thousandth}%
2121 \else
2122 \edef#2{\@fc@ordstr\ \@thousand}%
2123 \fi
2124 \fi

```



```

2125 \fi
2126 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
2127 \divide\@strctr by 100
2128 \ifnum\@strctr>0\relax
2129   \ifnum#1>1000\relax
2130     \let\@@fc@ordstr#2\relax
2131     \edef#2{\@@fc@ordstr\ }%
2132   \fi
2133   \let\@@fc@ordstr#2\relax
2134   \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
2135   \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
2136   \let\@@fc@ordstr#2\relax
2137   \ifnum\@strctr=0\relax
2138     \edef#2{\@@fc@ordstr\ \@hundredth}%
2139   \else
2140     \edef#2{\@@fc@ordstr\ \@hundred}%
2141   \fi
2142 \fi
2143 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
2144 \ifnum#1>100\relax
2145   \ifnum\@strctr>0\relax
2146     \let\@@fc@ordstr#2\relax
2147     \edef#2{\@@fc@ordstr\ \@andname\ }%
2148   \fi
2149 \fi
2150 \ifnum\@strctr>19\relax
2151   \@tmpstrctr=\@strctr
2152   \divide\@strctr by 10\relax
2153   \@FCmodulo{\@tmpstrctr}{10}%
2154   \let\@@fc@ordstr#2\relax
2155   \ifnum\@tmpstrctr=0\relax
2156     \edef#2{\@@fc@ordstr\@tenthstring{\@strctr}}%
2157   \else
2158     \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
2159   \fi
2160   \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
2161   \ifnum\@strctr>0\relax
2162     \let\@@fc@ordstr#2\relax
2163     \edef#2{\@@fc@ordstr-\@unitthstring{\@strctr}}%
2164   \fi
2165 \else
2166   \ifnum\@strctr<10\relax
2167     \ifnum\@strctr=0\relax
2168       \ifnum#1<100\relax
2169         \let\@@fc@ordstr#2\relax
2170         \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
2171       \fi
2172     \else
2173       \let\@@fc@ordstr#2\relax

```

```

2174 \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
2175 \fi
2176 \else
2177 \@FCmodulo{\@strctr}{10}%
2178 \let\@@fc@ordstr#2\relax
2179 \edef#2{\@@fc@ordstr\@teenthstring{\@strctr}}%
2180 \fi
2181 \fi
2182 }%
2183 \global\let\@@ordinalstringenglish\@@ordinalstringenglish

```

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

```

2184 \DeclareRobustCommand{\@ordinalstringMenglish}[2]{%
2185 \let\@unitthstring=\@unitthstringenglish
2186 \let\@teenthstring=\@teenthstringenglish
2187 \let\@tenthstring=\@tenthstringenglish
2188 \let\@unitstring=\@unitstringenglish
2189 \let\@teenstring=\@teenstringenglish
2190 \let\@tenstring=\@tenstringenglish
2191 \def\@andname{and}%
2192 \def\@hundred{hundred}\def\@thousand{thousand}%
2193 \def\@hundredth{hundredth}\def\@thousandth{thousandth}%
2194 \@ordinalstringenglish{#1}{#2}%
2195 }%
2196 \global\let\@ordinalstringMenglish\@ordinalstringMenglish

```

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

```

2197 \global\let\@ordinalstringFenglish=\@ordinalstringMenglish
2198 \global\let\@ordinalstringNenglish=\@ordinalstringMenglish

```

First letter of each word in upper case:

```

2199 \DeclareRobustCommand{\@OrdinalstringMenglish}[2]{%
2200 \let\@unitthstring=\@Unitthstringenglish
2201 \let\@teenthstring=\@Teenthstringenglish
2202 \let\@tenthstring=\@Tenthstringenglish
2203 \let\@unitstring=\@Unitstringenglish
2204 \let\@teenstring=\@Teenstringenglish
2205 \let\@tenstring=\@Tenstringenglish
2206 \def\@andname{and}%
2207 \def\@hundred{Hundred}\def\@thousand{Thousand}%
2208 \def\@hundredth{Hundredth}\def\@thousandth{Thousandth}%
2209 \@Ordinalstringenglish{#1}{#2}%
2210 }%
2211 \global\let\@OrdinalstringMenglish\@OrdinalstringMenglish

```

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

```

2212 \global\let\@OrdinalstringFenglish=\@OrdinalstringMenglish
2213 \global\let\@OrdinalstringNenglish=\@OrdinalstringMenglish

```

#### 9.4.4 fc-francais.def

```
2214 \ProvidesFCLanguage{francais}[2013/08/17]%
2215 \FCloadlang{french}%
```

Set francais to be equivalent to french.

```
2216 \global\let\@ordinalMfrancais=\@ordinalMfrench
2217 \global\let\@ordinalFfrancais=\@ordinalFfrench
2218 \global\let\@ordinalNfrancais=\@ordinalNfrench
2219 \global\let\@numberstringMfrancais=\@numberstringMfrench
2220 \global\let\@numberstringFfrancais=\@numberstringFfrench
2221 \global\let\@numberstringNfrancais=\@numberstringNfrench
2222 \global\let\@NumberstringMfrancais=\@NumberstringMfrench
2223 \global\let\@NumberstringFfrancais=\@NumberstringFfrench
2224 \global\let\@NumberstringNfrancais=\@NumberstringNfrench
2225 \global\let\@ordinalstringMfrancais=\@ordinalstringMfrench
2226 \global\let\@ordinalstringFfrancais=\@ordinalstringFfrench
2227 \global\let\@ordinalstringNfrancais=\@ordinalstringNfrench
2228 \global\let\@OrdinalstringMfrancais=\@OrdinalstringMfrench
2229 \global\let\@OrdinalstringFfrancais=\@OrdinalstringFfrench
2230 \global\let\@OrdinalstringNfrancais=\@OrdinalstringNfrench
```

#### 9.4.5 fc-french.def

Definitions for French.

```
2231 \ProvidesFCLanguage{french}[2012/10/24]%
```

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

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

```
#1 key name,
#2 key value,
#3 configuration index for 'reformed',
#4 configuration index for 'traditional',
#5 configuration index for 'reformed o', and
#6 configuration index for 'traditional o'.
2232 \def\fc@french@set@plural#1#2#3#4#5#6{%
2233   \ifthenelse{\equal{#2}{reformed}}{%
2234     \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#3}%
2235   }{%
2236     \ifthenelse{\equal{#2}{traditional}}{%
2237       \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#4}%
2238     }{%
2239       \ifthenelse{\equal{#2}{reformed o}}{%
2240         \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#5}%
2241       }{%
2242         \ifthenelse{\equal{#2}{traditional o}}{%
```

```

2243 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#6}%
2244 }{%
2245 \ifthenelse{\equal{#2}{always}}{%
2246 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{0}%
2247 }{%
2248 \ifthenelse{\equal{#2}{never}}{%
2249 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{1}%
2250 }{%
2251 \ifthenelse{\equal{#2}{multiple}}{%
2252 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{2}%
2253 }{%
2254 \ifthenelse{\equal{#2}{multiple g-last}}{%
2255 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{3}%
2256 }{%
2257 \ifthenelse{\equal{#2}{multiple l-last}}{%
2258 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{4}%
2259 }{%
2260 \ifthenelse{\equal{#2}{multiple lng-last}}{%
2261 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{5}%
2262 }{%
2263 \ifthenelse{\equal{#2}{multiple ng-last}}{%
2264 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{6}%
2265 }{%
2266 \PackageError{fmtcount}{Unexpected argument}{%
2267 '#2' was unexpected: french option '#1 plural' expects 'reformed'
2268 'reformed o', 'traditional o', 'always', 'never', 'multiple', 'mu
2269 'multiple l-last', 'multiple lng-last', or 'multiple ng-last'.%
2270 }}}}}}}}}}}}}

```

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’.

```

2271 \def\@tempa#1#2#3{%
2272 \define@key{fcfrench}{#1 plural}[reformed]{%
2273 \fc@french@set@plural{#1}{##1}{#2}{#3}{#3}%
2274 }%
2275 }
2276 \@tempa{vingt}{4}{5}
2277 \@tempa{cent}{4}{5}
2278 \@tempa{mil}{0}{0}
2279 \@tempa{n-illion}{2}{6}
2280 \@tempa{n-illiard}{2}{6}

```

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

```

2281 \define@key{fcfrench}{all plural}[reformed]{%
2282 \csname KV@fcfrench@vingt plural\endcsname{#1}%
2283 \csname KV@fcfrench@cent plural\endcsname{#1}%
2284 \csname KV@fcfrench@mil plural\endcsname{#1}%

```

```

2285 \csname KV@fcfrench@n-illion plural\endcsname{#1}%
2286 \csname KV@fcfrench@n-illiard plural\endcsname{#1}%
2287 }

```

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

```

2288 \define@key{fcfrench}{dash or space}[reformed]{%
2289 \ifthenelse{\equal{#1}{traditional}}{%
2290 \let\fc@frenchoptions@supermillion@dos\space%
2291 \let\fc@frenchoptions@submillion@dos\space
2292 }{%
2293 \ifthenelse{\equal{#1}{reformed}}{\or\equal{#1}{1990}}{%
2294 \let\fc@frenchoptions@supermillion@dos\space
2295 \def\fc@frenchoptions@submillion@dos{-}%
2296 }{%
2297 \ifthenelse{\equal{#1}{always}}{%
2298 \def\fc@frenchoptions@supermillion@dos{-}%
2299 \def\fc@frenchoptions@submillion@dos{-}%
2300 }{%
2301 \PackageError{fmtcount}{Unexpected argument}{%
2302 French option ‘dash or space’ expects ‘always’, ‘reformed’ or ‘traditional’
2303 }
2304 }%
2305 }%
2306 }%
2307 }

```

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

```

2308 \define@key{fcfrench}{scale}[recursive]{%
2309 \ifthenelse{\equal{#1}{long}}{%
2310 \let\fc@poweroften\fc@@pot@longscalefrench
2311 }{%
2312 \ifthenelse{\equal{#1}{recursive}}{%
2313 \let\fc@poweroften\fc@@pot@recursivefrench
2314 }{%
2315 \ifthenelse{\equal{#1}{short}}{%
2316 \let\fc@poweroften\fc@@pot@shortscalefrench
2317 }{%
2318 \PackageError{fmtcount}{Unexpected argument}{%
2319 French option ‘scale’ expects ‘long’, ‘recursive’ or ‘short’
2320 }

```

```

2321     }%
2322 }%
2323 }%
2324 }

```

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”

```

2325 \define@key{french}{n-illiard upto}[infinity]{%
2326   \ifthenelse{\equal{#1}{infinity}}{%
2327     \def\fc@longscale@nilliard@upto{0}%
2328   }{%
2329     \ifthenelse{\equal{#1}{infty}}{%
2330       \def\fc@longscale@nilliard@upto{0}%
2331     }{%
2332       \ifnum 9 < 1#1\fi\else
2333       \PackageError{fmtcount}{Unexpected argument}{%
2334         French option ‘milliard threshold’ expects ‘infinity’, or equivalently ‘infty’, or a
2335         integer.}%
2336       \fi
2337       \def\fc@longscale@nilliard@upto{#1}%
2338     }%
2339 }

```

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.

```

2340 \def\@tempa#1{%
2341   \define@key{french}{#1}[]{}%
2342   \PackageError{fmtcount}{Unexpected argument}{French option with key ‘#1’ does not take
2343     any value}%
2344   \expandafter\def\csname KV@french@#1@default\endcsname{%
2345     \def\fmtcount@french{#1}}%
2346 }%
2347 \@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.

```

2348 \define@key{french}{dialect}[france]{%
2349   \ifthenelse{\equal{#1}{france}}{
2350     \or\equal{#1}{swiss}
2351     \or\equal{#1}{belgian}}{%
2352     \def\fmtcount@french{#1}}%
2353   \PackageError{fmtcount}{Invalid value ‘#1’ to french option dialect key}
2354   {Option ‘french’ can only take the values ‘france’,
2355     ‘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`’.

```
2356 \define@key{fcfrench}{mil plural mark}[le]{%
2357   \def\fc@frenchoptions@mil@plural@mark{#1}}
```

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

```
2358 \def\fc@UpperCaseFirstLetter#1#2\@nil{%
2359   \uppercase{#1}#2}
2360
2361 \def\fc@CaseIden#1\@nil{%
2362   #1%
2363 }
2364 \def\fc@UpperCaseAll#1\@nil{%
2365   \uppercase{#1}%
2366 }
2367
2368 \let\fc@case\fc@CaseIden
2369
```

`\@ ordinalMfrench`

```
2370 \newcommand*{\@ordinalMfrench}[2]{%
2371 \iffmtord@abbrv
2372   \edef#2{\number#1\relax\noexpand\fmtord{e}}%
2373 \else
2374   \ifnum#1=1\relax
2375     \edef#2{\number#1\relax\noexpand\fmtord{er}}%
2376   \else
2377     \edef#2{\number#1\relax\noexpand\fmtord{eme}}%
2378   \fi
2379 \fi}
```

`\@ ordinalFfrench`

```
2380 \newcommand*{\@ordinalFfrench}[2]{%
2381 \iffmtord@abbrv
2382   \edef#2{\number#1\relax\noexpand\fmtord{e}}%
2383 \else
2384   \ifnum#1=1 %
2385     \edef#2{\number#1\relax\noexpand\fmtord{i\‘ere}}%
2386   \else
2387     \edef#2{\number#1\relax\noexpand\fmtord{i\‘eme}}%
2388   \fi
2389 \fi}
```

In French neutral gender and masculine gender are formally identical.

```
2390 \let\@ordinalNfrench\@ordinalMfrench
```

`\@ @unitstringfrench`

```
2391 \newcommand*{\@unitstringfrench}[1]{%
2392 \noexpand\fc@case
2393 \ifcase#1 %
```

```

2394 z\'ero%
2395 \or un%
2396 \or deux%
2397 \or trois%
2398 \or quatre%
2399 \or cinq%
2400 \or six%
2401 \or sept%
2402 \or huit%
2403 \or neuf%
2404 \fi
2405 \noexpand\@nil
2406 }

\@ @tenstringfrench
2407 \newcommand*{\@tenstringfrench}[1]{%
2408 \noexpand\fc@case
2409 \ifcase#1 %
2410 \or dix%
2411 \or vingt%
2412 \or trente%
2413 \or quarante%
2414 \or cinquante%
2415 \or soixante%
2416 \or septante%
2417 \or huitante%
2418 \or nonante%
2419 \or cent%
2420 \fi
2421 \noexpand\@nil
2422 }

\@ @teenstringfrench
2423 \newcommand*{\@teenstringfrench}[1]{%
2424 \noexpand\fc@case
2425 \ifcase#1 %
2426     dix%
2427 \or onze%
2428 \or douze%
2429 \or treize%
2430 \or quatorze%
2431 \or quinze%
2432 \or seize%
2433 \or dix\noexpand\@nil-\noexpand\fc@case sept%
2434 \or dix\noexpand\@nil-\noexpand\fc@case huit%
2435 \or dix\noexpand\@nil-\noexpand\fc@case neuf%
2436 \fi
2437 \noexpand\@nil
2438 }

\@ @seventiesfrench

```



```

2439 \newcommand*\@@seventiesfrench}[1]{%
2440 \@tenstring{6}%
2441 \ifnum#1=1 %
2442 \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
2443 \else
2444 -%
2445 \fi
2446 \@teenstring{#1}%
2447 }

```

\@ @eightiesfrench Macro \@eightiesfrench is used to format numbers in the interval [80..89]. Argument as follows:

#1 digit  $d_w$  such that the number to be formatted is  $80 + d_w$

Implicit arguments as:

\count0 weight  $w$  of the number  $d_{w+1}d_w$  to be formatted

\count1 same as \#1

\count6 input, counter giving the least weight of non zero digits in top level formatted number integral part, with rounding down to a multiple of 3,

\count9 input, counter giving the power type of the power of ten following the eighties to be formatted; that is '1' for "mil" and '2' for "< $nn$

```

2448 \newcommand*\@@eightiesfrench[1]{%
2449 \fc@case quatre\@nil-\noexpand\fc@case vingt%
2450 \ifnum#1>0 %
2451 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
2452 s%
2453 \fi
2454 \noexpand\@nil
2455 -\@unitstring{#1}%
2456 \else
2457 \ifcase\fc@frenchoptions@vingt@plural\space
2458 s% 0: always
2459 \or
2460 % 1: never
2461 \or
2462 s% 2: multiple
2463 \or
2464 % 3: multiple g-last
2465 \ifnum\count0=\count6\ifnum\count9=0 s\fi\fi
2466 \or
2467 % 4: multiple l-last
2468 \ifnum\count9=1 %
2469 \else
2470 s%
2471 \fi
2472 \or
2473 % 5: multiple lng-last

```

```

2474 \ifnum\count9=1 %
2475 \else
2476 \ifnum\count0>0 %
2477 s%
2478 \fi
2479 \fi
2480 \or
2481 % or 6: multiple ng-last
2482 \ifnum\count0>0 %
2483 s%
2484 \fi
2485 \fi
2486 \noexpand\@nil
2487 \fi
2488 }
2489 \newcommand*\@@ninetiesfrench}[1]{%
2490 \fc@case quatre\@nil-\noexpand\fc@case vingt%
2491 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
2492 s%
2493 \fi
2494 \noexpand\@nil
2495 -\@teenstring{#1}%
2496 }
2497 \newcommand*\@@seventiesfrenchswiss}[1]{%
2498 \@tenstring{7}%
2499 \ifnum#1=1\ \@andname\ \fi
2500 \ifnum#1>1-\fi
2501 \ifnum#1>0 \@unitstring{#1}\fi
2502 }
2503 \newcommand*\@@eightiesfrenchswiss}[1]{%
2504 \@tenstring{8}%
2505 \ifnum#1=1\ \@andname\ \fi
2506 \ifnum#1>1-\fi
2507 \ifnum#1>0 \@unitstring{#1}\fi
2508 }
2509 \newcommand*\@@ninetiesfrenchswiss}[1]{%
2510 \@tenstring{9}%
2511 \ifnum#1=1\ \@andname\ \fi
2512 \ifnum#1>1-\fi
2513 \ifnum#1>0 \@unitstring{#1}\fi
2514 }

```

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

```

2515 \newcommand*\fc@french@common{%
2516 \let\@unitstring=\@unitstringfrench
2517 \let\@teenstring=\@teenstringfrench
2518 \let\@tenstring=\@tenstringfrench
2519 \def\@hundred{cent}%
2520 \def\@andname{et}%

```

```

2521 }

2522 \DeclareRobustCommand{\@numberstringMfrenchswiss}[2]{%
2523 \let\fc@case\fc@CaseIden
2524 \fc@french@common
2525 \let\@seventies=\@seventiesfrenchswiss
2526 \let\@eighties=\@eightiesfrenchswiss
2527 \let\@nineties=\@ninetiesfrenchswiss
2528 \let\fc@nbrstr@preamble\@empty
2529 \let\fc@nbrstr@postamble\@empty
2530 \@numberstringfrench{#1}{#2}}
2531 \DeclareRobustCommand{\@numberstringMfrenchfrance}[2]{%
2532 \let\fc@case\fc@CaseIden
2533 \fc@french@common
2534 \let\@seventies=\@seventiesfrench
2535 \let\@eighties=\@eightiesfrench
2536 \let\@nineties=\@ninetiesfrench
2537 \let\fc@nbrstr@preamble\@empty
2538 \let\fc@nbrstr@postamble\@empty
2539 \@numberstringfrench{#1}{#2}}
2540 \DeclareRobustCommand{\@numberstringMfrenchbelgian}[2]{%
2541 \let\fc@case\fc@CaseIden
2542 \fc@french@common
2543 \let\@seventies=\@seventiesfrenchswiss
2544 \let\@eighties=\@eightiesfrench
2545 \let\@nineties=\@ninetiesfrench
2546 \let\fc@nbrstr@preamble\@empty
2547 \let\fc@nbrstr@postamble\@empty
2548 \@numberstringfrench{#1}{#2}}
2549 \let\@numberstringMfrench=\@numberstringMfrenchfrance
2550 \DeclareRobustCommand{\@numberstringFfrenchswiss}[2]{%
2551 \let\fc@case\fc@CaseIden
2552 \fc@french@common
2553 \let\@seventies=\@seventiesfrenchswiss
2554 \let\@eighties=\@eightiesfrenchswiss
2555 \let\@nineties=\@ninetiesfrenchswiss
2556 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
2557 \let\fc@nbrstr@postamble\@empty
2558 \@numberstringfrench{#1}{#2}}
2559 \DeclareRobustCommand{\@numberstringFfrenchfrance}[2]{%
2560 \let\fc@case\fc@CaseIden
2561 \fc@french@common
2562 \let\@seventies=\@seventiesfrench
2563 \let\@eighties=\@eightiesfrench
2564 \let\@nineties=\@ninetiesfrench
2565 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
2566 \let\fc@nbrstr@postamble\@empty
2567 \@numberstringfrench{#1}{#2}}
2568 \DeclareRobustCommand{\@numberstringFfrenchbelgian}[2]{%
2569 \let\fc@case\fc@CaseIden

```

```

2570 \fc@french@common
2571 \let\@seventies=\@seventiesfrenchswiss
2572 \let\@eighties=\@eightiesfrench
2573 \let\@nineties=\@ninetiesfrench
2574 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2575 \let\fc@nbrstr@postamble\@empty
2576 \@@numberstringfrench{#1}{#2}}
2577 \let\@numberstringFfrench=\@numberstringFfrenchfrance
2578 \let\@ordinalstringNfrench\@ordinalstringMfrench
2579 \DeclareRobustCommand{\@NumberstringMfrenchswiss}[2]{%
2580 \let\fc@case\fc@UpperCaseFirstLetter
2581 \fc@french@common
2582 \let\@seventies=\@seventiesfrenchswiss
2583 \let\@eighties=\@eightiesfrenchswiss
2584 \let\@nineties=\@ninetiesfrenchswiss
2585 \let\fc@nbrstr@preamble\@empty
2586 \let\fc@nbrstr@postamble\@empty
2587 \@@numberstringfrench{#1}{#2}}
2588 \DeclareRobustCommand{\@NumberstringMfrenchfrance}[2]{%
2589 \let\fc@case\fc@UpperCaseFirstLetter
2590 \fc@french@common
2591 \let\@seventies=\@seventiesfrench
2592 \let\@eighties=\@eightiesfrench
2593 \let\@nineties=\@ninetiesfrench
2594 \let\fc@nbrstr@preamble\@empty
2595 \let\fc@nbrstr@postamble\@empty
2596 \@@numberstringfrench{#1}{#2}}
2597 \DeclareRobustCommand{\@NumberstringMfrenchbelgian}[2]{%
2598 \let\fc@case\fc@UpperCaseFirstLetter
2599 \fc@french@common
2600 \let\@seventies=\@seventiesfrenchswiss
2601 \let\@eighties=\@eightiesfrench
2602 \let\@nineties=\@ninetiesfrench
2603 \let\fc@nbrstr@preamble\@empty
2604 \let\fc@nbrstr@postamble\@empty
2605 \@@numberstringfrench{#1}{#2}}
2606 \let\@NumberstringMfrench=\@NumberstringMfrenchfrance
2607 \DeclareRobustCommand{\@NumberstringFfrenchswiss}[2]{%
2608 \let\fc@case\fc@UpperCaseFirstLetter
2609 \fc@french@common
2610 \let\@seventies=\@seventiesfrenchswiss
2611 \let\@eighties=\@eightiesfrenchswiss
2612 \let\@nineties=\@ninetiesfrenchswiss
2613 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2614 \let\fc@nbrstr@postamble\@empty
2615 \@@numberstringfrench{#1}{#2}}
2616 \DeclareRobustCommand{\@NumberstringFfrenchfrance}[2]{%
2617 \let\fc@case\fc@UpperCaseFirstLetter
2618 \fc@french@common

```

```

2619 \let\@seventies=\@seventiesfrench
2620 \let\@eighties=\@eightiesfrench
2621 \let\@nineties=\@ninetiesfrench
2622 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
2623 \let\fc@nbrstr@postamble\@empty
2624 \@@numberstringfrench{#1}{#2}}
2625 \DeclareRobustCommand{\@NumberstringFfrenchbelgian}[2]{%
2626 \let\fc@case\fc@UpperCaseFirstLetter
2627 \fc@french@common
2628 \let\@seventies=\@seventiesfrenchswiss
2629 \let\@eighties=\@eightiesfrench
2630 \let\@nineties=\@ninetiesfrench
2631 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
2632 \let\fc@nbrstr@postamble\@empty
2633 \@@numberstringfrench{#1}{#2}}
2634 \let\@NumberstringFfrench=\@NumberstringFfrenchfrance
2635 \let\@NumberstringNfrench\@NumberstringMfrench
2636 \DeclareRobustCommand{\@OrdinalstringMfrenchswiss}[2]{%
2637 \let\fc@case\fc@CaseIden
2638 \let\fc@first=\fc@@firstfrench
2639 \fc@french@common
2640 \let\@seventies=\@seventiesfrenchswiss
2641 \let\@eighties=\@eightiesfrenchswiss
2642 \let\@nineties=\@ninetiesfrenchswiss
2643 \@@ordinalstringfrench{#1}{#2}%
2644 }
2645 \newcommand*\fc@@firstfrench{premier}
2646 \newcommand*\fc@@firstFfrench{premi'ere}
2647 \DeclareRobustCommand{\@OrdinalstringMfrenchfrance}[2]{%
2648 \let\fc@case\fc@CaseIden
2649 \let\fc@first=\fc@@firstfrench
2650 \fc@french@common
2651 \let\@seventies=\@seventiesfrench
2652 \let\@eighties=\@eightiesfrench
2653 \let\@nineties=\@ninetiesfrench
2654 \@@ordinalstringfrench{#1}{#2}}
2655 \DeclareRobustCommand{\@OrdinalstringMfrenchbelgian}[2]{%
2656 \let\fc@case\fc@CaseIden
2657 \let\fc@first=\fc@@firstfrench
2658 \fc@french@common
2659 \let\@seventies=\@seventiesfrench
2660 \let\@eighties=\@eightiesfrench
2661 \let\@nineties=\@ninetiesfrench
2662 \@@ordinalstringfrench{#1}{#2}%
2663 }
2664 \let\@OrdinalstringMfrench=\@OrdinalstringMfrenchfrance
2665 \DeclareRobustCommand{\@OrdinalstringFfrenchswiss}[2]{%
2666 \let\fc@case\fc@CaseIden
2667 \let\fc@first=\fc@@firstFfrench

```

```

2668 \fc@french@common
2669 \let\@seventies=\@seventiesfrenchswiss
2670 \let\@eighties=\@eightiesfrenchswiss
2671 \let\@nineties=\@ninetiesfrenchswiss
2672 \@@ordinalstringfrench{#1}{#2}%
2673 }
2674 \DeclareRobustCommand{\@ordinalstringFfrenchfrance}[2]{%
2675 \let\fc@case\fc@CaseIden
2676 \let\fc@first=\fc@@firstFfrench
2677 \fc@french@common
2678 \let\@seventies=\@seventiesfrench
2679 \let\@eighties=\@eightiesfrench
2680 \let\@nineties=\@ninetiesfrench
2681 \@@ordinalstringfrench{#1}{#2}%
2682 }
2683 \DeclareRobustCommand{\@ordinalstringFfrenchbelgian}[2]{%
2684 \let\fc@case\fc@CaseIden
2685 \let\fc@first=\fc@@firstFfrench
2686 \fc@french@common
2687 \let\@seventies=\@seventiesfrench
2688 \let\@eighties=\@eightiesfrench
2689 \let\@nineties=\@ninetiesfrench
2690 \@@ordinalstringfrench{#1}{#2}%
2691 }
2692 \let\@ordinalstringFfrench=\@ordinalstringFfrenchfrance
2693 \let\@ordinalstringNfrench\@ordinalstringMfrench
2694 \DeclareRobustCommand{\@OrdinalstringMfrenchswiss}[2]{%
2695 \let\fc@case\fc@UpperCaseFirstLetter
2696 \let\fc@first=\fc@@firstfrench
2697 \fc@french@common
2698 \let\@seventies=\@seventiesfrenchswiss
2699 \let\@eighties=\@eightiesfrenchswiss
2700 \let\@nineties=\@ninetiesfrenchswiss
2701 \@@ordinalstringfrench{#1}{#2}%
2702 }
2703 \DeclareRobustCommand{\@OrdinalstringMfrenchfrance}[2]{%
2704 \let\fc@case\fc@UpperCaseFirstLetter
2705 \let\fc@first=\fc@@firstfrench
2706 \fc@french@common
2707 \let\@seventies=\@seventiesfrench
2708 \let\@eighties=\@eightiesfrench
2709 \let\@nineties=\@ninetiesfrench
2710 \@@ordinalstringfrench{#1}{#2}%
2711 }
2712 \DeclareRobustCommand{\@OrdinalstringMfrenchbelgian}[2]{%
2713 \let\fc@case\fc@UpperCaseFirstLetter
2714 \let\fc@first=\fc@@firstfrench
2715 \fc@french@common
2716 \let\@seventies=\@seventiesfrench

```

```

2717 \let\@eighties=\@eightiesfrench
2718 \let\@nineties=\@ninetiesfrench
2719 \@ordinalstringfrench{#1}{#2}%
2720 }
2721 \let\@OrdinalstringMfrench=\@OrdinalstringMfrenchfrance
2722 \DeclareRobustCommand{\@OrdinalstringFfrenchswiss}[2]{%
2723 \let\fc@case\fc@UpperCaseFirstLetter
2724 \let\fc@first=\fc@@firstfrench
2725 \fc@french@common
2726 \let\@seventies=\@seventiesfrenchswiss
2727 \let\@eighties=\@eightiesfrenchswiss
2728 \let\@nineties=\@ninetiesfrenchswiss
2729 \@ordinalstringfrench{#1}{#2}%
2730 }
2731 \DeclareRobustCommand{\@OrdinalstringFfrenchfrance}[2]{%
2732 \let\fc@case\fc@UpperCaseFirstLetter
2733 \let\fc@first=\fc@@firstFfrench
2734 \fc@french@common
2735 \let\@seventies=\@seventiesfrench
2736 \let\@eighties=\@eightiesfrench
2737 \let\@nineties=\@ninetiesfrench
2738 \@ordinalstringfrench{#1}{#2}%
2739 }
2740 \DeclareRobustCommand{\@OrdinalstringFfrenchbelgian}[2]{%
2741 \let\fc@case\fc@UpperCaseFirstLetter
2742 \let\fc@first=\fc@@firstFfrench
2743 \fc@french@common
2744 \let\@seventies=\@seventiesfrench
2745 \let\@eighties=\@eightiesfrench
2746 \let\@nineties=\@ninetiesfrench
2747 \@ordinalstringfrench{#1}{#2}%
2748 }
2749 \let\@OrdinalstringFfrench=\@OrdinalstringFfrenchfrance
2750 \let\@OrdinalstringNfrench\@OrdinalstringMfrench

```

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

```

2751 \ifcsundef{fc@@do@plural@mark}{}%
2752 {\PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2753   'fc@@do@plural@mark'}}

```

Arguments as follows:

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

Implicit arguments as follows:

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

```

2754 \def\fc@@do@plural@mark#1{%
2755   \ifcase\count10 %
2756     #1% 0=always
2757     \or% 1=never
2758     \or% 2=multiple
2759       \ifnum\count1>1 %
2760         #1%
2761       \fi
2762     \or% 3= multiple g-last
2763       \ifnum\count1>1 %
2764         \ifnum\count0=\count6 %
2765           #1%
2766         \fi
2767       \fi
2768     \or% 4= multiple l-last
2769       \ifnum\count1>1 %
2770         \ifnum\count9=1 %
2771           \else
2772             #1%
2773           \fi
2774       \fi
2775     \or% 5= multiple lng-last
2776       \ifnum\count1>1 %
2777         \ifnum\count9=1 %
2778           \else
2779             \if\count0>\count6 %
2780               #1%
2781             \fi
2782           \fi
2783       \fi
2784     \or% 6= multiple ng-last
2785       \ifnum\count1>1 %
2786         \ifnum\count0>\count6 %
2787           #1%
2788         \fi
2789       \fi
2790   \fi
2791 }

```



```

\fc  @@nbrstr@Fpreamble Macro \fc@@nbrstr@Fpreamble do the necessary pre-
liminaries before formatting a cardinal with feminine gender.
2792 \ifcsundef{fc@@nbrstr@Fpreamble}{\}%
2793 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2794   'fc@@nbrstr@Fpreamble'}}}

\fc  @@nbrstr@Fpreamble
2795 \def\fc@@nbrstr@Fpreamble{%
2796   \fc@read@unit{\count1}{0}%
2797   \ifnum\count1=1 %
2798     \let\fc@case@save\fc@case
2799     \def\fc@case{\noexpand\fc@case}%
2800     \def\@nil{\noexpand\@nil}%
2801     \let\fc@nbrstr@postamble\fc@@nbrstr@Fpostamble
2802   \fi
2803 }

\fc  @@nbrstr@Fpostamble
2804 \def\fc@@nbrstr@Fpostamble{%
2805   \let\fc@case\fc@case@save
2806   \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
2807   \def\@tempd{un}%
2808   \ifx\@tempc\@tempd
2809     \let\@tempc\@tempa
2810     \edef\@tempa{\@tempb\fc@case une\@nil}%
2811   \fi
2812 }

\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.
2813 \ifcsundef{fc@@pot@longscalefrench}{\}%
2814 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2815   '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
2816 \def\fc@@pot@longscalefrench#1#2#3{%
2817   {%

```

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

```
2818 \edef\@tempb{\number#1}%
```

Let \count1 be the plural value.

```
2819 \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  $r$ .

```
2820 \count2\count0 %
2821 \divide\count2 by 6 %
2822 \count3\count2 %
2823 \multiply\count3 by 6 %
2824 \count3-\count3 %
2825 \advance\count3 by \count0 %
2826 \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)”.

```
2827 \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.

```
2828 \edef\@temp{%
2829 \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.

```
2830 1%
2831 \else
2832 \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.

```
2833 \ifnum\fc@longscale@nilliard@upto=0 %
```

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

```
2834 2%
2835 \else
```

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

```
2836 \ifnum\count2>\fc@longscale@nilliard@upto
2837 1%
2838 \else
2839 2%
2840 \fi
2841 \fi
```

```

2842         \else
2843         2%
2844         \fi
2845     \fi
2846 }%
2847 \ifnum \@temph=1 %

```

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

```

2848     \count10=\fc@frenchoptions@mil@plural\space
2849     \edef \@tempe{%
2850         \noexpand\fc@case
2851         mil%
2852         \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
2853         \noexpand\@nil
2854     }%
2855 \else
2856     % weight >= 6
2857     \expandafter\fc@@latin@cardinal@pexif\expandafter{\the\count2}\@tempg
2858     % now form the xxx-illion(s) or xxx-illiard(s) word
2859     \ifnum\count3>2 %
2860         \toks10{illiard}%
2861         \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
2862     \else
2863         \toks10{illion}%
2864         \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2865     \fi
2866     \edef \@tempe{%
2867         \noexpand\fc@case
2868         \@tempg
2869         \the\toks10 %
2870         \fc@@do@plural@mark s%
2871         \noexpand\@nil
2872     }%
2873     \fi
2874 \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.

```

2875     \let \@tempe \@empty
2876     \def \@temph{0}%
2877 \fi
2878 \else

```

Case of  $w = 0$ .

```

2879     \let \@tempe \@empty
2880     \def \@temph{0}%
2881 \fi

```

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

```

2882     \expandafter\toks\expandafter1\expandafter{\@tempe}%

```

```

2883 \toks0{#2}%
2884 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
2885 \expandafter
2886 }\@tempa
2887 }

```

\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.

```

2888 \ifcsundef{fc@@pot@shortscalefrench}{\}%
2889 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2890 '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

```

2891 \def\fc@@pot@shortscalefrench#1#2#3{%
2892   {%

```

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

```

2893 \edef\@tempb{\number#1}%

```

And let \count1 be the plural value.

```

2894 \count1=\@tempb

```

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

```

2895 \count2\count0 %
2896 \divide\count2 by 3 %
2897 \advance\count2 by -1 %

```

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

```

2898 \ifnum\count0>0 % If weight>=3, i.e we do have to append thousand or n-illion(s)/n-illi
2899 \ifnum\count1>0 % we have at least one thousand/n-illion/n-illiard
2900 \ifnum\count2=0 %
2901 \def\@temph{1}%
2902 \count1=\fc@frenchoptions@mil@plural\space
2903 \edef\@tempe{%

```

```

2904         mil%
2905         \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
2906     }%
2907 \else
2908     \def\@temph{2}%
2909     % weight >= 6
2910     \expandafter\fc@@latin@cardinal@pefix\expandafter{\the\count2}\@tempg
2911     \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2912     \edef\@tempe{%
2913         \noexpand\fc@case
2914         \@tempg
2915         illion%
2916         \fc@@do@plural@mark s%
2917         \noexpand\@nil
2918     }%
2919 \fi
2920 \else

```

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

```

2921     \def\@temph{0}%
2922     \let\@tempe\@empty
2923 \fi
2924 \else

```

Here  $w = 0$ .

```

2925     \def\@temph{0}%
2926     \let\@tempe\@empty
2927 \fi
2928 % now place into \@cs{@tempa} the assignment of results \@cs{@temph} and \@cs{@tempe} to to \@
2929 % \texttt{\#3} for further propagation after closing brace.
2930 % \begin{macrocode}
2931 \expandafter\toks\expandafter1\expandafter{\@tempe}%
2932 \toks0{\#2}%
2933 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
2934 \expandafter
2935 }\@tempa
2936 }

```

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

```

2937 \ifcsundef{fc@@pot@recursivefrench}{-}{%
2938 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2939 '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

```
2940 \def\fc@pot@recursivefrench#1#2#3{%
```

```
2941  {%
```

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

```
2942   \edef\@tempb{\number#1}%
```

```
2943   \let\@tempa\@tempa
```

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

```
2944   \count1=\@tempb\space
```

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

```
2945   \ifnum\count1>0 %
```

```
2946     \count2\count0 %
```

```
2947     \divide\count2 by 9 %
```

```
2948     \advance\count2 by -1 %
```

```
2949     \let\@tempe\@empty
```

```
2950     \edef\@tempf{\fc@frenchoptions@supermillion@dos
```

```
2951       de\fc@frenchoptions@supermillion@dos\fc@case milliards\@nil}%
```

```
2952     \count11\count0 %
```

```
2953     \ifnum\count2>0 %
```

```
2954       \count3\count2 %
```

```
2955       \count3-\count3 %
```

```
2956       \multiply\count3 by 9 %
```

```
2957       \advance\count11 by \count3 %
```

```
2958     \loop
```

```
2959       % (\count2, \count3) <- (\count2 div 2, \count2 mod 2)
```

```
2960       \count3\count2 %
```

```
2961       \divide\count3 by 2 %
```

```
2962       \multiply\count3 by 2 %
```

```
2963       \count3-\count3 %
```

```
2964       \advance\count3 by \count2 %
```

```
2965       \divide\count2 by 2 %
```

```
2966       \ifnum\count3=1 %
```

```
2967         \let\@tempg\@tempe
```

```
2968         \edef\@tempe{\@tempg\@tempf}%
```

```
2969       \fi
```

```
2970       \let\@tempg\@tempf
```

```

2971         \edef\@tempf{\@tempg\@tempg}%
2972         \ifnum\count2>0 %
2973         \repeat
2974     \fi
2975     \divide\count11 by 3 %
2976     \ifcase\count11 % 0 .. 5
2977         % 0 => d milliard(s) (de milliards)*
2978         \def\@temph{2}%
2979         \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
2980     \or % 1 => d mille milliard(s) (de milliards)*
2981         \def\@temph{1}%
2982         \count10=\fc@frenchoptions@mil@plural\space
2983     \or % 2 => d million(s) (de milliards)*
2984         \def\@temph{2}%
2985         \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2986     \or % 3 => d milliard(s) (de milliards)*
2987         \def\@temph{2}%
2988         \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
2989     \or % 4 => d mille milliards (de milliards)*
2990         \def\@temph{1}%
2991         \count10=\fc@frenchoptions@mil@plural\space
2992     \else % 5 => d million(s) (de milliards)*
2993         \def\@temph{2}%
2994         \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2995     \fi
2996     \let\@tempg\@tempe
2997     \edef\@tempf{%
2998         \ifcase\count11 % 0 .. 5
2999         \or
3000             mil\fc@@do@plural@mark \fc@frenchoptions@mil@plural@mark
3001         \or
3002             million\fc@@do@plural@mark s%
3003         \or
3004             milliard\fc@@do@plural@mark s%
3005         \or
3006             mil\fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
3007             \noexpand\@nil\fc@frenchoptions@supermillion@dos
3008             \noexpand\fc@case milliards% 4
3009         \or
3010             million\fc@@do@plural@mark s%
3011             \noexpand\@nil\fc@frenchoptions@supermillion@dos
3012             de\fc@frenchoptions@supermillion@dos\noexpand\fc@case milliards% 5
3013         \fi
3014     }%
3015     \edef\@tempe{%
3016         \ifx\@tempf\@empty\else
3017             \expandafter\fc@case\@tempf\@nil
3018         \fi
3019         \@tempg

```

```

3020     }%
3021   \else
3022     \def\@temph{0}%
3023     \let\@tempe\@empty
3024   \fi

```

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

```

3025   \expandafter\toks\expandafter1\expandafter{\@tempe}%
3026   \toks0{#2}%
3027   \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
3028   \expandafter
3029 } \@tempa
3030 }

```

\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.

```

3031 \ifcsundef{fc@muladdfrench}{-}{%
3032   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3033     '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

```

3034 \def\fc@muladdfrench#1#2#3{%
3035   {%

```

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

```

3036   \edef\@tempc{#1}%
3037   \edef\@tempd{#2}%
3038   \edef\@tempf{#3}%
3039   \let\@tempc\@tempc
3040   \let\@tempd\@tempd

```



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

```

3041    \ifnum\@tempc=1 %
3042        \ifnum\count9=1 %
3043            \let\@tempd\@empty
3044        \fi
3045    \fi

```

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

```

3046    \edef\@tempg{%
3047        \@tempd
3048        \ifx\@tempd\@empty\else
3049            \ifx\@tempf\@empty\else
3050                \ifcase\count9 %
3051                    \or
3052                        \fc@frenchoptions@submillion@dos
3053                    \or
3054                        \fc@frenchoptions@supermillion@dos
3055                \fi
3056            \fi
3057        \fi
3058    \@tempf
3059 }%

```

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

```

3060    \edef\@temph{%
3061        \@tempa
3062        \ifx\@tempa\@empty\else
3063            \ifx\@tempg\@empty\else
3064                \ifcase\count8 %
3065                    \or
3066                        \fc@frenchoptions@submillion@dos
3067                    \or
3068                        \fc@frenchoptions@supermillion@dos
3069                \fi
3070            \fi
3071        \fi
3072    \@tempg
3073 }%

```

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

```

3074    \def\@tempb##1{\def\@tempa{\def\@tempa{##1}}}%
3075    \expandafter\@tempb\expandafter{\@temph}%
3076    \expandafter
3077 } \@tempa

```

```

3078 }%
\fc @lthundredstringfrench Macro \fc@lthundredstringfrench is used to for-
mat a number in interval [0..99]. First we check that it is not already defined.
3079 \ifcsundef{fc@lthundredstringfrench}{}%
3080 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3081 'fc@lthundredstringfrench'}}

The number to format is not passed as an argument to this macro, instead each
digits of it is in a \fc@digit@<w> macro after this number has been parsed. So
the only thing that \fc@lthundredstringfrench needs is to know <w> 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.
3082 \def\fc@lthundredstringfrench#1{%
3083 {%

First save arguments into local temporary macro.
3084 \let\@tempc#1%

Read units  $d_w$  to \count1.
3085 \fc@read@unit{\count1}{\count0}%

Read tens  $d_{w+1}$  to \count2.
3086 \count3\count0 %
3087 \advance\count3 1 %
3088 \fc@read@unit{\count2}{\count3}%

Now do the real job, set macro \@tempa to #1 followed by  $d_{w+1}d_w$  formatted.
3089 \edef\@tempa{%
3090 \@tempc
3091 \ifnum\count2>1 %
3092 % 20 .. 99
3093 \ifnum\count2>6 %
3094 % 70 .. 99
3095 \ifnum\count2<8 %
3096 % 70 .. 79
3097 \@seventies{\count1}%
3098 \else
3099 % 80..99
3100 \ifnum\count2<9 %
3101 % 80 .. 89
3102 \@eighties{\count1}%
3103 \else
3104 % 90 .. 99
3105 \@nineties{\count1}%
3106 \fi
3107 \fi

```

```

3108         \else
3109             % 20..69
3110             \@tenstring{\count2}%
3111             \ifnum\count1>0 %
3112                 % x1 .. x0
3113                 \ifnum\count1=1 %
3114                     % x1
3115                     \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
3116                 \else
3117                     % x2 .. x9
3118                     -%
3119                 \fi
3120             \@unitstring{\count1}%
3121         \fi
3122     \fi
3123 \else
3124     % 0 .. 19
3125     \ifnum\count2=0 % when tens = 0
3126         % 0 .. 9
3127         \ifnum\count1=0 % when units = 0
3128             % \count3=1 when #1 = 0, i.e. only for the unit of the top level number
3129             \ifnum\count3=1 %
3130                 \ifnum\fc@max@weight=0 %
3131                     \@unitstring{0}%
3132                 \fi
3133             \fi
3134         \else
3135             % 1 .. 9
3136             \@unitstring{\count1}%
3137         \fi
3138     \else
3139         % 10 .. 19
3140         \@teenstring{\count1}%
3141     \fi
3142 \fi
3143 }%

```

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

```

3144     \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
3145     \expandafter\@tempb\expandafter{\@tempa}%
3146     \expandafter
3147 }\@tempa
3148 }

```

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

```

3149 \ifcsundef{fc@lthousandstringfrench}{\{%
3150   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3151     'fc@lthousandstringfrench'}}%

```

Output is empty for 0. Arguments as follows:

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

Implicit input arguments as follows:

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

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

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

```
3152 \def\fc@ltthousandstringfrench#1{%
```

```
3153   {%
```

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

```
3154   \count4\count0 %
```

```
3155   \advance\count4 by 2 %
```

```
3156   \fc@read@unit{\count2 }{\count4 }%
```

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

```
3157   \advance\count4 by -1 %
```

```
3158   \count3\count4 %
```

```
3159   \advance\count3 by -1 %
```

```
3160   \fc@check@nonzeros{\count3 }{\count4 }\@tempa
```

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

```
3161   \edef\@temps{%
```

```
3162     \ifcase\fc@frenchoptions@cent@plural\space
```

```
3163     % 0 => always
```

```
3164     s%
```

```
3165     \or
```

```
3166     % 1 => never
```

```
3167     \or
```

```
3168     % 2 => multiple
```

```
3169     \ifnum\count2>1s\fi
```

```
3170     \or
```

```
3171     % 3 => multiple g-last
```

```
3172     \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count0=\count6s\fi\fi\fi
```

```
3173     \or
```

```
3174     % 4 => multiple l-last
```

```
3175     \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count9=0s\else\ifnum\count9=2s\fi\fi\fi\fi
```

```
3176     \fi
```

```
3177   }%
```

```
3178   % compute spacing after cent(s?) into \@tempb
```

```
3179   \expandafter\let\expandafter\@tempb
```

```
3180   \ifnum\@tempa>0 \fc@frenchoptions@submillion@dos\else\@empty\fi
```

```
3181   % now place into \@tempa the hundreds
```

```
3182   \edef\@tempa{%
```

```
3183     \ifnum\count2=0 %
```

```
3184     \else
```

```
3185     \ifnum\count2=1 %
```

```
3186     \expandafter\fc@case\@hundred\@nil
```

```
3187     \else
```

```

3188         \@unitstring{\count2}\fc@frenchoptions@submillion@dos
3189         \noexpand\fc@case\@hundred\@temps\noexpand\@nil
3190     \fi
3191     \@tempb
3192 \fi
3193 }%
3194 % now append to \@tempa the ten and unit
3195 \fc@lthundredstringfrench\@tempa

```

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

```

3196     \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
3197     \expandafter\@tempb\expandafter{\@tempa}%
3198     \expandafter
3199 } \@tempa
3200 }

```

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

```

3201 \ifcsundef{@@numberstringfrench}{-}{%
3202   \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

```

3203 \def\@@numberstringfrench#1#2{%
3204   {%

```

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

```

3205     \edef\@tempa{#1}%
3206     \expandafter\fc@number@parser\expandafter{\@tempa}%
3207     \ifnum\fc@min@weight<0 %
3208         \PackageError{fmtcount}{Out of range}%
3209         {This macro does not work with fractional numbers}%
3210     \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'.

```

3211     \edef\@tempa{\ifcase\fc@sign@case\space\or\fc@case plus\@nil\or\fc@case moins\@nil\fi}%
3212     \fc@nbrstr@preamble
3213     \fc@@nbrstrfrench@inner
3214     \fc@nbrstr@postamble

```

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

```

3215     \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
3216     \expandafter\@tempb\expandafter{\@tempa}%
3217     \expandafter
3218 } \@tempa

```

```

3219 }
\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.
3220 \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.
3221     \count0=\fc@max@weight
3222     \divide\count0 by 3 %
3223     \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.
3224     \fc@intpart@find@last{\count5 }%
3225     \count6\count5 %
3226     \divide\count6 3 %
3227     \multiply\count6 3 %
3228     \count8=0 %
3229     \loop
    First we check whether digits in weight interval  $[w..(w+2)]$  are all zero and
    place check result into macro \@tempt.
3230         \count1\count0 %
3231         \advance\count1 by 2 %
3232         \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.
3233         \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
3234         \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  $10^w$ .
3235         \fc@muladdfrench\@tempt\@tempd\@tempb
    Then iterate.
3236         \count8\count9 %
3237         \advance\count0 by -3 %
3238         \ifnum\count6>\count0 \else
3239             \repeat
3240 }

```

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

```
3241 \ifcsundef{@@ordinalstringfrench}{}{%
3242   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3243     '@@ordinalstringfrench'}}
```

Arguments are as follows:

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

```
3244 \def\@@ordinalstringfrench#1#2{%
3245   {%
```

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

```
3246   \edef\@tempa{#1}%
3247   \expandafter\fc@number@parser\expandafter{\@tempa}%
3248   \ifnum\fc@min@weight<0 %
3249     \PackageError{fmtcount}{Out of range}%
3250     {This macro does not work with fractional numbers}%
3251   \fi
3252   \ifnum\fc@sign@case>0 %
3253     \PackageError{fmtcount}{Out of range}%
3254     {This macro does with negative or explicitly marked as positive numbers}%
3255   \fi
```

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

```
3256   \ifnum\fc@max@weight=0 %
3257     \ifnum\csname fc@digit0\endcsname=1 %
3258       \count0=1 %
3259     \else
3260       \count0=0 %
3261     \fi
3262   \else
3263     \count0=0 %
3264   \fi
3265   \ifnum\count0=1 %
3266     \edef\@tempa{\expandafter\fc@case\fc@first\@nil}%
3267   \else
```

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

```
3268   \def\@tempa##1{%
3269     \expandafter\edef\csname fc@frenchoptions@@@1@plural\endcsname{%
3270       \ifcase\csname fc@frenchoptions@@@1@plural\endcsname\space
3271         0% 0: always => always
3272       \or
3273         1% 1: never => never
3274       \or
3275         6% 2: multiple => multiple   ng-last
3276       \or
```

```

3277         1% 3: multiple g-last => never
3278         \or
3279         5% 4: multiple l-last => multiple lng-last
3280         \or
3281         5% 5: multiple lng-last => multiple lng-last
3282         \or
3283         6% 6: multiple ng-last => multiple ng-last
3284         \fi
3285     }%
3286 }%
3287 \@tempa{vingt}%
3288 \@tempa{cent}%
3289 \@tempa{mil}%
3290 \@tempa{n-illion}%
3291 \@tempa{n-illiard}%

```

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

```

3292 \let\fc@case@save\fc@case
3293 \def\fc@case{\noexpand\fc@case}%
3294 \def\@nil{\noexpand\@nil}%

```

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

```

3295 \let\@tempa\@empty
3296 \fc@@nbrstrfrench@inner

```

Now restore \fc@case

```

3297 \let\fc@case\fc@case@save

```

Now we add the “ième” ending

```

3298 \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
3299 \expandafter\fc@get@last@letter\expandafter{\@tempc}\@tempd\@tempe
3300 \def\@tempf{e}%
3301 \ifx\@tempe\@tempf
3302     \edef\@tempa{\@tempb\expandafter\fc@case\@tempd i\'eme\@nil}%
3303 \else
3304     \def\@tempf{q}%
3305     \ifx\@tempe\@tempf
3306         \edef\@tempa{\@tempb\expandafter\fc@case\@tempd qui\'eme\@nil}%
3307     \else
3308         \def\@tempf{f}%
3309         \ifx\@tempe\@tempf
3310             \edef\@tempa{\@tempb\expandafter\fc@case\@tempd vi\'eme\@nil}%
3311         \else
3312             \edef\@tempa{\@tempb\expandafter\fc@case\@tempc i\'eme\@nil}%
3313         \fi
3314     \fi
3315 \fi
3316 \fi

```

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



```

3317 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
3318 \expandafter\@tempb\expandafter{\@tempa}%
3319 \expandafter
3320 }\@tempa
3321 }

```

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

```

3322 \newcommand*\fc@frenchoptions@setdefaults{%
3323 \csname KV@fcfrench@all plural\endcsname{reformed}%
3324 \def\fc@frenchoptions@submillion@dos{-}%
3325 \let\fc@frenchoptions@supermillion@dos\space
3326 \let\fc@u@in@duo\@empty% Could be ‘u’
3327 % \let\fc@poweroften\fc@@pot@longscalefrench
3328 \let\fc@poweroften\fc@@pot@recursivefrench
3329 \def\fc@longscale@nilliard@upto{0}% infinity
3330 \def\fc@frenchoptions@mil@plural@mark{le}%
3331 }
3332 \fc@frenchoptions@setdefaults

```

#### 9.4.6 fc-frenchb.def

```

3333 \ProvidesFCLanguage{frenchb}[2013/08/17]%
3334 \FCloadlang{french}%

```

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

```

3335 \global\let\@ordinalMfrenchb=\@ordinalMfrench
3336 \global\let\@ordinalFfrenchb=\@ordinalFfrench
3337 \global\let\@ordinalNfrenchb=\@ordinalNfrench
3338 \global\let\@numberstringMfrenchb=\@numberstringMfrench
3339 \global\let\@numberstringFfrenchb=\@numberstringFfrench
3340 \global\let\@numberstringNfrenchb=\@numberstringNfrench
3341 \global\let\@NumberstringMfrenchb=\@NumberstringMfrench
3342 \global\let\@NumberstringFfrenchb=\@NumberstringFfrench
3343 \global\let\@NumberstringNfrenchb=\@NumberstringNfrench
3344 \global\let\@ordinalstringMfrenchb=\@ordinalstringMfrench
3345 \global\let\@ordinalstringFfrenchb=\@ordinalstringFfrench
3346 \global\let\@ordinalstringNfrenchb=\@ordinalstringNfrench
3347 \global\let\@OrdinalstringMfrenchb=\@OrdinalstringMfrench
3348 \global\let\@OrdinalstringFfrenchb=\@OrdinalstringFfrench
3349 \global\let\@OrdinalstringNfrenchb=\@OrdinalstringNfrench

```

#### 9.4.7 fc-german.def

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

```

3350 \ProvidesFCLanguage{german}[2014/06/09]%

```

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:

```

3351 \newcommand{\@ordinalMgerman}[2]{%

```

```

3352 \edef#2{\number#1\relax.}%
3353 }%
3354 \global\let\@ordinalMgerman\@ordinalMgerman

```

Feminine:

```

3355 \newcommand{\@ordinalFgerman}[2]{%
3356 \edef#2{\number#1\relax.}%
3357 }%
3358 \global\let\@ordinalFgerman\@ordinalFgerman

```

Neuter:

```

3359 \newcommand{\@ordinalNgerman}[2]{%
3360 \edef#2{\number#1\relax.}%
3361 }%
3362 \global\let\@ordinalNgerman\@ordinalNgerman

```

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

```

3363 \newcommand*\@@unitstringgerman[1]{%
3364 \ifcase#1%
3365 null%
3366 \or ein%
3367 \or zwei%
3368 \or drei%
3369 \or vier%
3370 \or f\"unf%
3371 \or sechs%
3372 \or sieben%
3373 \or acht%
3374 \or neun%
3375 \fi
3376 }%
3377 \global\let\@@unitstringgerman\@@unitstringgerman

```

Tens (argument must go from 1 to 10):

```

3378 \newcommand*\@@tenstringgerman[1]{%
3379 \ifcase#1%
3380 \or zehn%
3381 \or zwanzig%
3382 \or drei{\ss}ig%
3383 \or vierzig%
3384 \or f\"unfzig%
3385 \or sechzig%
3386 \or siebzig%
3387 \or achtzig%
3388 \or neunzig%
3389 \or einhundert%
3390 \fi
3391 }%
3392 \global\let\@@tenstringgerman\@@tenstringgerman

```

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

```
3393 \providecommand*\einhundert{\einhundert}%
3394 \providecommand*\eintausend{\eintausend}%
3395 \global\let\einhundert\einhundert
3396 \global\let\eintausend\eintausend
```

Teens:

```
3397 \newcommand*\@@teenstringgerman[1]{%
3398   \ifcase#1%
3399     zehn%
3400   \or elf%
3401   \or zw"olf%
3402   \or dreizehn%
3403   \or vierzehn%
3404   \or f"unfzehn%
3405   \or sechzehn%
3406   \or siebzehn%
3407   \or achtzehn%
3408   \or neunzehn%
3409   \fi
3410 }%
3411 \global\let\@@teenstringgerman\@@teenstringgerman
```

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

```
3412 \DeclareRobustCommand{\@numberstringMgerman}[2]{%
3413   \let\@unitstring=\@unitstringgerman
3414   \let\@teenstring=\@teenstringgerman
3415   \let\@tenstring=\@tenstringgerman
3416   \@numberstringgerman{#1}{#2}%
3417 }%
3418 \global\let\@numberstringMgerman\@numberstringMgerman
```

Feminine and neuter forms:

```
3419 \global\let\@numberstringFgerman=\@numberstringMgerman
3420 \global\let\@numberstringNgerman=\@numberstringMgerman
```

As above, but initial letters in upper case:

```
3421 \DeclareRobustCommand{\@NumberstringMgerman}[2]{%
3422   \@numberstringMgerman{#1}{\@num@str}%
3423   \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3424 }%
3425 \global\let\@NumberstringMgerman\@NumberstringMgerman
```

Feminine and neuter form:

```
3426 \global\let\@NumberstringFgerman=\@NumberstringMgerman
3427 \global\let\@NumberstringNgerman=\@NumberstringMgerman
```

As above, but for ordinals.

```
3428 \DeclareRobustCommand{\@ordinalstringMgerman}[2]{%
3429   \let\@unitthstring=\@unitthstringMgerman
```

```

3430 \let\@teenthstring=\@teenthstringMgerman
3431 \let\@tenthstring=\@tenthstringMgerman
3432 \let\@unitstring=\@unitstringgerman
3433 \let\@teenstring=\@teenstringgerman
3434 \let\@tenstring=\@tenstringgerman
3435 \def\@thousandth{tausendster}%
3436 \def\@hundredth{hundertster}%
3437 \@@ordinalstringgerman{#1}{#2}%
3438 }%
3439 \global\let\@ordinalstringMgerman\@ordinalstringMgerman

```

Feminine form:

```

3440 \DeclareRobustCommand{\@ordinalstringFgerman}[2]{%
3441 \let\@unitthstring=\@unitthstringFgerman
3442 \let\@teenthstring=\@teenthstringFgerman
3443 \let\@tenthstring=\@tenthstringFgerman
3444 \let\@unitstring=\@unitstringgerman
3445 \let\@teenstring=\@teenstringgerman
3446 \let\@tenstring=\@tenstringgerman
3447 \def\@thousandth{tausendste}%
3448 \def\@hundredth{hundertste}%
3449 \@@ordinalstringgerman{#1}{#2}%
3450 }%
3451 \global\let\@ordinalstringFgerman\@ordinalstringFgerman

```

Neuter form:

```

3452 \DeclareRobustCommand{\@ordinalstringNgerman}[2]{%
3453 \let\@unitthstring=\@unitthstringNgerman
3454 \let\@teenthstring=\@teenthstringNgerman
3455 \let\@tenthstring=\@tenthstringNgerman
3456 \let\@unitstring=\@unitstringgerman
3457 \let\@teenstring=\@teenstringgerman
3458 \let\@tenstring=\@tenstringgerman
3459 \def\@thousandth{tausendstes}%
3460 \def\@hundredth{hunderstes}%
3461 \@@ordinalstringgerman{#1}{#2}%
3462 }%
3463 \global\let\@ordinalstringNgerman\@ordinalstringNgerman

```

As above, but with initial letters in upper case.

```

3464 \DeclareRobustCommand{\@OrdinalstringMgerman}[2]{%
3465 \@ordinalstringMgerman{#1}{\@@num@str}%
3466 \edef#2{\noexpand\MakeUppercase\expandonce\@@num@str}%
3467 }%
3468 \global\let\@OrdinalstringMgerman\@OrdinalstringMgerman

```

Feminine form:

```

3469 \DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
3470 \@ordinalstringFgerman{#1}{\@@num@str}%
3471 \edef#2{\noexpand\MakeUppercase\expandonce\@@num@str}%
3472 }%

```

```
3473 \global\let\@OrdinalstringFgerman\@OrdinalstringFgerman
```

Neuter form:

```
3474 \DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
3475 \@OrdinalstringNgerman{#1}{\@num@str}%
3476 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3477 }%
3478 \global\let\@OrdinalstringNgerman\@OrdinalstringNgerman
```

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

```
3479 \newcommand*\@unitthstringMgerman[1]{%
3480 \ifcase#1%
3481 nullter%
3482 \or erster%
3483 \or zweiter%
3484 \or dritter%
3485 \or vierter%
3486 \or f\"unfter%
3487 \or sechster%
3488 \or siebter%
3489 \or achter%
3490 \or neunter%
3491 \fi
3492 }%
3493 \global\let\@unitthstringMgerman\@unitthstringMgerman
```

Tens:

```
3494 \newcommand*\@tenthstringMgerman[1]{%
3495 \ifcase#1%
3496 \or zehnter%
3497 \or zwanzigster%
3498 \or drei{\ss}igster%
3499 \or vierzigster%
3500 \or f\"unzigster%
3501 \or sechzigster%
3502 \or siebzigster%
3503 \or achtzigster%
3504 \or neunzigster%
3505 \fi
3506 }%
3507 \global\let\@tenthstringMgerman\@tenthstringMgerman
```

Teens:

```
3508 \newcommand*\@teenthstringMgerman[1]{%
3509 \ifcase#1%
3510 zehnter%
3511 \or elfter%
3512 \or zw\"olfter%
3513 \or dreizehnter%
3514 \or vierzehnter%
```

```

3515 \or f\"unfzehnter%
3516 \or sechzehnter%
3517 \or siebzehnter%
3518 \or achtzehnter%
3519 \or neunzehnter%
3520 \fi
3521 }%
3522 \global\let\@@teenthstringMgerman\@@teenthstringMgerman

Units (feminine):
3523 \newcommand*\@@unitthstringFgerman[1]{%
3524 \ifcase#1%
3525 nullte%
3526 \or erste%
3527 \or zweite%
3528 \or dritte%
3529 \or vierte%
3530 \or f\"unfte%
3531 \or sechste%
3532 \or siebte%
3533 \or achte%
3534 \or neunte%
3535 \fi
3536 }%
3537 \global\let\@@unitthstringFgerman\@@unitthstringFgerman

Tens (feminine):
3538 \newcommand*\@@tentthstringFgerman[1]{%
3539 \ifcase#1%
3540 \or zehnte%
3541 \or zwanzigste%
3542 \or drei{\ss}igste%
3543 \or vierzigste%
3544 \or f\"unfzigste%
3545 \or sechzigste%
3546 \or siebzigste%
3547 \or achtzigste%
3548 \or neunzigste%
3549 \fi
3550 }%
3551 \global\let\@@tentthstringFgerman\@@tentthstringFgerman

Teens (feminine)
3552 \newcommand*\@@teenthstringFgerman[1]{%
3553 \ifcase#1%
3554 zehnte%
3555 \or elfte%
3556 \or zw\"olfte%
3557 \or dreizehnte%
3558 \or vierzehnte%
3559 \or f\"unfzehnte%

```

```

3560 \or sechzehnte%
3561 \or siebzehnte%
3562 \or achtzehnte%
3563 \or neunzehnte%
3564 \fi
3565 }%
3566 \global\let\@@teenthstringFgerman\@@teenthstringFgerman

Units (neuter):
3567 \newcommand*\@@unitthstringNgerman[1]{%
3568 \ifcase#1%
3569 nulltes%
3570 \or erstes%
3571 \or zweites%
3572 \or drittes%
3573 \or viertes%
3574 \or f\unftes%
3575 \or sechstes%
3576 \or siebtes%
3577 \or achtes%
3578 \or neuntes%
3579 \fi
3580 }%
3581 \global\let\@@unitthstringNgerman\@@unitthstringNgerman

Tens (neuter):
3582 \newcommand*\@@tenthstringNgerman[1]{%
3583 \ifcase#1%
3584 \or zehntes%
3585 \or zwanzigstes%
3586 \or drei\ss}igstes%
3587 \or vierzigstes%
3588 \or f\unfzigstes%
3589 \or sechzigstes%
3590 \or siebzigstes%
3591 \or achtzigstes%
3592 \or neunzigstes%
3593 \fi
3594 }%
3595 \global\let\@@tenthstringNgerman\@@tenthstringNgerman

Teens (neuter)
3596 \newcommand*\@@teenthstringNgerman[1]{%
3597 \ifcase#1%
3598 zehntes%
3599 \or elftes%
3600 \or zw\olftes%
3601 \or dreizehntes%
3602 \or vierzehntes%
3603 \or f\unfzehntes%
3604 \or sechzehntes%

```

```

3605 \or siebzehntes%
3606 \or achtzehntes%
3607 \or neunzehntes%
3608 \fi
3609 }%
3610 \global\let\@@teenthstringNgerman\@@teenthstringNgerman

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

3611 \newcommand*\@@numberunderhundredgerman[2]{%
3612 \ifnum#1<10\relax
3613 \ifnum#1>0\relax
3614 \eappto#2{\@unitstring{#1}}%
3615 \fi
3616 \else
3617 \@tmpstrctr=#1\relax
3618 \@FCmodulo{\@tmpstrctr}{10}%
3619 \ifnum#1<20\relax
3620 \eappto#2{\@teenstring{\@tmpstrctr}}%
3621 \else
3622 \ifnum\@tmpstrctr=0\relax
3623 \else
3624 \eappto#2{\@unitstring{\@tmpstrctr}und}%
3625 \fi
3626 \@tmpstrctr=#1\relax
3627 \divide\@tmpstrctr by 10\relax
3628 \eappto#2{\@tenstring{\@tmpstrctr}}%
3629 \fi
3630 \fi
3631 }%
3632 \global\let\@@numberunderhundredgerman\@@numberunderhundredgerman

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

3633 \newcommand*\@@numberstringgerman[2]{%
3634 \ifnum#1>99999\relax
3635 \PackageError{fmtcount}{Out of range}%
3636 {This macro only works for values less than 100000}%
3637 \else
3638 \ifnum#1<0\relax
3639 \PackageError{fmtcount}{Negative numbers not permitted}%
3640 {This macro does not work for negative numbers, however
3641 you can try typing "minus" first, and then pass the modulus of
3642 this number}%
3643 \fi
3644 \fi
3645 \def#2{}%
3646 \@strctr=#1\relax \divide\@strctr by 1000\relax
3647 \ifnum\@strctr>1\relax

```



```

#1 is  $\geq 2000$ , \@strctr now contains the number of thousands
3648 \@numberunderhundredgerman{\@strctr}{#2}%
3649 \appto#2{tausend}%
3650 \else
    #1 lies in range [1000,1999]
3651 \ifnum\@strctr=1\relax
3652     \eappto#2{eintausend}%
3653 \fi
3654 \fi
3655 \@strctr=#1\relax
3656 \@FCmodulo{\@strctr}{1000}%
3657 \divide\@strctr by 100\relax
3658 \ifnum\@strctr>1\relax
    now dealing with number in range [200,999]
3659 \eappto#2{\@unitstring{\@strctr}hundert}%
3660 \else
3661     \ifnum\@strctr=1\relax
        dealing with number in range [100,199]
3662         \ifnum#1>1000\relax
            if original number > 1000, use einhundert
3663             \appto#2{einhundert}%
3664         \else
            otherwise use \einhundert
3665             \eappto#2{\einhundert}%
3666         \fi
3667     \fi
3668 \fi
3669 \@strctr=#1\relax
3670 \@FCmodulo{\@strctr}{100}%
3671 \ifnum#1=0\relax
3672     \def#2{null}%
3673 \else
3674     \ifnum\@strctr=1\relax
3675         \appto#2{eins}%
3676     \else
3677         \@numberunderhundredgerman{\@strctr}{#2}%
3678     \fi
3679 \fi
3680 }%
3681 \global\let\@numberstringgerman\@numberstringgerman
    As above, but for ordinals
3682 \newcommand*\@numberunderhundredthgerman[2]{%
3683 \ifnum#1<10\relax
3684 \eappto#2{\@unitthstring{#1}}%
3685 \else
3686 \@tmpstrctr=#1\relax

```

```

3687 \@FCmodulo{\@tmpstrctr}{10}%
3688 \ifnum#1<20\relax
3689 \eappto#2{\@teenthsstring{\@tmpstrctr}}%
3690 \else
3691 \ifnum\@tmpstrctr=0\relax
3692 \else
3693 \eappto#2{\@unitstring{\@tmpstrctr}und}%
3694 \fi
3695 \@tmpstrctr=#1\relax
3696 \divide\@tmpstrctr by 10\relax
3697 \eappto#2{\@tenthsstring{\@tmpstrctr}}%
3698 \fi
3699 \fi
3700 }%
3701 \global\let\@@numberunderhundredthgerman\@@numberunderhundredthgerman

3702 \newcommand*\@@ordinalstringgerman[2]{%
3703 \ifnum#1>99999\relax
3704 \PackageError{fmtcount}{Out of range}%
3705 {This macro only works for values less than 100000}%
3706 \else
3707 \ifnum#1<0\relax
3708 \PackageError{fmtcount}{Negative numbers not permitted}%
3709 {This macro does not work for negative numbers, however
3710 you can try typing "minus" first, and then pass the modulus of
3711 this number}%
3712 \fi
3713 \fi
3714 \def#2{}%
3715 \@strctr=#1\relax \divide\@strctr by 1000\relax
3716 \ifnum\@strctr>1\relax

#1 is  $\geq 2000$ , \@strctr now contains the number of thousands
3717 \@@numberunderhundredgerman{\@strctr}{#2}%

is that it, or is there more?
3718 \@tmpstrctr=#1\relax \@FCmodulo{\@tmpstrctr}{1000}%
3719 \ifnum\@tmpstrctr=0\relax
3720 \eappto#2{\@thousandth}%
3721 \else
3722 \appto#2{tausend}%
3723 \fi
3724 \else

#1 lies in range [1000,1999]
3725 \ifnum\@strctr=1\relax
3726 \ifnum#1=1000\relax
3727 \eappto#2{\@thousandth}%
3728 \else
3729 \eappto#2{eintausend}%
3730 \fi

```

```

3731 \fi
3732 \fi
3733 \@strctr=#1\relax
3734 \@FCmodulo{\@strctr}{1000}%
3735 \divide\@strctr by 100\relax
3736 \ifnum\@strctr>1\relax
    now dealing with number in range [200,999] is that it, or is there more?
3737 \@tmpstrctr=#1\relax \@FCmodulo{\@tmpstrctr}{100}%
3738 \ifnum\@tmpstrctr=0\relax
3739 \ifnum\@strctr=1\relax
3740 \eappto#2{\@hundredth}%
3741 \else
3742 \eappto#2{\@unitstring{\@strctr}\@hundredth}%
3743 \fi
3744 \else
3745 \eappto#2{\@unitstring{\@strctr}hundert}%
3746 \fi
3747 \else
3748 \ifnum\@strctr=1\relax
    dealing with number in range [100,199] is that it, or is there more?
3749 \@tmpstrctr=#1\relax \@FCmodulo{\@tmpstrctr}{100}%
3750 \ifnum\@tmpstrctr=0\relax
3751 \eappto#2{\@hundredth}%
3752 \else
3753 \ifnum#1>1000\relax
3754 \appto#2{einhundert}%
3755 \else
3756 \eappto#2{\einhundert}%
3757 \fi
3758 \fi
3759 \fi
3760 \fi
3761 \@strctr=#1\relax
3762 \@FCmodulo{\@strctr}{100}%
3763 \ifthenelse{\@strctr=0 \and #1>0}{-}{%
3764 \@@numberunderhundredthgerman{\@strctr}{#2}%
3765 }%
3766 }%
3767 \global\let\@@ordinalstringgerman\@@ordinalstringgerman
    Load fc-germanb.def if not already loaded
3768 \FCloadlang{germanb}%

```

#### 9.4.8 fc-germanb.def

```

3769 \ProvidesFCLanguage{germanb}[2013/08/17]%
    Load fc-german.def if not already loaded
3770 \FCloadlang{german}%

```

Set germanb to be equivalent to german.

```

3771 \global\let\@ordinalMgermanb=\@ordinalMgerman
3772 \global\let\@ordinalFgermanb=\@ordinalFgerman
3773 \global\let\@ordinalNgermanb=\@ordinalNgerman
3774 \global\let\@numberstringMgermanb=\@numberstringMgerman
3775 \global\let\@numberstringFgermanb=\@numberstringFgerman
3776 \global\let\@numberstringNgermanb=\@numberstringNgerman
3777 \global\let\@NumberstringMgermanb=\@NumberstringMgerman
3778 \global\let\@NumberstringFgermanb=\@NumberstringFgerman
3779 \global\let\@NumberstringNgermanb=\@NumberstringNgerman
3780 \global\let\@ordinalstringMgermanb=\@ordinalstringMgerman
3781 \global\let\@ordinalstringFgermanb=\@ordinalstringFgerman
3782 \global\let\@ordinalstringNgermanb=\@ordinalstringNgerman
3783 \global\let\@OrdinalstringMgermanb=\@OrdinalstringMgerman
3784 \global\let\@OrdinalstringFgermanb=\@OrdinalstringFgerman
3785 \global\let\@OrdinalstringNgermanb=\@OrdinalstringNgerman

```

#### 9.4.9 fc-italian

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

```

3786 \ProvidesFCLanguage{italian}[2013/08/17]
3787
3788 \RequirePackage{itnumpar}
3789
3790 \newcommand{\@numberstringMitalian}[2]{%
3791   \edef#2{\noexpand\printnumeroinparole{#1}}%
3792 }
3793 \global\let\@numberstringMitalian\@numberstringMitalian
3794
3795 \newcommand{\@numberstringFitalian}[2]{%
3796   \edef#2{\noexpand\printnumeroinparole{#1}}%
3797 }
3798 \global\let\@numberstringFitalian\@numberstringFitalian
3799
3800 \newcommand{\@NumberstringMitalian}[2]{%
3801   \edef#2{\noexpand\printNumeroinparole{#1}}%
3802 }
3803 \global\let\@NumberstringMitalian\@NumberstringMitalian
3804
3805 \newcommand{\@NumberstringFitalian}[2]{%
3806   \edef#2{\noexpand\printNumeroinparole{#1}}%
3807 }
3808 \global\let\@NumberstringFitalian\@NumberstringFitalian
3809
3810 \newcommand{\@ordinalstringMitalian}[2]{%
3811   \edef#2{\noexpand\printordinalem{#1}}%
3812 }
3813 \global\let\@ordinalstringMitalian\@ordinalstringMitalian
3814

```

```

3815 \newcommand{\@ordinalstringFitalian}[2]{%
3816   \edef#2{\noexpand\printordinalef{#1}}%
3817 }
3818 \global\let\@ordinalstringFitalian\@ordinalstringFitalian
3819
3820 \newcommand{\@OrdinalstringMitalian}[2]{%
3821   \edef#2{\noexpand\printOrdinalem{#1}}%
3822 }
3823 \global\let\@OrdinalstringMitalian\@OrdinalstringMitalian
3824
3825 \newcommand{\@OrdinalstringFitalian}[2]{%
3826   \edef#2{\noexpand\printOrdinalef{#1}}%
3827 }
3828 \global\let\@OrdinalstringFitalian\@OrdinalstringFitalian
3829
3830 \newcommand{\@ordinalMitalian}[2]{%
3831   \edef#2{#1\relax\noexpand\fmtord{o}}%
3832 }
3833 \global\let\@ordinalMitalian\@ordinalMitalian
3834
3835 \newcommand{\@ordinalFitalian}[2]{%
3836   \edef#2{#1\relax\noexpand\fmtord{a}}%
3837 }
3838 \global\let\@ordinalFitalian\@ordinalFitalian

```

#### 9.4.10 fc-ngerman.def

```

3838 \ProvidesFCLanguage{ngerman}[2012/06/18]%
3839 \FCloadlang{german}%
3840 \FCloadlang{ngermanb}%

```

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

```

3841 \global\let\@ordinalMngerman=\@ordinalMgerman
3842 \global\let\@ordinalFngerman=\@ordinalFgerman
3843 \global\let\@ordinalNngerman=\@ordinalNgerman
3844 \global\let\@numberstringMngerman=\@numberstringMgerman
3845 \global\let\@numberstringFngerman=\@numberstringFgerman
3846 \global\let\@numberstringNngerman=\@numberstringNgerman
3847 \global\let\@NumberstringMngerman=\@NumberstringMgerman
3848 \global\let\@NumberstringFngerman=\@NumberstringFgerman
3849 \global\let\@NumberstringNngerman=\@NumberstringNgerman
3850 \global\let\@ordinalstringMngerman=\@ordinalstringMgerman
3851 \global\let\@ordinalstringFngerman=\@ordinalstringFgerman
3852 \global\let\@ordinalstringNngerman=\@ordinalstringNgerman
3853 \global\let\@OrdinalstringMngerman=\@OrdinalstringMgerman
3854 \global\let\@OrdinalstringFngerman=\@OrdinalstringFgerman
3855 \global\let\@OrdinalstringNngerman=\@OrdinalstringNgerman

```

#### 9.4.11 fc-ngermanb.def

```

3856 \ProvidesFCLanguage{ngermanb}[2013/08/17]%

```

3857 \FCloadlang{german}%

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

```
3858 \global\let\@ordinalMngermanb=\@ordinalMgerman
3859 \global\let\@ordinalFngermanb=\@ordinalFgerman
3860 \global\let\@ordinalNngermanb=\@ordinalNgerman
3861 \global\let\@numberstringMngermanb=\@numberstringMgerman
3862 \global\let\@numberstringFngermanb=\@numberstringFgerman
3863 \global\let\@numberstringNngermanb=\@numberstringNgerman
3864 \global\let\@NumberstringMngermanb=\@NumberstringMgerman
3865 \global\let\@NumberstringFngermanb=\@NumberstringFgerman
3866 \global\let\@NumberstringNngermanb=\@NumberstringNgerman
3867 \global\let\@ordinalstringMngermanb=\@ordinalstringMgerman
3868 \global\let\@ordinalstringFngermanb=\@ordinalstringFgerman
3869 \global\let\@ordinalstringNngermanb=\@ordinalstringNgerman
3870 \global\let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
3871 \global\let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
3872 \global\let\@OrdinalstringNngermanb=\@OrdinalstringNgerman
```

Load fc-ngerman.def if not already loaded

3873 \FCloadlang{ngerman}%

#### 9.4.12 fc-portuges.def

Portuguese definitions

3874 \ProvidesFCLanguage{portuges}[2014/06/09]%

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:

```
3875 \newcommand*\@ordinalMportuges[2]{%
3876   \ifnum#1=0\relax
3877     \edef#2{\number#1}%
3878   \else
3879     \edef#2{\number#1\relax\noexpand\fmtord{o}}%
3880   \fi
3881 }%
3882 \global\let\@ordinalMportuges\@ordinalMportuges
```

Feminine:

```
3883 \newcommand*\@ordinalFportuges[2]{%
3884   \ifnum#1=0\relax
3885     \edef#2{\number#1}%
3886   \else
3887     \edef#2{\number#1\relax\noexpand\fmtord{a}}%
3888   \fi
3889 }%
3890 \global\let\@ordinalFportuges\@ordinalFportuges
```

Make neuter same as masculine:

```
3891 \global\let\@ordinalNportuges\@ordinalMportuges
```

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

```

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 \global\let\@@unitstringportuges\@@unitstringportuges
3907 % \end{macrocode}
3908 % As above, but for feminine:
3909 % \begin{macrocode}
3910 \newcommand*\@@unitstringFportuges[1]{%
3911   \ifcase#1\relax
3912     zero%
3913     \or uma%
3914     \or duas%
3915     \or tr\^es%
3916     \or quatro%
3917     \or cinco%
3918     \or seis%
3919     \or sete%
3920     \or oito%
3921     \or nove%
3922   \fi
3923 }%
3924 \global\let\@@unitstringFportuges\@@unitstringFportuges

```

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

```

3925 \newcommand*\@@tenstringportuges[1]{%
3926   \ifcase#1\relax
3927     \or dez%
3928     \or vinte%
3929     \or trinta%
3930     \or quarenta%
3931     \or cinq\"uenta%
3932     \or sessenta%
3933     \or setenta%
3934     \or oitenta%
3935     \or noventa%

```

```

3936     \or cem%
3937   \fi
3938 }%
3939 \global\let\@@tenstringportuges\@@tenstringportuges

```

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

```

3940 \newcommand*\@@teenstringportuges[1]{%
3941   \ifcase#1\relax
3942     dez%
3943     \or onze%
3944     \or doze%
3945     \or treze%
3946     \or quatorze%
3947     \or quinze%
3948     \or dezesesseis%
3949     \or dezessete%
3950     \or dezoito%
3951     \or dezenove%
3952   \fi
3953 }%
3954 \global\let\@@teenstringportuges\@@teenstringportuges

```

Hundreds:

```

3955 \newcommand*\@@hundredstringportuges[1]{%
3956   \ifcase#1\relax
3957     \or cento%
3958     \or duzentos%
3959     \or trezentos%
3960     \or quatrocentos%
3961     \or quinhentos%
3962     \or seiscentos%
3963     \or setecentos%
3964     \or oitocentos%
3965     \or novecentos%
3966   \fi
3967 }%
3968 \global\let\@@hundredstringportuges\@@hundredstringportuges

```

Hundreds (feminine):

```

3969 \newcommand*\@@hundredstringFportuges[1]{%
3970   \ifcase#1\relax
3971     \or cento%
3972     \or duzentas%
3973     \or trezentas%
3974     \or quatrocentas%
3975     \or quinhentas%
3976     \or seiscentas%
3977     \or setecentas%
3978     \or oitocentas%
3979     \or novecentas%
3980   \fi

```



```

3981 }%
3982 \global\let\@@hundredstringFportuges\@@hundredstringFportuges

```

Units (initial letter in upper case):

```

3983 \newcommand*\@@Unitstringportuges[1]{%
3984   \ifcase#1\relax
3985     Zero%
3986     \or Um%
3987     \or Dois%
3988     \or Tr\^es%
3989     \or Quatro%
3990     \or Cinco%
3991     \or Seis%
3992     \or Sete%
3993     \or Oito%
3994     \or Nove%
3995   \fi
3996 }%
3997 \global\let\@@Unitstringportuges\@@Unitstringportuges

```

As above, but feminine:

```

3998 \newcommand*\@@UnitstringFportuges[1]{%
3999   \ifcase#1\relax
4000     Zera%
4001     \or Uma%
4002     \or Duas%
4003     \or Tr\^es%
4004     \or Quatro%
4005     \or Cinco%
4006     \or Seis%
4007     \or Sete%
4008     \or Oito%
4009     \or Nove%
4010   \fi
4011 }%
4012 \global\let\@@UnitstringFportuges\@@UnitstringFportuges

```

Tens (with initial letter in upper case):

```

4013 \newcommand*\@@Tenstringportuges[1]{%
4014   \ifcase#1\relax
4015     \or Dez%
4016     \or Vinte%
4017     \or Trinta%
4018     \or Quarenta%
4019     \or Cinq\"uenta%
4020     \or Sessenta%
4021     \or Setenta%
4022     \or Oitenta%
4023     \or Noventa%
4024     \or Cem%
4025   \fi

```

```

4026 }%
4027 \global\let\@@Teenstringportuges\@@Teenstringportuges

  Teens (with initial letter in upper case):
4028 \newcommand*\@@Teenstringportuges[1]{%
4029   \ifcase#1\relax
4030     Dez%
4031     \or Onze%
4032     \or Doze%
4033     \or Treze%
4034     \or Quatorze%
4035     \or Quinze%
4036     \or Dezesesseis%
4037     \or Dezessete%
4038     \or Dezoito%
4039     \or Dezenove%
4040   \fi
4041 }%
4042 \global\let\@@Teenstringportuges\@@Teenstringportuges

  Hundreds (with initial letter in upper case):
4043 \newcommand*\@@Hundredstringportuges[1]{%
4044   \ifcase#1\relax
4045     \or Cento%
4046     \or Duzentos%
4047     \or Trezentos%
4048     \or Quatrocentos%
4049     \or Quinhentos%
4050     \or Seiscentos%
4051     \or Setecentos%
4052     \or Oitocentos%
4053     \or Novecentos%
4054   \fi
4055 }%
4056 \global\let\@@Hundredstringportuges\@@Hundredstringportuges

  As above, but feminine:
4057 \newcommand*\@@HundredstringFportuges[1]{%
4058   \ifcase#1\relax
4059     \or Cento%
4060     \or Duzentas%
4061     \or Trezentas%
4062     \or Quatrocentas%
4063     \or Quinhentas%
4064     \or Seiscentas%
4065     \or Setecentas%
4066     \or Oitocentas%
4067     \or Novecentas%
4068   \fi
4069 }%
4070 \global\let\@@HundredstringFportuges\@@HundredstringFportuges

```

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

```

4071 \DeclareRobustCommand{\@numberstringMportuges}[2]{%
4072   \let\@unitstring=\@unitstringportuges
4073   \let\@teenstring=\@teenstringportuges
4074   \let\@tenstring=\@tenstringportuges
4075   \let\@hundredstring=\@hundredstringportuges
4076   \def\@hundred{cem}\def\@thousand{mil}%
4077   \def\@andname{e}%
4078   \@numberstringportuges{#1}{#2}%
4079 }%
4080 \global\let\@numberstringMportuges\@numberstringMportuges

```

As above, but feminine form:

```

4081 \DeclareRobustCommand{\@numberstringFportuges}[2]{%
4082   \let\@unitstring=\@unitstringFportuges
4083   \let\@teenstring=\@teenstringportuges
4084   \let\@tenstring=\@tenstringportuges
4085   \let\@hundredstring=\@hundredstringFportuges
4086   \def\@hundred{cem}\def\@thousand{mil}%
4087   \def\@andname{e}%
4088   \@numberstringportuges{#1}{#2}%
4089 }%
4090 \global\let\@numberstringFportuges\@numberstringFportuges

```

Make neuter same as masculine:

```

4091 \global\let\@numberstringNportuges\@numberstringMportuges

```

As above, but initial letters in upper case:

```

4092 \DeclareRobustCommand{\@NumberstringMportuges}[2]{%
4093   \let\@unitstring=\@Unitstringportuges
4094   \let\@teenstring=\@Teenstringportuges
4095   \let\@tenstring=\@Tenstringportuges
4096   \let\@hundredstring=\@Hundredstringportuges
4097   \def\@hundred{Cem}\def\@thousand{Mil}%
4098   \def\@andname{e}%
4099   \@numberstringportuges{#1}{#2}%
4100 }%
4101 \global\let\@NumberstringMportuges\@NumberstringMportuges

```

As above, but feminine form:

```

4102 \DeclareRobustCommand{\@NumberstringFportuges}[2]{%
4103   \let\@unitstring=\@UnitstringFportuges
4104   \let\@teenstring=\@Teenstringportuges
4105   \let\@tenstring=\@Tenstringportuges
4106   \let\@hundredstring=\@HundredstringFportuges
4107   \def\@hundred{Cem}\def\@thousand{Mil}%
4108   \def\@andname{e}%

```

```

4109 \@@numberstringportuges{#1}{#2}%
4110 }%
4111 \global\let\@NumberstringFportuges\@NumberstringFportuges

    Make neuter same as masculine:
4112 \global\let\@NumberstringNportuges\@NumberstringMportuges

    As above, but for ordinals.
4113 \DeclareRobustCommand{\@OrdinalstringMportuges}[2]{%
4114   \let\@unitthstring=\@unitthstringportuges
4115   \let\@unitstring=\@unitstringportuges
4116   \let\@teenthstring=\@teenthstringportuges
4117   \let\@tenthstring=\@tenthstringportuges
4118   \let\@hundredthstring=\@hundredthstringportuges
4119   \def\@thousandth{mil\'esimo}%
4120   \@@ordinalstringportuges{#1}{#2}%
4121 }%
4122 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges

    Feminine form:
4123 \DeclareRobustCommand{\@OrdinalstringFportuges}[2]{%
4124   \let\@unitthstring=\@unitthstringFportuges
4125   \let\@unitstring=\@unitstringFportuges
4126   \let\@teenthstring=\@teenthstringFportuges
4127   \let\@tenthstring=\@tenthstringFportuges
4128   \let\@hundredthstring=\@hundredthstringFportuges
4129   \def\@thousandth{mil\'esima}%
4130   \@@ordinalstringportuges{#1}{#2}%
4131 }%
4132 \global\let\@OrdinalstringFportuges\@OrdinalstringFportuges

    Make neuter same as masculine:
4133 \global\let\@OrdinalstringNportuges\@OrdinalstringMportuges

    As above, but initial letters in upper case (masculine):
4134 \DeclareRobustCommand{\@OrdinalstringMportuges}[2]{%
4135   \let\@unitthstring=\@Unitthstringportuges
4136   \let\@unitstring=\@Unitstringportuges
4137   \let\@teenthstring=\@Teenthstringportuges
4138   \let\@tenthstring=\@Tenthstringportuges
4139   \let\@hundredthstring=\@Hundredthstringportuges
4140   \def\@thousandth{Mil\'esimo}%
4141   \@@ordinalstringportuges{#1}{#2}%
4142 }%
4143 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges

    Feminine form:
4144 \DeclareRobustCommand{\@OrdinalstringFportuges}[2]{%
4145   \let\@unitthstring=\@UnitthstringFportuges
4146   \let\@unitstring=\@UnitstringFportuges
4147   \let\@teenthstring=\@Teenthstringportuges
4148   \let\@tenthstring=\@TenthstringFportuges

```

```

4149 \let\@hundredthstring=\@HundredthstringFportuges
4150 \def\@thousandth{Mil\'esima}%
4151 \@ordinalstringportuges{#1}{#2}%
4152 }%
4153 \global\let\@OrdinalstringFportuges\@OrdinalstringFportuges

    Make neuter same as masculine:
4154 \global\let\@OrdinalstringNportuges\@OrdinalstringMportuges

    In order to do the ordinals, split into units, teens, tens and hundreds. Units:
4155 \newcommand*\@unitthstringportuges[1]{%
4156   \ifcase#1\relax
4157     zero%
4158     \or primeiro%
4159     \or segundo%
4160     \or terceiro%
4161     \or quarto%
4162     \or quinto%
4163     \or sexto%
4164     \or s\'etimo%
4165     \or oitavo%
4166     \or nono%
4167   \fi
4168 }%
4169 \global\let\@unitthstringportuges\@unitthstringportuges

    Tens:
4170 \newcommand*\@tenthstringportuges[1]{%
4171   \ifcase#1\relax
4172     \or d\'ecimo%
4173     \or vig\'esimo%
4174     \or trig\'esimo%
4175     \or quadrag\'esimo%
4176     \or q\'uinquag\'esimo%
4177     \or sexag\'esimo%
4178     \or setuag\'esimo%
4179     \or octog\'esimo%
4180     \or nonag\'esimo%
4181   \fi
4182 }%
4183 \global\let\@tenthstringportuges\@tenthstringportuges

    Teens:
4184 \newcommand*\@teenthstringportuges[1]{%
4185   \@tenthstring{1}%
4186   \ifnum#1>0\relax
4187     -\@unitthstring{#1}%
4188   \fi
4189 }%
4190 \global\let\@teenthstringportuges\@teenthstringportuges

```

### Hundreds:

```
4191 \newcommand*\@@hundredthstringportuges[1]{%
4192   \ifcase#1\relax
4193     \or cent\'esimo%
4194     \or ducent\'esimo%
4195     \or trecent\'esimo%
4196     \or quadringent\'esimo%
4197     \or q\'uingent\'esimo%
4198     \or seiscent\'esimo%
4199     \or setingent\'esimo%
4200     \or octingent\'esimo%
4201     \or nongent\'esimo%
4202   \fi
4203 }%
4204 \global\let\@@hundredthstringportuges\@@hundredthstringportuges
```

### Units (feminine):

```
4205 \newcommand*\@@unitthstringFportuges[1]{%
4206   \ifcase#1\relax
4207     zero%
4208     \or primeira%
4209     \or segunda%
4210     \or terceira%
4211     \or quarta%
4212     \or quinta%
4213     \or sexta%
4214     \or s\'etima%
4215     \or oitava%
4216     \or nona%
4217   \fi
4218 }%
4219 \global\let\@@unitthstringFportuges\@@unitthstringFportuges
```

### Tens (feminine):

```
4220 \newcommand*\@@tenthstringFportuges[1]{%
4221   \ifcase#1\relax
4222     \or d\'ecima%
4223     \or vig\'esima%
4224     \or trig\'esima%
4225     \or quadrag\'esima%
4226     \or q\'uinquag\'esima%
4227     \or sexag\'esima%
4228     \or setuag\'esima%
4229     \or octog\'esima%
4230     \or nonag\'esima%
4231   \fi
4232 }%
4233 \global\let\@@tenthstringFportuges\@@tenthstringFportuges
```

### Hundreds (feminine):

```

4234 \newcommand*\@@hundredthstringFportuges[1]{%
4235   \ifcase#1\relax
4236     \or cent\'esima%
4237     \or ducent\'esima%
4238     \or trecent\'esima%
4239     \or quadringent\'esima%
4240     \or q\'uingent\'esima%
4241     \or seiscent\'esima%
4242     \or setingent\'esima%
4243     \or octingent\'esima%
4244     \or nongent\'esima%
4245   \fi
4246 }%
4247 \global\let\@@hundredthstringFportuges\@@hundredthstringFportuges

```

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

```

4248 \newcommand*\@@Unitthstringportuges[1]{%
4249   \ifcase#1\relax
4250     Zero%
4251     \or Primeiro%
4252     \or Segundo%
4253     \or Terceiro%
4254     \or Quarto%
4255     \or Quinto%
4256     \or Sexto%
4257     \or S\'etimo%
4258     \or Oitavo%
4259     \or Nono%
4260   \fi
4261 }%
4262 \global\let\@@Unitthstringportuges\@@Unitthstringportuges

```

Tens:

```

4263 \newcommand*\@@Tenthstringportuges[1]{%
4264   \ifcase#1\relax
4265     \or D\'ecimo%
4266     \or Vig\'esimo%
4267     \or Trig\'esimo%
4268     \or Quadrag\'esimo%
4269     \or Q\'uinquag\'esimo%
4270     \or Sexag\'esimo%
4271     \or Setuag\'esimo%
4272     \or Octog\'esimo%
4273     \or Nonag\'esimo%
4274   \fi
4275 }%
4276 \global\let\@@Tenthstringportuges\@@Tenthstringportuges

```

Hundreds:

```

4277 \newcommand*\@@Hundredthstringportuges[1]{%
4278   \ifcase#1\relax

```

```

4279 \or Cent\'esimo%
4280 \or Ducent\'esimo%
4281 \or Trecent\'esimo%
4282 \or Quadringent\'esimo%
4283 \or Q\'uingent\'esimo%
4284 \or Seiscent\'esimo%
4285 \or Setingent\'esimo%
4286 \or Octingent\'esimo%
4287 \or Nongent\'esimo%
4288 \fi
4289 }%
4290 \global\let\@@Hundredthstringportuges\@@Hundredthstringportuges

```

As above, but feminine. Units:

```

4291 \newcommand*\@@UnitthstringFportuges[1]{%
4292 \ifcase#1\relax
4293 Zera%
4294 \or Primeira%
4295 \or Segunda%
4296 \or Terceira%
4297 \or Quarta%
4298 \or Quinta%
4299 \or Sexta%
4300 \or S\'etima%
4301 \or Oitava%
4302 \or Nona%
4303 \fi
4304 }%
4305 \global\let\@@UnitthstringFportuges\@@UnitthstringFportuges

```

Tens (feminine);

```

4306 \newcommand*\@@TenthstringFportuges[1]{%
4307 \ifcase#1\relax
4308 \or D\'ecima%
4309 \or Vig\'esima%
4310 \or Trig\'esima%
4311 \or Quadrag\'esima%
4312 \or Q\'uinquag\'esima%
4313 \or Sexag\'esima%
4314 \or Setuag\'esima%
4315 \or Octog\'esima%
4316 \or Nonag\'esima%
4317 \fi
4318 }%
4319 \global\let\@@TenthstringFportuges\@@TenthstringFportuges

```

Hundreds (feminine):

```

4320 \newcommand*\@@HundredthstringFportuges[1]{%
4321 \ifcase#1\relax
4322 \or Cent\'esima%
4323 \or Ducent\'esima%

```



```

4324 \or Trecent\'esima%
4325 \or Quadringent\'esima%
4326 \or Q\'uigent\'esima%
4327 \or Seiscent\'esima%
4328 \or Setingent\'esima%
4329 \or Octingent\'esima%
4330 \or Nongent\'esima%
4331 \fi
4332 }%

```

```

4333 \global\let\@@HundredthstringFportuges\@@HundredthstringFportuges

```

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

(These internal macros are not meant for use in documents.)

```

4334 \newcommand*\@@numberstringportuges[2]{%
4335 \ifnum#1>99999\relax
4336 \PackageError{fmtcount}{Out of range}%
4337 {This macro only works for values less than 100000}%
4338 \else
4339 \ifnum#1<0\relax
4340 \PackageError{fmtcount}{Negative numbers not permitted}%
4341 {This macro does not work for negative numbers, however
4342 you can try typing "minus" first, and then pass the modulus of
4343 this number}%
4344 \fi
4345 \fi
4346 \def#2{}%
4347 \@strctr=#1\relax \divide\@strctr by 1000\relax
4348 \ifnum\@strctr>9\relax
4349 \divide\@strctr by 10\relax
4350 \ifnum\@strctr>1\relax
4351 \let\@@fc@numstr#2\relax
4352 \protected@edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
4353 \@strctr=#1 \divide\@strctr by 1000\relax
4354 \@FCmodulo{\@strctr}{10}%
4355 \ifnum\@strctr>0
4356 \ifnum\@strctr=1\relax
4357 \let\@@fc@numstr#2\relax
4358 \protected@edef#2{\@@fc@numstr\ \@andname}%
4359 \fi
4360 \let\@@fc@numstr#2\relax
4361 \protected@edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
4362 \fi
4363 \else
4364 \@strctr=#1\relax
4365 \divide\@strctr by 1000\relax
4366 \@FCmodulo{\@strctr}{10}%

```

```

4367 \let\@@fc@numstr#2\relax
4368 \protected@edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
4369 \fi
4370 \let\@@fc@numstr#2\relax
4371 \protected@edef#2{\@@fc@numstr\ \@thousand}%
4372 \else
4373 \ifnum\@strctr>0\relax
4374 \ifnum\@strctr>1\relax
4375 \let\@@fc@numstr#2\relax
4376 \protected@edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
4377 \fi
4378 \let\@@fc@numstr#2\relax
4379 \protected@edef#2{\@@fc@numstr\@thousand}%
4380 \fi
4381 \fi
4382 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
4383 \divide\@strctr by 100\relax
4384 \ifnum\@strctr>0\relax
4385 \ifnum#1>1000 \relax
4386 \let\@@fc@numstr#2\relax
4387 \protected@edef#2{\@@fc@numstr\ }%
4388 \fi
4389 \@tmpstrctr=#1\relax
4390 \@FCmodulo{\@tmpstrctr}{1000}%
4391 \let\@@fc@numstr#2\relax
4392 \ifnum\@tmpstrctr=100\relax
4393 \protected@edef#2{\@@fc@numstr\@tenstring{10}}%
4394 \else
4395 \protected@edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
4396 \fi%
4397 \fi
4398 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
4399 \ifnum#1>100\relax
4400 \ifnum\@strctr>0\relax
4401 \let\@@fc@numstr#2\relax
4402 \protected@edef#2{\@@fc@numstr\ \@andname\ }%
4403 \fi
4404 \fi
4405 \ifnum\@strctr>19\relax
4406 \divide\@strctr by 10\relax
4407 \let\@@fc@numstr#2\relax
4408 \protected@edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
4409 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
4410 \ifnum\@strctr>0
4411 \ifnum\@strctr=1\relax
4412 \let\@@fc@numstr#2\relax
4413 \protected@edef#2{\@@fc@numstr\ \@andname}%
4414 \else
4415 \ifnum#1>100\relax

```

```

4416         \let\@@fc@numstr#2\relax
4417         \protected@edef#2{\@@fc@numstr\ \@andname}%
4418     \fi
4419 \fi
4420 \let\@@fc@numstr#2\relax
4421 \protected@edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
4422 \fi
4423 \else
4424     \ifnum\@strctr<10\relax
4425         \ifnum\@strctr=0\relax
4426             \ifnum#1<100\relax
4427                 \let\@@fc@numstr#2\relax
4428                 \protected@edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
4429             \fi
4430         \else %(>0,<10)
4431             \let\@@fc@numstr#2\relax
4432             \protected@edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
4433         \fi
4434     \else%>10
4435         \@FCmodulo{\@strctr}{10}%
4436         \let\@@fc@numstr#2\relax
4437         \protected@edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
4438     \fi
4439 \fi
4440 }%
4441 \global\let\@@numberstringportuges\@@numberstringportuges

```

As above, but for ordinals.

```

4442 \newcommand*\@@ordinalstringportuges[2]{%
4443 \@strctr=#1\relax
4444 \ifnum#1>99999
4445 \PackageError{fmtcount}{Out of range}%
4446 {This macro only works for values less than 100000}%
4447 \else
4448 \ifnum#1<0
4449 \PackageError{fmtcount}{Negative numbers not permitted}%
4450 {This macro does not work for negative numbers, however
4451 you can try typing "minus" first, and then pass the modulus of
4452 this number}%
4453 \else
4454 \def#2{}%
4455 \ifnum\@strctr>999\relax
4456     \divide\@strctr by 1000\relax
4457     \ifnum\@strctr>1\relax
4458         \ifnum\@strctr>9\relax
4459             \@tmpstrctr=\@strctr
4460             \ifnum\@strctr<20
4461                 \@FCmodulo{\@tmpstrctr}{10}%
4462                 \let\@@fc@ordstr#2\relax
4463                 \protected@edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%

```

```

4464 \else
4465 \divide\@tmpstrctr by 10\relax
4466 \let\@@fc@ordstr#2\relax
4467 \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4468 \@tmpstrctr=\@strctr
4469 \@FCmodulo{\@tmpstrctr}{10}%
4470 \ifnum\@tmpstrctr>0\relax
4471 \let\@@fc@ordstr#2\relax
4472 \protected@edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
4473 \fi
4474 \fi
4475 \else
4476 \let\@@fc@ordstr#2\relax
4477 \protected@edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
4478 \fi
4479 \fi
4480 \let\@@fc@ordstr#2\relax
4481 \protected@edef#2{\@@fc@ordstr\@thousandth}%
4482 \fi
4483 \@strctr=#1\relax
4484 \@FCmodulo{\@strctr}{1000}%
4485 \ifnum\@strctr>99\relax
4486 \@tmpstrctr=\@strctr
4487 \divide\@tmpstrctr by 100\relax
4488 \ifnum#1>1000\relax
4489 \let\@@fc@ordstr#2\relax
4490 \protected@edef#2{\@@fc@ordstr-}%
4491 \fi
4492 \let\@@fc@ordstr#2\relax
4493 \protected@edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
4494 \fi
4495 \@FCmodulo{\@strctr}{100}%
4496 \ifnum#1>99\relax
4497 \ifnum\@strctr>0\relax
4498 \let\@@fc@ordstr#2\relax
4499 \protected@edef#2{\@@fc@ordstr-}%
4500 \fi
4501 \fi
4502 \ifnum\@strctr>9\relax
4503 \@tmpstrctr=\@strctr
4504 \divide\@tmpstrctr by 10\relax
4505 \let\@@fc@ordstr#2\relax
4506 \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4507 \@tmpstrctr=\@strctr
4508 \@FCmodulo{\@tmpstrctr}{10}%
4509 \ifnum\@tmpstrctr>0\relax
4510 \let\@@fc@ordstr#2\relax
4511 \protected@edef#2{\@@fc@ordstr-\@unitthstring{\@tmpstrctr}}%
4512 \fi

```

```

4513 \else
4514   \ifnum \@strctr=0 \relax
4515     \ifnum #1=0 \relax
4516       \let \@fc@ordstr#2 \relax
4517       \protected@edef#2{\@fc@ordstr \@unitstring{0}}%
4518     \fi
4519   \else
4520     \let \@fc@ordstr#2 \relax
4521     \protected@edef#2{\@fc@ordstr \@unitthstring{\@strctr}}%
4522   \fi
4523 \fi
4524 \fi
4525 \fi
4526 }%
4527 \global \let \@ordinalstringportuges \@ordinalstringportuges

```

#### 9.4.13 fc-portuguese.def

```

4528 \ProvidesFCLanguage{portuguese}[2014/06/09]%

```

Load fc-portuges.def if not already loaded

```

4529 \FCloadlang{portuges}%

```

Set portuguese to be equivalent to portuges.

```

4530 \global \let \@ordinalMportuguese \@ordinalMportuges
4531 \global \let \@ordinalFportuguese \@ordinalFportuges
4532 \global \let \@ordinalNportuguese \@ordinalNportuges
4533 \global \let \@numberstringMportuguese \@numberstringMportuges
4534 \global \let \@numberstringFportuguese \@numberstringFportuges
4535 \global \let \@numberstringNportuguese \@numberstringNportuges
4536 \global \let \@NumberstringMportuguese \@NumberstringMportuges
4537 \global \let \@NumberstringFportuguese \@NumberstringFportuges
4538 \global \let \@NumberstringNportuguese \@NumberstringNportuges
4539 \global \let \@ordinalstringMportuguese \@ordinalstringMportuges
4540 \global \let \@ordinalstringFportuguese \@ordinalstringFportuges
4541 \global \let \@ordinalstringNportuguese \@ordinalstringNportuges
4542 \global \let \@OrdinalstringMportuguese \@OrdinalstringMportuges
4543 \global \let \@OrdinalstringFportuguese \@OrdinalstringFportuges
4544 \global \let \@OrdinalstringNportuguese \@OrdinalstringNportuges

```

#### 9.4.14 fc-spanish.def

Spanish definitions

```

4545 \ProvidesFCLanguage{spanish}[2013/08/17]%

```

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

```

4546 \newcommand* \@ordinalMspanish[2]{%
4547   \edef#2{\number#1 \relax \noexpand \fmtord{o}}%
4548 }%

```

```

4549 \global\let\@ordinalMspanish\@ordinalMspanish
    Feminine:
4550 \newcommand{\@ordinalFspanish}[2]{%
4551   \edef#2{\number#1\relax\noexpand\fmtord{a}}%
4552 }%
4553 \global\let\@ordinalFspanish\@ordinalFspanish
    Make neuter same as masculine:
4554 \global\let\@ordinalNspanish\@ordinalMspanish
    Convert a number to text. The easiest way to do this is to break it up into units,
    tens, teens, twenties and hundreds. Units (argument must be a number from 0
    to 9):
4555 \newcommand*\@@unitstringspanish[1]{%
4556   \ifcase#1\relax
4557     cero%
4558     \or uno%
4559     \or dos%
4560     \or tres%
4561     \or cuatro%
4562     \or cinco%
4563     \or seis%
4564     \or siete%
4565     \or ocho%
4566     \or nueve%
4567   \fi
4568 }%
4569 \global\let\@@unitstringspanish\@@unitstringspanish
    Feminine:
4570 \newcommand*\@@unitstringFspanish[1]{%
4571   \ifcase#1\relax
4572     cera%
4573     \or una%
4574     \or dos%
4575     \or tres%
4576     \or cuatro%
4577     \or cinco%
4578     \or seis%
4579     \or siete%
4580     \or ocho%
4581     \or nueve%
4582   \fi
4583 }%
4584 \global\let\@@unitstringFspanish\@@unitstringFspanish
    Tens (argument must go from 1 to 10):
4585 \newcommand*\@@tenstringspanish[1]{%
4586   \ifcase#1\relax
4587     \or diez%

```

```

4588 \or veinte%
4589 \or treinta%
4590 \or cuarenta%
4591 \or cincuenta%
4592 \or sesenta%
4593 \or setenta%
4594 \or ochenta%
4595 \or noventa%
4596 \or cien%
4597 \fi
4598 }%
4599 \global\let\@@tenstringspanish\@@tenstringspanish

```

#### Teens:

```

4600 \newcommand*\@@teenstringspanish[1]{%
4601 \ifcase#1\relax
4602 diez%
4603 \or once%
4604 \or doce%
4605 \or trece%
4606 \or catorce%
4607 \or quince%
4608 \or diecis\'eis%
4609 \or diecisiete%
4610 \or dieciocho%
4611 \or diecinueve%
4612 \fi
4613 }%
4614 \global\let\@@teenstringspanish\@@teenstringspanish

```

#### Twenties:

```

4615 \newcommand*\@@twentystringspanish[1]{%
4616 \ifcase#1\relax
4617 veinte%
4618 \or veintiuno%
4619 \or veintid\'os%
4620 \or veintitr\'es%
4621 \or veinticuatro%
4622 \or veinticinco%
4623 \or veintis\'eis%
4624 \or veintisiete%
4625 \or veintiocho%
4626 \or veintinueve%
4627 \fi
4628 }%
4629 \global\let\@@twentystringspanish\@@twentystringspanish

```

#### Feminine form:

```

4630 \newcommand*\@@twentystringFspanish[1]{%
4631 \ifcase#1\relax
4632 veinte%

```

```

4633 \or veintiuna%
4634 \or veintid\'os%
4635 \or veintitr\'es%
4636 \or veinticuatro%
4637 \or veinticinco%
4638 \or veintis\'eis%
4639 \or veintisiete%
4640 \or veintiocho%
4641 \or veintinueve%
4642 \fi
4643 }%
4644 \global\let\@@twentystringFspanish\@@twentystringFspanish

```

Hundreds:

```

4645 \newcommand*\@@hundredstringspanish[1]{%
4646 \ifcase#1\relax
4647 \or ciento%
4648 \or doscientos%
4649 \or trescientos%
4650 \or cuatrocientos%
4651 \or quinientos%
4652 \or seiscientos%
4653 \or setecientos%
4654 \or ochocientos%
4655 \or novecientos%
4656 \fi
4657 }%
4658 \global\let\@@hundredstringspanish\@@hundredstringspanish

```

Feminine form:

```

4659 \newcommand*\@@hundredstringFspanish[1]{%
4660 \ifcase#1\relax
4661 \or cienta%
4662 \or doscientas%
4663 \or trescientas%
4664 \or cuatrocientas%
4665 \or quinientas%
4666 \or seiscientas%
4667 \or setecientas%
4668 \or ochocientas%
4669 \or novecientas%
4670 \fi
4671 }%
4672 \global\let\@@hundredstringFspanish\@@hundredstringFspanish

```

As above, but with initial letter uppercase:

```

4673 \newcommand*\@@Unitstringspanish[1]{%
4674 \ifcase#1\relax
4675 Cero%
4676 \or Uno%
4677 \or Dos%

```



```

4678 \or Tres%
4679 \or Cuatro%
4680 \or Cinco%
4681 \or Seis%
4682 \or Siete%
4683 \or Ocho%
4684 \or Nueve%
4685 \fi
4686 }%
4687 \global\let\@@Unitstringspanish\@@Unitstringspanish

Feminine form:
4688 \newcommand*\@@UnitstringFspanish[1]{%
4689 \ifcase#1\relax
4690 Cera%
4691 \or Una%
4692 \or Dos%
4693 \or Tres%
4694 \or Cuatro%
4695 \or Cinco%
4696 \or Seis%
4697 \or Siete%
4698 \or Ocho%
4699 \or Nueve%
4700 \fi
4701 }%
4702 \global\let\@@UnitstringFspanish\@@UnitstringFspanish

Tens:
4703 %\changes{2.0}{2012-06-18}{fixed spelling mistake (correction
4704 %provided by Fernando Maldonado)}
4705 \newcommand*\@@Tenstringspanish[1]{%
4706 \ifcase#1\relax
4707 \or Diez%
4708 \or Veinte%
4709 \or Treinta%
4710 \or Cuarenta%
4711 \or Cincuenta%
4712 \or Sesenta%
4713 \or Setenta%
4714 \or Ochenta%
4715 \or Noventa%
4716 \or Cien%
4717 \fi
4718 }%
4719 \global\let\@@Tenstringspanish\@@Tenstringspanish

Teens:
4720 \newcommand*\@@Teenstringspanish[1]{%
4721 \ifcase#1\relax
4722 Diez%

```

```

4723 \or Once%
4724 \or Doce%
4725 \or Trece%
4726 \or Catorce%
4727 \or Quince%
4728 \or Diecis\'eis%
4729 \or Diecisiete%
4730 \or Dieciocho%
4731 \or Diecinueve%
4732 \fi
4733 }%
4734 \global\let\@@Teenstringspanish\@@Teenstringspanish

Twenties:
4735 \newcommand*\@@Twentystingspanish[1]{%
4736 \ifcase#1\relax
4737 Veinte%
4738 \or Veintiuno%
4739 \or Veintid\'os%
4740 \or Veintitr\'es%
4741 \or Veinticuatro%
4742 \or Veinticinco%
4743 \or Veintis\'eis%
4744 \or Veintisiete%
4745 \or Veintiocho%
4746 \or Veintinueve%
4747 \fi
4748 }%
4749 \global\let\@@Twentystingspanish\@@Twentystingspanish

Feminine form:
4750 \newcommand*\@@TwentystringFspanish[1]{%
4751 \ifcase#1\relax
4752 Veinte%
4753 \or Veintiuna%
4754 \or Veintid\'os%
4755 \or Veintitr\'es%
4756 \or Veinticuatro%
4757 \or Veinticinco%
4758 \or Veintis\'eis%
4759 \or Veintisiete%
4760 \or Veintiocho%
4761 \or Veintinueve%
4762 \fi
4763 }%
4764 \global\let\@@TwentystringFspanish\@@TwentystringFspanish

Hundreds:
4765 \newcommand*\@@Hundredstringspanish[1]{%
4766 \ifcase#1\relax
4767 \or Ciento%

```

```

4768 \or Doscientos%
4769 \or Trescientos%
4770 \or Cuatrocientos%
4771 \or Quinientos%
4772 \or Seiscientos%
4773 \or Setecientos%
4774 \or Ochocientos%
4775 \or Novecientos%
4776 \fi
4777 }%
4778 \global\let\@@Hundredstringspanish\@@Hundredstringspanish

```

Feminine form:

```

4779 \newcommand*\@@HundredstringFspanish[1]{%
4780 \ifcase#1\relax
4781 \or Cienta%
4782 \or Doscientas%
4783 \or Trescientas%
4784 \or Cuatrocientas%
4785 \or Quinientas%
4786 \or Seiscientas%
4787 \or Setecientas%
4788 \or Ochocientas%
4789 \or Novecientas%
4790 \fi
4791 }%
4792 \global\let\@@HundredstringFspanish\@@HundredstringFspanish

```

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

```

4793 \DeclareRobustCommand{\@numberstringMspanish}[2]{%
4794 \let\@unitstring=\@unitstringspanish
4795 \let\@teenstring=\@teenstringspanish
4796 \let\@tenstring=\@tenstringspanish
4797 \let\@twentystring=\@twentystringspanish
4798 \let\@hundredstring=\@hundredstringspanish
4799 \def\@hundred{cien}\def\@thousand{mil}%
4800 \def\@andname{y}%
4801 \@numberstringspanish{#1}{#2}%
4802 }%
4803 \global\let\@numberstringMspanish\@numberstringMspanish

```

Feminine form:

```

4804 \DeclareRobustCommand{\@numberstringFspanish}[2]{%
4805 \let\@unitstring=\@unitstringFspanish
4806 \let\@teenstring=\@teenstringspanish
4807 \let\@tenstring=\@tenstringspanish
4808 \let\@twentystring=\@twentystringFspanish

```

```

4809 \let\@hundredstring=\@hundredstringFspanish
4810 \def\@hundred{cien}\def\@thousand{mil}%
4811 \def\@andname{b}%
4812 \@@numberstringspanish{#1}{#2}%
4813 }%
4814 \global\let\@numberstringFspanish\@numberstringFspanish

    Make neuter same as masculine:
4815 \global\let\@numberstringNspanish\@numberstringMspanish

    As above, but initial letters in upper case:
4816 \DeclareRobustCommand{\@NumberstringMspanish}[2]{%
4817 \let\@unitstring=\@Unitstringspanish
4818 \let\@teenstring=\@Teenstringspanish
4819 \let\@tenstring=\@Tenstringspanish
4820 \let\@twentystring=\@Twentystringspanish
4821 \let\@hundredstring=\@Hundredstringspanish
4822 \def\@andname{y}%
4823 \def\@hundred{Cien}\def\@thousand{Mil}%
4824 \@@numberstringspanish{#1}{#2}%
4825 }%
4826 \global\let\@NumberstringMspanish\@NumberstringMspanish

    Feminine form:
4827 \DeclareRobustCommand{\@NumberstringFspanish}[2]{%
4828 \let\@unitstring=\@UnitstringFspanish
4829 \let\@teenstring=\@Teenstringspanish
4830 \let\@tenstring=\@Tenstringspanish
4831 \let\@twentystring=\@TwentystringFspanish
4832 \let\@hundredstring=\@HundredstringFspanish
4833 \def\@andname{b}%
4834 \def\@hundred{Cien}\def\@thousand{Mil}%
4835 \@@numberstringspanish{#1}{#2}%
4836 }%
4837 \global\let\@NumberstringFspanish\@NumberstringFspanish

    Make neuter same as masculine:
4838 \global\let\@NumberstringNspanish\@NumberstringMspanish

    As above, but for ordinals.
4839 \DeclareRobustCommand{\@OrdinalstringMspanish}[2]{%
4840 \let\@unitthstring=\@Unitthstringspanish
4841 \let\@unitstring=\@Unitstringspanish
4842 \let\@teenthstring=\@Teenthstringspanish
4843 \let\@tenthstring=\@Tenthstringspanish
4844 \let\@hundredthstring=\@Hundredthstringspanish
4845 \def\@thousandth{mil'esimo}%
4846 \@@ordinalstringspanish{#1}{#2}%
4847 }%
4848 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish

    Feminine form:

```

```

4849 \DeclareRobustCommand{\@OrdinalstringFspanish}[2]{%
4850   \let\@unitthstring=\@@unitthstringFspanish
4851   \let\@unitstring=\@@unitstringFspanish
4852   \let\@teenthstring=\@@teenthstringFspanish
4853   \let\@tenthstring=\@@tenthstringFspanish
4854   \let\@hundredthstring=\@@hundredthstringFspanish
4855   \def\@thousandth{mil\'esima}%
4856   \@@ordinalstringspanish{#1}{#2}%
4857 }%
4858 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish

```

Make neuter same as masculine:

```

4859 \global\let\@OrdinalstringNspanish\@OrdinalstringMspanish

```

As above, but with initial letters in upper case.

```

4860 \DeclareRobustCommand{\@OrdinalstringMspanish}[2]{%
4861   \let\@unitthstring=\@@Unitthstringspanish
4862   \let\@unitstring=\@@Unitstringspanish
4863   \let\@teenthstring=\@@Teenthstringspanish
4864   \let\@tenthstring=\@@Tenthstringspanish
4865   \let\@hundredthstring=\@@Hundredthstringspanish
4866   \def\@thousandth{Mil\'esimo}%
4867   \@@ordinalstringspanish{#1}{#2}%
4868 }
4869 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish

```

Feminine form:

```

4870 \DeclareRobustCommand{\@OrdinalstringFspanish}[2]{%
4871   \let\@unitthstring=\@@UnitthstringFspanish
4872   \let\@unitstring=\@@UnitstringFspanish
4873   \let\@teenthstring=\@@TeenthstringFspanish
4874   \let\@tenthstring=\@@TenthstringFspanish
4875   \let\@hundredthstring=\@@HundredthstringFspanish
4876   \def\@thousandth{Mil\'esima}%
4877   \@@ordinalstringspanish{#1}{#2}%
4878 }%
4879 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish

```

Make neuter same as masculine:

```

4880 \global\let\@OrdinalstringNspanish\@OrdinalstringMspanish

```

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

```

4881 \newcommand*\@@unitthstringspanish[1]{%
4882   \ifcase#1\relax
4883     cero%
4884     \or primero%
4885     \or segundo%
4886     \or tercero%
4887     \or cuarto%
4888     \or quinto%

```

```

4889 \or sexto%
4890 \or s\'eptimo%
4891 \or octavo%
4892 \or noveno%
4893 \fi
4894 }%
4895 \global\let\@@unitthstringspanish\@@unitthstringspanish

```

#### Tens:

```

4896 \newcommand*\@@tenthstringspanish[1]{%
4897 \ifcase#1\relax
4898 \or d\'ecimo%
4899 \or vig\'esimo%
4900 \or trig\'esimo%
4901 \or cuadrag\'esimo%
4902 \or quincuag\'esimo%
4903 \or sexag\'esimo%
4904 \or septuag\'esimo%
4905 \or octog\'esimo%
4906 \or nonag\'esimo%
4907 \fi
4908 }%
4909 \global\let\@@tenthstringspanish\@@tenthstringspanish

```

#### Teens:

```

4910 \newcommand*\@@teenthstringspanish[1]{%
4911 \ifcase#1\relax
4912 d\'ecimo%
4913 \or und\'ecimo%
4914 \or duod\'ecimo%
4915 \or decimotercero%
4916 \or decimocuarto%
4917 \or decimoquinto%
4918 \or decimosexto%
4919 \or decimos\'eptimo%
4920 \or decimoctavo%
4921 \or decimonoveno%
4922 \fi
4923 }%
4924 \global\let\@@teenthstringspanish\@@teenthstringspanish

```

#### Hundreds:

```

4925 \newcommand*\@@hundredthstringspanish[1]{%
4926 \ifcase#1\relax
4927 \or cent\'esimo%
4928 \or ducent\'esimo%
4929 \or tricent\'esimo%
4930 \or cuadringent\'esimo%
4931 \or quingent\'esimo%
4932 \or sexcent\'esimo%
4933 \or septing\'esimo%

```

```

4934 \or octingent\'esimo%
4935 \or noningent\'esimo%
4936 \fi
4937 }%
4938 \global\let\@@hundredthstringFspanish\@@hundredthstringFspanish

```

Units (feminine):

```

4939 \newcommand*\@@unitthstringFspanish[1]{%
4940 \ifcase#1\relax
4941 \or primera%
4942 \or segunda%
4943 \or tercera%
4944 \or cuarta%
4945 \or quinta%
4946 \or sexta%
4947 \or septima%
4948 \or octava%
4949 \or novena%
4950 \or decima%
4951 \fi
4952 }%
4953 \global\let\@@unitthstringFspanish\@@unitthstringFspanish

```

Tens (feminine):

```

4954 \newcommand*\@@tenthstringFspanish[1]{%
4955 \ifcase#1\relax
4956 \or d\'ecima%
4957 \or vig\'esima%
4958 \or trig\'esima%
4959 \or cuadrag\'esima%
4960 \or quincuag\'esima%
4961 \or sexag\'esima%
4962 \or septuag\'esima%
4963 \or octog\'esima%
4964 \or nonag\'esima%
4965 \fi
4966 }%
4967 \global\let\@@tenthstringFspanish\@@tenthstringFspanish

```

Teens (feminine)

```

4968 \newcommand*\@@teenthstringFspanish[1]{%
4969 \ifcase#1\relax
4970 \or und\'ecima%
4971 \or duod\'ecima%
4972 \or decimotercera%
4973 \or decimocuarta%
4974 \or decimoquinta%
4975 \or decimosexta%
4976 \or decimos\'eptima%
4977 \or decimo\'octava%
4978 \or decimo\'novenaria%

```

```

4979     \or decimonovena%
4980   \fi
4981 }%
4982 \global\let\@@teenthstringFspanish\@@teenthstringFspanish

  Hundreds (feminine)
4983 \newcommand*\@@hundredthstringFspanish[1]{%
4984   \ifcase#1\relax
4985     \or cent\'esima%
4986     \or ducent\'esima%
4987     \or tricent\'esima%
4988     \or cuadringent\'esima%
4989     \or quingent\'esima%
4990     \or sexcent\'esima%
4991     \or septing\'esima%
4992     \or octingent\'esima%
4993     \or noningent\'esima%
4994   \fi
4995 }%
4996 \global\let\@@hundredthstringFspanish\@@hundredthstringFspanish

  As above, but with initial letters in upper case
4997 \newcommand*\@@Unitthstringspanish[1]{%
4998   \ifcase#1\relax
4999     Cero%
5000     \or Primero%
5001     \or Segundo%
5002     \or Tercero%
5003     \or Cuarto%
5004     \or Quinto%
5005     \or Sexto%
5006     \or S\'eptimo%
5007     \or Octavo%
5008     \or Noveno%
5009   \fi
5010 }%
5011 \global\let\@@Unitthstringspanish\@@Unitthstringspanish

  Tens:
5012 \newcommand*\@@Tenthstringspanish[1]{%
5013   \ifcase#1\relax
5014     \or D\'ecimo%
5015     \or Vig\'esimo%
5016     \or Trig\'esimo%
5017     \or Cuadrag\'esimo%
5018     \or Quincuag\'esimo%
5019     \or Sexag\'esimo%
5020     \or Septuag\'esimo%
5021     \or Octog\'esimo%
5022     \or Nonag\'esimo%
5023   \fi

```



```

5024 }%
5025 \global\let\@@Tenthstringspanish\@@Tenthstringspanish

  Teens:
5026 \newcommand*\@@Teenthstringspanish[1]{%
5027   \ifcase#1\relax
5028     D\'ecimo%
5029     \or Und\'ecimo%
5030     \or Duod\'ecimo%
5031     \or Decimotercero%
5032     \or Decimocuarto%
5033     \or Decimoquinto%
5034     \or Decimosexto%
5035     \or Decimos\'eptimo%
5036     \or Decimoctavo%
5037     \or Decimonoveno%
5038   \fi
5039 }%
5040 \global\let\@@Teenthstringspanish\@@Teenthstringspanish

  Hundreds
5041 \newcommand*\@@Hundredthstringspanish[1]{%
5042   \ifcase#1\relax
5043     \or Cent\'esimo%
5044     \or Ducent\'esimo%
5045     \or Tricent\'esimo%
5046     \or Cuadringent\'esimo%
5047     \or Quingent\'esimo%
5048     \or Sexcent\'esimo%
5049     \or Septing\'esimo%
5050     \or Octingent\'esimo%
5051     \or Noningent\'esimo%
5052   \fi
5053 }%
5054 \global\let\@@Hundredthstringspanish\@@Hundredthstringspanish

  As above, but feminine.
5055 \newcommand*\@@UnitthstringFspanish[1]{%
5056   \ifcase#1\relax
5057     Cera%
5058     \or Primera%
5059     \or Segunda%
5060     \or Tercera%
5061     \or Cuarta%
5062     \or Quinta%
5063     \or Sexta%
5064     \or S\'eptima%
5065     \or Octava%
5066     \or Novena%
5067   \fi
5068 }%

```

```

5069 \global\let\@@UnitthstringFspanish\@@UnitthstringFspanish
      Tens (feminine)
5070 \newcommand*\@@TenthstringFspanish[1]{%
5071   \ifcase#1\relax
5072     \or D\'ecima%
5073     \or Vig\'esima%
5074     \or Trig\'esima%
5075     \or Cuadrag\'esima%
5076     \or Quincuag\'esima%
5077     \or Sexag\'esima%
5078     \or Septuag\'esima%
5079     \or Octog\'esima%
5080     \or Nonag\'esima%
5081   \fi
5082 }%
5083 \global\let\@@TenthstringFspanish\@@TenthstringFspanish
      Teens (feminine):
5084 \newcommand*\@@TeenthstringFspanish[1]{%
5085   \ifcase#1\relax
5086     D\'ecima%
5087     \or Und\'ecima%
5088     \or Duod\'ecima%
5089     \or Decimotercera%
5090     \or Decimocuarta%
5091     \or Decimoquinta%
5092     \or Decimosexta%
5093     \or Decimos\'eptima%
5094     \or Decimoctava%
5095     \or Decimonovena%
5096   \fi
5097 }%
5098 \global\let\@@TeenthstringFspanish\@@TeenthstringFspanish
      Hundreds (feminine):
5099 \newcommand*\@@HundredthstringFspanish[1]{%
5100   \ifcase#1\relax
5101     \or Cent\'esima%
5102     \or Ducent\'esima%
5103     \or Tricent\'esima%
5104     \or Cuadringent\'esima%
5105     \or Quingent\'esima%
5106     \or Sexcent\'esima%
5107     \or Septing\'esima%
5108     \or Octingent\'esima%
5109     \or Noningent\'esima%
5110   \fi
5111 }%
5112 \global\let\@@HundredthstringFspanish\@@HundredthstringFspanish

```

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

```

5113 \newcommand*\@@numberstringspanish[2]{%
5114 \ifnum#1>99999
5115 \PackageError{fmtcount}{Out of range}%
5116 {This macro only works for values less than 100000}%
5117 \else
5118 \ifnum#1<0
5119 \PackageError{fmtcount}{Negative numbers not permitted}%
5120 {This macro does not work for negative numbers, however
5121 you can try typing "minus" first, and then pass the modulus of
5122 this number}%
5123 \fi
5124 \fi
5125 \def#2{}%
5126 \@strctr=#1\relax \divide\@strctr by 1000\relax
5127 \ifnum\@strctr>9
    #1 is greater or equal to 10000
5128 \divide\@strctr by 10
5129 \ifnum\@strctr>1
5130 \let\@@fc@numstr#2\relax
5131 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
5132 \@strctr=#1 \divide\@strctr by 1000\relax
5133 \@FCmodulo{\@strctr}{10}%
5134 \ifnum\@strctr>0\relax
5135 \let\@@fc@numstr#2\relax
5136 \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
5137 \fi
5138 \else
5139 \@strctr=#1\relax
5140 \divide\@strctr by 1000\relax
5141 \@FCmodulo{\@strctr}{10}%
5142 \let\@@fc@numstr#2\relax
5143 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
5144 \fi
5145 \let\@@fc@numstr#2\relax
5146 \edef#2{\@@fc@numstr\ \@thousand}%
5147 \else
5148 \ifnum\@strctr>0\relax
5149 \ifnum\@strctr>1\relax
5150 \let\@@fc@numstr#2\relax
5151 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
5152 \fi
5153 \let\@@fc@numstr#2\relax
5154 \edef#2{\@@fc@numstr\@thousand}%
5155 \fi

```

```

5156 \fi
5157 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
5158 \divide\@strctr by 100\relax
5159 \ifnum\@strctr>0\relax
5160   \ifnum#1>1000\relax
5161     \let\@fc@numstr#2\relax
5162     \edef#2{\@fc@numstr\ }%
5163   \fi
5164   \@tmpstrctr=#1\relax
5165   \@FCmodulo{\@tmpstrctr}{1000}%
5166   \ifnum\@tmpstrctr=100\relax
5167     \let\@fc@numstr#2\relax
5168     \edef#2{\@fc@numstr\@tenstring{10}}%
5169   \else
5170     \let\@fc@numstr#2\relax
5171     \edef#2{\@fc@numstr\@hundredstring{\@strctr}}%
5172   \fi
5173 \fi
5174 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
5175 \ifnum#1>100\relax
5176   \ifnum\@strctr>0\relax
5177     \let\@fc@numstr#2\relax
5178     \edef#2{\@fc@numstr\ }%
5179   \fi
5180 \fi
5181 \ifnum\@strctr>29\relax
5182   \divide\@strctr by 10\relax
5183   \let\@fc@numstr#2\relax
5184   \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
5185   \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
5186   \ifnum\@strctr>0\relax
5187     \let\@fc@numstr#2\relax
5188     \edef#2{\@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
5189   \fi
5190 \else
5191   \ifnum\@strctr<10\relax
5192     \ifnum\@strctr=0\relax
5193       \ifnum#1<100\relax
5194         \let\@fc@numstr#2\relax
5195         \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
5196       \fi
5197     \else
5198       \let\@fc@numstr#2\relax
5199       \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
5200     \fi
5201   \else
5202     \ifnum\@strctr>19\relax
5203       \@FCmodulo{\@strctr}{10}%
5204       \let\@fc@numstr#2\relax

```

```

5205     \edef#2{\@fc@numstr\@twentystring{\@strctr}}%
5206     \else
5207         \@FCmodulo{\@strctr}{10}%
5208         \let\@fc@numstr#2\relax
5209         \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
5210     \fi
5211 \fi
5212 \fi
5213 }%
5214 \global\let\@numberstringspanish\@numberstringspanish

```

As above, but for ordinals

```

5215 \newcommand*\@ordinalstringspanish[2]{%
5216 \@strctr=#1\relax
5217 \ifnum#1>99999
5218 \PackageError{fmtcount}{Out of range}%
5219 {This macro only works for values less than 100000}%
5220 \else
5221 \ifnum#1<0
5222 \PackageError{fmtcount}{Negative numbers not permitted}%
5223 {This macro does not work for negative numbers, however
5224 you can try typing "minus" first, and then pass the modulus of
5225 this number}%
5226 \else
5227 \def#2{}%
5228 \ifnum\@strctr>999\relax
5229     \divide\@strctr by 1000\relax
5230     \ifnum\@strctr>1\relax
5231         \ifnum\@strctr>9\relax
5232             \@tmpstrctr=\@strctr
5233             \ifnum\@strctr<20
5234                 \@FCmodulo{\@tmpstrctr}{10}%
5235                 \let\@fc@ordstr#2\relax
5236                 \edef#2{\@fc@ordstr\@teenthstring{\@tmpstrctr}}%
5237             \else
5238                 \divide\@tmpstrctr by 10\relax
5239                 \let\@fc@ordstr#2\relax
5240                 \edef#2{\@fc@ordstr\@tenthstring{\@tmpstrctr}}%
5241                 \@tmpstrctr=\@strctr
5242                 \@FCmodulo{\@tmpstrctr}{10}%
5243                 \ifnum\@tmpstrctr>0\relax
5244                     \let\@fc@ordstr#2\relax
5245                     \edef#2{\@fc@ordstr\@unitthstring{\@tmpstrctr}}%
5246                 \fi
5247             \fi
5248         \else
5249             \let\@fc@ordstr#2\relax
5250             \edef#2{\@fc@ordstr\@unitstring{\@strctr}}%
5251         \fi
5252     \fi

```

```

5253 \let\@@fc@ordstr#2\relax
5254 \edef#2{\@@fc@ordstr\@thousandth}%
5255 \fi
5256 \@strctr=#1\relax
5257 \@FCmodulo{\@strctr}{1000}%
5258 \ifnum\@strctr>99\relax
5259 \@tmpstrctr=\@strctr
5260 \divide\@tmpstrctr by 100\relax
5261 \ifnum#1>1000\relax
5262 \let\@@fc@ordstr#2\relax
5263 \edef#2{\@@fc@ordstr\ }%
5264 \fi
5265 \let\@@fc@ordstr#2\relax
5266 \edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
5267 \fi
5268 \@FCmodulo{\@strctr}{100}%
5269 \ifnum#1>99\relax
5270 \ifnum\@strctr>0\relax
5271 \let\@@fc@ordstr#2\relax
5272 \edef#2{\@@fc@ordstr\ }%
5273 \fi
5274 \fi
5275 \ifnum\@strctr>19\relax
5276 \@tmpstrctr=\@strctr
5277 \divide\@tmpstrctr by 10\relax
5278 \let\@@fc@ordstr#2\relax
5279 \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
5280 \@tmpstrctr=\@strctr
5281 \@FCmodulo{\@tmpstrctr}{10}%
5282 \ifnum\@tmpstrctr>0\relax
5283 \let\@@fc@ordstr#2\relax
5284 \edef#2{\@@fc@ordstr\ \@unitthstring{\@tmpstrctr}}%
5285 \fi
5286 \else
5287 \ifnum\@strctr>9\relax
5288 \@FCmodulo{\@strctr}{10}%
5289 \let\@@fc@ordstr#2\relax
5290 \edef#2{\@@fc@ordstr\@teenthstring{\@strctr}}%
5291 \else
5292 \ifnum\@strctr=0\relax
5293 \ifnum#1=0\relax
5294 \let\@@fc@ordstr#2\relax
5295 \edef#2{\@@fc@ordstr\@unitstring{0}}%
5296 \fi
5297 \else
5298 \let\@@fc@ordstr#2\relax
5299 \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
5300 \fi
5301 \fi

```

```

5302 \fi
5303 \fi
5304 \fi
5305 }%
5306 \global\let\@@ordinalstringspanish\@@ordinalstringspanish

```

#### 9.4.15 fc-UKenglish.def

English definitions

```
5307 \ProvidesFCLanguage{UKenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
5308 \FCLoadlang{english}%
```

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

```

5309 \global\let\@@ordinalMUKenglish\@@ordinalMenglish
5310 \global\let\@@ordinalFUKenglish\@@ordinalMenglish
5311 \global\let\@@ordinalNUKenglish\@@ordinalMenglish
5312 \global\let\@@numberstringMUKenglish\@@numberstringMenglish
5313 \global\let\@@numberstringFUKenglish\@@numberstringMenglish
5314 \global\let\@@numberstringNUKenglish\@@numberstringMenglish
5315 \global\let\@@NumberstringMUKenglish\@@NumberstringMenglish
5316 \global\let\@@NumberstringFUKenglish\@@NumberstringMenglish
5317 \global\let\@@NumberstringNUKenglish\@@NumberstringMenglish
5318 \global\let\@@ordinalstringMUKenglish\@@ordinalstringMenglish
5319 \global\let\@@ordinalstringFUKenglish\@@ordinalstringMenglish
5320 \global\let\@@ordinalstringNUKenglish\@@ordinalstringMenglish
5321 \global\let\@@OrdinalstringMUKenglish\@@OrdinalstringMenglish
5322 \global\let\@@OrdinalstringFUKenglish\@@OrdinalstringMenglish
5323 \global\let\@@OrdinalstringNUKenglish\@@OrdinalstringMenglish

```

#### 9.4.16 fc-USenglish.def

US English definitions

```
5324 \ProvidesFCLanguage{USenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
5325 \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.)

```

5326 \global\let\@@ordinalMUSenglish\@@ordinalMenglish
5327 \global\let\@@ordinalFUSenglish\@@ordinalMenglish
5328 \global\let\@@ordinalNUSenglish\@@ordinalMenglish
5329 \global\let\@@numberstringMUSenglish\@@numberstringMenglish
5330 \global\let\@@numberstringFUSenglish\@@numberstringMenglish
5331 \global\let\@@numberstringNUSenglish\@@numberstringMenglish
5332 \global\let\@@NumberstringMUSenglish\@@NumberstringMenglish
5333 \global\let\@@NumberstringFUSenglish\@@NumberstringMenglish
5334 \global\let\@@NumberstringNUSenglish\@@NumberstringMenglish

```

5335 \global\let\@ordinalstringMUSenglish\@ordinalstringMenglish  
5336 \global\let\@ordinalstringFUSenglish\@ordinalstringMenglish  
5337 \global\let\@ordinalstringNUSenglish\@ordinalstringMenglish  
5338 \global\let\@OrdinalstringMUSenglish\@OrdinalstringMenglish  
5339 \global\let\@OrdinalstringFUSenglish\@OrdinalstringMenglish  
5340 \global\let\@OrdinalstringNUSenglish\@OrdinalstringMenglish